







Article

GPU Rasterization-Based 3D LiDAR Simulation for Deep Learning

Leon Denis ^{1,2,*}, Remco Royen ^{1,2}, Quentin Bolsée ^{1,2}, Nicolas Vercheval ^{3,4}, Aleksandra Pižurica ³
and Adrian Munteanu ^{1,2}

¹ Department of Electronics and Informatics (ETRO), Vrije Universiteit Brussel, Pleinlaan 2, 1050 Brussels, Belgium; remco.royen@vub.be (R.R.); qbolsee@etrovub.be (Q.B.); adrian.munteanu@vub.be (A.M.)

² Imec, Kapeldreef 75, 3001 Leuven, Belgium

³ Department of Telecommunications and Information Processing (TELIN-GAIM), Ghent University, 9000 Ghent, Belgium; nicolas.vercheval@ugent.be (N.V.); aleksandra.pizurica@ugent.be (A.P.)

⁴ Department of Electronics and Information Systems, Clifford Research Group, Ghent University, 9000 Ghent, Belgium

* Correspondence: ldenis@etrovub.be

Abstract: High-quality data are of utmost importance for any deep-learning application. However, acquiring such data and their annotation is challenging. This paper presents a GPU-accelerated simulator that enables the generation of high-quality, perfectly labelled data for any Time-of-Flight sensor, including LiDAR. Our approach optimally exploits the 3D graphics pipeline of the GPU, significantly decreasing data generation time while preserving compatibility with all real-time rendering engines. The presented algorithms are generic and allow users to perfectly mimic the unique sampling pattern of any such sensor. To validate our simulator, two neural networks are trained for denoising and semantic segmentation. To bridge the gap between reality and simulation, a novel loss function is introduced that requires only a small set of partially annotated real data. It enables the learning of classes for which no labels are provided in the real data, hence dramatically reducing annotation efforts. With this work, we hope to provide means for alleviating the data acquisition problem that is pertinent to deep-learning applications.

Keywords: LiDAR; GPU; simulation; data generation; neural networks



Citation: Denis, L.; Royen, R.; Bolsée, Q.; Vercheval, N.; Pižurica, A.; Munteanu, A. GPU Rasterization-Based 3D LiDAR Simulation for Deep Learning. *Sensors* **2023**, *23*, 8130. <https://doi.org/10.3390/s23198130>

Academic Editor: Ikhlas Abdel-Qader

Received: 15 August 2023
Revised: 7 September 2023
Accepted: 26 September 2023
Published: 28 September 2023



Copyright: © 2023 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

The field of AI has witnessed rapid development in recent years [1–3]. The increasing flexibility of GPUs and their ability to consume vast amounts of data have led to the birth of deep neural networks and have stirred the AI landscape. This is more specifically also the case for the domain of Autonomous Vehicles, where artificial-intelligence-based solutions are indispensable [4–10]. One can regard the development of deep neural networks (DNNs) as a two-phased process: designing the architecture and acquiring and annotating data. The majority of research focuses on the former. This can be explained by the high-quality datasets publicly available but also by the challenge imposed by acquiring and annotating data. Though publicly available datasets are suitable for obtaining generalized models, the industry often has specific requirements and prefers specialized solutions that work optimally with their hardware. This, in turn, imposes obtaining specific, sometimes non-existent, annotated datasets. This is especially true for LiDARs, as hardware specifications can differ greatly. It has been shown that not taking the specifications of the LiDAR into account during training results in decreased performance due to sensor bias [11]. The main question then becomes how to acquire and annotate LiDAR-specific data. One has the choice between annotating real-world samples or generating synthetic data. Both have their merits and flaws. When leveraging real data, no mismatch exists between the training and test sets. However, compiling sufficiently large real-world datasets is very challenging given the enormous amount of accurately labelled data required to train reliable

models. When performing more advanced tasks, such as segmentation, manual annotation becomes borderline unfeasible, if not impractical. An overview of the processes involved for obtaining high-quality annotated datasets is provided in [12]. The processes rely on very precisely calibrated and synchronized hardware combined with state-of-the-art traditional parsing methods and deep-learning solutions. After processing, roughly 30% still needs to be labelled manually, which, in turn, almost guarantees the introduction of human errors. Lastly, real-world datasets tend to be prone to class imbalance. For instance, on average, cars are much more represented in traffic scenarios compared to other vehicles [13–17]. By contrast, simulated data do not suffer from any of the aforementioned limitations and can be used to generate well-balanced, perfectly annotated datasets. However, deviations from reality pose significant risks, and mismatches between training and test sets must be limited and closely observed.

This paper describes a novel GPU-accelerated Time-of-Flight (ToF) sensor simulator. The idea stems from the desire to apply deep-learning solutions for a custom LiDAR (Light Detected and Ranging) for which no annotated data are available. Though focusing on AI for solid-state LiDAR, we introduce concepts that are generic and can provide means to enable deep-learning on other sensing devices for which no annotated data are available. From a technical perspective, the proposed simulator harvests the computational power of the GPU by exploiting the real-time graphics rendering pipeline, hence significantly increasing data generating speeds compared to the state-of-the-art ToF simulators, which rely on ray casting [18,19]. Experiments reveal that the proposed simulator mimics reality accurately and is suitable for generating datasets to train neural networks. Summarised, our main contributions are as follows:

- The proposed simulator exploits the highly optimised rasterization pipeline of the GPU, increasing data generation speeds one hundredfold compared to the state-of-the-art. This is particularly useful for data-driven applications demanding vast amounts of data, such as deep-learning.
- The presented algorithms are generic and allow simulation of any ToF sensor, including those with unique sampling patterns, which has not been considered in any prior work to the best of our knowledge.
- We introduce a novel loss function leveraging synthetic and partially annotated real data to alleviate the mismatch between simulation and reality. It furthermore allows the model to learn classes for which no labels are provided in the real-world training set.
- Our techniques greatly alleviate, and in some cases completely eliminate, the tedious annotation process of real data, even for difficult tasks such as segmentation, unlike any prior work focusing on LiDAR. That is, we present a working pipeline for training reliable models for specific ToF devices with unique hardware specifications when no or limited annotated data are available.
- Two neural networks operating on real data for denoising and semantic segmentation are trained using synthetic point clouds generated by a digital twin of a real-world prototype LiDAR.

The remainder of this paper is organized as follows. Related works are presented first. Next, the proposed methodology and implementation details of the digital twin are detailed. This is followed by the experimental evaluation which discusses two deep-learning applications. Limitations are discussed in Section 5, which is followed by the conclusions of our work.

2. Related Work

AI for LiDAR has gained a lot of interest in recent years. The active developments in the research domain are most likely fuelled by the high interest expressed by the autonomous navigation industry. Most of those developments employ real-world data captured using a physical LiDAR [20–28]. However, realistic and performant LiDAR simulations are currently highly desirable given the difficulties with data gathering and annotation. Much

effort has therefore been made to create accurate ToF simulations. One can classify them as either being ray casting, rasterization, or AI-based.

Ray-casting-based techniques naturally stem from the fundamental principles of LiDARs, where multiple laser pulses are emitted to capture depth measurements. These simulators leverage ray casting algorithms, which have a well-established history in the field of computer graphics and were originally employed for tracing light rays to generate realistic imagery [29]. Given their intrinsic connection with the underlying hardware, it is unsurprising that the majority of LiDAR simulators fall within this category. Early works primarily concentrated on airborne LiDARs for terrestrial applications. Initial efforts predominantly focused on characterizing the properties of reflected laser beams, which is a crucial factor in determining the position of leaves during canopy scanning [30]. Subsequently, more attention was directed towards considering atmospheric effects on the laser beams [31]. With the advent of increased computational power, higher-resolution 3D models were incorporated, yielding more precise simulations for both terrestrial and urban scanning [32,33]. Later on, the fidelity of the underlying physical simulations improved through the integration of radiative transfer models into the simulations [34]. As these simulations became more accurate, virtual simulations were employed to determine the optimal scanning paths, particularly for canopy scanning to maximize leaf area coverage [35]. Later on, GPU flexibility increased and significant acceleration of ToF simulations was achieved through hardware-accelerated ray casting [36].

The recent emergence of autonomous driving has significantly redirected research efforts towards the automotive domain, particularly with a strong emphasis on the integration of deep-learning techniques. In the study presented in [37], the authors propose a ray tracing simulation within a traffic environment as a means to develop occupancy grid mapping algorithms. Their simulation's validity is substantiated through a virtual reconstruction of a real-world environment. Building upon this concept, ref. [38] takes a step further by employing video games in conjunction with ray-casting-based LiDAR simulations to generate simulated traffic data to train neural networks. A similar philosophy is embraced in [19], where the focus lies on constructing a traffic simulator using the Unreal Engine [39]. The authors' platform supports various sensors, including ray-casting-based LiDAR [39]. Subsequently, this simulator is utilized for training neural networks geared towards autonomous driving applications [40]. The use of synthetic data was also adopted by [41], in which real-world data are employed to enhance the accuracy of the physical properties of virtual environments and actors, such as reflected beam intensities and noise.

The rising success of data-driven AI for point clouds in turn led to the development of publicly available ray-casting-based ToF simulators. One of the pioneering efforts in the field was Helios [42]: an all-purpose ToF sensor simulator implemented in Java. Subsequently, an enhanced version was developed in C++, which led to a substantial reduction in simulation time, as discussed in [43]. More recently, the latest generic ToF simulator to emerge is Blainder [18], a simulator integrated with Blender [44]. Blainder offers the capability to generate meticulously annotated synthetic point data, marking a significant advancement in ToF simulation technology.

The second category of virtual ToF sensors comprises **rasterization-based** methods. Such simulators harvest the power of the GPU by exploiting the 3D rendering pipeline popularized by real-time graphics applications. The principle idea is to parallelize the computation of the first hit of the laser pulses by reverse-transforming the values of the depth map. The main advantage over the aforementioned ray-casting-based techniques is the dramatic reduction in computation time due to massive parallelization.

