Contents lists available at ScienceDirect

ISPRS Journal of Photogrammetry and Remote Sensing

journal homepage: www.elsevier.com/locate/isprsjprs

Enhancing LiDAR point cloud generation with BRDF-based appearance modelling

Alfonso López *, Carlos J. Ogayar, Rafael J. Segura, Juan C. Casas-Rosa

Department of Computer Science, University of Jaén, Campus Las Lagunillas s/n, Jaén, 23071, Spain

A R T I C L E I N F O

Keywords:
LiDAR simulation
Virtual laser scanner
Bidirectional reflectance distribution function
Graphics processing unit

A B S T R A C T

This work presents an approach to generating LiDAR point clouds with empirical intensity data on a massively parallel scale. Our primary aim is to complement existing real-world LiDAR datasets by simulating a wide spectrum of attributes, ensuring our generated data can be directly compared to real point clouds. However, our emphasis lies in intensity data, which conventionally has been generated using non-photorealistic shading functions. In contrast, we represent surfaces with Bidirectional Reflectance Distribution Functions (BRDF) obtained through goniophotometer measurements. We also incorporate refractivity indices derived from prior research. Beyond this, we simulate other attributes commonly found in LiDAR datasets, including RGB values, normal vectors, GPS timestamps, semantic labels, instance IDs, and return data. Our simulations extend beyond terrestrial scenarios; we encompass mobile and aerial scans as well. Our results demonstrate the efficiency of our solution compared to other state-of-the-art simulators, achieving an average decrease in simulation time of 85.62%. Notably, our approach introduces greater variability in the generated intensity data, accounting for material properties and variations caused by the incident and viewing vectors. The source code is available on GitHub (https://github.com/AlfonsoLRz/LiDAR_BRDF).

1. Introduction

LiDAR (Light Detection and Ranging) point clouds have become increasingly popular in various applications such as autonomous driving (Manivasagam et al., 2020), land mapping and urban planning (Zhou et al., 2022). Current trends in these applications involve training Machine Learning (ML) models using large point cloud datasets. However, successfully training these networks often requires additional data, particularly in supervised learning scenarios. Semantic labels, instance numbers (Rozenberszki et al., 2024), and intensity (Díaz-Medina et al., 2023) are crucial. From these, only semantic labels are available in the most used datasets (see Table 1). Although manual annotation as well as unsupervised and supervised ML models can infer attributes such as semantic labels, these approaches may introduce erroneous data that can mislead the algorithms trained with this information. These limitations highlight the importance of virtual laser scanners (VLS), which is the primary focus of this work.

A plethora of repositories gathers LiDAR point clouds, offering valuable insights into this field (Cai et al., 2022). However, many of these repositories primarily focus on urban scenarios for autonomous driving applications. Despite LiDAR remaining a powerful sensing technology, photogrammetry is a cost-efficient alternative. Nevertheless, it comes with some trade-offs, including the generation of lower-quality point clouds and increased susceptibility to geometrical errors. Moreover, the pixel-level features derived from photogrammetry differ from those obtained through LiDAR sensing. Still, both approaches share some common challenges: the time-consuming processes of collecting, cleaning, and augmenting the data, which typically requires human supervision. In addition, publicly available datasets may not suffix specific application requirements regarding data density, attributes, kinds of scenes, recording platforms, etc. For example, most of the airborne datasets are collected by governmental institutions for land inventorying, using mid-altitude sensors that record only a few points per squared metre (U.S. Geological Survey, 2012; Instituto Geográfico de Información Geográfica, 2023). Instead, synthetic datasets created from realistic scenarios emulating real-world conditions offer a more reasonable alternative. Unlike real-world objects, synthetic models lack uncertainty and can be associated with semantic labels and materials, among other features attached to 3D points via VLS.

In recent years, simulators have gained interest as time and cost-effective tools for generating large datasets that enable trainable models to seek patterns in various scene perception tasks such as semantic segmentation, instance segmentation and classification applied over indoor and outdoor scenarios. Most of them emulate LiDAR and Radar

* Corresponding author.
E-mail address: allopezr@ujaen.es (A. López).

https://doi.org/10.1016/j.isprsjprs.2025.02.010
Received 27 October 2023; Received in revised form 9 February 2025; Accepted 9 February 2025
Available online 27 February 2025

Table 1
Comparison of features provided by the most frequently used LiDAR datasets, LiDAR simulators (*), and our study.$^+$ implies it can be computed in post-processing.

	Dataset/Software tool*							
Attribute	Ours	SemanticKITTI	nuScenes	SemanticPOSS	Toronto-3D	Semantic3D	DALES	HELIOS++ *
x, y, z	✓	✓	✓	✓	✓	✓	✓	✓
r, g, b	✓	✓	✓	✗	✓	✓	✗	✗
Intensity	✓	✗	✗	✗	✓	✓	✗	✓
Return number	✓	✗	✗	✗	✗	✗	✓	✓
Number of returns	✓	✗	✗	✗	✗	✗	✓	✓
Scan angle rank$^+$	✓	✗	✗	✗	✓	✗	✗	✗
GPS time	✓(Normalized)	✗	✗	✗	✓	✗	✗	✓
Normal vector	✓	✗	✗	✗	✗	✗	✗	✗
Custom semantic label	✓	✓(28)	✓(23)	✓(14)	✗(8)	✓(8)	✓(8)	✓
LAS semantic label	✓	✗	✗	✗	✗	✗	✗	✗
Instance	✓	✗	✗	✗	✓	✗	✗	✓
Sensor platform	Static/Mobile/Aerial	Mobile	Mobile	Mobile	Mobile	Static	Aerial	Static/Mobile/Aerial

sensors to acquire information on surfaces, objects and phenomena, using a wide variety of sensors regarding the platform they are mounted on (static or mobile), manufacturers and capabilities (data acquisition speed, number of channels, maximum distance, field of view, precision, etc.) (Poux, 2019). Among the available platforms, terrestrial, airborne and mobile LiDAR scanning are the most frequently simulated, mainly addressing the geometric features of resulting point clouds. However, other features relevant to the classification of LiDAR point clouds, such as intensity and return number (ordered id of the return within a pulse), are not as frequently simulated.

This study presents a GPU-based (Graphics Processing Unit) LiDAR simulator that enables the generation of large-scale point cloud datasets by operating on highly detailed static and procedural scenes. The simulator encompasses terrestrial and airborne scans and various scan deflectors to replicate the two most prevalent types of LiDAR point clouds. Simulations can be conducted in a single location or following a path. For airborne scenarios, automated path computation ensures coverage with the necessary overlapping. The efficiency of the proposed simulator is particularly noteworthy when it comes to larger missions involving multiple locations. To achieve this, the simulator utilizes state-of-the-art spatial indexing data structures for efficient ray tracing. In addition to providing semantic labels and instance numbers, the simulator integrates models with materials characterized by reflectance and refractivity signatures across a wide spectral range. These material signatures are derived from prior research focused on measuring the light scattering of different materials. In this manner, the simulator benefits from a solid foundation and enhances the realism of the generated point clouds. It is important to highlight that this simulator represents an advancement over the previous work described in López et al. (2022). The predecessor faced significant time bottlenecks generating LiDAR beams and did not emulate intensity using empirical Bidirectional Reflectance Distribution Functions (BRDF). This work addresses these limitations, improving efficiency and a more accurate representation of LiDAR intensity values through BRDFs.

In summary, this GPU-based LiDAR simulator offers a solution for generating large-scale point cloud datasets with a wide number of features, including spatial coordinates, RGB shading, intensity, return number, number of returns, scan rank, normal vector, semantic labels and instances' id. Some of these features are not provided by real datasets, e.g., normal vectors. On the other hand, the availability of the rest of the features is inconsistent across most used LiDAR datasets, as reported in Table 1.

2. Previous work

Previous work regarding LiDAR simulation will be discussed according to the following five factors: (1) input scenarios, (2) spatial indexing of scenarios, (3) simulation of the Time of Flight (ToF) principle, (4) simulation of intensity and (5) efficiency.

2.1. Input data and sampling

Input scenarios. The versatility of the proposed LiDAR simulator in adapting to different digital models and generating procedural scenarios opens up the possibility of producing extensive datasets, regardless of the simulation process itself. It is important to note that the majority of previous works have primarily focused on ad-hoc environments that lack realism (Winiwarter et al., 2022; Bechtold and Höfle, 2016; Haider et al., 2022; Dayal et al., 2021; Kukko and Hyyppä, 2009). Amongst these, procedural forests, urban scenarios and ad-hoc setups with low level of detail (LOD) are the most frequently found in the literature. Otherwise, more detailed forestry can be generated from inventory information (Schäfer et al., 2023), and triangle meshes can be modelled by professionals with a high LOD (Dosovitskiy et al., 2017; Xiao et al., 2021). Scenarios with lower complexity are easier to traverse and enable faster simulations at the expense of producing less realistic point clouds. On the other hand, working with scenes of higher LOD is far more time-consuming and they have been previously simplified to voxels (Winiwarter et al., 2022). In addition, the intersections of rays and voxels are more efficiently solved (Majercik et al., 2018) than collisions between rays and triangles (Möller and Trumbore, 1997). Another possible drawback of input scenarios is the degree of knowledge about them. For example, realistic scenarios from videogames have been previously used in the generation of LiDAR datasets; however, there is limited semantic information associated with intersected models (Yue et al., 2018; Wu et al., 2019). Similarly, combining real scans with simulated ones introduces limitations in semantic information due to the uncertainty in real point clouds (Fang et al., 2020; Manivasagam et al., 2020).

Spatial indexing. Many laser scanning simulators model beams as rays originating from the sensor's emitter, bounded by the maximum range. Therefore, efficient data structures are required to organize the scenario and enable rapid intersection calculations between rays and objects. Ray tracers commonly adopt the Boundary Volume Hierarchy (BVH) as a popular solution for addressing this problem (Riordan et al., 2021; Chen and Müller, 2022). In this regard, extensive research has been dedicated to optimizing the construction and traversal of BVHs. Other not-that-frequent and less efficient data structures are kd-trees (Winiwarter et al., 2022; Bechtold and Höfle, 2016) and octrees (Weiser et al., 2021). kd-trees and octrees calculate even partitions of the scenario, while BVHs generate tighter partitions, diminishing the traversal time.

2.2. LiDAR simulation and computational complexity

Simulation of ToF principle. The earliest LiDAR simulators generated visually plausible point clouds (Gschwandtner et al., 2011) by simulating the sensor's field of view, object occlusion and beam divergence. To this end, LiDAR beams were represented as rays and the collisions were solved with ray-casting, also referred to as ray-tracing in the literature. However, ray tracers emulate light interaction

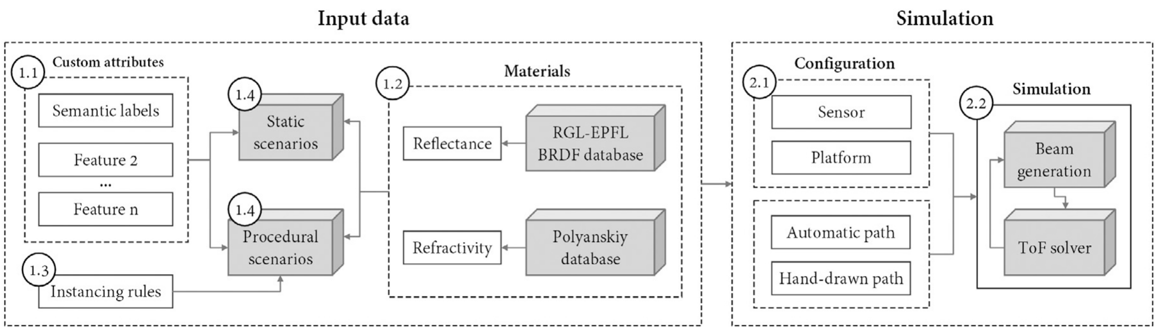


Fig. 1. Overview of the proposed LiDAR simulation. First, semantic labels and materials from a BRDF database are attached to objects from static and procedural scenes. Then, a virtual LiDAR system recreates the simulation with (1) scanning specifications, (2) from a platform and (3) following a path or placed in a static location. The simulation is split into two stages: beam generation and Time of Flight (ToF) solver. In the last stage, intensity and other relevant attributes are attached to 3D points.

between surrounding surfaces, which is hard to recreate in real-time. Modelling each LiDAR pulse with multiple rays is not that frequent, and instead, beams have been modelled with a single ray if only the first impact is necessary. If pulses are recreated using a set of rays, these carry out energy and spread to detect one or multiple collisions (Zohdi, 2020; Winiwarter et al., 2022; Bechtold and Höfle, 2016). In addition, Winiwarter et al. (2022) normalized the power weight within the beam cone, hence simulating higher power in the centre. Primarily, simulated rays spread according to the sensor's location and the scan deflection that indicates the outgoing direction. Another critical aspect is the beam divergence and radius resulting from collimated and diverging rays. Scanning patterns, including parallel, zig-zag and elliptical, hold particular significance in airborne missions.

Full-waveform simulators have also had a notable presence in previous research. Instead of providing intensity data in a single wavelength, full-waveform LiDAR sensors record the signature across a wide spectral range. Well-known full-waveform LiDAR simulators are HELIOS++ (Winiwarter et al., 2022) and DART (Discrete Anisotropic Radiative Transfer) (Yang et al., 2022). For instance, HELIOS++ has been studied in the inventory of forestry areas, helping to determine the vegetation density at different altitudes (Schäfer et al., 2023). While their output is notably different, beams are simulated using multiple subrays.

Sensor imperfections resulting from systematic and random errors are crucial in LiDAR simulation. Multiple effects can be covered and even tailored to specific sensor models (Haider et al., 2022). Systematic errors arise from various sources, including laser detector bias, deviations of beams caused by surface shininess and slope, misalignments of the inertial navigation system (INS), etc. Random errors, on the other hand, are influenced by various factors such as the electronics accuracy (including INS and GPS), as well as inherent LiDAR features like laser beam divergence, wavelength, and object reflectivity. Simulation of errors related to environmental conditions such as rain, snow, fog and haze (Zhao et al., 2021; Dosovitskiy et al., 2017; Bechtold and Höfle, 2016; Hanke et al., 2017), the drop-off in intensity (Ahn et al., 2020; Zhao et al., 2021), sensor noise (López et al., 2022; Haider et al., 2022), ranging errors (Zhao et al., 2021; López et al., 2022), atmospheric attenuation (Haider et al., 2022), return losses (Dosovitskiy et al., 2017) and fast motion scan effect (dSPACE, 2024; Chen and Müller, 2022) are the most frequent. While physically modelled rays are the primary approach, errors can also be simulated by perturbing ideal point clouds with Deep Learning (DL) (Manivasagam et al., 2020; Xiao et al., 2021). The combination of actual LiDAR data and virtually scanned objects has also been explored (Manivasagam et al., 2020), although this approach still faces challenges related to human supervision for real data. Despite the relevance of these effects, it is important to note that point clouds are frequently voxelized in deep learning training pipelines. This process leads to information loss unless models employ alternative approaches, such as adapting the data convolution pipeline to handle sparse point clouds (e.g., sparse convolutions) or transforming the data feeding method (e.g., using raw unstructured point clouds or spherical frustums (Zheng et al., 2023)).

LiDAR scans are typically performed following sequential steps across a vehicle path. Consequently, the emitter location varies, and the path can be determined automatically, defined by the user, or randomized using a set of navigable roads. However, simulating LiDAR scans with varying emitter locations and realistic vehicle paths poses certain challenges. The automatic computation of paths is a more complex approach, particularly when considering occlusion effects (Bechtold and Höfle, 2016), and leads to another research field called Planning for Scanning (P4S). On the other hand, allowing users to define paths for LiDAR scans has been proposed in previous works (Bechtold and Höfle, 2016; Winiwarter et al., 2022). This approach provides more control over the simulated scans, enabling users to define specific trajectories that align with their requirements. Additionally, the generation of completely procedural paths has been explored in works such as Dosovitskiy et al. (2017). This solution can generate paths in either a synchronous or asynchronous manner, with the asynchronous generation being faster as it is not constrained to a real-time simulation.

Simulation of intensity. The intensity simulation has been addressed in previous works; however, many focus on ad-hoc simulators designed for specific wavelengths or sensors. For example, Zohdi (2020) computed relative reflectivity without considering material properties. In some studies, analytic BRDFs have been employed to compute the returned reflectance using different models, including Lambertian, Oren-Nayar, and Blinn-Phong reflectance models (Chen and Müller, 2022; Gschwandtner et al., 2011). These are simplifications intended to emulate specific surfaces and do not consider the operating wavelength. Similarly, the earliest BRDF databases only recorded RGB values (Serrano et al., 2016), whereas the most recent collect a wider spectral interval. Still, RGB values are relevant for recording data similar to cameras coupled to LiDAR systems, despite not being directly integrated into scanners. More recently, Winiwarter et al. (2022) calculated received power based on factors such as incidence angle, beam divergence and sensor properties. They used the BRDFs of Meerdink et al. (2019), covering a wide range of wavelengths, but they are not dependent on the viewing and outgoing vectors. Even though the viewing and outgoing vectors are parallel in LiDAR systems, real-world BRDFs vary the scattered light based on the viewing/outgoing vector. Nonetheless, Winiwarter et al. (2022) incorporated full-waveform signatures besides discrete returns. In contrast, the DART simulator (Yang et al., 2022) comprises a physically-based radiative model that can process the BRDF hemispheres with custom angular resolution. However, they can only handle one wavelength in the input data, whereas our work can trivially adapt to sensors operating in a different wavelength. Although DART provides the most sophisticated radiative model, Winiwarter et al. (2022) proved their simulator produced similar results in full-waveform-based LiDAR simulations. Accordingly, we have utilized HELIOS++ as the reference work in our experiments due to its recent development and ease of replication.