One of the earlier rasterization-based LiDAR simulators is discussed in [45]: it introduces a novel approach involving blending real-world backgrounds with synthetic actors, which are subsequently synthesized into the final images. This simulation method entails simulating a virtual LiDAR by initially projecting the virtual world onto cube-maps, which are then sampled. A similar methodology is employed in [46], where a combination of real-world data and synthetic actors is achieved through surface splatting techniques. In

contrast, the work presented in [47] exclusively relies on virtual data for LiDAR simulation. This approach employs equirectangular rendering and utilizes spherical projection and wrapping techniques to expedite the simulation process. Like the previously mentioned methods, it necessitates rendering the entire virtual world prior to simulation. Lastly, the research detailed in [48] introduces a LiDAR simulation technique that involves the insertion of virtual actors into real point clouds. This approach updates the existing rays within the original data by identifying ray correspondences through laser projection onto the synthetic image, offering a unique perspective on LiDAR simulation.

This work also belongs to this group, but it differs substantially from the aforementioned methods in the sense that it generalizes the ToF simulation. First, it allows for the virtually mimicking of the behaviour of both solid-state and electromechanical LiDARs. By contrast, the aforementioned citations only consider the latter and are not suitable for solid-state devices. Secondly, the unique sampling pattern is not accounted for in any existing works. In other words, no prior work enables the implementation of an exact digital twin, which is crucial for doing realistic simulations. Furthermore, the proposed simulator does not require the whole virtual world to be rendered for the simulation. Only portions visible by the LiDAR are rendered, thus significantly increasing simulation efficiency. Though those works can indeed be improved upon, it is fair to mention that they do not focus on the simulation. It is simply a byproduct for data augmentation. Simply put, the proposed methodology has the flexibility, realism, and generality of ray-casting-based simulators with the additional benefit of the computational efficiency imposed by the rasterization-based solutions.

Most recent advancements in the field have endeavoured to harness deep-learning techniques for **AI-based** simulating of LiDAR. One noteworthy contribution is exemplified by UniSim [49]. In this research, deep neural networks are applied to generate novel training data, accommodating dynamic actor modifications and limited sensor adjustments. A comparable approach is adopted in [50], which primarily concentrates on introducing new dynamic actors into real-world-captured footage. In contrast, ref. [51] takes a distinctive approach by enhancing the attributes of synthetic ray-casting-based LiDAR simulation through neural networks. This enhancement enables a more accurate representation of physical characteristics such as noise, reflected intensity, and dropped points. However, it is imperative to note that all these methodologies necessitate high-quality real-world data for training the neural networks, despite their ability to yield highly realistic results.

All related works have in common that they alleviate the **data acquisition problem**. Recent studies confirm that acquiring and labelling data indeed poses one of the main challenges for reaching Level 5 vehicle autonomy [52,53]. This is particularly more challenging for LiDAR, as each device has its own sampling pattern, noise model, and pulse intensity measurements. Ignoring such parameters is possible, and potentially mismatched training data can be extracted from existing datasets [12,54,55]. However, another solution is necessary when the required data simply does not exist. Prior to ApolloScape [12], no labels for flora were available for point clouds. The increasing realism of real-time graphics and the required efforts of data gathering have motivated researchers to leverage virtual simulators for both unmanned aerial vehicles [56,57] and autonomous driving [58,59]. Recent works in the field of deep-learning have also benefited from such simulators [60–63]. Others synthesize training samples by combining real data with virtual actors [45,64]. To the best of our knowledge, all previous works involving LiDAR simulations require real annotated data for producing acceptable performance [38,40,41,46,60–62]. We will show later that this is also true for one of our applications. However, unlike any previous work, we will introduce a novel training methodology in which labels for new classes can be completely omitted in the real training data, thus significantly reducing annotation efforts.

3. GPU-Accelerated LiDAR Simulation

This section thoroughly discusses the proposed GPU-accelerated simulator. We start by briefly explaining the underlying mechanics of ToF sensors and by providing generic al-

gorithms suitable for simulating such sensors. Thereafter, we explicate the implementation of a virtual digital twin based on the hardware specifications of its real world counterpart.

3.1. Basic Principles ToF Sensors

Figure 1 illustrates the fundamental principles of a ToF sensor. At its core, a ToF sensor comprises a cost-effective Complementary Metal-Oxide-Semiconductor (CMOS) pixel array sensor paired with an active infrared light source. Depth measurements are derived by illuminating the scene with modulated light and measuring the time it takes for the reflected light to return to the CMOS sensor. These time measurements are subsequently translated into depth values. The modulated light source typically encompasses projected infrared laser beams, with each laser pulse yielding a depth measurement. ToF cameras, such as the Microsoft Kinect v2, construct depth maps wherein depth measurements are organized in a rectangular grid format, analogous to an image. Conversely, LiDARs often employ distinctive sampling patterns, resulting in the creation of point clouds.

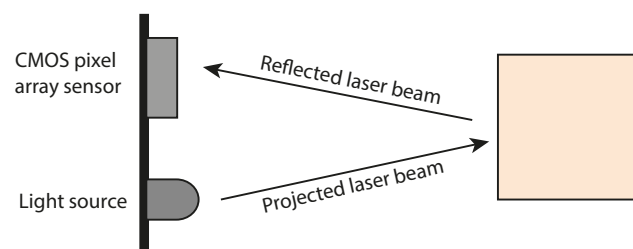


Figure 1. A visual schematic of a Time-of-Flight sensor. Depth measurement is achieved by emitting a laser beam into the scene and subsequently calculating the time required for the laser's reflection to reach a complementary CMOS pixel array sensor.

As mentioned before, the projection of rays originating from a light source has been extensively studied in the past, particularly in the context of computer-generated imagery and notably within the realm of ray tracing. Considerable effort has been dedicated to optimizing ray casting algorithms due to the substantial time investment required for generating realistic images, which involves tracing millions of light rays [65]. Simulation of ToF sensors can be achieved by adapting these methods, effectively simulating the firing of rays into the virtual scene for each laser pulse to retrieve depth information. Given the close correspondence between the simulation process and the physical ToF device, it is unsurprising that a significant portion of ToF simulators rely on such ray casting methodologies. As far as our knowledge extends, all generic ToF simulators adopt a ray-casting-based approach, including examples like Helios [42], its improved version [43], and Blainder [18]. These simulators are strictly implemented on the CPU and often cast rays either sequentially or with limited parallelization.

3.2. Rasterization-Based ToF Sensor Simulation

Unlike ray-casting-based methods, which sequentially simulate laser pulses, the proposed method simulates *all* rays in parallel by exploiting the rasterization pipeline of the GPU for determining the first-hit of each laser pulse. More specifically, our simulator reverses the forward 3D rendering pipeline by employing data stored in the framebuffer for reconstructing the input geometry of the virtual 3D space in a point-wise manner, hence simulating the behaviour of ToF sensors. A simplified version of the pipeline is depicted in Figure 2. The blue blocks in the figure indicate reprogrammable stages of the rendering pipeline. As will be explained later, the proposed simulator only requires the use of the vertex and fragment shaders, making it compatible with older hardware. In a general context, the task of the vertex shader is to project vertices from local coordinate space to the image plane using a series of affine transformations. The transformed vertices are passed to the geometry shader, after which the input polygons are rasterized, colourized, and

clipped in order to fit the image plane. The geometrical transformations applied to each input vertex in the forward rendering pipeline can be mathematically expressed as

$$p_i = v(M_{proj} \cdot M_{view} \cdot M_{model} \cdot p_l), \quad (1)$$

with p_l representing an input vertex in its local coordinate system, also referred to as model space. M_{proj} , M_{view} , and M_{model} denote the projection, view, and model matrices, respectively. The variable v transforms data from clipping space to image space. The resulting point p_i is a homogeneous coordinate on the image plane, with z corresponding to the z -buffer value. The proposed simulator computes the first hit of cast rays by partially reversing the chain of transformations to obtain points in camera space:

$$p_c = M_{proj}^{-1} \cdot v^{-1}(p_i). \quad (2)$$

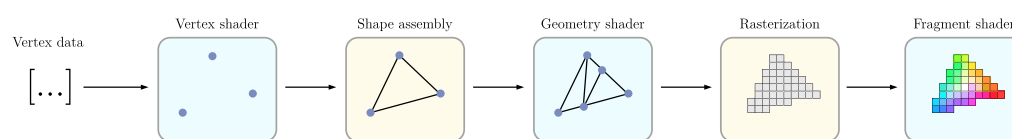


Figure 2. The 3D real-time rendering pipeline. Vertices are transformed to world space by the vertex shader, and faces are constructed. The embedded pixels defined through rasterization are coloured by the fragment shader. The proposed method constructs point clouds by reversing the pipeline. More specifically, the coordinates of each fragment are transformed back to camera space and stored in the fragment data, which is subsequently written to a texture and retrieved from the GPU.

As will be explicated later in Section 3.3, the projection matrix must be constructed following specific hardware guidelines dictated by the simulated ToF sensor and must remain constant through the simulation. We therefore compute M_{proj}^{-1} once on the CPU and make it available to the fragment shader rather than computing it on the GPU.

Synthetic point clouds are produced through the process of rendering a specific three-dimensional scene via the forward rendering pipeline, yielding a 32-bit floating-point texture as the output. In accordance with standard practices, the primary responsibility of the vertex shader lies in the projection of input vertices from the virtual world onto the image plane, as dictated by Equation (1). Subsequently, the simulation of LiDAR rays is carried out at pixel-level granularity within the fragment shader. This simulation involves the partial reversal of the transformation sequence. Specifically, camera space coordinates are obtained using the corresponding pixel's image space coordinates together with the inverse projection matrix and reversing the forward rendering pass by applying the reverse transformations according to Equation (2). Depth information can be obtained directly since the coordinate system's origin and the camera's position coincide. Throughout this procedure, all other attributes pertaining to the point cloud are calculated at pixel-level in the fragment shader and are subsequently stored in the framebuffer object. After rendering, the output texture is retrieved from the GPU to the main memory, from which point clouds are constructed by associating on a per-pixel basis the camera space coordinates with their respective point attributes. Every constructed point corresponds to a laser pulse of the virtual LiDAR, with its position coinciding with that of the camera. The camera thus acts as the ToF sensor.

3.3. Matching Hardware Specifications

To realistically mimic real hardware, camera parameters should be selected according to sensor specifications. The aspect ratio, rendering resolution, and Field-Of-View (FOV) must perfectly match that of the sensor. Near and far planes must be chosen to reflect the

minimum and maximum sensing range, respectively. The following perspective-projection matrix can serve that purpose:

$$M_{proj} = \begin{bmatrix} \frac{h}{w \tan(\alpha/2)} & 0 & 0 & 0 \\ 0 & \frac{1}{\tan(\alpha/2)} & 0 & 0 \\ 0 & 0 & \frac{-(f+n)}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix}. \quad (3)$$

This is derived from the standard projection matrix commonly used in real-time hardware-accelerated rendering. The variables w and h symbolise the width and height of the framebuffer, respectively. The near and far planes are denoted by n and f , respectively. The vertical FOV is represented by α . M_{proj} projects the virtual input geometry on the image plane according to the exact specifications of the simulated ToF sensor. As elaborated in Section 3, the output texture is subject to sampling after its retrieval from the GPU, which is a step integral to the point cloud construction process following forward rendering. To match hardware specifications, the image plane must be sampled akin to the depth-measuring pattern of the real-world device. The Microsoft Kinect v2, for example, produces depth maps of resolution 512×424 . In this case, the output texture can match that resolution, after which pixel-wise sampling can be employed for point-data generation. LiDAR laser pulses typically are not as structured. To imitate the pattern of their laser projections, nearest-neighbour sampling can be done using uv -coordinates derived from hardware specifications. If such information is unavailable, one can point the LiDAR towards a flat surface and derive the uv -coordinates by normalizing the x and y values of the generated point cloud along their respective dimensions. Optionally, the utilization of higher-resolution rendering can be considered as a means to ensure that each laser pulse corresponds to a different pixel. However, it remains imperative to maintain the perspective projection as described by Equation (3). In our specific case, we render the output image at a resolution twice that of the real-world device. Figure 3 compares the results obtained through pixel-wise and uv -sampling. The image on the right mimics our LiDAR, for which uv -coordinates were derived from hardware specifications.

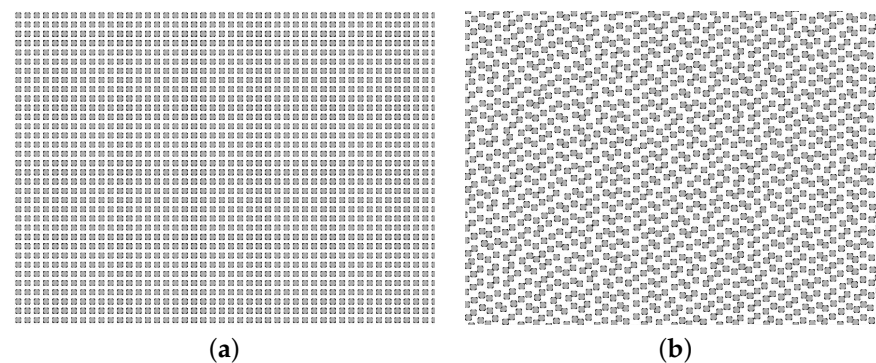


Figure 3. An example of a point cloud generated directly from the rasterization grid (a) and when using the sampling pattern of the prototype LiDAR (b). The grid-like pattern in (a) is the result of the equidistant pixels of the rendered output image. By contrast, LiDARs exhibit their own unique sampling pattern, which can be mimicked by associating uv -coordinates to each laser pulse. The uv -coordinates can be extracted by directing the real-world LiDAR perpendicular towards a flat surface, resulting in the patterns shown in (b).

3.4. Noise Model

Real-world depth measurements of ToF devices are subject to noise. The measurement uncertainty is caused by numerous factors such as photon shot, dark current noise, readout noise, etc. [66]. Previous research has established that the perturbations to the depth values are normally distributed [66,67] and are dependent on the distance of the hit surface, its

reflectance, and the ambient light. Though the exact noise model varies among sensors, all manifest noise along their respective rays. This characteristic is an inherent outcome of the underlying mechanics of ToF sensors. As the directions of the laser pulses are known, the perturbation $\vec{p}_{n,i}$ of a given point p_i can be formulated as

$$\vec{p}_{n,i} = p_i + \mathcal{N}(0, Z(d_i, \varphi_i, l_i)) \vec{R}_{e,i}, \quad (4)$$

with $\vec{R}_{e,i}$ denoting the vector from the origin of the emitter to p_i . $\mathcal{N}(0, Z(d_i, \varphi_i, l_i))$ represents a random variable distributed normally with zero mean and variance $Z(d_i, \varphi_i, l_i)$. The variables d_i , φ_i , and l_i denote the depth, albedo, and ambient light, respectively, associated with p_i . An example obtained when applying noise on a point cloud generated by our digital twin is shown in Figure 4b. Clean point clouds such as the one depicted in Figure 4a may serve as ground truth for training neural networks for denoising purposes. Interestingly, they can never be acquired with real-world LiDARs—they can only be obtained through simulation using a virtual digital twin.

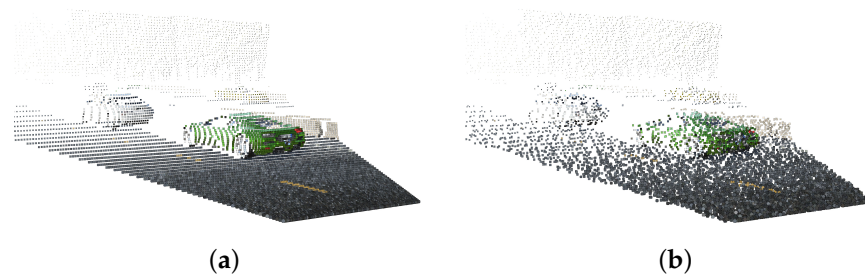


Figure 4. An example of a clean (a) and noisy (b) point cloud generated by the digital twin of our LiDAR. The clean point cloud directly results from reversing the rasterization pipeline of Figure 2. In reality, points captured by real-world LiDARs exhibit spatial noise in the form of displacements along their respective laser pulses, as shown in (b). This is the result of imprecise measurements and/or ambient light interaction or due to the reflectance properties of the hit surface.

3.5. GPU Implementation Details

The close relationship of the proposed methodology with 3D hardware-accelerated rendering makes it highly suitable for a GPU implementation. In fact, our simulator can be almost fully processed by the programmable graphics pipeline. The only exception is the simulation of the unique LiDAR sampling patterns, which requires non-uniform reading of the output texture, which, in turn, dictates CPU processing. Nevertheless, in all cases, the computation of the first-hit point for each ray is carried out on the GPU, making use of its rasterization hardware. This approach yields a substantial reduction in execution time, as this particular step demands the most significant computational resources. We will further substantiate this assertion through our experimental evaluation, as outlined in Section 4. We emphasize that the reduction in execution time is particularly important for data-driven applications, such as deep-learning, which require vast amounts of data to produce reliable results.

As mentioned before, our implementation uses a GPU program that outputs for each pixel fragment its corresponding camera space position to a texture, which is subsequently retrieved to the main memory and is sampled to match the hardware specifications. In our implementation, z-buffer values are directly retrieved in the fragment shader as supported by the OpenGL Shading Language (GLSL) [68]. Not all GPU programming languages support this feature. If not supported, the depth texture must be generated using a custom GPU program in an additional rendering phase and be made available to the GPU program.

To avoid value clipping and quantization artefacts, 32-bit floating-point textures are employed. Using textures with limited bit-depth negatively impacts the quality of the produced point clouds. This is also true for the bit-depth of the z-buffer, which must be increased from the standard 16 bits to 32 bits. We note that this is supported by all

current real-time-rendering APIs. Failure to use sufficient bit resolution in any point of the rendering pipeline results in aliasing artefacts, as demonstrated by Figure 5a. In our experiments, we noticed that a 32-bit process suffices to generate clean point clouds, such as the one shown in Figure 5b.

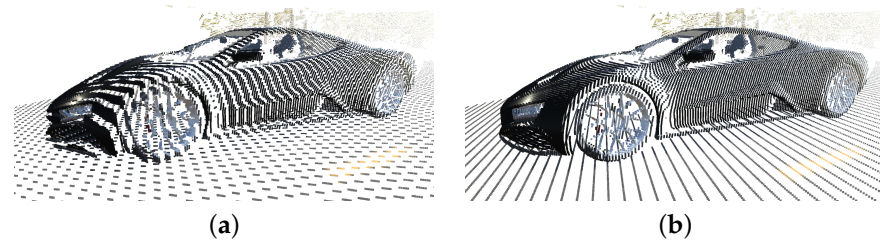


Figure 5. This figure demonstrates the impact of the bit-depth of the textures used for point cloud generation. Limiting the bit-depth in any phase of the rendering pipeline will result in aliasing artefacts (a). Employing 32 bits for the z-buffer and output textures produces clean point clouds (b).

4. Experimental Evaluation

This section evaluates the performance of our simulator. We compare data generation speeds with the state-of-the-art and assess the validity of the synthetic data for two deep-learning applications, namely, denoising and semantic segmentation. We note that the main purpose of the discussed applications is to validate the simulator and to prove that training can be done successfully using the generated data.

4.1. Data Generation Speed

We compare our method with two different techniques from the literature: Blainder [18], which is the most recent ToF simulator for multiple sensors, and Carla [19], which is the state-of-the-art in driving simulation and also supports numerous sensors, including LiDAR. Execution times for different amounts of virtual laser pulses are summarized in Table 1. Experiments were carried out on a laptop with an Intel i7-8950 and an RTX 2070 max-q. The table shows that the proposed methodology allows generating point clouds consisting of 1 million samples at roughly 90 Hz. In other words, real-time high-resolution simulation can be achieved with relatively modest hardware. Resolution and execution times are linearly correlated for all methods except at very low resolutions for which rendering overhead may become dominant. Comparing with Blainder [18], we notice the average data generation time is reduced by a factor of 300. For Carla [19], the speed improvements add up to a factor of 100. The large performance increase of the proposed method is due to fully employing the GPU rasterization pipeline. In contrast, both Blainder [18] and Carla [19] rely on ray-casting methods. We do acknowledge that a GPU implementation of Blainder [18] or Carla [19] will narrow the gap in terms of execution times. We have therefore implemented a rudimentary simulator using NVIDIA Optix [69], which employs GPU-accelerated ray casting. Our simulator vastly outperforms this method as well, especially for low resolutions, where the overhead of building acceleration structures used by the ray-casting methods becomes dominant.