Materials have also been modelled using a reflectivity percentage (Zhao et al., 2021). For instance, Haider et al. (2022) conducted experiments with 5% reflective point scatter targets. More recent studies have incorporated diffuse, specular, and transmissive factors obtained from commercial frameworks like CarMaker (Haider et al., 2022). Finally, another approach is to estimate intensity with DL from LiDAR point clouds projected into images (Vacek et al., 2022; Xiao et al., 2021). However, it is important to note that these models are conditioned by the available training data, including lighting conditions and materials observed during training.

Efficiency. Simulating LiDAR point clouds can be time-consuming, particularly when applied to complex scenarios or scanning sessions across paths. While efficient data structures are commonly used to improve performance, evaluating the overall efficiency of the simulation goes beyond data structures. Previous works have employed different frameworks and techniques to improve simulation efficiency. Peinecke et al. (2008) utilized OpenGL's shaders to recreate a LiDAR sensor, while Riordan et al. (2021) and Chen and Müller (2022) employed the Nvidia OptiX ray-tracing framework. Among these, only the work of Chen and Müller (2022) demonstrated real-time performance, achieving simulation times ranging from 19 ms/scan to 200 ms/scan. In another approach, Su et al. (2019) simulated a LiDAR sensor using OpenGL's fragment shaders to emulate the ToF principle. They projected geometry into an image and estimated distances from the depth information, thus enabling the direct use of DL models. However, this approach may lack precision due to the limited geometric and radiometric resolution. The work of Winiwarter et al. (2022) was presented as an improvement over their predecessor (Bechtold and Höfle, 2016), and reported their performance over three different scenes. The simulation time ranged from less than 1 s to several dozens of seconds in more complex scenarios.

2.3. LiDAR datasets

The vast majority of LiDAR datasets focus on autonomous driving and perception tasks such as classification, semantic segmentation, instance segmentation and tracking of mobile objects (Chen et al., 2022), using geometrical (Behley et al., 2021) and intensity information (Tan et al., 2020). Most of them record metropolitan environments from a mobile vehicle coupled with a LiDAR, including visible and infrared cameras in a few case studies (Choi et al., 2018). Recently, LiDAR datasets have been released with video and audio feed (Piadyk et al., 2023). On the other hand, aerial LiDAR datasets covering large portions of the Earth's surface are frequently published by governmental institutions at land inventorying portals. Since these are intended to cover large areas, they are acquired from mid-altitude platforms with a density of a few points per squared metre (approximately 10 points/m^2 on average in the fourth Dutch airborne laser scanning campaign (Gao et al., 2024)). Otherwise, there are denser airborne LiDAR datasets (Varney et al., 2020).

Both kinds of datasets must be annotated to train DL networks. Previously revised datasets are recorded by real sensors, and the classification is performed manually (Behley et al., 2021; Pan et al., 2020; Tan et al., 2020) or using models trained with limited data (Wu et al., 2019). Manual labelling induces errors, but another shortcoming is the lack of LOD. In this regard, these are the number of semantic labels of some widespread real datasets (Cai et al., 2022): SemanticKITTI (28 labels) (Behley et al., 2021), nuScenes (23) (Caesar et al., 2019), SemanticPOSS (14) (Pan et al., 2020), Toronto-3D (8) (Tan et al., 2020) and Semantic3D (8) (Hackel et al., 2017). Available datasets also present considerable differences regarding (1) the recorded scenario, (2) the recording device and (3) the path planning. Therefore, these differences have their impact on the available features, shown in Table 1, as well as in the dimensionality and density of the captured point clouds. These gaps harden the training of DL networks over multiple

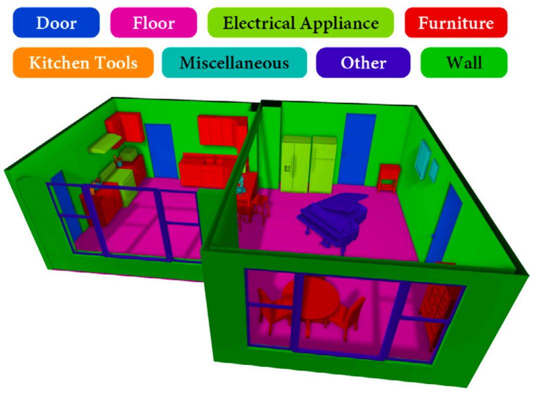


Fig. 2. Semantic labelling of a CAD scenario with 8 and 6 tags, respectively.

datasets, whereas VLS helps to alleviate this problem by emulating the recording conditions of one dataset to increase its size.

This paper extends our previous work (López et al., 2022) by efficiently simulating radiometric data in 3D point clouds obtained from models linked to semantic labels and materials. These features are intended to build more complete synthetic LiDAR datasets that can be leveraged with real-world datasets in training pipelines. Scenarios modelled by professionals and procedural environments, together with GPU-based scanning, enable the generation of large datasets with semantic and radiometric data. In addition, the pipeline has been adapted to lower the simulation time by minimizing the data transfers between CPU and GPU.

3. Methodology

This section describes the components of the system, depicted in Fig. 1, and provides an overview of the simulator. The methodology is split as follows: (1) the use of synthetic environments, (2) the indexing of virtual scenes in an efficient spatial data structure, and (3) the scan simulation over virtual environments in the GPU.

3.1. Virtual scenarios

Virtual scans can be performed over indoor and outdoor scenarios. This work uses indoor scenarios derived from Computer-Aided Design (CAD) models and procedurally generated outdoor scenarios. Although the latter falls outside the primary focus of this study, it enables a diverse array of scenarios. In contrast, CAD scenarios such as those published by McGuire (2017) are static and only enable collecting a few point clouds. Instead of manually labelling large LiDAR point clouds, human operators are required to annotate the digital models with semantic labels. In this manner, scenarios are rapidly labelled by looking for a pattern in the names of models or materials. Names of objects are primarily used; otherwise, the procedure falls back on the names of the materials. A name is considered not representative if it collides with the names of other digital models (e.g., `Mesh.001`). The links between semantic labels and names are provided as an XML together with the 3D digital model, reaching any LOD (see Fig. 2). In this work, virtual models are annotated with (1) labels with a custom LOD and (2) standard labels from the LAS (LASer) file format (American Society for Photogrammetry and Remote Sensing (ASPRS), 2019). Besides labels, models are linked to materials that help to compute the returned intensity in Section Surface modelling.

The LiDAR returns are estimated by ray-casting from the sensor towards the virtual geometry. To efficiently solve intersections between millions of rays and triangles, scenarios are organized into a BVH constructed in the GPU as described by Meister and Bittner (2018). This binary tree comprises leaf nodes containing polygons bounded

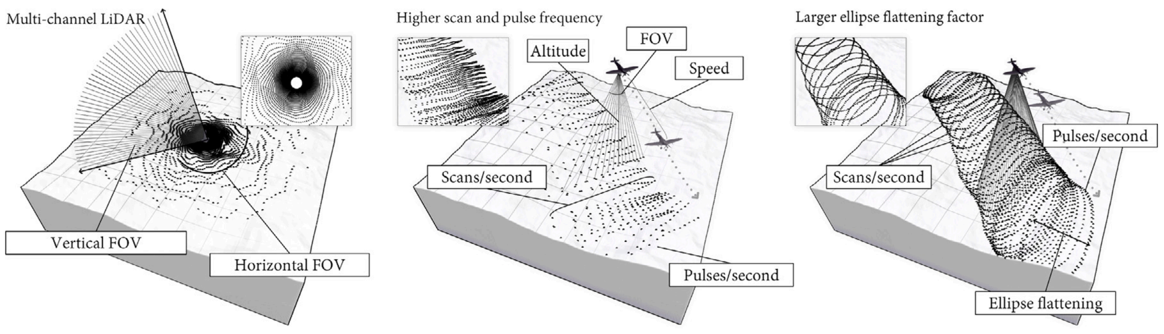


Fig. 3. Overview of TLS and ALS simulations. Aerial scans are depicted with two different scan deflectors: zigzag and ellipsoidal.

by their Axis-Aligned Bounding Box (AABB). Intermediate nodes are created by collapsing AABBs from the bottom up. Once constructed, multiple threads can work in parallel to traverse the scene for each ray, starting from the root and descending into the leaves. Ray-AABB and ray-triangle intersection tests are efficiently solved with Majercik et al. (2018) and Möller and Trumbore (1997) algorithms, respectively.

3.2. Simulation

The implementation and rendering of the proposed GPU-based LiDAR system are explained in this section, covering the scan principles and platforms. This simulation offers physically-based intensity for ALS and TLS over medium and large-scale datasets. This section is organized as follows: first, the structure of LiDAR beams is described, and then, the workflow of the GPU-based scan is explained.

3.2.1. Pulse modelling

LiDAR pulses are modelled as rays produced from the sensor's emitter, spreading towards uniformly sampled points from a unit sphere. It is assumed that the sensor emitter and receiver occupy the same location since this will simplify this process and later stages. Eq. (1) shows how ToF LiDAR systems work. First, the distance of the surface that returned the backscattered energy, R, is calculated according to the time delay, t_s, and the speed of light, c.

$$R = \frac{1}{2} \cdot c \cdot t_s \tag{1}$$

Furthermore, LiDAR pulses have a footprint that must be considered for simulating multiple returns. Therefore, pulses are better simulated with multiple rays (Zohdi, 2020; Winiwarter et al., 2022), as depicted in Fig. 4, that spreads within a radius in a parallel (collimated beams) or diverging manner. Rays from the same pulse are computed using an orthonormal basis composed by the ray direction, $\hat{r_d}$, an up vector, $\hat{up}$, and a vector $\hat{u}$ that is orthogonal to the previous two. The randomization in this work follows a uniform distribution in $[-1, 1]$ that, however, can be trivially transformed into other distributions (e.g., Gaussian) (see Fig. 5). This is possible since a large buffer comprising random values is generated to be accessed by individual threads. The overall size is a multiple of the number of subrays, albeit generally much smaller than the number of rays (hence, threads access circularly using the *module* operator). Also, the same random buffer can be applied to other randomized stages.

Then, $\hat{u}$ and $\hat{up}$ are randomly scaled to generate points within a circumference of radius p_r. In TLS simulations, the space subdivision is mainly guided by the vertical and horizontal resolution, with the first being related to the number of channels. Also, the field of view (FOV) does not always cover 360°× 90°; indeed, it is frequent to have a very narrow vertical FOV.

Accordingly, the horizontal FOV covers from $\varrho_{orig_{xy}}$ to $\varrho_{orig_{xy}} + \varrho_{xy}$, with $\varrho_{orig_{xy}}$ being the starting angle and ϱ_{xy} the covered horizontal angle, with $\varrho_{xy} \in [0, 360[$. The vertical FOV is calculated similarly.

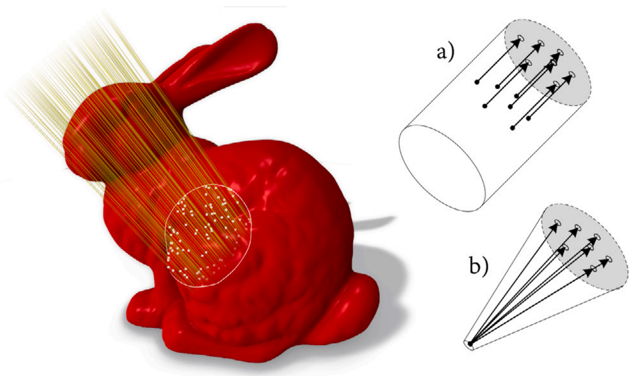
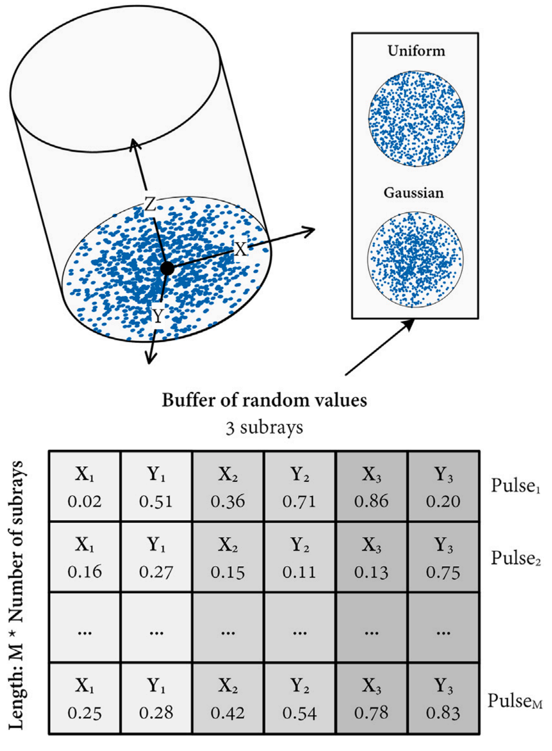


Fig. 4. Collimated rays simulating a pulse. The right side shows the scheme of (a) collimated and (b) diverging rays.



X_1 0.02	Y_1 0.51	X_2 0.36	Y_2 0.71	X_3 0.86	Y_3 0.20	Pulse_1
X_1 0.16	Y_1 0.27	X_2 0.15	Y_2 0.11	X_3 0.13	Y_3 0.75	Pulse_2
...	...	...	...	...	...	
X_1 0.25	Y_1 0.28	X_2 0.42	Y_2 0.54	X_3 0.78	Y_3 0.83	Pulse_M

Fig. 5. Random buffer for generating subrays in a LiDAR pulse. The distribution can be adjusted to represent the behaviour of LiDAR pulses, e.g., by focusing rays on the centre as in Winiwarter et al. (2022). It is depicted as a matrix, while the underlying form is a linear buffer that can be used for other randomized processes.

Nonetheless, equidistant sampling leads to a visually unpleasant aliasing effect due to an insufficient sampling rate, hence not meeting the Nyquist criterion. Visually, this is frequently handled by jittering the rays' target (Akenine-Möller et al., 2018), a technique that proves useful for emulating pulse jittering and minor inaccuracies observed in real sensors (McManamon, 2019).

In contrast to TLS, ALS simulations always follow a path. The FOV is typically more narrowed to avoid capturing atmospheric returns, and the jittering affects the emitted rays and the height of the mobile platform. The resolution is determined by the number of scans and pulses per second. Instead of exclusively performing parallel scans over the environment, other patterns help to optimize the point cloud coverage. Fig. 3 shows zigzag and elliptical patterns as described by Dong and Chen (2018), together with some of the configurable parameters.

The computational demands of these simulations are substantial for scans with high resolution or extended durations. Therefore, beams can be massively generated in the GPU, with the primary challenge being the limited GPU memory. In this regard, OpenGL buffers can expand up to a few gigabytes regardless of the GPU VRAM (Video Random Access Memory). Consequently, long scans must be completed in several iterations, working over buffers allocated only once. The maximum number of rays that can be simultaneously operated is computed according to the maximum capacity of an OpenGL buffer, the size of the information comprised in a ray (in bytes) and the number of subrays simulating a pulse, n_r. The overall number of rays must be a multiple of n_r since subrays behave synchronously and cannot be split across different executions. Unlike modern programming languages and libraries, OpenGL's compute shaders do not provide a built-in generator of pseudorandom numbers. Therefore, we lighten this computational effort by using the previous random buffer. It is generated in the CPU once, transferred to the GPU and accessed by threads.

3.2.2. Path design

Another key in mobile simulations is the design of paths. TLS simulations can be performed by following a user-defined route at a constant height (e.g., the height at which a LiDAR sensor is mounted on the roof of a vehicle). On the other hand, the path of ALS simulations can be automatically calculated, with parallel lines whose distance is computed with the extent of the scenario, the sensor's FOV and the required overlapping. Otherwise, the path can be designed over a canvas (the rendering surface of our application). Similarly to TLS, the height is constant, though it can vary with noise of configurable magnitude. One shortcoming of paths designed in a canvas is that they are captured in every screen frame, leading to many similar and noisy points. We coped with this by simplifying the path using the Douglas–Peucker algorithm (Douglas and Peucker, 1973) with a variable threshold, and then, these points were used as control points for a Catmull–Rom spline curve that will be later sampled according to the scanning resolution. This routine is easily integrated into the LiDAR solver as it simply involves changing the sensor's location and propagating beams according to the platform navigation. Manual and automatically generated paths are subsampled in the CPU and transferred to a GPU buffer, where paths composed of multiple lines are transferred as different buffers to avoid miscalculating the platform direction.

The path design enables the simulation of different mobile sensors, including ALS, MLS and TLS coupled on vehicles. A few examples of this are illustrated in Fig. 6. Similarly, Winiwarter et al. (2022) support trajectories which are later interpolated during simulation; however, we facilitate this by providing a canvas where they can be designed.

3.2.3. ToF solver

Propagated TLS and ALS rays interact with geometry to construct dense point clouds with augmented data (semantic, instances, etc.). Rays are processed by following the set of synchronous stages depicted in Fig. 7, regardless of the scanning platform. Particularities of TLS and ALS are addressed using shader subroutines instead of different pipelines. The GPU-based stages manipulate and share the SSBO (Shader Storage Buffer Object) buffers to avoid CPU data transfers, with an explicit barrier preventing different stages from working asynchronously.