We emphasize that the results of the proposed simulation are identical to those produced by ray-casting techniques. That is, all methods presented in this section generate identical point clouds given the same sensor configurations and virtual environment. This is due to all of them being physically accurate. However, it is important to note that Carla [19] does not provide a general solution for ToF sensor simulation and that a solid-state LiDAR, for instance, cannot be simulated, unlike the proposed work. Therefore, both Blainder [18] and Carla [19] would benefit greatly by adopting the proposed method in terms of execution times. For Carla [19], this also comes with increased flexibility, allowing it to simulate any ToF sensor. Both findings demonstrate the value of the proposed techniques. Furthermore, our method is compatible with even the most basic GPUs, as no specialized hardware is required. Lastly, we remark that the proposed methodology can

be integrated into any rasterization-based rendering engine, hence providing access to its modelling tools.

Table 1. Data generation speed comparison with the state-of-the-art for different number of simulated laser pulses. The numbers are expressed in milliseconds. Note that the proposed method reduces data generation time by a factor of 300 compared to Blainder [18], which is the state-of-the-art in ToF sensor simulation. For the state-of-the-art driving simulator Carla [19], execution times are reduced by a factor of 100. A custom implementation revealed that the proposed method still vastly outperforms a lightweight ray-casting-based ToF simulator relying on NVIDIA OptiX [69] for GPU acceleration.

Simulator	Simulated Laser Pulses				
	1K	10K	100K	1M	10M
Blainder [18]	45.20	79.76	566.73	5381.91	53,750.80
Carla [19]	4.24	5.78	109.75	1386.94	11,653.42
NVIDIA OptiX [69]	18.35	19.02	26.72	74.76	559.74
Proposed	0.61	0.81	2.27	10.87	113.41

4.2. Point Cloud Denoising

The first deep-learning application considers spatial denoising. We discuss the data generation process and present results for real-world data.

4.2.1. Dataset Generation

Training data are generated using the previously explained methodology. Ground truth is obtained by simply omitting the virtually added noise during point cloud generation. It is important to note that in this context, obtaining ground-truth data from the real-world device is not feasible due to the inherent noise always present in such devices, as mentioned previously. Generic meshes with varying geometrical features are employed to obtain robustness against all kinds of surface shapes. Point clouds are obtained through random scene compositions and by recovering the point data using our digital twin. For each point, KNN patches are extracted and an affine transformation is applied to locate the central point at the origin, with the laser pulse being aligned with the Z-axis of the coordinate system. Rotation is ignored, as it is not relevant for local denoising. Next, data are normalized independently along all axes to fit the [0, 1] interval. The normalized patches serve as input for the neural network, which outputs the offset along the z-axis with respect to the ground truth.

4.2.2. Performance Evaluation

We evaluate the performance with real data through controlled calibration environments and a captured traffic scene. The first environment consists of the LiDAR pointing perpendicularly towards a wall and placed at a distance of 20 m. PointNet [70] is employed to perform the denoising task. The network has been trained using the training data previously mentioned and thus solely relies on synthetic data. Figure 6 illustrates the obtained point clouds before and after denoising. The figure clearly demonstrates the denoising capability of the network, which significantly improves the depth measurements of the LiDAR. Concretely, the root-mean-square error is reduced from 284.16 mm to 54.44 mm. The variance is reduced from 150.6 to 65.9 mm.

A second experiment positions three boards with known reflective properties at 100 m distance in an outdoor environment. An image of the setup is shown in Figure 7. Table 2 provides the standard deviation before and after the denoising. From the table, the denoising capabilities of the network are clear, which in turn proves the validity of the generated synthetic data and thus the simulator. As summarized by the table, noise is reduced by approximately 20%.

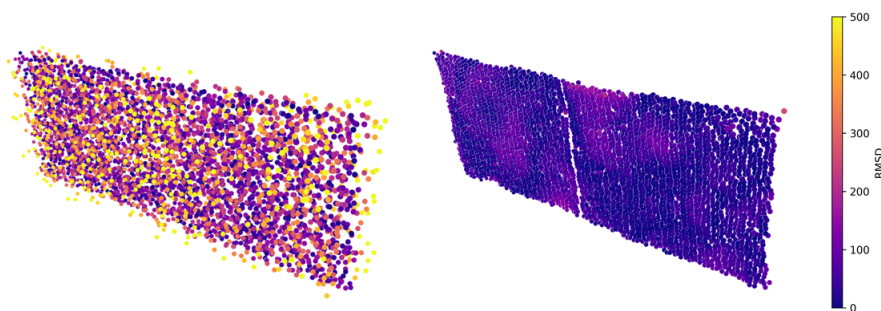


Figure 6. Visual results for our first experiment. The point clouds are captured by our real-world LiDAR for a calibration setup. Results before and after denoising are shown on the left and right, respectively. After denoising, the variance is reduced from 150.6 mm to 65.9 mm. Denoising was performed using a deep neural network trained with only synthetic data generated by our digital twin.



Figure 7. This figure shows the controlled environment for a second denoising experiment. The setup consists of 3 boards of different colours with known reflective properties and positioned at a distance of 100 m from the real-world LiDAR. Point clouds are captured and denoised using a neural network trained on synthetic data generated by our digital twin. Denoising results are shown in Table 2.

Table 2. This table shows the standard deviation of the measured distance of the test setup shown in Figure 7. Results are provided in meters. Point clouds were captured by our real-world LiDAR from a distance of 100 m and denoised using the deep neural network trained solely on synthetic data. The experiment reveals that the neural network is successfully trained and reduces noise for all 3 coloured boards by approximately 20%.

Target	Before Denoising	After Denoising	Noise Reduction
Dark	6.54	5.24	19.88%
Gray	4.71	3.61	23.35%
Light	2.46	1.96	20.33%

The third experiment comprises a real driving scenario. We only provide qualitative results as no ground truth data are known. Figure 8 shows a bird’s-eye view of two frames captured while driving. The colours indicate the intensity of the reflected laser pulses. After denoising, more structure is obtained in the point clouds. Edges are more refined, and driveable areas are more easily distinguishable. The vehicle on the left in Figure 8d is also more clearly defined compared to Figure 8c. From these observation, it is clear that the synthetic data generated by our digital twin can be leveraged to train neural networks for denoising purposes, thus, in turn, validating the proposed simulator.

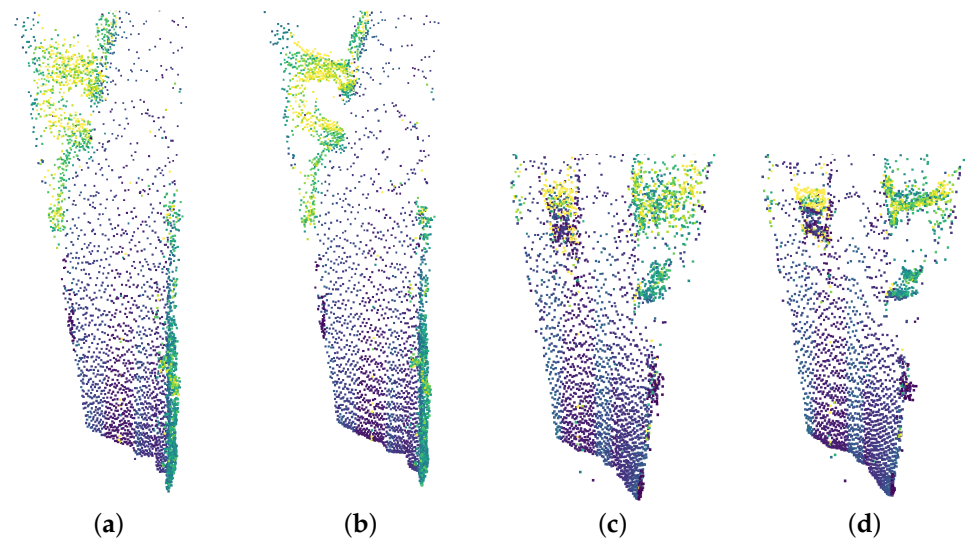


Figure 8. Bird’s-eye view of two frames captured by the real-world LiDAR. The colours indicate the intensity of the reflected laser beams. The noisy point clouds are shown in (a,c). The corresponding denoised frames are given in (b,d), respectively. Notice the more refined edges after denoising in (b), which reveals the open space in the top left corner. For the point cloud in (d), one can notice the vehicle at the top left becoming more defined. This is also true for the structures at the top right.

4.3. Semantic Segmentation

To further assess our synthetic data, semantic segmentation is performed. Point clouds and annotations are extracted using a digital twin implemented in Unity [71].

4.3.1. Dataset Generation

A rudimentary virtual world has been built to resemble the test data captured by our real-world LiDAR during a driving scenario. Figure 9 shows the 3D scene in more detail. Training data consist of 10,000 generated samples with cars randomly—but realistically—placed on the road. Camera parameters are slightly randomized to prevent overfitting and to obtain network robustness. Specifically, we enhance the network by introducing uniform noise within the range of -1° to 1° for the camera orientation. We apply the same principle to the spatial position, where uniform noise ranging from -0.1 to 0.1 m is introduced. This deliberate noise addition aims to prepare the network for handling variations in LiDAR placement that can occur in real-world settings. Projection parameters are retained and perfectly match those of the real LiDAR. Three different classes are used in our experiments: car, street, and flora. Point clouds are automatically annotated by rendering the segmentation map using colours corresponding to each class. We note that anti-aliasing features should be disabled for acquiring clean labels void of pixel blending. Figure 10 illustrates the different data layers associated with our annotated point cloud extraction method.

As mentioned previously, our simulator closely simulates the behaviour of the real-world LiDAR. Figure 11b shows a render mimicking the CMOS sensor of our LiDAR. The reference image in Figure 11a illustrates the resemblance with the synthetic data, both in terms of viewing parameters and in captured luminance. We note that the images are included for illustration purposes and that only depth information is used for point cloud generation. The choice is deliberate in order to void the incoherence of night and day when performing segmentation.