According to the illustrated stages, the virtual LiDAR system behaves as follows:

1. The energy carried out by each ray within a pulse is calculated, which will affect the measured intensity. The id of the return is set to zero.
2. Each ray finds the nearest intersected surface and polygon, if any. Ray-AABB and ray-triangle intersection tests are used while traversing the BVH.
3. Following, rays within a pulse behave synchronously. This is especially relevant to simulate multiple returns in scenarios such as forests, where rays that do not terminate early penetrate high and low vegetation, and therefore, are more likely to reach the ground surface. Although multiple, disparate, collisions can be detected, only the nearest one is valid in each iteration. Thus, threads that impacted the same piece of surface or did not collide are terminated, whereas the rest still propagate. Several collisions are considered to impact the same piece whether they belong to surrounding polygons, bounded by a distance of $2 \cdot d \cdot p_r(2 - \hat{n} \cdot \hat{r}_d)$, with d being the distance to the sensor's origin, p_r the pulse radius and $\hat{n}, \hat{r}_d$ the surface normal and ray direction, respectively. Note that the amount of collided rays determines the returned intensity.
   The detected collisions are discarded if they exceed the sensor's maximum range. It is randomized with a user-defined magnitude to avoid sharp boundaries. Surface slope also influences the distortion of returned collisions, both on vertical and horizontal axes (Deems et al., 2013). The magnitude of these errors is highly dependent on sensor distance, and therefore, this error mainly affects aerial surveys (Hodgson and Bresnahan, 2004). Finally, a significant number of material errors are derived from shiny, highly reflective surfaces, that were described in a previous work.
4. Next, radiometric intensity data is computed for objects whose reflectance is modelled with the so-called BRDF. The intensity values are influenced by several factors, defined in the LiDAR equation which will be presented below. Therefore, this stage depends on the collision metadata as well as the sensor capabilities. For example, LiDAR sensors can operate at multiple wavelengths, which notably conditions the outcomes. An example is the bathymetric LiDAR (532 nm), which can capture points from shallow underwater surfaces.
5. Outliers simulate errors caused by environmental conditions, including temperature, atmospheric pressure variations, dust or steam (Boehler and Marbs, 2018). To this end, a variable part of the recorded collisions is translated using the ray parametric form ($p = r_o + r_d \cdot t$) and a random value assigned to the distance t, which is particularly relevant to determine the spatial distribution of outliers.
6. Finally, returns within a pulse are sorted to assign the return id, from 1 to $n_{returns}$, by following the pointer between consecutive impacts. This feature is relevant for filtering out returns but the last one from each pulse, thus extracting the ground points to build Digital Elevation Models (DEM). This can be done using the factor calculated from $id_{return}/n_{returns}$.

3.3. Surface modelling

In this work, synthetic models are associated with materials and semantic labels at different LODs. The materials include reflectance and refractivity signatures. Reflectance models the ratio between incoming and outgoing radiance across the wavelength spectrum and is obtained

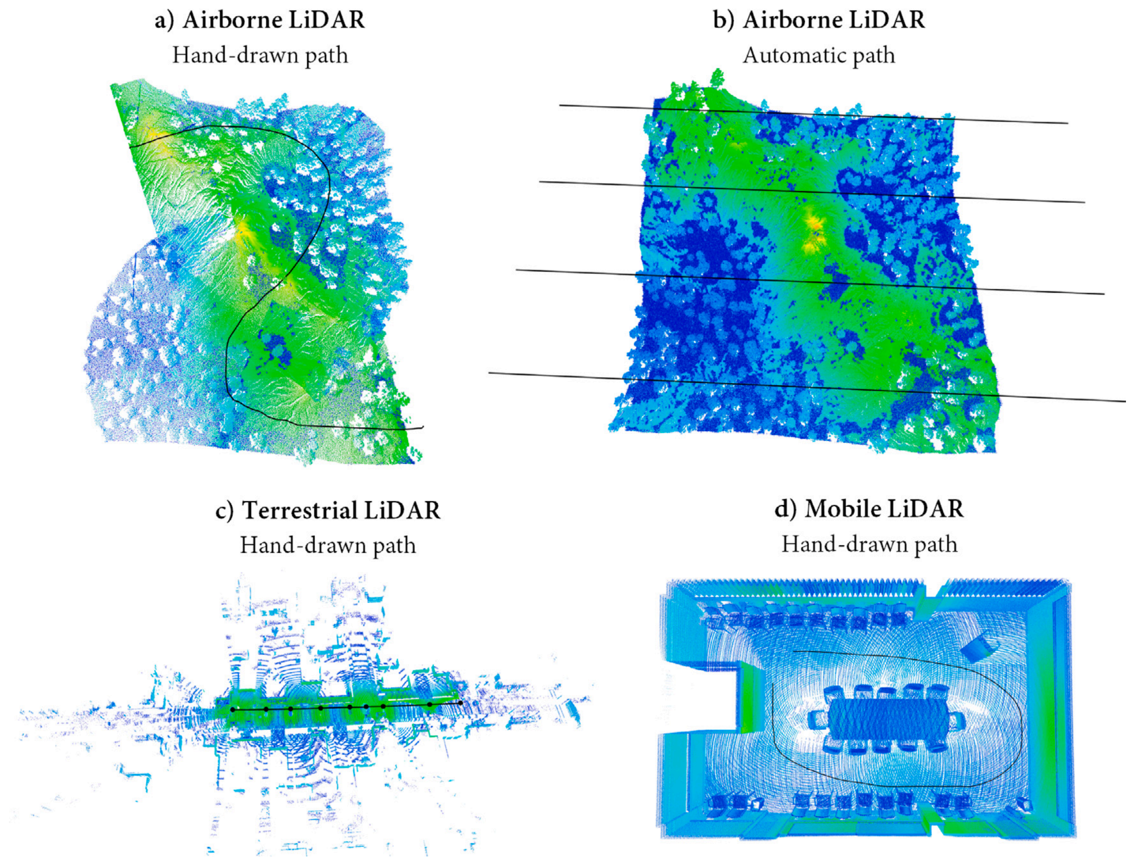


Fig. 6. Four different path-driven simulations. (a) ALS conducted with a manually sketched path, (b) ALS performed across a computer-generated path, (c) TLS operating at a height of 2 m in an urban context and (d) MLS following a manually sketched path in an indoor scenario.

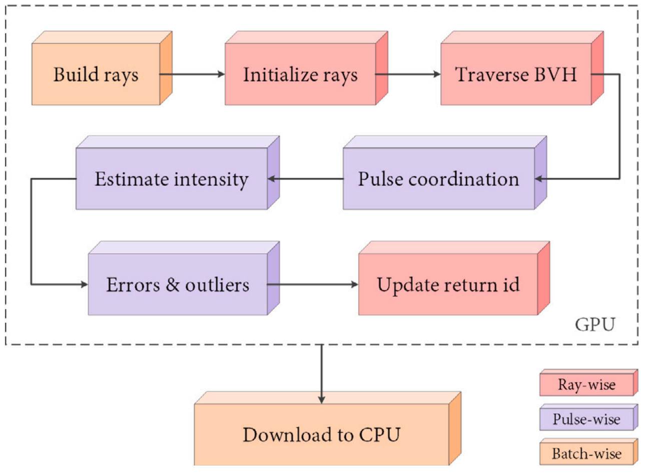


Fig. 7. Overview of LiDAR workflow. GPU and CPU stages do not overlap to avoid data transfers.

from a material database that Dupuy and Jakob (2018) collected using a goniophotometer. This database, though not densely populated, discretizes the hemispheres for each material with a variable number of samples, typically fewer than 360×90 for materials such as anisotropic ones. Note that 360×90 samples imply an angular resolution of 1° in both azimuth (θ) and elevation (ϕ). Despite not always achieving this resolution, we assume the hemispheres are discretized with 360×90 samples for simplicity. This sampling is replicated for every possible incoming ($\vec{w}_i$) and outgoing ($\vec{w}_o$) direction.

A few samples from this database are illustrated in Fig. 8. Unlike analytic BRDFs, data from a goniophotometer results in a larger memory footprint $(360{\times}90)^2$. However, we mitigate this by assuming the sensor's emitter and receiver position are equal ($\vec{w}_i = \vec{w}_o = \vec{w}$), reducing the storage requirement to 360×90 samples per material. Consequently, the memory footprint of this approach is limited to $62 \times 360 \times 90 \times 195$ floating-point values, totalling 1.45 GB, where 62 is the number of materials and 195 is the number of wavelengths collected, ranging from 358 nm to 1001 nm. Since LiDAR sensors operate at a specific wavelength, the footprint is further reduced to 7.66 MB on the GPU when using a single wavelength. To achieve this, the spectral signatures of materials are fitted to a spline curve, retrieving the reflectance for specific wavelengths that may not be directly available in the database.

This database is queried across the hemisphere of every material with vectors such as $(\cos\phi, -\sin\phi, \sin\theta)$. Note that available BRDFs are cosine-weighted, and the raw reflectance must be extracted by unweighting the collected data with the cosine of the angled enclosed by the surface normal and a vector determined by θ. Once in the GPU, the θ and ϕ angles for each return are calculated as shown in Eq. (2), where $\hat{n}$ is the normal vector of the surface at an arbitrary point p_i and r_o is the position of the sensor's emitter and receiver components. Remark that sampled BRDFs work under a local coordinate system which must be replicated for every return. This coordinate system is built with p_i being the origin of the new coordinate system, and $\hat{n}$ the Z-axis. Accordingly, the ray direction, r_d, must be reformulated relative to $\hat{n}$.

$$\theta = |-\hat{r_d} \cdot \hat{n}| \cdot \frac{\pi}{2}$$
$$\phi = 2\left(\arctan\frac{-\hat{r}_{d\,z}}{-\hat{r}_{d\,x}} + \frac{\pi}{2}\right) \tag{2}$$

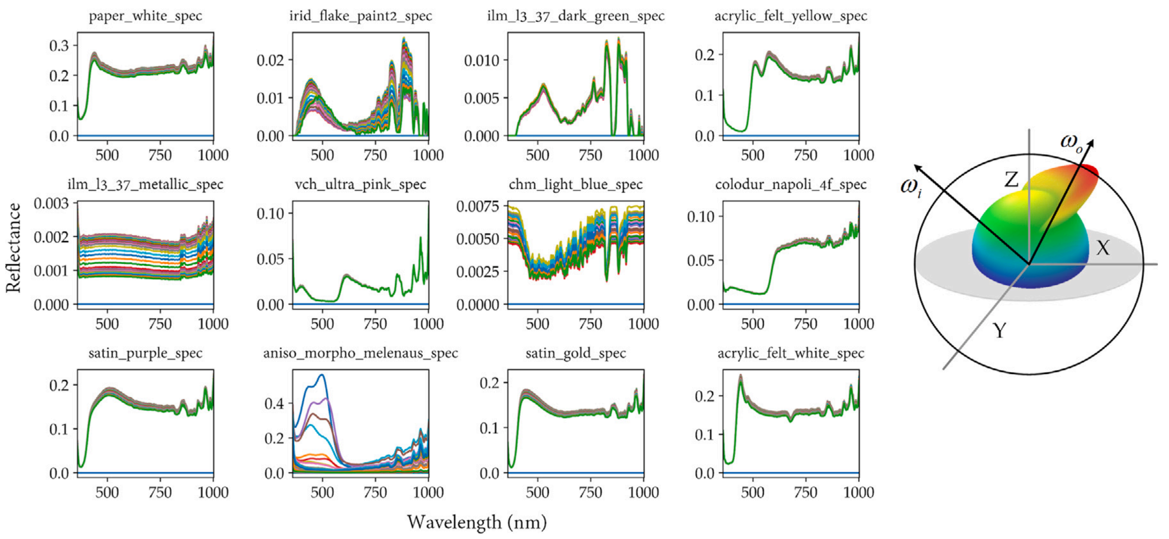


Fig. 8. Sample of BRDFs published by Dupuy and Jakob (2018). On the left side, the grid displays some of the materials and the spectral signature for random incoming vectors ($\vec{\omega}_i$), with incoming vectors parallel to outgoing ones (i.e., $\vec{\omega}_i = \vec{\omega}_o$). There are considerable variations whether $\vec{\omega}_i$ and $\vec{\omega}_o$ vary, with lower elevation obtaining signatures of smaller magnitude. On the right side, the scheme of reflections is depicted (the energy coming from $\vec{\omega}_i$ is observed at the direction $\vec{\omega}_o$).

As previously mentioned, the BRDF database has low resolution, whereas LiDAR returns require data with higher angular resolution. Therefore, intermediate values must be computed by interpolating the database samples. In this work, we used linear interpolations and Hermite splines. The first has a simple logic as it interpolates from two values. On the other hand, Hermite polynomials are harder to construct in real-time as they require finding the polynomial coefficients from their derivatives, and their complexity grows for polynomials with a higher degree. An efficient solution is to pre-calculate the coefficient matrix and transfer it to the GPU in an SSBO. Accordingly, the logic of Hermite interpolations is simplified as in Eq. (3) whether four points are used to interpolate the reflectance. The Listing 2 shows the shader logic for smoothing the goniophotometer-based BRDF with Hermite interpolation, in contrast to the linear interpolation in Listing 1.

$$\begin{pmatrix} A \\ B \\ C \\ D \end{pmatrix} = \begin{pmatrix} x_{0,0} & x_{0,1} & \cdots & x_{0,3} \\ x_{1,0} & x_{1,1} & \cdots & x_{1,3} \\ \vdots & \vdots & \ddots & \vdots \\ x_{3,0} & x_{2,1} & \cdots & x_{3,3} \end{pmatrix} \begin{pmatrix} f_{r_0} \\ f_{r_1} \\ f_{r_2} \\ f_{r_3} \end{pmatrix}$$
$$H(t, \mathcal{F}) = At^3 + Bt^2 + Ct + D, \quad t \in [0, 1] \tag{3}$$

Another relevant factor for a LiDAR system is the refractivity of materials. It also varies through the spectrum, as can be observed in the refractivity database from Polyanskiy (2022) (Fig. 9). We have primarily applied this feature for the refractions coming from shallow underwater surfaces in bathymetric surveys. Also, note that these signatures are considerably more sparse than reflectance signatures, and the spline curve is not as well approximated.

Using the previous data, the returned energy is computed from the estimated reflectance, which depends on the emitter and receiver components and is expressed as shown in Eq. (4). There are multiple formulae in the literature to calculate the LiDAR intensity, though most are analogous (Höfle and Pfeifer, 2007; Bolkas and Martinez, 2018; Dong and Chen, 2018). $\bar{I}$ is the emitted energy, D_r^2 is the receiver diameter (m), η_{atm}, η_{sys} are atmospheric and system transmission factors, ρ is the target reflectance, A_t is the target area, R is the distance to receiver in m, β is the transmit beam width (rad) and Ω is the scattering solid angle (sr). π controls the amount of energy in a specific point since it

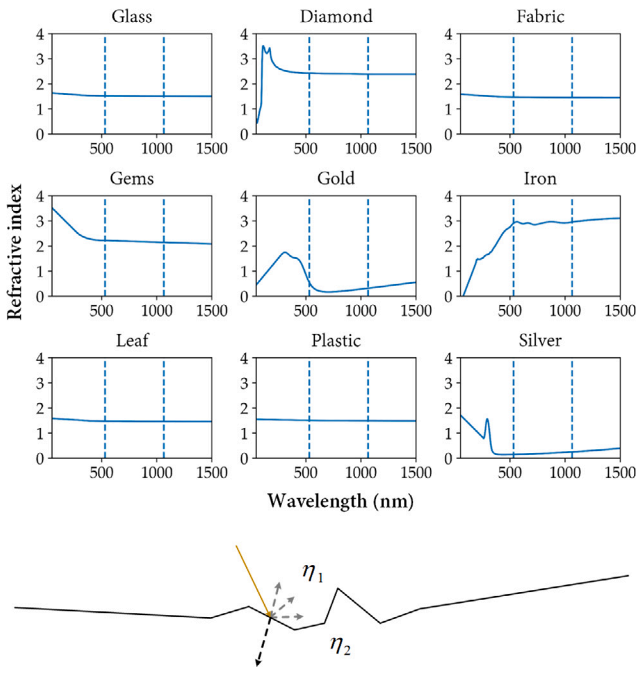


Fig. 9. Database of refractivity signatures collected by Polyanskiy (2022). The top chart grid shows the refractivity of different materials, with two frequent LiDAR wavelengths annotated as guidelines. The bottom image shows the reflection and refraction phenomena for two media with different refractivity.

spreads in every possible direction within a hemisphere.

$$\sigma = \frac{4\pi\rho A_t}{\Omega}$$
$$A_t = \frac{\pi R^2 \beta_t^2}{4} \tag{4}$$
$$P_r = \frac{\bar{I} D_r^2 \eta_{atm} \eta_{sys} \sigma}{4\pi R^4} = \frac{4\pi^2 \bar{I} D_r^2 \eta_{atm} \eta_{sys} \rho}{4\Omega R^2 \beta_t^2}$$

From here, the scattering solid angle is computed as A_t/R^2, therefore it can also be expressed as $\Omega = \pi R^2/R^2 = \pi$. The attenuation factor η_{sys} is a system-dependent factor that varies over time and with different systems. Hence, it is 1 by default. η_{atm} also stands for a transmission factor, although it depends on atmospheric conditions. As described by Höfle and Pfeifer (2007), the attenuation factor of horizontal and vertical propagation differs, and we included the attenuation factors measured by such work, ranging from clear to haze conditions. Additionally, η_{atm} is also influenced by the distance and is frequently approximated as $10^{-2\cdot R\cdot v_{atm}}$, provided that v_{atm} is the atmospheric attenuation factor. With these assumptions, Eq. (4) is simplified as shown in Eq. (5).

$$P_r = \frac{\bar{I} D_r^2 \rho \cdot 10^{-2\cdot d\cdot v_{atm}} \eta_{sys}}{4R^2} \tag{5}$$

Similar to these formulas, other variations for bathymetric LiDAR surveys can be found in the literature (Narayanan et al., 2009). Given that alternative forms of Eq. (4) are designed for simulating specific scenarios, we have applied it exclusively to the simulation of bathymetry for shallow underwater surfaces. We adhere to Eq. (5) for other scenarios where refractivity also plays a role.

3.3.1. Tone mapping

Previously computed intensity, driven by the pulse energy, among other factors such as the reflectivity of the target surface and atmospheric conditions, comprises values in a wide interval. Accordingly, the rendering of intensity data is frequently intricate due to showing values in a wide range. This shortcoming hardens the observation of subtle changes due to materials and observation angle. Therefore, it is impractical to normalize intensity in [0, 1]. Otherwise, the amplitude is sometimes provided as an exponential function covering the dynamic range of the instrument (RIEGL Laser Measurement Systems GmbH, 2017). However, improving shading tones is a prevalent topic in Computer Graphics and has been thoughtfully treated for rendering. More advanced solutions use tone-mapping function (Akenine-Möller et al., 2018) that helps to emphasize changes, especially in values far from zero. Tone-mapping functions such as the one in Hable (2010) can be modelled as flexible and efficient curves, parameterized by exposure (λ) and gamma (γ) factors. It is important to adjust λ and γ according to the specific screen. Fig. 10 compares the rendering of the point cloud replicated from an Ultra-Puck LiDAR with lower peak power ($\bar{I}$, in W units), and its improved visualization using $\lambda \leftarrow 2.2$ and $\gamma \leftarrow 1$.

3.3.2. Analytic BRDFs

In contrast to real-world BRDFs, surfaces can also be modelled with analytic BRDFs, i.e., functions intended to fake a physically-based shading to humans in real time. However, these functions do not change according to wavelengths; instead, surfaces are modelled with RGB-based components, such as the albedo and specular components.

A notable list of proposed analytic BRDFs can be found in related reviews (Guarnera et al., 2016). We implemented six BRDFs that help to emulate different kinds of materials; from anisotropic surfaces to dull-like, nearly Lambertian objects. These BRDFs are showcased in Fig. 11 in two ways. First, analytic BRDFs were used to shade the dragon model. Then, using an incoming vector, w_i, and an outgoing vector, w_o, the analytic BRDF helps to shape a hemisphere to show how energy spreads at every possible normal vector. Accordingly, the Lambertian model distributes the same energy across all normal vectors, regardless of the incoming radiance or viewing direction. In contrast, the Cook-Torrance, Blinn-Phong, and Ward-anisotropic models feature more complex specular lobes that depend on the viewing direction.

Analytic BRDFs also get considerably simplified whether $w_i = w_o = w$, as in a LiDAR system. In this regard, the formulae for the implemented BRDFs are listed below, where ρ_d and ρ_s are diffuse and specular colours of the material, α_m is the roughness factor and h is the halfway vector ($\hat{w}_i + \hat{w}_o = 2\cdot\hat{w}$). Further insight into the following BRDFs can be found in Guarnera et al. (2016).

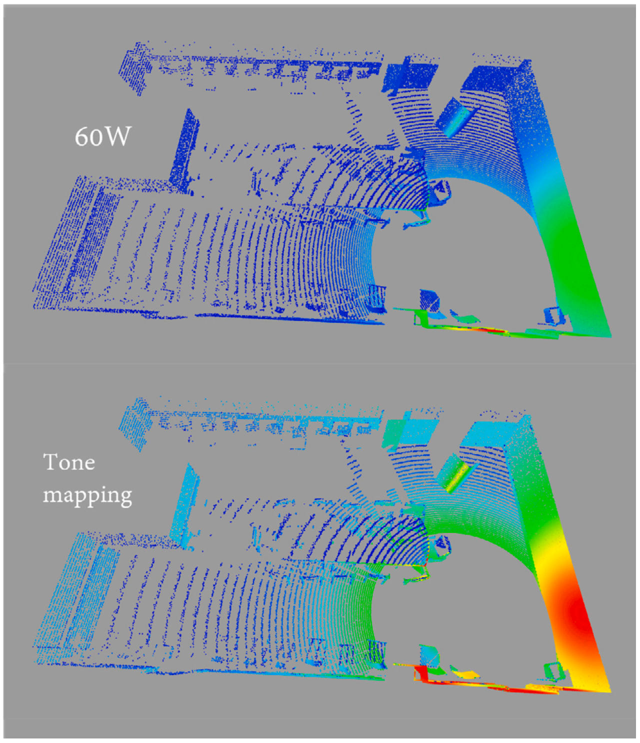


Fig. 10. Simulation of HDL-64E LiDAR with 60 W, and rendering of the resulting point cloud (1) without and (2) with tone-mapping.

(i) Lambertian:

$$\rho = \frac{\rho_d}{\pi} \tag{6}$$

(ii) Oren-Nayar:

$$\begin{aligned} \rho &= \frac{\rho_d}{\pi}(K + J)\sin\theta_w \tan\theta_w \\ K &= 1 - 0.5\frac{\alpha_m^2}{{\alpha_m}^2 + 0.33} \\ J &= 0.45\frac{\alpha_m^2}{{\alpha_m}^2 + 0.09} \end{aligned} \tag{7}$$

(iii) Minnaert:

$$\rho = \frac{\rho_d}{\pi}(\hat{n}\cdot\hat{w})^{2(k-1)} \tag{8}$$

(iv) Blinn-Phong:

$$\rho = \rho_s(\hat{n}\cdot\hat{h})^k = \rho_s(\hat{n}\cdot 2\hat{w})^k \tag{9}$$

(v) Cook-Torrance:

$$\begin{aligned} \rho &= \frac{F(\beta)D(h)G(\vec{w})}{\pi(\hat{n}\cdot\hat{w})^2} \\ F(\beta) &= F_0 + (1 - F_0)(1 - \hat{n}\cdot\hat{w})^5 \\ D(h) &= \frac{1}{\alpha_m^2(\hat{h}\cdot\hat{n})^4}\exp\frac{(\hat{h}\cdot\hat{n})^2 - 1}{\alpha_m^2(\hat{h}\cdot\hat{n})^4} \\ G(\vec{w}) &= \min\left(1, \frac{4(\hat{n}\cdot\hat{w})^2}{\hat{h}\cdot\hat{w}}\right) \end{aligned} \tag{10}$$

where $F(\beta)$ is the Schlick approximation described in Akenine-Möller et al. (2018), $D(h)$ is the Beckmann distribution (Montes Soldado and Ureña Almagro, 2012; Guarnera et al., 2016) and $G(h)$ is the geometric attenuation factor. Also, note

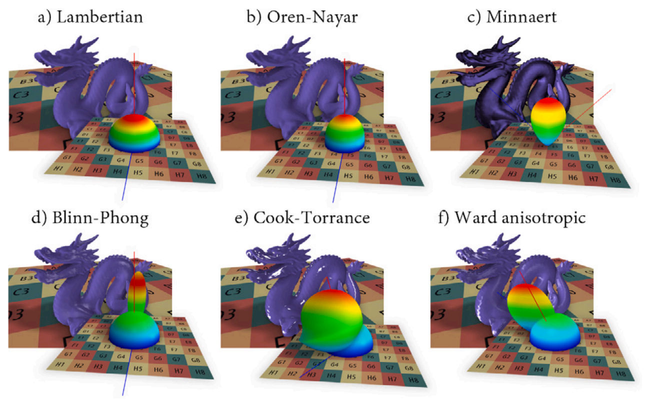


Fig. 11. BRDFs depicted (1) by shading a 3D model and (2) by distorting a hemisphere with the estimated intensity. $f_r(\vec{w})$ is represented by the distance from each vertex of the distorted hemisphere to the origin, [0, 0, 0]. From left to right, and from top to bottom: (a) Lambertian, (b) Oren-Nayar with $\alpha_m = 0.5$, (c) Minnaert with a darkening factor (k) of 1.47, whereas ρ_d is increased by a factor of A = 2.6, (d) Blinn-Phong with $\alpha = 11$, (e) Cook-Torrance with $\alpha = 0.685$ and $F_0 = 0.40$, and (f) Ward anisotropic with $\alpha_x = 0.15$ and $\alpha_y = 0.75$. $\rho_d = 1$ for all the illustrations.

that F_0 is the external reflection under normal incidence casuistic ($\vec{w} = \vec{n}$), therefore the rest of the values are approximated through interpolation between F_0 and 1.

(vi) Ward anisotropic:

$$\rho = \frac{\rho_s \exp\left(-\frac{\left(\frac{h\cdot x}{\alpha_x}\right)^2 + \left(\frac{h\cdot y}{\alpha_y}\right)^2}{(\hat{h}\cdot\hat{n})^2}\right)}{4\pi\alpha_x\alpha_y(\hat{n}\cdot\hat{w})} \tag{11}$$

3.4. Return losses

LiDAR returns are also affected by the surface roughness. In this regard, surfaces with lower roughness are more likely to cause the 'time-walk' effect, which makes returns appear more distant, or even not be captured by the receiver component (Ullrich and Pfennigbauer, 2019). Although BRDF databases provide the scattered light in a specific direction, there is no trivial manner to decompose these values into the terms that led to it, including roughness. To overcome this, materials are also linked to a roughness factor, between zero and one, that, together with a global loss function, helps to model return losses as depicted in Fig. 12.

The custom function, $l(\alpha_m)$, has an exponential shape based on roughness α_m, and is parameterized according to a, b, c, p factors and a threshold m, as shown in Eq. (12). Accordingly, returns can be discarded whether their roughness, plus a random factor, is above m. Whether an intersection is not discarded but belongs to a glossy surface, the simulated distance is higher due to the cited 'time-walk' effect. The additional distance is emulated considering (1) a random factor belonging to the object and (2) a random factor linked to the polygon. These considerations help to randomize the 'time-walk' effect while preserving the object shape. Both factors can be adjusted, and a, b, c and p should be adapted to fit a specific LiDAR sensor.

$$l(k_s) = \begin{cases} c + a(k_s + b)^p & \alpha_m > m \\ c & \alpha_m \leq m \end{cases} \tag{12}$$

where c introduces some randomness even for diffuse materials (it is zero by default).

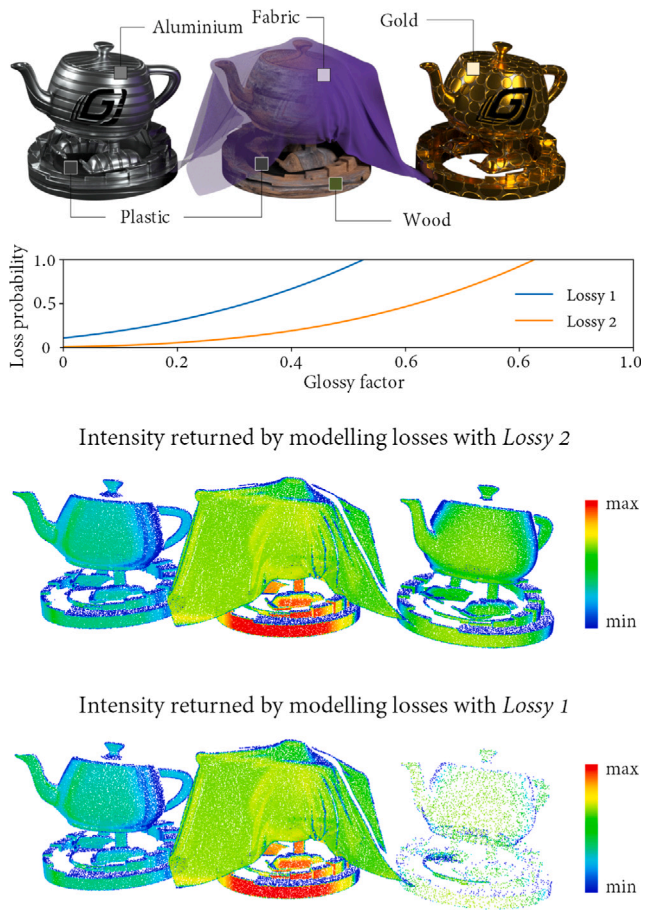


Fig. 12. Configuration of the loss function over three different materials. The depicted functions are represented in terms of $x = \alpha_m$, from zero to one, and $y = l(\alpha_m)$. This way, we get the probability of a return being lost for a given roughness value. The gold material is the one with lower roughness and therefore is more likely to lose returns. The first exponential function, *Lossy 1*, is more strict and provokes more return losses. The colour encoding represents the recorded intensity.

3.5. RGB shading

RGB cameras are frequently integrated into scanning systems, and as a post-processing stage, LiDAR point clouds can be enhanced with RGB data. This enhancement is more informative than intensity alone because it represents three wavelengths, making it more intuitive for human inspection. Therefore, RGB data was simulated by coupling a camera and adding it as an additional step after the LiDAR simulation. While previous empirical BRDFs are realistic, the database is incomplete and lacks a wide variety of RGB colours. In contrast, available scenarios often utilize materials defined by albedo, metallic, and roughness factors in the Physically-based Rendering (PBR) theory. Additionally, one or several light sources can be defined in the GUI, with at least one acting as a shadowing source. We implemented this post-processing shading using traditional rendering techniques (vertex shader followed by fragment shader), although it can also be achieved using compute shaders.

Fig. 13 shows the followed pipeline. The RGB shading is performed for each model since only one material was provided simultaneously. The fragment shader stage is required for shadowing the scenario since it provides smoothing operators that enable accessing surrounding pixels and avoiding artefacts from shadow mapping. On the other hand, most of the calculations are performed per point (vertex shader) for efficiency (the number of points is much lower than the number of pixels on the screen). During rendering, the camera is placed at the

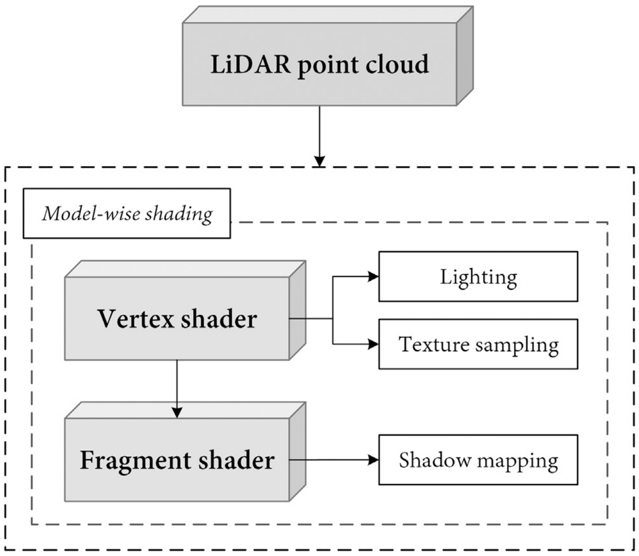


Fig. 13. Overview of procedure for calculating the RGB shading after the main simulation.

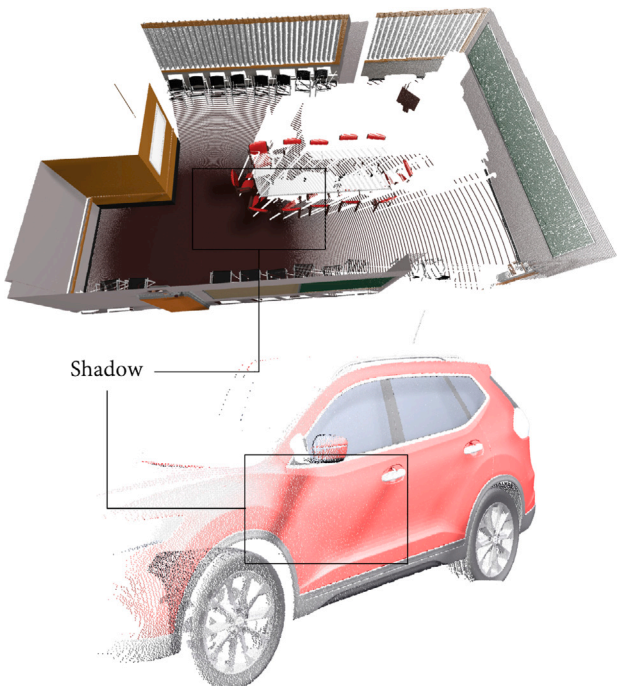


Fig. 14. RGB shading of LiDAR point clouds from the conference room and a car model.

position of the light source to generate the shadow map. To obtain the final rendering points do not necessarily get projected into the screen, but can be processed offline to calculate the point-wise RGB colour. Nonetheless, other factors such as specularity, influenced by the viewing angle, must be computed according to the LiDAR location rather than the lighting source. The results of shading the conference room and a car model are illustrated in Fig. 14.

4. Results and discussion

The proposed simulation is evaluated by (1) measuring the overhead of intensity calculations, (2) comparing the simulation time and intensity returned by our simulator against other notable VLS, (3) showing the outcomes of different commercial sensors in terms of intensity, (4) analysing the similarity of class-wise histograms in comparison with a real-world point cloud of an urban scenario and (5) varying the LiDAR wavelength across a significant spectral interval. To this end, different scenarios will be used across this section: a conference room (330k triangles), a bedroom (1.5M triangles), the San Miguel scene (5.6M triangles) (McGuire, 2017) and an urban scenario (4M triangles) that can be scanned following a path. Also, analytic BRDFs will be compared to real-world BRDF data by measuring the simulation time of each approach and rendering the outcome of each one, in contrast to real-world point clouds. For the sake of simplicity, these experiments will be mainly carried out using TLS simulations. The parameterization of the proposed LiDAR enables simulating a vast number of commercial sensors. In our experiments, we used the following sensors: Velodyne HDL-64E (Velodyne, 2018a), Pandar64 (Hesai, 2020), HDL-32E (Velodyne, 2019b), Puck (Velodyne, 2019c), Puck Lite (Velodyne, 2018c), Puck Hi-Res (Velodyne, 2018b), Ultra Puck (Velodyne, 2019d), Alpha Prime (Velodyne, 2019a) and Zenmuse L1 (ALS) (DJI, 2021).

Measurements were performed on a PC with AMD Ryzen Threadripper 3970X 3.6 GHz, 256 GB RAM, Nvidia RTX A6000 GPU and Windows 10 OS. The proposed methodology is implemented in C++ 20 and OpenGL (Open Graphics Library). Massively parallel implementations were developed in GLSL (OpenGL Shading Language) using general-purpose compute shaders and OpenMP for the multi-core CPU approach. The reported values concerning simulation time are obtained by averaging five different results.