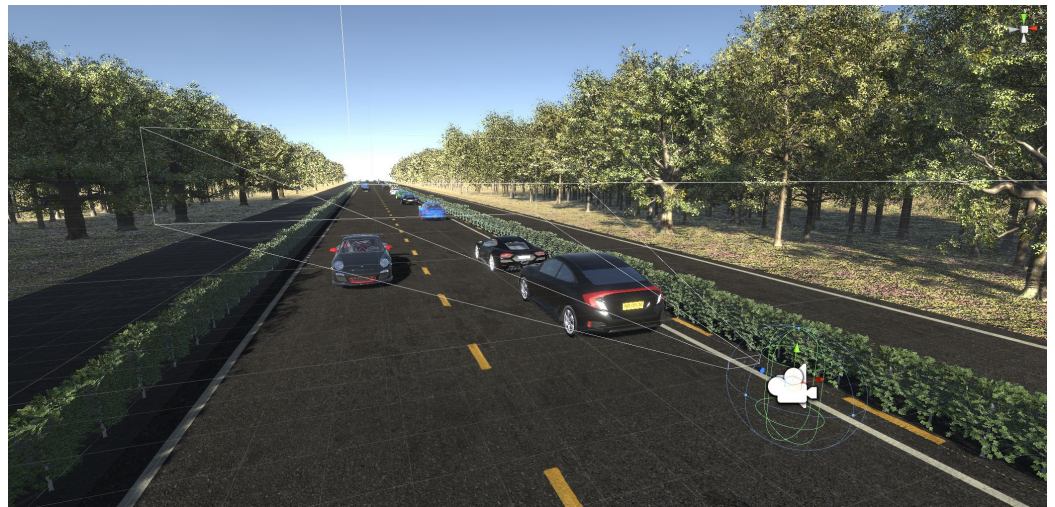


Figure 9. The proposed digital twin of our LiDAR is implemented in the Unity engine [71]. The figure demonstrates the scene used for data generation to train a neural network for semantic segmentation purposes. The spatial position of the camera coincides with that of the virtual LiDAR. The scene is modelled to resemble the environment of a real driving scenario captured using a real-world LiDAR, which is utilized for validating and testing the AI model.

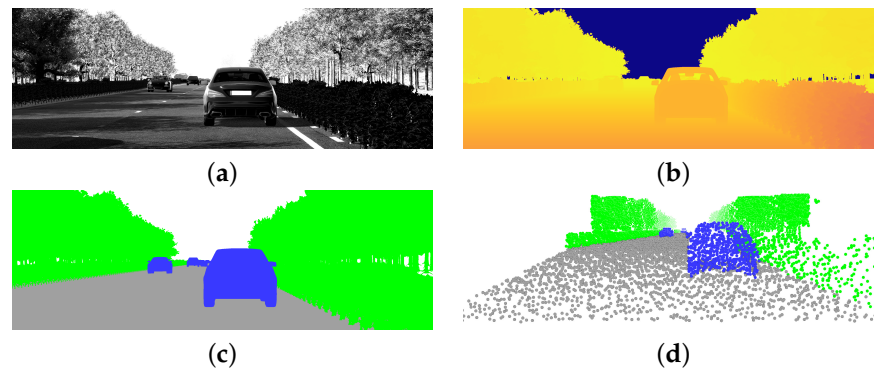


Figure 10. Data layers for point cloud generation using our digital twin implemented in Unity [71]. The luminance is visualised in (a). The corresponding depth and segmentation maps, shown in (b,c), are used to construct the auto-annotated point cloud of (d). Specifically, points are constructed by reversing the rasterization pipeline according to Equation (2) and subsequently mimicking the real-world LiDAR's laser pattern by sampling the output textures (b,c) using the LiDAR's associated uv-coordinates. Annotations are assigned on a nearest-neighbour-basis using the segmentation map shown in (c), resulting in a perfectly annotated point cloud according to the specifications of the real-world LiDAR.



Figure 11. Comparison between images captured by the CMOS sensor of our real-world LiDAR (a) and its digital twin (b). Though the synthetic images are not used during training, they clearly demonstrate the resemblance of the output of the proposed digital twin and the real-world device. Resolution, field-of-view, sensor sensitivity, LiDAR position, and orientation are all accounted for during simulation, which is mandatory for successfully training neural networks operating on real data.

Unlike the previous application, we have enriched the training dataset with real-world *partially* labelled data. Annotations for *cars only* are obtained by inferring the CMOS image with a retrained Xception 65 model [72] trained on the Cityscape dataset [14] and transferring the labels from the CMOS image to the point clouds. Labels are manually corrected. The dataset enrichment is done to bridge the gap between reality and simulation. The incoherence in both datasets is not necessarily caused by the digital twin, but rather, it is due to the somewhat simplistic 3D scene, which strays from reality. Concretely, the training data comprise 10,000 artificial samples annotated with the classes car, street, and flora, and a small set of real data with only the cars labelled.

4.3.2. Custom Loss

The annotation mismatch between real and synthetic data imposes a new training methodology: we propose a novel loss function that combines samples with differently labelled classes:

$$L = \|Y_{synth} - h(X_{synth})\|_2 + \lambda \|Y_{real} - h(X_{real})\|_2, \quad (5)$$

with X_{real} and $X_{synth} \in \mathbb{R}^{N \times 3}$ denoting the real and synthetic input point clouds, respectively. $Y_{synth} \in \mathbb{R}^{N \times 3}$ symbolises the perfect synthetic semantic labels, whereas $Y_{real} \in \mathbb{R}^{N \times 1}$ represents the real semantic car labels. The regularization factor is denoted by λ and the model by h . Since two prediction steps are required per optimisation step, each batch is subdivided into two sub-batches containing either real or synthetic samples. Predictions for each are retrieved thereafter. To guarantee processing the same number of synthetic and real samples, circular linked lists are employed. For each list, samples are reshuffled after processing the number of different samples in the list. The optimization step is done jointly to preserve smooth convergence.

4.3.3. Performance Evaluation

We once more employ PointNet to test our approach. Purely for evaluation purposes, we obtain a test set containing 824 real-world samples annotated similarly to the real training set but with the inclusion of all categories instead of solely the cars. In this case, labels are not manually corrected, as this proved impractical. A representative sample is illustrated in Figure 12. Table 3 summarises the mean Intersection-over-Union (mIoU), overall accuracy (oAcc), and mean accuracy (mAcc) for three different experiments. The table indicates that solely relying on synthetic data leads to acceptable performance for flora and street. The IoU for cars shows room for improvement despite the high accuracy. Utilising only real data improves the IoU from 32.7 to 50.8. However, in this case, street and flora cannot be segmented, as such labels were not available during training. Employing the enriched dataset of 800 real samples dramatically improves the model, resulting in increased performance for all classes—even for street and flora, for which no extra labels are provided.

Visual results are presented in Figure 13. The CMOS image is included for reference. Results are obtained using the enriched dataset comprised of 1200 real-world samples. From the figure, it is clear that the trained network is able to segment the real captured point clouds well. Multiple cars are correctly and simultaneously segmented, as illustrated in Figure 13a,b. Minor confusion for the traffic sign is noticeable, which can be explained by such data not being present in the training set. Figure 13f shows a zoomed-in portion of the point cloud of Figure 13e. It illustrates the network being able to distinguish between car and road even when having a limited set of points. Overall, the street is accurately segmented as well. Most notable, the exit lane and T-junction in Figure 13c,d are accurately predicted even though such scenarios were not present in the virtual scene. Lastly, we also notice good inference for flora. Though perhaps not necessary in practice, similar as for the street, its prediction was freely obtained, as no manual annotation was involved. This example clearly shows the benefit of the proposed techniques.

Table 3. Quantitative semantic segmentation results for a driving scenario captured using our real-world LiDAR. We note that only cars are annotated for the real data. The model trained solely on such data is therefore unable to predict other classes. The table demonstrates that the network trained solely on synthetic point clouds achieves satisfactory results in semantic segmentation. Furthermore, this model outperforms the network trained on 800 real-world samples, specifically for the “car” class, in terms of accuracy. The best results are obtained when incorporating both synthetic and a small amount of real data using the proposed loss function defined in Equation (5).

Training Data		Results								
Synth	Real	Car		Street		Flora		Average		
		Acc	IoU	Acc	IoU	Acc	IoU	oAcc	mAcc	mIoU
10,000	0	77.38	32.72	72.48	67.83	89.74	74.71	79.90	78.86	58.35
0	800	74.02	50.81	-	-	-	-	-	-	-
10,000	800	79.72	62.88	90.73	83.29	89.18	80.55	89.36	86.51	75.64

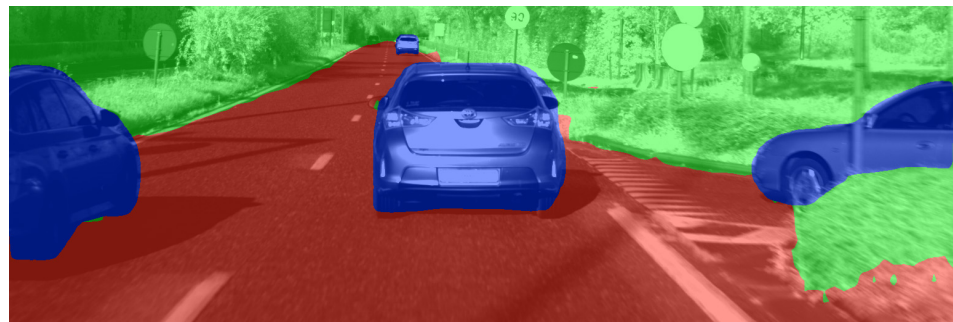


Figure 12. A representative real-world sample of the ground truth for the CMOS image captured by our LiDAR. The annotations are obtained using a retrained Xception 65 model [72] trained on the Cityscape dataset [14]. Note that annotations are not manually corrected, as this proved impractical. Annotated point clouds are obtained by transferring the labels from the image to the associated point clouds, which, in turn, serve as ground truth for model testing.