4.1. Computational cost from calculating intensity

After the LiDAR collisions are detected, we attach data to them, either coming from the intersected surface or calculated from available data. The following experiments measure the computational cost from calculating intensity over three scenes and six LiDAR scanners. This setup helps to shed light on the stability of intensity calculations, even with larger scenarios. These tests do not only show the overhead of intensity calculations but also compare the response time obtained from (1) using analytic (ABRDF) and empirical BRDFs (PBRDF), as well as (2) different interpolation methods (naive, linear and Hermite interpolations). The naive method selects the value from the nearest spherical point, without interpolating. Table 3 compares the performance of using PBRDF and ABRDF, whereas Table 2 compares the computational cost of different interpolations. Fig. 15 summarizes the results of both tables visually. From this figure, it can be concluded that PBRDFs are less time-consuming, at the expense of slightly increasing the use of VRAM. On the other hand, using one interpolation or another has no relevant effects on the response time. Indeed, the measured time fluctuates from one test to another at the level of ms. Still, the Hermite interpolation seems to be more unstable according to the reported variance. However, it must be noted that the overhead introduced by calculating intensity is irrelevant when compared with the overall simulation, which is analysed below.

4.2. Overall computational cost

The following experiments highlight the efficiency of this system relative to our previous work and HELIOS++. Our previous version (López et al., 2022) involved computing the entire buffer of rays on the GPU, transferring them to the CPU, and iteratively uploading them back to the GPU to perform the simulation. In contrast, our current pipeline computes the rays during the simulation itself, eliminating the need for this separate preprocessing stage.

In these experiments, our work and HELIOS++ were evaluated in four different scenarios using specifications of commercial LiDAR sensors. Similarly to our solution, pulses were simulated using nineteen rays, equivalent to 3 subcircles in HELIOS++ (1 + 6 + 12) (coined as beam sample quality in HELIOS++). Table 4 provides further insight

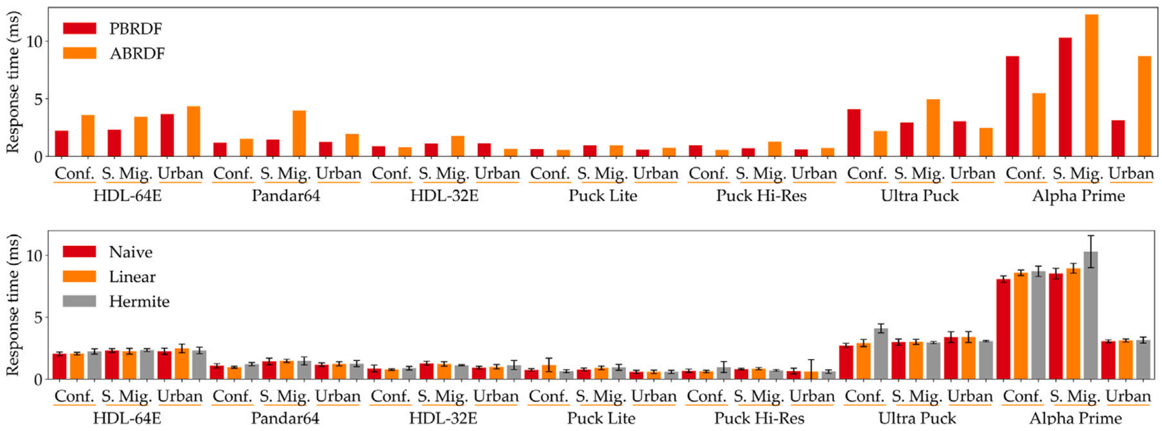


Fig. 15. Response time in ms measured by calculating intensity with PBRDFs (Hermite) and ABRDFs, on the top chart, and using different interpolation operators, on the bottom chart.

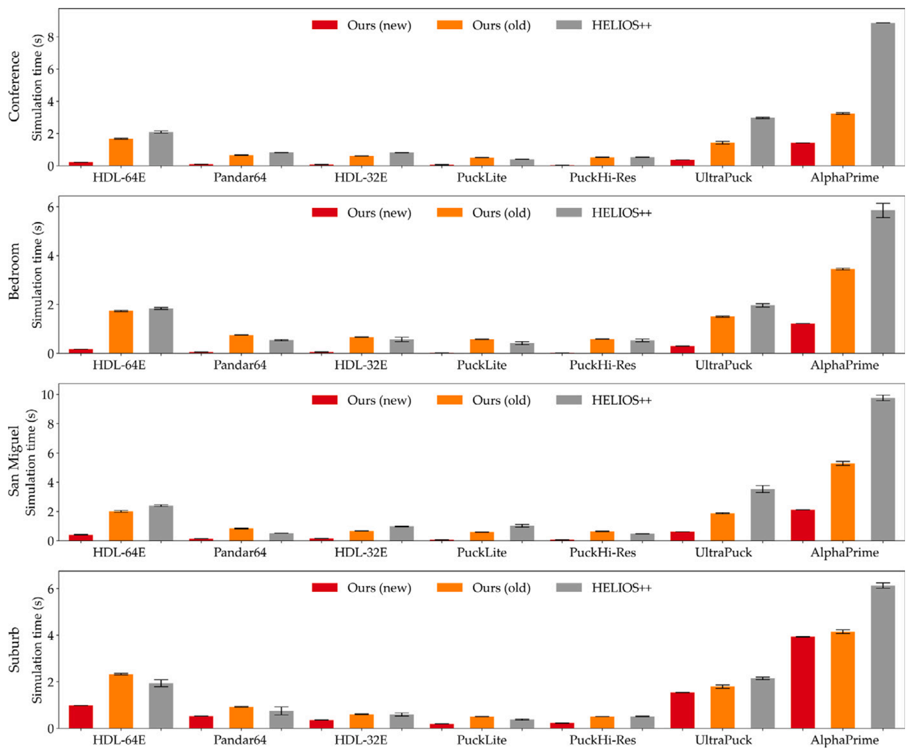


Fig. 16. Simulation time, in seconds, measured in four scenarios scanned by seven LiDAR scanners. Three approaches are compared: the one explained in this manuscript, our previous work, and HELIOS++.

into the measured time, and Fig. 16 summarizes it. The improvement in the simulation time of our VLS against HELIOS++ ranged from 77.79% (suburb) to 91.5% (conference room), whereas it goes from 51.14% (suburb) to 71.89% (bedroom) in comparison with the previous version of our VLS. On average, the simulation time was improved by 85.62% relative to HELIOS++ and 65.93% w.r.t. our previous work. We hypothesize that the simulation time of HELIOS++ is notably higher due to solving the spatial searches in the CPU. The improvement relative to our previous work comes from diminishing data transfers between CPU and GPU, and from immediately operating over calculated data (such as the rays) instead of downloading and uploading it later. Still, data transfers represent a notable bottleneck, although the GPU does not stall as much as in our previous approach. Furthermore, this workflow helps to conduct large LiDAR scans, either in a mobile platform or guided by measuring time, by splitting LiDAR surveys into many iterations. Note that this number depends on the number of maximum returns and the size of a single ray object in the GPU.

4.3. Relevant factors in intensity calculation

The distribution of intensity data was analysed in the conference room by changing the operating device and location. Accordingly, Fig. 17 shows how intensity histograms vary with four different configurations. First, HDL-64E and UltraPuck devices are emulated using the same location (experiments (a) and (b)), leading to similar histograms. However, the peak power of the HDL-64E sensor is considerably higher than in UltraPuck, and so is the magnitude in the horizontal axis of

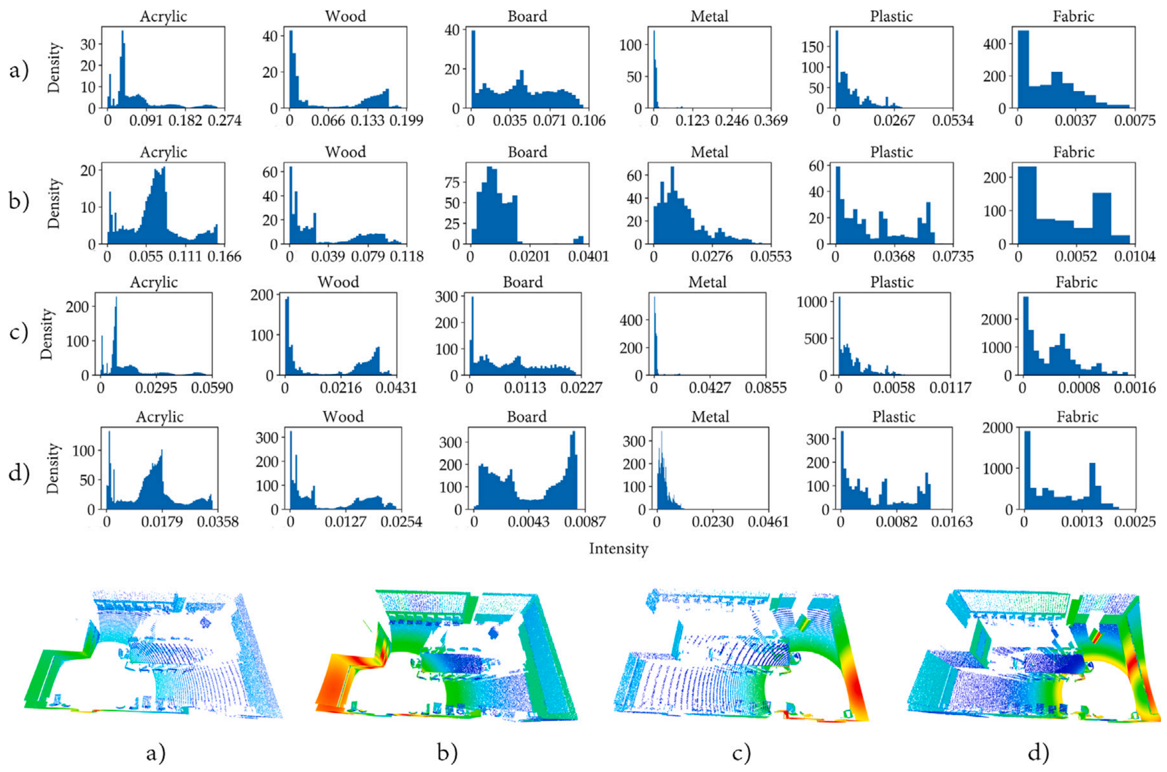


Fig. 17. Intensity histograms for different sensors and locations. From top to bottom, HDL-64E and UltraPuck are simulated in the same location ((a) and (b)), and then, the same sensors are simulated in another position ((c) and (d)).

the depicted histograms. Then, the last tests, (c) and (d), replicate this experiment with a different location and height. They produce more disparate point clouds due to the field of view covered by both sensors; while the HDL-64E goes from −24.8°to 2°, UltraPuck covers 40°starting from −25°. One of the main variations is on the board material, corresponding to panels in the walls, from which the UltraPuck sensor produced more points. In addition, these points were found with an incident vector close to the normal vector of the walls, hence returning higher intensity.

4.4. Intensity measurement

The following experiment compares the graphical results of intensity simulation. First, the analytic BRDFs are used as traditionally proposed. The material's albedo colour is weighted according to a shading function such as Oren-Nayar and Cook-Torrance. Otherwise, materials are linked to BRDFs from the goniophotometer database, thus getting rid of albedo colour. Fig. 18 compares the graphical results of ABRDF and PBRDF, and the resulting histograms for every material. Note that ABRDFs tend to obtain a decreasing function identical for all the materials, despite having disparate intensity intervals. On the other hand, PBRDFs offer a wider variety of distribution shapes, as real-world BRDFs are typically more complex and less uniform than traditional rendering techniques. Fig. 19 shows the histograms of four point clouds recorded by real-world LiDAR scans. Unlike the steadily decreasing functions observed in ABRDFs, these histograms do not exhibit a clear pattern, even between point clouds recorded by the same sensor in similar scenarios. This lack of consistency underscores the complexity and unpredictability of BRDFs compared to ABRDFs due to the numerous parameters influencing the results. These parameters include recording height, material appearance, distance, positioning, and orientation of surfaces. Consequently, these factors complicate the comparison between simulated and real-world scans, making it challenging to achieve accurate simulations.

We have also compared our work against HELIOS++, which originally used empirical BRDFs from Meerdink et al. (2019). These BRDFs collect values from a wide wavelength interval, but they are not dependent on the ongoing and outgoing vectors. Instead of using the BRDFs from Meerdink et al. (2019), we translated the database of Dupuy and Jakob (2018) into their format. We only considered the returned light when the incident vector is the surface's normal vector. The percentage of returned light for every wavelength was obtained by summing values for every w_o while fixing w_i.

The intensity and semantic maps of the scene are depicted in Fig. 20. The intensity recorded by HELIOS++ is mainly affected by the distance and the incident angle (cosine factor from $-\hat{w}_i\hat{n}$). By contrast, our solution demonstrates greater intensity variability among different materials, including intricate intensity responses from anisotropic materials like fabric. The latter BRDF was used to model the curtains on the left side of the conference room. In Fig. 20, the bottom figures compare fabric and cardboard (Lambertian-like) materials, highlighting the complexity of the first.

4.5. Path recording

Designing paths helps to solve scans extended over time according to the vehicle speed. In this regard, we conducted two experiments showing the recorded intensity through terrestrial and airborne surveys. Fig. 22 depicts the mean intensity across a path composed of 232 locations (TLS; HDL-64E) over the urban scenario previously used. The ALS path was automatically calculated and the simulation was performed using a DJI Zenmuse L1 sensor flying at 40 m. The class with the higher number of occurrences is depicted every ten frames, and its average intensity is shown as a red line. The overall average intensity is, on the other hand, represented as a blue line.

The average intensity in ALS is notably higher due to objects being captured with incident vectors similar to the normal vector. Since it was recorded over a procedural scenario, the most frequent labels are high vegetation, low vegetation and water, with ground barely

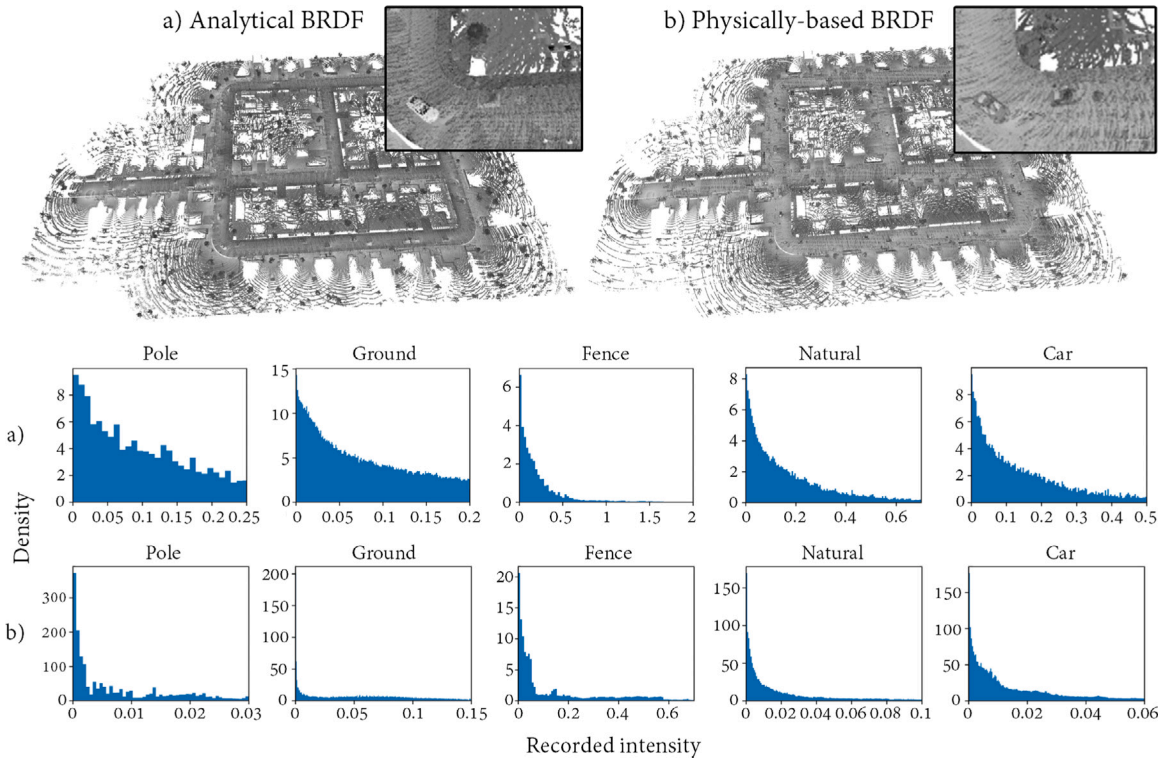


Fig. 18. (a) Intensity obtained from a Velodyne HDL-64E coupled on a vehicle, using analytic BRDFs. (b) Same configuration as (a), though intensity calculation is performed with empirical BRDFs.

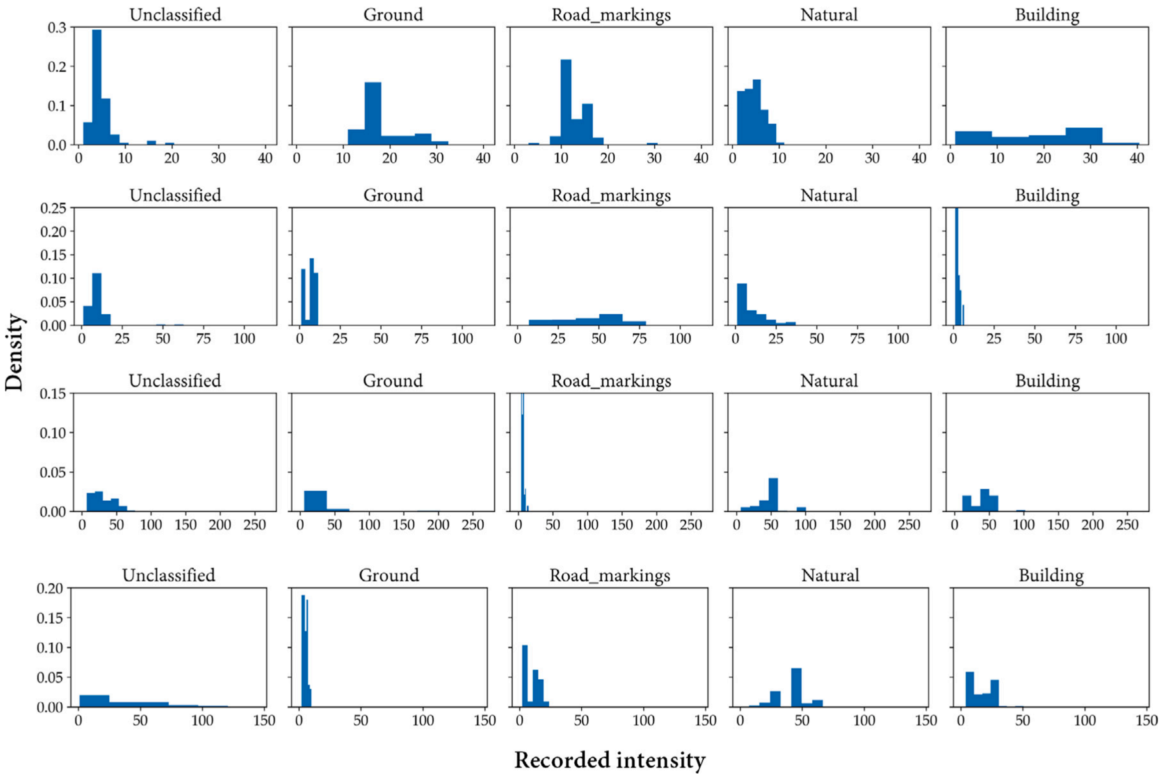


Fig. 19. Intensity histograms of the five most frequent labels found in four point clouds in the Toronto-3D dataset (Tan et al., 2020). Note that, even in point clouds captured by the same sensor and similar scenarios, there are notable changes in the recorded intensity data.

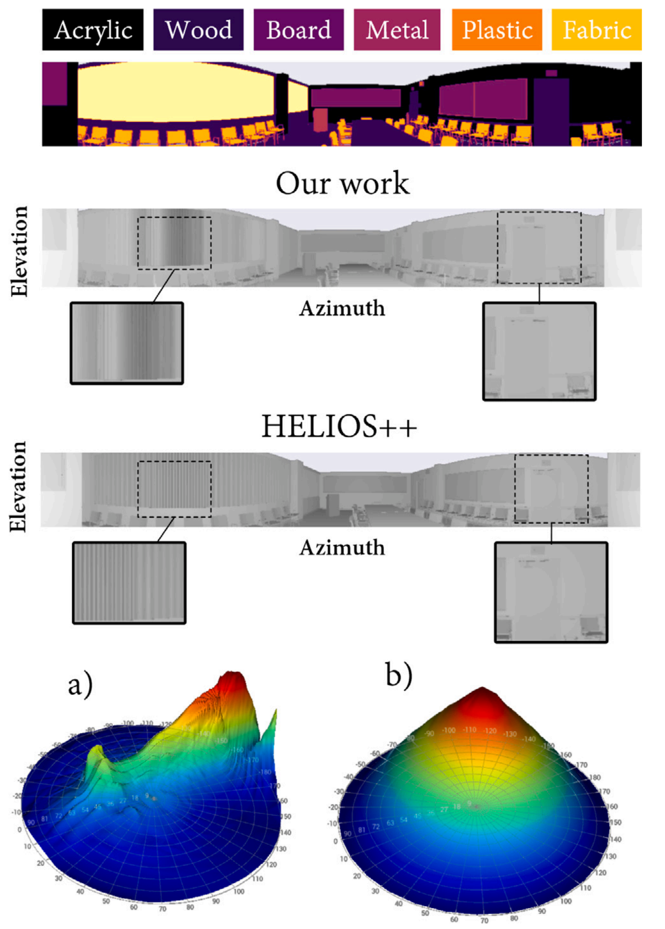


Fig. 20. Semantic and intensity maps obtained from HELIOS++ and our solution. Zoom-ins show details concerning an anisotropic material that cannot be represented in HELIOS++, and another part of the scene where materials are notably similar due to the Lambertian behaviour. Below, (a) fabric and (b) cardboard BRDFs are compared using the same incident angle, illustrated in Tekari (Dupuy and Jakob, 2018).

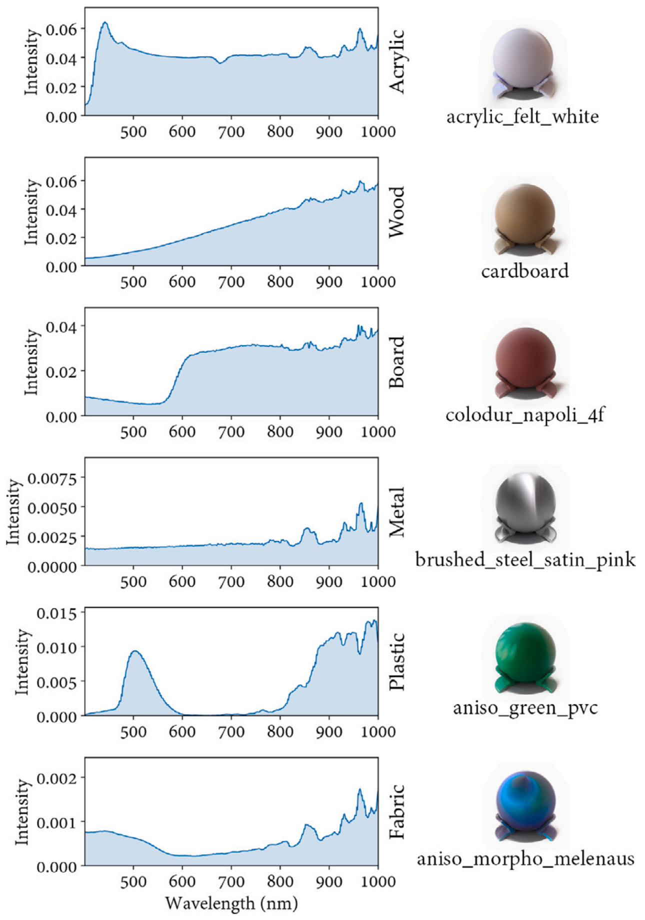


Fig. 21. Average intensity obtained for six materials by scanning them within a wavelength interval in [400, 1000] nm. The BRDFs linked to each material are depicted on the right side of every chart.

appearing. Due to the FOV, the number of points belonging to each class is considerably imbalanced. Hence, the majority of points per scan belong to the depicted class, while others are barely observed and contribute to lowering the average intensity. Note that the latter are typically recorded with incidence vectors far from the nadir vector.

On the other hand, individual TLS scans are given by 360°scans of an urban scenario. In this simulation, the appearance of different labels is much more balanced. Although buildings and high vegetation are the most frequent, with BRDFs similar to Lambertian, these are commonly mixed with other BRDFs of higher intensity, such as metallic ones found in cars. Since the scenario is not too large and has some visible boundaries, the ground was the most frequent label in at least one scan. Note that in these, the average intensity is considerably lower since barely any other labels are found and if so, the returned intensity is low due to distance and incidence angle.

4.6. Operating wavelength

LiDAR sensors are widely used for various applications and operate over several wavelengths. In non-scientific applications, the focus is typically on the 600–1000 nm wavelength range. Still, several commonly used LiDAR sensors operate at wavelengths such as 532 nm (bathymetric), 905 nm, 1064 nm, and 1550 nm. The choice of wavelength depends on the target objects and whether the sensor needs to be eye-safe. An experiment was conducted in the conference room to study the effects of different wavelengths on LiDAR measurements. The active wavelength was shifted from 400 nm to 1000 nm, and the LiDAR sensor was configured as an HDL-64E device. Fig. 21 illustrates the average intensity obtained for different materials, including wood, acrylic, board, metal, plastic, and fabric. Each material is associated with its corresponding BRDF, annotated on the right side of the figure.

While most of the objects used in this experiment had corresponding materials in the BRDF database, having complete coverage is not always possible. The average intensity of the LiDAR returns exhibited uneven variations for the set of materials tested. Some materials displayed a steady increase in intensity as the wavelength increased, while others presented a notable number of intensity peaks or crests. This experiment highlights the significance of simulating LiDAR systems at different wavelengths. Understanding how different materials respond to varying wavelengths can aid in the development of effective LiDAR systems and improve the interpretation of LiDAR data for a wide range of applications.

4.7. Completeness of LiDAR features

Finally, VLS offers a distinct advantage in its ability to flawlessly simulate an extensive array of features derived from synthetic models. All the refined datasets encompass a subset of the myriad possible features, and as such, VLS empowers the integration of novel features that would be otherwise challenging to incorporate. An illustrative example is the inclusion of normal vectors, a task considerably more intricate for human operators. When calculated automatically, these

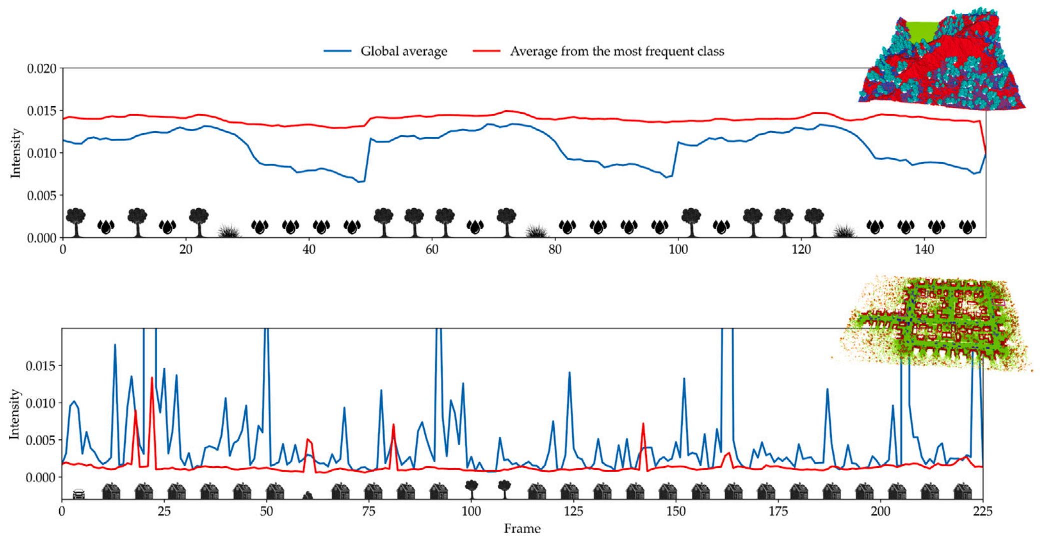


Fig. 22. Frame-wise average intensity during airborne (top) and terrestrial (bottom) simulations. The label with the highest number of returns is depicted every eight frames.

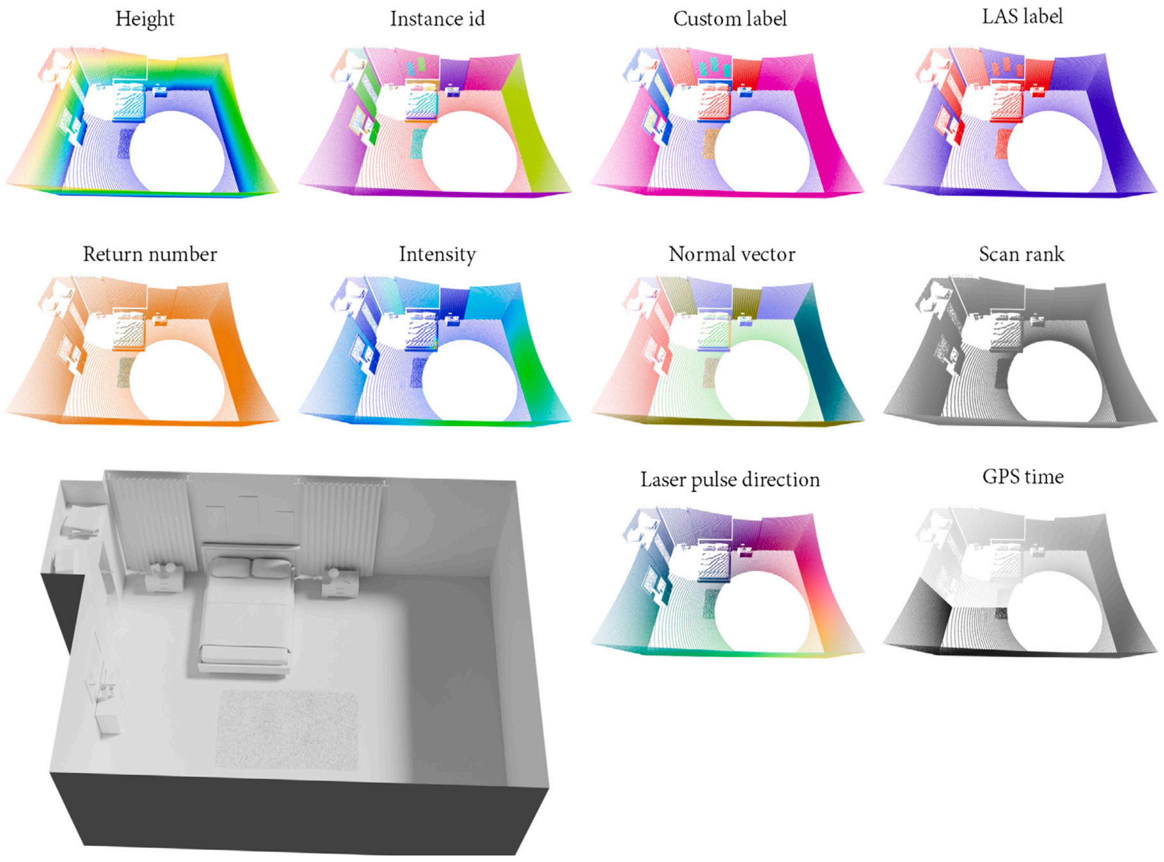


Fig. 23. Rendering of features obtained by scanning the bedroom scenario with an Ultra Puck sensor. The gaps below the scanning location are derived from the limited FOV. The scenario lacks albedo colours and therefore, RGB shading was omitted in this figure.

vectors often rely on Deep Learning or geometric algorithms that may introduce erroneous data. In contrast, synthetic models reliably provide data such as normal vectors, instance IDs, GPS timestamps, semantic labels at any LOD, scanning rank, intensity, and more.

Fig. 23 provides a visual representation of the features achievable by our work. We evaluated its performance using a bedroom scene consisting of 1.5 million vertices. GPS timestamps have been normalized in $[0, 1]$, while scan ranks are scaled within the interval $[0, \frac{\pi}{2}]$. Notice that the laser pulse direction does not refer to the scan direction flag in the LAS standard. It represents the vectors going from the sensor position to the captured points. It is worth noting that LAS labels are typically not well-suited for indoor scenes. Consequently, we have only employed two labels (`Building` and `Created`).

5. Conclusions and future work

This paper introduces a GPU-based LiDAR simulator to generate comprehensive point clouds with diverse attributes, including point

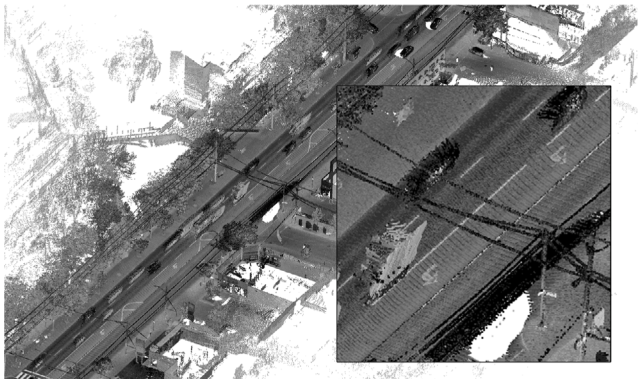


Fig. 24. Road marks visible in the Toronto-3D dataset.

coordinates, return number, number of returns, semantic labels, and GPS time, and other less common but crucial attributes such as normal vectors, intensity values, scan rank, laser pulse direction, and individual instances. While many published LiDAR datasets touch upon semantics, they are often limited to a limited number of classes, and may even contain erroneous data. In contrast, our work offers the flexibility to simulate semantics with any LOD. Moreover, we employ BRDFs involving both radiometric and spatial dimensions to compute intensity values, as opposed to prior approaches relying on shading-based BRDFs like Phong or Cook-Torrance or BRDFs neglecting the incidence angle of energy. To this end, we leverage a recent BRDF database acquired from a goniophotometer for accurate intensity computation. Additionally, our VLS introduces dynamic simulations, allowing platforms to undergo translation by specifying a set of control points through a user-friendly GUI. This facilitates the design of custom paths, whether for TLS, ALS, or MLS. While other VLS systems have addressed this capability to some extent, they often lack intuitive tools for path design.

The complete simulation procedure was implemented on a GPU and benchmarked against another state-of-the-art VLS system incorporating semantic and intensity data. The conducted experiments demonstrated the superior performance of our approach, decreasing the simulation time on an average of 85.62% in comparison to Winiwarter et al. (2022). Furthermore, our sensor parameterization framework empowers the simulation of a wide spectrum of LiDAR sensors and platforms. This versatility can serve dual purposes: firstly, optimization for future inspection methodologies, and secondly, the generation of datasets that mimic currently published real-world datasets, effectively augmenting their scale.