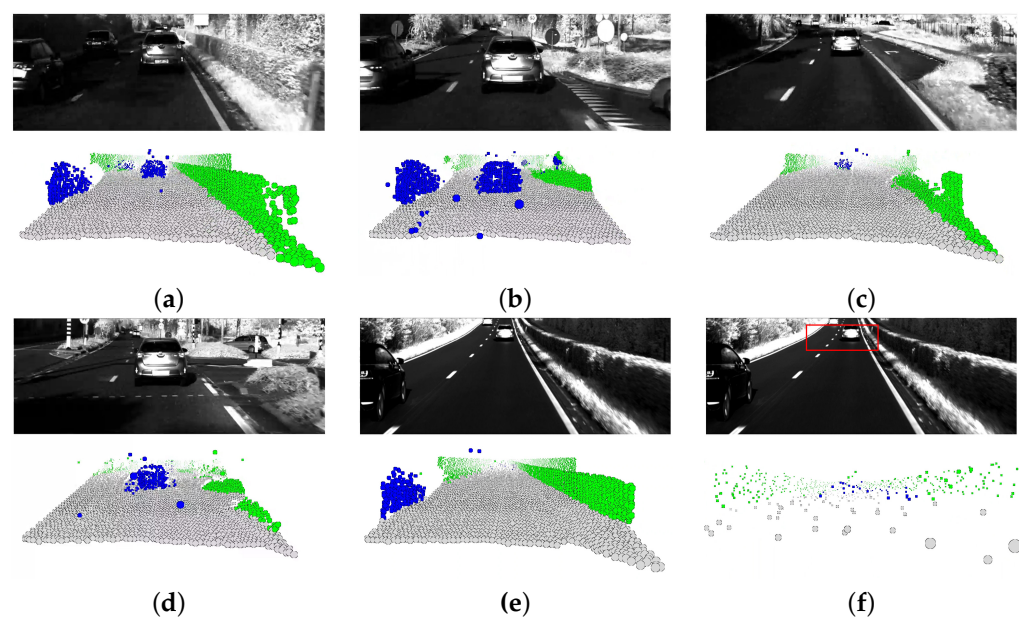


Figure 13. Visual segmentation results for data captured by our real-world LiDAR. The colours blue, gray, and green represent cars, the street, and flora, respectively. The employed model was

obtained using the synthetic dataset enriched with 1200 real-world samples. The figures clearly demonstrate that the trained neural network performs well in segmenting point clouds obtained from the real-world LiDAR system. Notably, the network is capable of simultaneously detecting multiple cars, as evidenced in (a,b), and it accurately defines the boundaries of the street, even when it is not simply straight, as exemplified in (c,d). An interesting observation is that the network is able to distinguish between the street and cars with a high degree of accuracy using only a small subset of selected points, as demonstrated in (f), which is a zoomed-in section of the point cloud shown in (e) and corresponds with the area inside the red rectangle of (f).

As a last experiment, we provide results of an ablation study demonstrating the effect of the number of real-world samples with respect to prediction accuracy. Figure 14 summarizes the results. It is noteworthy that only 25 of such samples suffice for dramatically improving prediction accuracy, increasing the mIoU from 58.35 to 68.62.

The results presented here once more show the successful training of a deep network, which, in turn, validates the proposed simulator and proves that the generated data resemble reality close enough even for segmentation tasks, which are particularly sensitive to data mismatch.

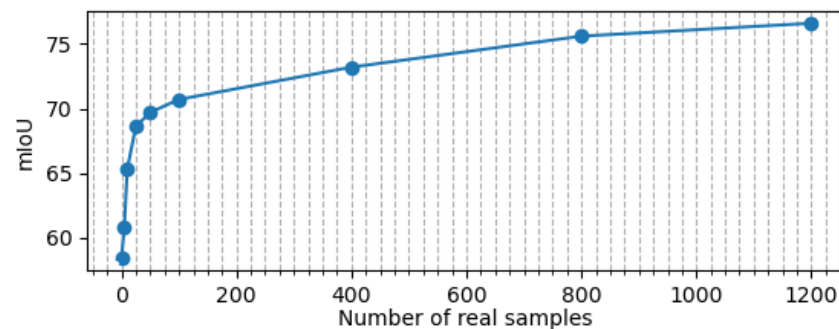


Figure 14. This figure illustrates the impact of the quantity of partially annotated real-world samples on the mean Intersection-over-Union (mIoU). The dots displayed in the graph correspond to different sample sizes, including [0, 5, 10, 25, 50, 100, 400, 800, 1200] samples. It is noteworthy that there is a substantial and rapid increase in the mIoU when relatively small quantities of real samples are introduced. For instance, the utilization of just 25 such samples results in a significant boost in mIoU, elevating it from 58.35% to 68.62%. However, as the number of real samples continues to increase, the rate of improvement gradually diminishes. When the dataset is enriched with 1200 real samples, an mIoU of 76.54% is achieved.

5. Limitations and Discussion

Though the presented methodology very accurately mimics ToF sensors, a small deviation from reality exists. Similar to all the cited studies, be they ray-casting- or rasterization-based, we assume co-location of the sensor emitter and receiver, whereas in reality a baseline between both exists (2 cm in our case). This, in essence, ever so slightly perturbs the uv-coordinates of the laser pulses depending on the measured depths. It is also the reason why we advocate pointing the laser scanner perpendicular towards a vertical flat surface to extract the sampling pattern. Newly occluded points are also possible. A potential solution projects the resulting point cloud to the image plane of the receiver. However, this incurs a cost of a decrease in computational efficiency by a factor two with arguably no notable improvement in terms of network performance considering the very minor perturbations.

Another drawback arises from the utilization of synthetic data. We acknowledge that the preference should generally lean towards employing perfectly annotated real-world data, as bridging the gap between virtual and real data can be challenging. However, this approach is not always feasible, particularly when dealing with complex tasks like segmentation. One of the primary advantages of the proposed techniques is their ability to

facilitate deep learning for ToF sensors when no real-world data are available. Nevertheless, as our experiments demonstrate, the inclusion of real-world data can enhance model accuracy. Therefore, a positive feedback loop can be established wherein initial predictions are recursively used as training data to further refine the model.

An additional strength of the proposed methods lies in their generality. They not only enable the creation of digital twins for virtually any ToF sensor but can also find applicability across various domains. While applications like autonomous driving may have relatively abundant data, other fields utilizing LiDAR, such as tree canopy estimation [73], crop classification [74], and wind farm efficiency prediction [75], may not possess readily available extensive datasets. The proposed methods could assist in simplifying the data collection and annotation processes in such contexts. Furthermore, even in domains with abundant data, our methods offer future-proofing benefits. Should the standard for traffic lights change, for example, new training data can be readily generated by updating the 3D model and running the simulator. This underscores one of the key advantages of employing synthetic data.

Lastly, while all the simulators discussed in this work generate the same point clouds under identical conditions, the reduction in data generation time should not be underestimated. This aspect holds significant importance, especially in data-driven applications like deep learning. The determination of class distributions in training data often necessitates multiple iterations to produce accurate models. Accelerating data generation by a factor of 100 can drastically reduce the resources required for obtaining such models, thereby saving time and money and reducing power consumption.

6. Conclusions

This paper presents a novel GPU-accelerated Time-of-Flight simulator. The proposed techniques are generic and can simulate any ToF sensor. Unlike any prior work, the proposed simulator allows the mimicking of the unique sampling patterns of each sensor. This enables the implementation of an exact digital twin, which is crucial for realistic simulations. Compared to the state-of-the-art, the proposed methodology decreases data generation times one hundredfold. It vastly outperforms GPU-accelerated ray-casting simulators as well. This is particularly important for data-driven applications, such as deep learning, which often rely on enormous amounts of data. Using the presented methodology, a digital twin of a custom in-house LiDAR is devised and used for generating perfectly annotated datasets. Two applications are considered. A denoising neural network is trained that leverages only synthetic data and which reduces the root-mean-square error of the depth measurements for our wall calibration test from 284.16 mm to 54.44 mm. A second experiment shows a 20% reduction in variance for materials with different reflectivity. In terms of qualitative results, point clouds captured during a real-world driving scenario exhibit much more refined edges after denoising, making it easier to distinguish vehicles and drivable areas.

We further assess the proposed methods by training a deep network for semantic segmentation. Leveraging only synthetic data yields decent performance for two of the three classes. Dataset enrichment with real data is proposed to void the gap between reality and simulation. To enable training with the differently annotated samples, a novel loss function is introduced. It allows learning classes for which no labels are provided in the real training set, hence significantly reducing annotation efforts. An ablation study reveals that solely 25 *partially* annotated real samples suffice to already significantly increase prediction accuracy for *all* classes. When increasing the number of real-world samples with annotated cars to 800, our network is able to achieve an overall accuracy, mean accuracy, and mean intersection over union of 89.36%, 86.51%, and 75.64%, respectively when averaged over the classes car, street, and flora.

Both applications demonstrate the validity of our simulation and prove that the generated synthetic data resemble reality even close enough for training neural networks, which are particularly sensitive to data mismatch. However, despite addressing denoising and

semantic segmentation, the proposed solutions are not limited to the discussed application domains. The simulator can be employed for research targeting LiDAR-based SLAM or vehicle navigation for example, whereas the proposed loss function can bridge the gap between synthetic and real data other than point clouds. Perhaps the proposed methods can find applications in thermal imaging, where not much data are available. Given the broad range of applications, we are convinced that the proposed techniques can help to alleviate the data acquisition problem, which is pertinent in many domains.

Author Contributions: Conceptualization, L.D., R.R. and A.M.; methodology, L.D., R.R. and Q.B.; software, L.D., R.R. and Q.B.; validation, L.D., R.R. and Q.B.; formal analysis, L.D., R.R., Q.B. and N.V.; investigation, L.D., R.R. and Q.B.; resources, A.M. and A.P.; writing—original draft preparation, L.D., R.R., Q.B., N.V., A.M. and A.P.; writing—review and editing, L.D., R.R., N.V., Q.B., A.M. and A.P.; visualization, L.D., R.R. and Q.B.; supervision, A.M. and A.P.; project administration, L.D., A.M. and A.P.; funding acquisition, A.M. and A.P. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Fonds Wetenschappelijk Onderzoek (FWO) (projects G094122N, FWOSB88 - PhD fellowship R. Royen).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results. All authors read and approved the final manuscript.

References

1. Krizhevsky, A.; Sutskever, I.; Hinton, G.E. ImageNet Classification with Deep Convolutional Neural Networks. In Proceedings of the Advances in Neural Information Processing Systems, Dayton, OH, USA, 15–19 July 2012; Pereira, F., Burges, C.J.C., Bottou, L., Weinberger, K.Q., Eds.; Curran Associates, Inc.: Sydney, Australia, 2012; Volume 25.
2. Liu, X.; Liu, T.; Zhou, J.; Liu, H. High-resolution facial expression image restoration via adaptive total variation regularization for classroom learning environment. *Infrared Phys. Technol.* **2023**, *128*, 104482. [[CrossRef](#)]
3. Liu, H.; Zhang, C.; Deng, Y.; Xie, B.; Liu, T.; Zhang, Z.; Li, Y.F. TransIFC: Invariant Cues-aware Feature Concentration Learning for Efficient Fine-grained Bird Image Classification. *IEEE Trans. Multimed.* **2023**, *1*, 8548. [[CrossRef](#)]
4. Ivanovs, M.; Ozols, K.; Dobrajs, A.; Kadikis, R. Improving Semantic Segmentation of Urban Scenes for Self-Driving Cars with Synthetic Images. *Sensors* **2022**, *22*, 2252. [[CrossRef](#)] [[PubMed](#)]
5. Wu, S.; Yan, Y.; Wang, W. CF-YOLOX: An Autonomous Driving Detection Model for Multi-Scale Object Detection. *Sensors* **2023**, *23*, 3794. [[CrossRef](#)]
6. Yuan, Z.; Wang, Z.; Li, X.; Li, L.; Zhang, L. Hierarchical Trajectory Planning for Narrow-Space Automated Parking with Deep Reinforcement Learning: A Federated Learning Scheme. *Sensors* **2023**, *23*, 4087. [[CrossRef](#)]
7. Shi, J.; Li, K.; Piao, C.; Gao, J.; Chen, L. Model-Based Predictive Control and Reinforcement Learning for Planning Vehicle-Parking Trajectories for Vertical Parking Spaces. *Sensors* **2023**, *23*, 7124. [[CrossRef](#)]
8. Gu, Z.; Liu, Z.; Wang, Q.; Mao, Q.; Shuai, Z.; Ma, Z. Reinforcement Learning-Based Approach for Minimizing Energy Loss of Driving Platoon Decisions. *Sensors* **2023**, *23*, 4176. [[CrossRef](#)]
9. Yang, L.; Babayi Semirami, M.; Xing, Y.; Lv, C.; Brighton, J.; Zhao, Y. The Identification of Non-Driving Activities with Associated Implication on the Take-Over Process. *Sensors* **2022**, *22*, 42. [[CrossRef](#)]
10. Boulch, A.; Sautier, C.; Michele, B.; Puy, G.; Marlet, R. ALSO: Automotive Lidar Self-Supervision by Occupancy Estimation. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Vancouver, BC, Canada, 17–24 June 2023; pp. 13455–13465.
11. Ryu, K.; Hwang, S.; Park, J. Instant Domain Augmentation for LiDAR Semantic Segmentation. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Vancouver, BC, Canada, 17–24 June 2023; pp. 9350–9360.
12. Wang, P.; Huang, X.; Cheng, X.; Zhou, D.; Geng, Q.; Yang, R. The ApolloScape Open Dataset for Autonomous Driving and Its Application. *IEEE Trans. Pattern Anal. Mach. Intell.* **2019**, *29*, 6463. [[CrossRef](#)]
13. Huang, X.; Cheng, X.; Geng, Q.; Cao, B.; Zhou, D.; Wang, P.; Lin, Y.; Yang, R. The ApolloScape Dataset for Autonomous Driving. In Proceedings of the 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), Salt Lake City, UT, USA, 18–22 June 2018; pp. 1067–10676. [[CrossRef](#)]

14. Cordts, M.; Omran, M.; Ramos, S.; Rehfeld, T.; Enzweiler, M.; Benenson, R.; Franke, U.; Roth, S.; Schiele, B. The cityscapes dataset for semantic urban scene understanding. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 27–30 June 2016; pp. 3213–3223.
15. Yu, F.; Chen, H.; Wang, X.; Xian, W.; Chen, Y.; Liu, F.; Madhavan, V.; Darrell, T. BDD100K: A Diverse Driving Dataset for Heterogeneous Multitask Learning. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Seattle, WA, USA, 13–16 June 2020.
16. Geiger, A.; Lenz, P.; Urtasun, R. Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite. In Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR), Providence, RI, Canada, 16–21 June 2012.
17. Caesar, H.; Bankiti, V.; Lang, A.H.; Vora, S.; Liong, V.E.; Xu, Q.; Krishnan, A.; Pan, Y.; Baldan, G.; Beijbom, O. nuScenes: A multimodal dataset for autonomous driving. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Seattle, WA, USA, 13–16 June 2020; pp. 11621–11631.
18. Reitmann, S.; Neumann, L.; Jung, B. BLAINDER—A Blender AI Add-On for Generation of Semantically Labeled Depth-Sensing Data. *Sensors* **2021**, *21*, 2144. [[CrossRef](#)]
19. Dosovitskiy, A.; Ros, G.; Codevilla, F.; Lopez, A.; Koltun, V. CARLA: An Open Urban Driving Simulator. In Proceedings of the 1st Annual Conference on Robot Learning, Los Angeles, CA, USA, 13–15 November 2017; pp. 1–16.
20. Razani, R.; Cheng, R.; Taghavi, E.; Bingbing, L. Lite-HDseg: LiDAR Semantic Segmentation Using Lite Harmonic Dense Convolutions. In Proceedings of the 2021 IEEE International Conference on Robotics and Automation (ICRA), Xi'an China, 12–15 June 2021; pp. 9550–9556. [[CrossRef](#)]
21. Cheng, R.; Razani, R.; Ren, Y.; Bingbing, L. S3Net: 3D LiDAR Sparse Semantic Segmentation Network. In Proceedings of the 2021 IEEE International Conference on Robotics and Automation (ICRA), Xi'an, China, 12–15 June 2021; pp. 14040–14046. [[CrossRef](#)]
22. Zhou, D.; Fang, J.; Song, X.; Liu, L.; Yin, J.; Dai, Y.; Li, H.; Yang, R. Joint 3D Instance Segmentation and Object Detection for Autonomous Driving. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Seattle, WA, USA, 13–19 June 2020.
23. Yang, B.; Luo, W.; Urtasun, R. PIXOR: Real-time 3D Object Detection from Point Clouds. In Proceedings of the 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–23 June 2018; pp. 7652–7660.
24. Zhou, Z.; Zhang, Y.; Foroosh, H. Panoptic-PolarNet: Proposal-Free LiDAR Point Cloud Panoptic Segmentation. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Vancouver, BC, Canada, 12–15 June 2021; pp. 13194–13203.
25. Hu, J.S.K.; Kuai, T.; Waslander, S.L. Point Density-Aware Voxels for LiDAR 3D Object Detection. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), New Orleans, LA, USA, 18–24 June 2022; pp. 8469–8478.
26. Lai, X.; Chen, Y.; Lu, F.; Liu, J.; Jia, J. Spherical Transformer for LiDAR-Based 3D Recognition. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Vancouver, BC, Canada, 12–15 June 2023; pp. 17545–17555.
27. Li, J.; Luo, C.; Yang, X. PillarNeXt: Rethinking Network Designs for 3D Object Detection in LiDAR Point Clouds. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Vancouver, BC, Canada, 12–15 June 2023; pp. 17567–17576.
28. Erabati, G.K.; Araujo, H. Li3DeTr: A LiDAR Based 3D Detection Transformer. In Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV), London, UK, 5–7 January 2023; pp. 4250–4259.
29. Whitted, T. An Improved Illumination Model for Shaded Display. *Commun. ACM* **1980**, *23*, 343–349. [[CrossRef](#)]
30. Morsdorf, F.; Frey, O.; Koetz, B.; Meier, E. Ray tracing for modeling of small footprint airborne laser scanning returns. *Int. Arch. Photogramm. Remote Sens. Spat. Inf. Sci.* **2007**, *36*, 249–299. [[CrossRef](#)]
31. Kukko, A.; Hyypä, J. Small-footprint Laser Scanning Simulator for System Validation, Error Assessment, and Algorithm Development. *Photogramm. Eng. Remote Sens.* **2009**, *75*, 1177. [[CrossRef](#)]
32. Kim, S.; Min, S.; Kim, G.; Lee, I.; Jun, C. Data simulation of an airborne lidar system. *Proc. SPIE* **2009**, *12*, 8545. [[CrossRef](#)]
33. Wang, Y.; Xie, D.; Yan, G.; Zhang, W.; Mu, X. Analysis on the inversion accuracy of LAI based on simulated point clouds of terrestrial LiDAR of tree by ray tracing algorithm. In Proceedings of the 2013 IEEE International Geoscience and Remote Sensing Symposium, IGARSS, Melbourne, Australia, 21–26 July 2013; pp. 532–535. [[CrossRef](#)]
34. Gastellu-Etchegorry, J.P.; Yin, T.; Lauret, N.; Cajgfinger, T.; Gregoire, T.; Grau, E.; Féret, J.B.; Lopes, M.; Guilleux, J.; Dedieu, G.; et al. Discrete Anisotropic Radiative Transfer (DART 5) for Modeling Airborne and Satellite Spectroradiometer and LIDAR Acquisitions of Natural and Urban Landscapes. *Remote Sens.* **2015**, *7*, 1667–1701. [[CrossRef](#)]
35. Yun, T.; Cao, L.; An, F.; Chen, B.; Xue, L.; Li, W.; Pincebourde, S.; Smith, M.J.; Eichhorn, M.P. Simulation of multi-platform LiDAR for assessing total leaf area in tree crowns. *Agric. For. Meteorol.* **2019**, 276–277, 107610. [[CrossRef](#)]
36. Gusmão, G.; Barbosa, C.; Raposo, A. Development and Validation of LiDAR Sensor Simulators Based on Parallel Raycasting. *Sensors* **2020**, *20*, 7186. [[CrossRef](#)]
37. Hanke, T.; Schaermann, A.; Geiger, M.; Weiler, K.; Hirsenkorn, N.; Rauch, A.; Schneider, S.A.; Biebl, E. Generation and validation of virtual point cloud data for automated driving systems. In Proceedings of the 2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC), Yokohama, Japan, 16–19 October 2017; pp. 1–6. [[CrossRef](#)]
38. Yue, X.; Wu, B.; Seshia, S.A.; Keutzer, K.; Sangiovanni-Vincentelli, A.L. A LiDAR Point Cloud Generator: From a Virtual World to Autonomous Driving. In Proceedings of the 2018 ACM on International Conference on Multimedia Retrieval, ICMR '18, New York, NY, USA, 17–21 April 2018; pp. 458–464. [[CrossRef](#)]