Nevertheless, LiDAR simulations pose significant challenges while still having certain drawbacks and limitations, which warrant attention in future research. A key deficiency is the level of detail within a single model, as illustrated in Fig. 24. The pavement is commonly modelled as a plane, thus overlooking the presence of different materials, such as acrylic for crosswalks. Furthermore, another problem in 3D modelling is the labelling of objects. Although semantic labelling is achieved through regular expressions applied to object and material names, there remain instances where proper naming conventions are lacking. For instance, the bedroom scene comprises objects with generic labels like `Mesh001` that fail to represent the semantics of 3D scenes. Therefore, the integration of semantic segmentation techniques holds promise for naming scenarios within publicly available datasets where conventional semantics may fall short.

Although empirical BRDFs have been included, the intensity modelling is much more complex, as observed in the DART simulator (Yang et al., 2022). Therefore, this system ought to be extended to include many more phenomena affecting real LiDAR scans. Moreover, the refinement of this methodology could benefit from incorporating full-waveform and single-photon LiDAR systems (Tachella et al., 2019), as well as other non-included technologies. In Deep Learning, a more in-depth investigation is crucial to elucidate the advantages of employing VLS over procedural environments for generating extensive datasets spanning natural and urban scenarios.

CRediT authorship contribution statement

Alfonso López: Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Resources, Methodology, Investigation, Funding acquisition, Formal analysis, Data curation, Conceptualization. **Carlos J. Ogayar:** Writing – review & editing, Methodology, Formal analysis. **Rafael J. Segura:** Writing – review & editing, Project administration, Investigation, Formal analysis, Conceptualization. **Juan C. Casas-Rosa:** Writing – review & editing, Validation, Software.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This result has been partially supported by the Spanish Ministry of Science, Innovation and Universities via a predoctoral grant (FPU19/00100) and a postdoctoral grant (JDC2023-051785-I) to the first author.

Code listings

```
float x_i, y_i, x_f = modf(x, x_i), y_f = modf(y, y_i)
int x0 = int(x_i), y0 = int(y_i), x1 = (x0 + 1) % 360, y1 = clamp(y0 + 1, 0, 90)

float reflectance = brdfData[materialID * 32760 + x0 * 91 + y0]
                    * (1.0f - x_f) * (1.0f - y_f) +
                    brdfData[materialID * 32760 + x1 * 91 + y0]
                    * x_f * (1.0f - y_f) +
                    brdfData[materialID * 32760 + x0 * 91 + y1]
                    * (1.0f - x_f) * y_f +
                    brdfData[materialID * 32760 + x1 * 91 + y1]
                    * x_f * y_f
```

Listing 1: Linear interpolation of BRDF data.

```
L = normalize(ray.origin - collision);

y = (PI / 2.0f - abs(asin(clamp(L.y, -1.0f, 1.0)))) * 2.0f * 90.0f / PI
x = (atan(L.z, L.x) + PI) * 2.0f * 90.0f / PI
x_f = modf(x, x_i)
y_f = modf(y, y_i)
x0 = int(x_i) % 360
x1 = (x0 + 1)
x2 = (x1 + 1) % 360
x3 = (x2 + 1) % 360
y0 = clamp(int(y_i), 0, 90)
y1 = clamp(y0 + 1, 0, 90)
y2 = clamp(y1 + 1, 0, 90)
y3 = clamp(y2 + 1, 0, 90)

rx0 = brdfData[materialID * 32760 + x0 * 91 + y0]
rx1 = brdfData[materialID * 32760 + x1 * 91 + y0]
rx2 = brdfData[materialID * 32760 + x2 * 91 + y0]
rx3 = brdfData[materialID * 32760 + x3 * 91 + y0]
ry0 = brdfData[materialID * 32760 + x0 * 91 + y0]
ry1 = brdfData[materialID * 32760 + x0 * 91 + y1]
ry2 = brdfData[materialID * 32760 + x0 * 91 + y2]
ry3 = brdfData[materialID * 32760 + x0 * 91 + y3];

ax = rx0 * hermiteTensor[0] + rx1 * hermiteTensor[1] + rx2 * hermiteTensor[2] + rx3 * hermiteTensor[3]
bx = rx0 * hermiteTensor[4] + rx1 * hermiteTensor[5] + rx2 * hermiteTensor[6] + rx3 * hermiteTensor[7]
cx = rx0 * hermiteTensor[8] + rx1 * hermiteTensor[9] + rx2 * hermiteTensor[10] + rx3 * hermiteTensor[11]
dx = rx0 * hermiteTensor[12] + rx1 * hermiteTensor[13] + rx2 * hermiteTensor[14] + rx3 * hermiteTensor[15]

ay = ry0 * hermiteTensor[0] + ry1 * hermiteTensor[1] + ry2 * hermiteTensor[2] + ry3 * hermiteTensor[3]
by = ry0 * hermiteTensor[4] + ry1 * hermiteTensor[5] + ry2 * hermiteTensor[6] + ry3 * hermiteTensor[7]
cy = ry0 * hermiteTensor[8] + ry1 * hermiteTensor[9] + ry2 * hermiteTensor[10] + ry3 * hermiteTensor[11]
dy = ry0 * hermiteTensor[12] + ry1 * hermiteTensor[13] + ry2 * hermiteTensor[14] + ry3 * hermiteTensor[15]

return (x_f * (x_f * (x_f * ax + bx) + cx) + dx) + (y_f * (y_f *
(y_f * ay + by) + cy) + dy)
```

Listing 2: Hermite interpolation of BRDF data, with coefficients being accessed from a GPU buffer, instead of calculating them.

Table 2
Overhead introduced by the intensity calculation stage using different interpolation approaches.

Sensor/Approach	Scenario								
	Conference 331k triangles			San Miguel 5.6 m triangles			Urban[a] 4.1 m triangles		
	Naive	Linear	Hermite	Naive	Linear	Hermite	Naive	Linear	Hermite
HDL-64E (2.880 m rays)	2.033 ± 0.144 ms	2.044 ± 0.106 ms	2.226 ± 0.219 ms	2.303 ± 0.153 ms	2.243 ± 0.233 ms	2.327 ± 0.119 ms	2.240 ± 0.252 ms	2.472 ± 0.353 ms	2.305 ± 0.265 ms
Pandar64 (1.334 m rays)	1.080 ± 0.175 ms	0.963 ± 0.079 ms	1.200 ± 0.132 ms	1.419 ± 0.260 ms	1.474 ± 0.121 ms	1.463 ± 0.319 ms	1.171 ± 0.137 ms	1.231 ± 0.163 ms	1.243 ± 0.247 ms
HDL-32E (446.4k rays)	0.863 ± 0.267 ms	0.778 ± 0.060 ms	0.880 ± 0.142 ms	1.278 ± 0.157 ms	1.220 ± 0.173 ms	1.118 ± 0.046 ms	0.922 ± 0.113 ms	1.002 ± 0.177 ms	1.135 ± 0.360 ms
Puck Lite (216k rays)	0.737 ± 0.106 ms	1.150 ± 0.540 ms	0.642 ± 0.123 ms	0.766 ± 0.128 ms	0.908 ± 0.141 ms	0.955 ± 0.239 ms	0.582 ± 0.133 ms	0.588 ± 0.136 ms	0.581 ± 0.128 ms
Puck Hi-Res (216k rays)	0.674 ± 0.133 ms	0.617 ± 0.101 ms	0.968 ± 0.436 ms	0.808 ± 0.056 ms	0.851 ± 0.094 ms	0.703 ± 0.074 ms	0.649 ± 0.229 ms	0.599 ± 0.965 ms	0.604 ± 0.144 ms
Ultra Puck (1.756 m rays)	2.708 ± 0.171 ms	2.906 ± 0.289 ms	4.091 ± 0.359 ms	2.973 ± 0.258 ms	3.000 ± 0.206 ms	2.943 ± 0.082 ms	3.382 ± 0.431 ms	3.394 ± 0.437 ms	3.059 ± 0.059 ms
Alpha Prime (5.241 m rays)	8.071 ± 0.268 ms	8.592 ± 0.223 ms	8.704 ± 0.415 ms	8.522 ± 0.424 ms	8.955 ± 0.399 ms	10.296 ± 1.293 ms	3.059 ± 0.116 ms	3.112 ± 0.119 ms	3.139 ± 0.247 ms

[a] LiDAR simulations over the urban scenario are not conducted with a single scan, but following a path.

Table 3
Overhead introduced by the intensity calculation stage using PBRDFs and ABRDFs. Response time is reported as a global metric and normalized based on the number of rays.

Sensor/Approach	Scenario											
	Conference 331k triangles				San Miguel 5.6 m triangles				Urban 4.1 m triangles			
	PBRDF		ABRF		PBRDF		ABRF		PBRDF		ABRF	
HDL-64E (2.880 m rays)	2.226 ms	0.77 ns/ray	3.588 ms	1.24 ns/ray	2.327 ms	0.80 ns/ray	3.432 ms	2.64 ns/ray	3.691 ms	1.28 ns/ray	4.351 ms	1.51 ns/ray
Pandar64 (1.334 m rays)	1.200 ms	0.89 ns/ray	1.535 ms	1.15 ns/ray	1.463 ms	1.09 ns/ray	3.999 ms	2.99 ns/ray	1.243 ms	0.93 ns/ray	1.949 ms	1.46 ns/ray
HDL-32E (446.4k rays)	0.880 ms	1.97 ns/ray	0.789 ms	1.76 ns/ray	1.118 ms	2.50 ns/ray	1.764 ms	3.95 ns/ray	1.135 ms	2.54 ns/ray	0.669 ms	1.49 ns/ray
Puck Lite (216k rays)	0.642 ms	2.97 ns/ray	0.565 ms	2.61 ns/ray	0.955 ms	4.42 ns/ray	0.952 ms	4.40 ns/ray	0.581 ms	2.68 ns/ray	0.734 ms	3.39 ns/ray
Puck Hi-Res (216k rays)	0.968 ms	4.48 ns/ray	0.557 ms	2.57 ns/ray	0.703 ms	3.25 ns/ray	1.270 ms	5.87 ns/ray	0.604 ms	2.79 ns/ray	0.714 ms	3.30 ns/ray
Ultra Puck (1.756 m rays)	4.091 ms	2.32 ns/ray	2.203 ms	1.25 ns/ray	2.943 ms	1.67 ns/ray	4.958 ms	2.82 ns/ray	3.059 ms	1.74 ns/ray	2.484 ms	1.41 ns/ray
Alpha Prime (5.241 m rays)	8.704 ms	1.66 ns/ray	5.507 ms	1.05 ns/ray	10.296 ms	1.96 ns/ray	12.308 ms	2.34 ns/ray	3.139 ms	0.59 ns/ray	8.702 ms	1.66 ns/ray

Table 4
Simulation time in seconds of three different VLS tested over four scenarios and seven LiDAR devices.

		Conference #triangles 330k	Bedroom #triangles 1.5M	San Miguel #triangles 5.6M	Suburb #triangles 4.1M
LiDAR	Approach	Simulation time (seconds)			
HDL-64E	GPU$_{new}$	0.3781 ± 0.050 s	0.3559 ± 0.008 s	0.4931 ± 0.008 s	0.7008 ± 0.061 s
	GPU$_{old}$	1.9561 ± 0.027 s	1.8456 ± 0.021 s	2.1220 ± 0.046 s	2.6589 ± 0.045 s
	HELIOS++	4.0919 ± 0.2028 s	3.0243 ± 0.0969 s	3.7222 ± 0.3522 s	3.2765 ± 0.2138 s
Pandar64	GPU$_{new}$	0.1426 ± 0.007 s	0.1426 ± 0.008 s	0.1962 ± 0.008 s	0.4432 ± 0.032 s
	GPU$_{old}$	0.7894 ± 0.004 s	0.8584 ± 0.025 s	0.9632 ± 0.016 s	1.1060 ± 0.026 s
	HELIOS++	1.7133 ± 0.0634 s	0.9489 ± 0.0194 s	1.6594 ± 0.2481 s	1.5708 ± 0.1868 s
HDL-32E	GPU$_{new}$	0.1492 ± 0.008 s	0.1413 ± 0.008 s	0.2042 ± 0.008 s	0.3252 ± 0.027 s
	GPU$_{old}$	0.7830 ± 0.006 s	0.8543 ± 0.012 s	0.8611 ± 0.012 s	0.8321 ± 0.015 s
	HELIOS++	1.6054 ± 0.1012 s	0.8987 ± 0.0666 s	1.6207 ± 0.1148 s	1.6127 ± 0.0751 s
Puck Lite	GPU$_{new}$	0.0529 ± 0.006 s	0.0538 ± 0.005 s	0.0776 ± 0.005 s	0.1536 ± 0.013 s
	GPU$_{old}$	0.7211 ± 0.007 s	0.6888 ± 0.013 s	0.7531 ± 0.011 s	0.7832 ± 0.011 s
	HELIOS++	0.8302 ± 0.0397 s	0.5217 ± 0.0440 s	0.9388 ± 0.1150 s	1.0754 ± 0.1012 s
Puck Hi-Res	GPU$_{new}$	0.0523 ± 0.005 s	0.0523 ± 0.005 s	0.0807 ± 0.006 s	0.1711 ± 0.014 s
	GPU$_{old}$	0.7189 ± 0.010 s	0.6756 ± 0.011 s	0.7435 ± 0.013 s	0.7784 ± 0.009 s
	HELIOS++	0.9598 ± 0.0518 s	0.7314 ± 0.0410 s	1.0005 ± 0.1703 s	1.0318 ± 0.1309 s
UltraPuck	GPU$_{new}$	0.5362 ± 0.008 s	0.5446 ± 0.008 s	0.7341 ± 0.007 s	1.0370 ± 0.029 s
	GPU$_{old}$	1.6101 ± 0.051 s	1.7780 ± 0.017 s	2.1071 ± 0.034 s	1.8437 ± 0.056 s
	HELIOS++	6.3608 ± 0.2083 s	3.5286 ± 0.1044 s	5.7765 ± 0.6631 s	4.5759 ± 0.3793 s
Alpha Prime	GPU$_{new}$	1.5987 ± 0.008 s	1.6260 ± 0.008 s	2.1919 ± 0.006 s	3.2030 ± 0.235 s
	GPU$_{old}$	3.4567 ± 0.035 s	3.6781 ± 0.008 s	5.577 ± 0.084 s	4.3485 ± 0.041 s
	HELIOS++	18.6971 ± 0.5502 s	10.4146 ± 0.1541 s	17.6918 ± 0.2321 s	14.0254 ± 0.8954 s

Simulation time

See Tables 2–4.

References

Ahn, N., Höfer, A., Herrmann, M., Donn, C., 2020. Real-time simulation of physical multi-sensor setups. ATZelectronics Worldw. 15 (6), 8–11. http://dx.doi.org/10.1007/s38314-020-0207-1.

Akenine-Möller, T., Haines, E., Hoffman, N., Pesce, A., Iwanicki, M., Hillaire, S., 2018. Real-Time Rendering 4th Edition. A K Peters/CRC Press, Boca Raton, FL, USA.

American Society for Photogrammetry and Remote Sensing (ASPRS), 2019. LASer (LAS) file format exchange activities – ASPRS. URL: https://www.asprs.org/divisions-committees/lidar-division/laser-las-file-format-exchange-activities.

Bechtold, S., Höfle, B., 2016. Helios: a multi-purpose LIDAR simulation framework for research, planning and training of laser scanning operations with airborne, ground-based mobile and stationary platforms. ISPRS Ann. Photogramm. Remote. Sens. Spat. Inf. Sci. III3, 161–168. http://dx.doi.org/10.5194/isprs-annals-III-3-161-2016, ADS Bibcode: 2016ISPAnIII3..161B.

Behley, J., Garbade, M., Milioto, A., Quenzel, J., Behnke, S., Gall, J., Stachniss, C., 2021. Towards 3D LiDAR-based semantic scene understanding of 3D point cloud sequences: The SemanticKITTI dataset. Int. J. Robot. Res. 40 (8–9), 959–967. http://dx.doi.org/10.1177/02783649211006735.

Boehler, W., Marbs, M.B.V., 2018. Investigating LASER Scanner Accuracy. Technical Report, i3mainz, Institute for Spatial Information and Surveying Technology, FH Mainz, Holzstrasse 36, 55116 Mainz, Germany.

Bolkas, D., Martinez, A., 2018. Effect of target color and scanning geometry on terrestrial LiDAR point-cloud noise and plane fitting. J. Appl. Geod. 12 (1), 109–127. http://dx.doi.org/10.1515/jag-2017-0034, Publisher: Walter de Gruyter GmbH.

Caesar, H., Bankiti, V., Lang, A.H., Vora, S., Liong, V.E., Xu, Q., Krishnan, A., Pan, Y., Baldan, G., Beijbom, O., 2019. nuScenes: A multimodal dataset for autonomous driving. arXiv preprint arXiv:1903.11027.

Cai, J., Deng, W., Guang, H., Wang, Y., Li, J., Ding, J., 2022. A survey on data-driven scenario generation for automated vehicle testing. Machines 10 (11), 1101. http://dx.doi.org/10.3390/machines10111101, Number: 11 Publisher: Multidisciplinary Digital Publishing Institute.

Chen, X., Mersch, B., Nunes, L., Marcuzzi, R., Vizzo, I., Behley, J., Stachniss, C., 2022. Automatic labeling to generate training data for online LiDAR-based moving object segmentation. IEEE Robot. Autom. Lett. 7 (3), 6107–6114. http://dx.doi.org/10.1109/LRA.2022.3166544, Conference Name: IEEE Robotics and Automation Letters.

Chen, H., Müller, S., 2022. Analysis of real-time lidar sensor simulation for testing automated driving functions on a vehicle-in-the-loop testbench. In: 2022 IEEE Intelligent Vehicles Symposium. IV, pp. 1605–1614. http://dx.doi.org/10.1109/IV51971.2022.9827048.