39. Unreal Technologies. Unreal Engine. Available online: <https://www.unrealengine.com> (accessed on 10 August 2023).
40. Wang, F.; Zhuang, Y.; Gu, H.; Hu, H. Automatic Generation of Synthetic LiDAR Point Clouds for 3-D Data Analysis. *IEEE Trans. Instrum. Meas.* **2019**, *68*, 2671–2673. [[CrossRef](#)]
41. Manivasagam, S.; Wang, S.; Wong, K.; Zeng, W.; Sazanovich, M.; Tan, S.; Yang, B.; Ma, W.; Urtasun, R. LiDARsim: Realistic LiDAR Simulation by Leveraging the Real World. In Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, Seattle, WA, USA, 14–19 June 2020; pp. 11164–11173. [[CrossRef](#)]
42. Bechtold, S.; Höfle, B. 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.* **2016**, *III-3*, 161–168. [[CrossRef](#)]
43. Winiwarter, L.; Pena, A.M.E.; Weiser, H.; Anders, K.; Sanchez, J.M.; Searle, M.; Höfle, B. Virtual laser scanning with HELIOS++: A novel take on ray tracing-based simulation of topographic 3D laser scanning. *arXiv* **2021**, arXiv:2101.09154.
44. Community, B.O. *Blender—A 3D Modelling and Rendering Package*; Blender Foundation, Stichting Blender Foundation: Amsterdam, The Netherlands, 2018.
45. Li, W.; Pan, C.W.; Zhang, R.; Ren, J.P.; Ma, Y.X.; Fang, J.; Yan, F.L.; Geng, Q.C.; Huang, X.Y.; Gong, H.J.; et al. AADS: Augmented autonomous driving simulation using data-driven algorithms. *Sci. Robot.* **2019**, *4*, eaaw0863. [[CrossRef](#)] [[PubMed](#)]
46. Fang, J.; Zhou, D.; Yan, F.; Tongtong, Z.; Zhang, F.; Ma, Y.; Wang, L.; Yang, R. Augmented LiDAR Simulator for Autonomous Driving. *IEEE Robot. Autom. Lett.* **2020**, *29PP*, 9927. [[CrossRef](#)]
47. Hossny, M.; Saleh, K.; Attia, M.H.; Abobakr, A.; Iskander, J. Fast Synthetic LiDAR Rendering via Spherical UV Unwrapping of Equirectangular Z-Buffer Images. *arXiv* **2020**, arXiv:2006.04345
48. Fang, J.; Zuo, X.; Zhou, D.; Jin, S.; Wang, S.; Zhang, L. LiDAR-Aug: A General Rendering-Based Augmentation Framework for 3D Object Detection. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Nashville, TN, USA, 20–25 June 2021; pp. 4710–4720.
49. Yang, Z.; Chen, Y.; Wang, J.; Manivasagam, S.; Ma, W.C.; Yang, A.J.; Urtasun, R. UniSim: A Neural Closed-Loop Sensor Simulator. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Vancouver, BC, Canada, 17–24 June 2023; pp. 1389–1399.
50. Yang, Z.; Manivasagam, S.; Chen, Y.; Wang, J.; Hu, R.; Urtasun, R. Reconstructing Objects in-the-wild for Realistic Sensor Simulation. In Proceedings of the 2023 IEEE International Conference on Robotics and Automation (ICRA), London, UK, 29 May–2 June 2023; pp. 11661–11668. [[CrossRef](#)]
51. Guillard, B.; Vemprala, S.; Gupta, J.K.; Miksik, O.; Vineet, V.; Fua, P.; Kapoor, A. Learning to Simulate Realistic LiDARs. In Proceedings of the 2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Kyoto, Japan, 23–27 October 2022; pp. 8173–8180. [[CrossRef](#)]
52. Li, Y.; Ma, L.; Zhong, Z.; Liu, F.; Chapman, M.; Cao, D.; Li, J. Deep Learning for LiDAR Point Clouds in Autonomous Driving: A Review. *IEEE Trans. Neural Netw. Learn. Syst.* **2020**, *30*, 5992. [[CrossRef](#)]
53. Feng, D.; Haase-Schutz, C.; Rosenbaum, L.; Hertlein, H.; Gläser, C.; Timm, F.; Wiesbeck, W.; Dietmayer, K. Deep Multi-Modal Object Detection and Semantic Segmentation for Autonomous Driving: Datasets, Methods, and Challenges. *IEEE Trans. Intell. Transp. Syst.* **2020**, *29*, 2974. [[CrossRef](#)]
54. Ros, G.; Sellart, L.; Materzynska, J.; Vazquez, D.; Lopez, A.M. The SYNTHIA Dataset: A Large Collection of Synthetic Images for Semantic Segmentation of Urban Scenes. In Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 27–30 June 2016; pp. 3234–3243. [[CrossRef](#)]
55. Behley, J.; Garbade, M.; Milioto, A.; Quenzel, J.; Behnke, S.; Stachniss, C.; Gall, J. SemanticKITTI: A Dataset for Semantic Scene Understanding of LiDAR Sequences. In Proceedings of the IEEE/CVF International Conf. on Computer Vision (ICCV), Seoul, Republic of Korea, 27 October–2 November 2019.
56. Shah, S.; Dey, D.; Lovett, C.; Kapoor, A. AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles. In Proceedings of the Field and Service Robotics, Cham, Switzerland, 2–9 August 2018; Hutter, M., Siegwart, R., Eds.; pp. 621–635.
57. Mueller, M.; Casser, V.; Lahoud, J.; Smith, N.; Ghanem, B. Sim4CV: A Photo-Realistic Simulator for Computer Vision Applications. *Int. J. Comput. Vis.* **2018**, *126*, 902–919. [[CrossRef](#)]
58. Richter, S.R.; Vineet, V.; Roth, S.; Koltun, V. Playing for Data: Ground Truth from Computer Games. In Proceedings of the Computer Vision—ECCV, Cham, Switzerland, 14–19 June 2016; Leibe, B., Matas, J., Sebe, N., Welling, M., Eds.; pp. 102–118.
59. NVIDIA. Self-Driving Cars Technology & Solutions from NVIDIA Automotive. Available online: <https://www.nvidia.com/en-us/self-driving-cars/> (accessed on 10 August 2023).
60. Johnson-Roberson, M.; Barto, C.; Mehta, R.; Sridhar, S.N.; Rosaen, K.; Vasudevan, R. Driving in the Matrix: Can virtual worlds replace human-generated annotations for real world tasks? In Proceedings of the 2017 IEEE International Conference on Robotics and Automation (ICRA), Singapore, 29 May–3 June 2017; pp. 746–753.
61. Wu, B.; Wan, A.; Yue, X.; Keutzer, K. SqueezeSeg: Convolutional Neural Nets with Recurrent CRF for Real-Time Road-Object Segmentation from 3D LiDAR Point Cloud. In Proceedings of the 2018 IEEE International Conference on Robotics and Automation (ICRA), Brisbane, Australia, 21–25 May 2018; pp. 1887–1893. [[CrossRef](#)]
62. Wu, B.; Zhou, X.; Zhao, S.; Yue, X.; Keutzer, K. SqueezeSegV2: Improved Model Structure and Unsupervised Domain Adaptation for Road-Object Segmentation from a LiDAR Point Cloud. In Proceedings of the ICRA, Montreal, QC, Canada, 20–24 May 2019.

63. Zhao, S.; Wang, Y.; Li, B.; Wu, B.; Gao, Y.; Xu, P.; Darrell, T.; Keutzer, K. ePointDA: An End-to-End Simulation-to-Real Domain Adaptation Framework for LiDAR Point Cloud Segmentation. In Proceedings of the AAAI Conference on Artificial Intelligence (AAAI), Washington, DC, USA, 7–14 February 2021.
64. Alhaja, H.; Mustikovela, S.; Mescheder, L.; Geiger, A.; Rother, C. Augmented Reality Meets Computer Vision: Efficient Data Generation for Urban Driving Scenes. *Int. J. Comput. Vis.* **2018**, *126*, 11263. [[CrossRef](#)]
65. Schmitt, A.; Leister, W.; Müller, H. *Ray Tracing Algorithms—Theory and Practice*; Springer: Berlin/Heidelberg, Germany, 1988; pp. 997–1030. [[CrossRef](#)]
66. Mei, L.; Zhang, L.; Kong, Z.; Li, H. Noise modeling, evaluation and reduction for the atmospheric lidar technique employing an image sensor. *Opt. Commun.* **2018**, *426*, 463–470. [[CrossRef](#)]
67. Falie, D.; Buzuloiu, V. Noise Characteristics of 3D Time-of-Flight Cameras. In Proceedings of the 2007 International Symposium on Signals, Circuits and Systems, Iasi, Romania, 13–14 July 2007; Volume 1; pp. 1–4. [[CrossRef](#)]
68. Khronos Group. OpenGL 4.5 Reference Pages. Available online: <https://www.khronos.org/registry/OpenGL-Refpages/gl4/> (accessed on 10 August 2023).
69. Nvidia OptiX™. Available online: <https://developer.nvidia.com/optix> (accessed on 10 August 2023).
70. Qi, C.R.; Su, H.; Mo, K.; Guibas, L.J. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, 21–26 July 2017.
71. Unity Technologies. Unity Real-Time Development Platform 3D, 2D VR & AR Engine. Available online: <https://unity.com/> (accessed on 10 August 2023).
72. Chollet, F. Xception: Deep learning with depthwise separable convolutions. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, HI, USA, 21–26 July 2017; pp. 1251–1258.
73. Patil, S.S.; Patil, Y.M.; Patil, S.B. Detection and Estimation of Tree Canopy using Deep Learning and Sensor Fusion. In Proceedings of the 2023 International Conference for Advancement in Technology (ICONAT), Goa, India, 24–26 January 2023; pp. 1–5. [[CrossRef](#)]
74. Reji, J.; Nidamanuri, R.R. Deep Learning based Fusion of LiDAR Point Cloud and Multispectral Imagery for Crop Classification Sensitive to Nitrogen Level. In Proceedings of the 2023 International Conference on Machine Intelligence for GeoAnalytics and Remote Sensing (MIGARS), Hyderabad, India, 27–29 January 2023; Volume 1, pp. 1–4. [[CrossRef](#)]
75. Zhang, J.; Zhao, X. Spatiotemporal wind field prediction based on physics-informed deep learning and LIDAR measurements. *Appl. Energy* **2021**, *288*, 116641. [[CrossRef](#)]

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.