Choi, Y., Kim, N., Hwang, S., Park, K., Yoon, J.S., An, K., Kweon, I.S., 2018. KAIST multi-spectral day/Night data set for autonomous and assisted driving. IEEE Trans. Intell. Transp. Syst. 19 (3), 934–948. http://dx.doi.org/10.1109/TITS.2018.2791533, Conference Name: IEEE Transactions on Intelligent Transportation Systems.

Dayal, S., Goel, S., Lohani, B., Mittal, N., Mishra, R.K., 2021. Comprehensive airborne laser scanning (ALS) simulation. J. the Indian Soc. Remote. Sens. 49 (7), 1603–1622. http://dx.doi.org/10.1007/s12524-021-01334-5.

Deems, J.S., Painter, T.H., Finnegan, D.C., 2013. Lidar measurement of snow depth: a review. J. Glaciol. 59 (215), 467–479. http://dx.doi.org/10.3189/2013jog12j154, Publisher: International Glaciological Society.

Díaz-Medina, M., Fuertes, J.M., Segura-Sánchez, R.J., Lucena, M., Ogayar-Anguita, C.J., 2023. LiDAR attribute based point cloud labeling using CNNs with 3D convolution layers. Comput. Geosci. 105453. http://dx.doi.org/10.1016/j.cageo.2023.105453.

DJI, 2021. Zenmuse L1. URL: https://enterprise.dji.com/es/zenmuse-l1/photo.

Dong, P., Chen, Q., 2018. LiDAR Remote Sensing and Applications. CRC Press, http://dx.doi.org/10.4324/9781351233354.

Dosovitskiy, A., Ros, G., Codevilla, F., Lopez, A., Koltun, V., 2017. CARLA: An open urban driving simulator. In: Proceedings of the 1st Annual Conference on Robot Learning. PMLR, pp. 1–16, ISSN: 2640-3498.

Douglas, D.H., Peucker, T.K., 1973. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. Cartographica: Int. J. Geogr. Inf. Geovisualization 10 (2), 112–122. http://dx.doi.org/10.3138/FM57-6770-U75U-7727, Publisher: University of Toronto Press.

dSPACE, 2024. AURELION, URL: https://www.dspace.com/en/pub/home/products/sw/experimentandvisualization/aurelion_sensor-realistic_sim.cfm.

Dupuy, J., Jakob, W., 2018. An adaptive parameterization for efficient material acquisition and rendering. ACM Trans. Graph. 37 (6), 274:1–274:14. http://dx.doi.org/10.1145/3272127.3275059.

Fang, J., Zhou, D., Yan, F., Zhao, T., Zhang, F., Ma, Y., Wang, L., Yang, R., 2020. Augmented LiDAR simulator for autonomous driving. IEEE Robot. Autom. Lett. 5 (2), 1931–1938. http://dx.doi.org/10.1109/LRA.2020.2969927.

Gao, W., Peters, R., Stoter, J., 2024. Building-PCC: Building point cloud completion benchmarks. ISPRS Ann. Photogramm. Remote. Sens. Spat. Inf. Sci. X-4/W5-2024, 179–186. http://dx.doi.org/10.5194/isprs-annals-X-4-W5-2024-179-2024.

Gschwandtner, M., Kwitt, R., Uhl, A., Pree, W., 2011. BlenSor: Blender sensor simulation toolbox. In: Bebis, G., Boyle, R., Parvin, B., Koracin, D., Wang, S., Kyungnam, K., Benes, B., Moreland, K., Borst, C., DiVerdi, S., Yi-Jen, C., Ming, J. (Eds.), Advances in Visual Computing. In: Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, pp. 199–208. http://dx.doi.org/10.1007/978-3-642-24031-7_20.

Guarnera, D., Guarnera, G., Ghosh, A., Denk, C., Glencross, M., 2016. BRDF representation and acquisition. Comput. Graph. Forum 35 (2), 625–650. http://dx.doi.org/10.1111/cgf.12867, _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.12867.

Hable, J., 2010. Uncharted 2: HDR lighting.

Hackel, T., Savinov, N., Ladicky, L., Wegner, J.D., Schindler, K., Pollefeys, M., 2017. SEMANTIC3D: A new large-scale point cloud classification benchmark. In: ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences. IV-1-W1, pp. 91–98.

Haider, A., Pigniczki, M., Köhler, M.H., Fink, M., Schardt, M., Cichy, Y., Zeh, T., Haas, L., Poguntke, T., Jakobi, M., Koch, A.W., 2022. Development of high-fidelity automotive LiDAR sensor model with standardized interfaces. Sensors 22 (19), 7556. http://dx.doi.org/10.3390/s22197556, Number: 19 Publisher: Multidisciplinary Digital Publishing Institute.

Hanke, T., Schaermann, A., Geiger, M., Weiler, K., Hirsenkorn, N., Rauch, A., Schneider, S.-A., Biebl, E., 2017. Generation and validation of virtual point cloud data for automated driving systems. In: 2017 IEEE 20th International Conference on Intelligent Transportation Systems. ITSC, pp. 1–6. http://dx.doi.org/10.1109/ITSC.2017.8317864, ISSN: 2153-0017.

Hesai, 2020. Pandar64 user's manual.pdf. URL: https://hesaiweb2019.blob.core.chinacloudapi.cn/uploads/Pandar64_User's_Manual.pdf.

Hodgson, M.E., Bresnahan, P., 2004. Accuracy of airborne lidar-derived elevation. Photogramm. Eng. Remote Sens. 70 (3), 331–339. http://dx.doi.org/10.14358/pers.70.3.331, Publisher: American Society for Photogrammetry and Remote Sensing.

Höfle, B., Pfeifer, N., 2007. Correction of laser scanning intensity data: Data and model-driven approaches. ISPRS J. Photogramm. Remote Sens. 62 (6), 415–433. http://dx.doi.org/10.1016/j.isprsjprs.2007.05.008, Publisher: Elsevier BV.

Instituto Geográfico de Información Geográfica, 2023. PNOA lidar. URL: https://pnoa.ign.es/.

Kukko, A., Hyyppä, J., 2009. Small-footprint laser scanning simulator for system validation, error assessment, and algorithm development. Photogramm. Eng. Remote Sens. 75 (10), 1177–1189. http://dx.doi.org/10.14358/PERS.75.10.1177.

López, A., Ogayar, C.J., Jurado, J.M., Feito, F.R., 2022. A GPU-accelerated framework for simulating lidar scanning. IEEE Trans. Geosci. Remote Sens. 60, 1–18. http://dx.doi.org/10.1109/TGRS.2022.3165746, Conference Name: IEEE Transactions on Geoscience and Remote Sensing.

Majercik, A., Crassin, C., Shirley, P., McGuire, M., 2018. A ray-box intersection algorithm and efficient dynamic voxel rendering. J. Comput. Graph. Tech. (JCGT) 7 (3), 66–81.

Manivasagam, S., Wang, S., Wong, K., Zeng, W., Sazanovich, M., Tan, S., Yang, B., Ma, W.-C., Urtasun, R., 2020. LiDARsim: Realistic lidar simulation by leveraging the real world. In: CVPR 2020. IEEE, pp. 11164–11173. http://dx.doi.org/10.1109/cvpr42600.2020.01118.

McGuire, M., 2017. Computer graphics archive.

McManamon, P., 2019. LiDAR Technologies and Systems. SPIE Press, http://dx.doi.org/10.1117/3.2518254.

Meerdink, S.K., Hook, S.J., Roberts, D.A., Abbott, E.A., 2019. The ECOSTRESS spectral library version 1.0. Remote Sens. Environ. 230, 111196. http://dx.doi.org/10.1016/j.rse.2019.05.015.

Meister, D., Bittner, J., 2018. Parallel locally-ordered clustering for bounding volume hierarchy construction. IEEE Trans. Vis. Comput. Graphics 24 (3), 1345–1353. http://dx.doi.org/10.1109/tvcg.2017.2669983, Publisher: Institute of Electrical and Electronics Engineers (IEEE).

Möller, T., Trumbore, B., 1997. Fast, minimum storage ray-triangle intersection. J. Graph. Tools 2 (1), 21–28. http://dx.doi.org/10.1080/10867651.1997.10487468, Publisher: Informa UK Limited.

Montes Soldado, R., Ureña Almagro, C., 2012. An Overview of BRDF Models. Technical Report, University of Granada, Granada, URL: http://hdl.handle.net/10481/19751.

Narayanan, R., Kim, H.B., Sohn, G., 2009. Classification of SHOALS 3000 bathymetric lidar signals using decision tree and ensemble techniques. In: 2009 IEEE Toronto International Conference Science and Technology for Humanity. TIC-STH, pp. 462–467. http://dx.doi.org/10.1109/TIC-STH.2009.5444456.

Pan, Y., Gao, B., Mei, J., Geng, S., Li, C., Zhao, H., 2020. SemanticPOSS: A point cloud dataset with large quantity of dynamic instances. _eprint: 2002.09147.

Peinecke, N., Lueken, T., Korn, B.R., 2008. Lidar simulation using graphics hardware acceleration. In: 2008 IEEE/AIAA 27th Digital Avionics Systems Conference. IEEE, pp. 4.D.4–1–4.D.4–8. http://dx.doi.org/10.1109/dasc.2008.4702838.

Piadyk, Y., Rulff, J., Brewer, E., Hosseini, M., Ozbay, K., Sankaradas, M., Chakradhar, S., Silva, C., 2023. StreetAware: A high-resolution synchronized multimodal urban scene dataset. Sensors 23 (7), 3710. http://dx.doi.org/10.3390/s23073710, Number: 7 Publisher: Multidisciplinary Digital Publishing Institute.

Polyanskiy, M.N., 2022. Refractive index database. URL: https://refractiveindex.info.

Poux, F., 2019. The Smart Point Cloud: Structuring 3D Intelligent Point Data (Ph.D. thesis). Université de Liège, Liège, Belgique.

RIEGL Laser Measurement Systems GmbH, 2017. LAS Extrabytes Implementation in RIEGL Software. Technical Report, RIEGL Laser Measurement Systems GmbH, Horn, URL: https://data.4tu.nl/file/1aac46fb-7900-4d4c-a099-d2ce354811d2/7ade80c4-aa45-4e87-b887-ee8478c96181.

Riordan, J., Manduhu, M., Black, J., Dow, A., Dooly, G., Matalonga, S., 2021. LiDAR simulation for performance evaluation of UAS detect and avoid. In: 2021 International Conference on Unmanned Aircraft Systems. ICUAS, pp. 1355–1363. http://dx.doi.org/10.1109/ICUAS51884.2021.9476817.

Rozenberszki, D., Litany, O., Dai, A., 2024. UnScene3D: Unsupervised 3D instance segmentation for indoor scenes. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 19957–19967.

Schäfer, J., Weiser, H., Winiwarter, L., Höfle, B., Schmidtlein, S., Fassnacht, F.E., 2023. Generating synthetic laser scanning data of forests by combining forest inventory information, a tree point cloud database and an open-source laser scanning simulator. Forestry: An Int. J. For. Res. http://dx.doi.org/10.1093/forestry/cpad006.

Serrano, A., Gutierrez, D., Myszkowski, K., Seidel, H.-P., Masia, B., 2016. An intuitive control space for material appearance. ACM Trans. Graph. 35 (6), 186:1–186:12. http://dx.doi.org/10.1145/2980179.2980242.

Su, H., Wang, R., Chen, K., Chen, Y., 2019. A simulation method for LIDAR of autonomous cars. IOP Conf. Ser.: Earth Environ. Sci. 234, 012055. http://dx.doi.org/10.1088/1755-1315/234/1/012055, Publisher: IOP Publishing.

Tachella, J., Altmann, Y., Mellado, N., McCarthy, A., Tobin, R., Buller, G.S., Tourneret, J.-Y., McLaughlin, S., 2019. Real-time 3D reconstruction from single-photon lidar data using plug-and-play point cloud denoisers. Nat. Commun. 10 (1), 4984. http://dx.doi.org/10.1038/s41467-019-12943-7, Number: 1 Publisher: Nature Publishing Group.

Tan, W., Qin, N., Ma, L., Li, Y., Du, J., Cai, G., Yang, K., Li, J., 2020. Toronto-3D: A large-scale mobile lidar dataset for semantic segmentation of urban roadways. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops. pp. 202–203.

Ullrich, A., Pfennigbauer, M., 2019. Advances in lidar point cloud processing. In: Turner, M.D., Kamerman, G.W. (Eds.), Laser Radar Technology and Applications XXIV. SPIE, pp. 157–166. http://dx.doi.org/10.1117/12.2518856.

U.S. Geological Survey, 2012. Lidar Base Specification: Collection Requirements. Techniques and Methods, U.S. Geological Survey, Series: Techniques and Methods.

Vacek, P., Jašek, O., Zimmermann, K., Svoboda, T., 2022. Learning to predict Lidar intensities. IEEE Trans. Intell. Transp. Syst. 23 (4), 3556–3564. http://dx.doi.org/10.1109/TITS.2020.3037980, Conference Name: IEEE Transactions on Intelligent Transportation Systems.

Varney, N., Asari, V.K., Graehling, Q., 2020. DALES: A large-scale aerial lidar data set for semantic segmentation. http://dx.doi.org/10.48550/arXiv.2004.11985, arXiv:2004.11985 [cs, stat].

Velodyne, 2018a. Velodyne LiDAR HDL-64E S3. URL: https://www.goetting-agv.com/dateien/downloads/63-9194_Rev-G_HDL-64E_S3_Spec%20Sheet_Web.pdf.

Velodyne, 2018b. Velodyne LiDAR puck hi-res. URL: https://www.mapix.com/wp-content/uploads/2018/07/63-9318_Rev-E_Puck-Hi-Res_Datasheet_Web.pdf.

Velodyne, 2018c. Velodyne LiDAR puck lite. URL: https://www.mapix.com/wp-content/uploads/2018/07/63-9286_Rev-H_Puck-LITE_Datasheet_Web.pdf.

Velodyne, 2019a. Velodyne LiDAR alpha prime. URL: https://www.mapix.com/wp-content/uploads/2019/11/VelodyneLidar_AlphaPrime_Datasheet.pdf.

Velodyne, 2019b. Velodyne LiDAR HDL-32e. URL: https://velodynelidar.com/wp-content/uploads/2019/12/97-0038-Rev-N-97-0038-DATASHEETWEBHDL32E_Web.pdf.

Velodyne, 2019c. Velodyne LiDAR puck. URL: https://velodynelidar.com/wp-content/uploads/2019/12/63-9229_Rev-K_Puck-_Datasheet_Web.pdf.

Velodyne, 2019d. Velodyne LiDAR ultra-puck. URL: https://velodynelidar.com/wp-content/uploads/2019/12/63-9378_Rev-F_Ultra-Puck_Datasheet_Web.pdf.

Weiser, H., Winiwarter, L., Anders, K., Fassnacht, F.E., Höfle, B., 2021. Opaque voxel-based tree models for virtual laser scanning in forestry applications. Remote Sens. Environ. 265, 112641. http://dx.doi.org/10.1016/j.rse.2021.112641.

Winiwarter, L., Esmorís Pena, A.M., Weiser, H., Anders, K., Martínez Sánchez, J., Searle, M., Höfle, B., 2022. Virtual laser scanning with HELIOS++: A novel take on ray tracing-based simulation of topographic full-waveform 3D laser scanning. Remote Sens. Environ. 269, 112772. http://dx.doi.org/10.1016/j.rse.2021.112772.

Wu, B., Zhou, X., Zhao, S., Yue, X., Keutzer, K., 2019. SqueezeSegV2: Improved model structure and unsupervised domain adaptation for road-object segmentation from a LiDAR point cloud. In: 2019 International Conference on Robotics and Automation. ICRA, pp. 4376–4382. http://dx.doi.org/10.1109/ICRA.2019.8793495, ISSN: 2577-087X.

Xiao, A., Huang, J., Guan, D., Zhan, F., Lu, S., 2021. SynLiDAR: Learning from synthetic lidar sequential point cloud for semantic segmentation. arXiv preprint arXiv:2107.05399.

Yang, X., Wang, Y., Yin, T., Wang, C., Lauret, N., Regaieg, O., Xi, X., Gastellu-Etchegorry, J.P., 2022. Comprehensive LiDAR simulation with efficient physically-based DART-Lux model (I): Theory, novelty, and consistency validation. Remote Sens. Environ. 272, 112952. http://dx.doi.org/10.1016/j.rse.2022.112952.

Yue, X., Wu, B., Seshia, S.A., Keutzer, K., Sangiovanni-Vincentelli, A.L., 2018. A lidar point cloud generator: from a virtual world to autonomous driving. In: ICMR. ACM, pp. 458–464.

Zhao, J., Li, Y., Zhu, B., Deng, W., Sun, B., 2021. Method and applications of lidar modeling for virtual testing of intelligent vehicles. IEEE Trans. Intell. Transp. Syst. 22 (5), 2990–3000. http://dx.doi.org/10.1109/TITS.2020.2978438, Conference Name: IEEE Transactions on Intelligent Transportation Systems.

Zheng, Y., Wang, G., Liu, J., Pollefeys, M., Wang, H., 2023. Spherical frustum sparse convolution network for lidar point cloud semantic segmentation. http://dx.doi.org/10.48550/arXiv.2311.17491, arXiv:2311.17491 [cs].

Zhou, Y., Han, X., Peng, M., Li, H., Yang, B., Dong, Z., Yang, B., 2022. Street-view imagery guided street furniture inventory from mobile laser scanning point clouds. ISPRS J. Photogramm. Remote Sens. 189, 63–77. http://dx.doi.org/10.1016/j.isprsjprs.2022.04.023.

Zohdi, T.I., 2020. Rapid simulation-based uncertainty quantification of flash-type time-of-flight and Lidar-based body-scanning processes. Comput. Methods Appl. Mech. Engrg. 359, 112–386. http://dx.doi.org/10.1016/j.cma.2019.03.056, Publisher: Elsevier BV.