CEIG – Spanish Computer Graphics Conference (2021)
A. Chica and L. Ortega (Editors)

A GPU-accelerated LiDAR sensor for generating labelled datasets

Alfonso López[1], Carlos J. Ogayar[1] and Francisco R. Feito[1]

[1]Department of Computer Science, University of Jaén

Abstract

This paper presents a GPU-based LiDAR simulator to generate large datasets of ground-truth point clouds. LiDAR technology has significantly increased its impact on academic and industrial environments. However, some of its applications require a large amount of annotated LiDAR data. Furthermore, there exist many types of LiDAR sensors. Therefore, developing a parametric LiDAR model allows simulating a wide range of LiDAR scanning technologies and obtaining a significant number of points clouds at no cost. Beyond their intensity data, these synthetic point clouds can be classified with any level of detail.

CCS Concepts

• ***Computing methodologies*** → ***Simulation environments;*** *Massively parallel algorithms; Rendering;*

1. Introduction

LiDAR (Light Detection and Ranging) provides valuable information about surfaces, objects or phenomenon without physical contact. This active remote sensing technique provides precise 3D coordinates as well as reflectance data for a laser wavelength. Thus, this technology presents a wide range of applications, from quality control or structural damage to autonomous navigation. Furthermore, it has greatly evolved through improvements based on data acquisition speed, maximum range, precision or accuracy. There also exists a wide variety of LiDAR sensors whether we consider their capabilities or the platform from where they are operated.

The use of LiDAR data has significantly increased for multiple processes and applications, e.g. autonomous driving [LMZ*20] or Deep Learning methods [LSL*19], where the main objective is to identify surveyed surfaces. Nevertheless, acquiring ground-truth data is a challenging task due to time and hardware requirements. Therefore, simulating a LiDAR allows obtaining as many datasets as needed at no cost. Also note that resulting points can be enriched with further information beyond their 3D position and intensity data, such as their class. On the other hand, a naive simulator can produce more realistic results by considering both systematic and random errors reported for LiDAR sensors.

Our work presents a generic simulator accelerated with Graphics Processing Unit (GPU) hardware to reduce the response time, as it is capable of casting millions of beams. To achieve this, we use the OpenGL framework for rendering the environment and the LiDAR results, as well as for developing massively parallel methods through general-purpose compute shaders. In addition, we also aim to generate labelled datasets that could be used as input data for Deep Learning (DL) based methods, mainly for segmentation and recognition of structures.

2. Related works

Over the last decades, several works have proposed LiDAR simulators, although their applications and results differ significantly from each other. Some studies are focused on calibrating a LiDAR sensor [LCL20], optimizing the scanning process [WBU20] or supporting autonomous driving [LMZ*20]. However, most of the works benefit from the use of synthetic environments. Beyond research purposes, there are other systems aimed at the configuration of complex simulated sensors in order to fine-tune all their parameters. Physically-based propagation of laser beams has also been widely studied [Zoh20].

Regarding simulation performance, few works have previously assessed the use of GPU acceleration [PLK08]. Furthermore, we can find commercial software applied to autonomous driving, such as LGSVL (LG) or Simcenter (Siemens Software), with a limited capacity of ray emission. Another relevant part of a LiDAR simulator is to include artificial errors during the process. Both systematic and random errors depend on several aspects [LM07], including missing points, non-uniform density, cluttering, occlusion or distortion of point properties. Thus, a precise simulation of a LiDAR sensor is a challenging task, both algorithmically and in terms of behaviour. The goal of this work is to develop a GPU-based simulator that outputs an annotated point cloud with intensity data as fast as possible. As our work is aimed at the training of neural networks, we propose a simulation that obtains discrete laser returns instead of full-waveform profiles. Furthermore, providing results that are not completely physically accurate tends not to be a significant problem, as only the position and colour of points are used. Not to mention that point clouds are typically oversegmentated into small regions due to their size and noise.

DOI: 10.2312/ceig.20211360

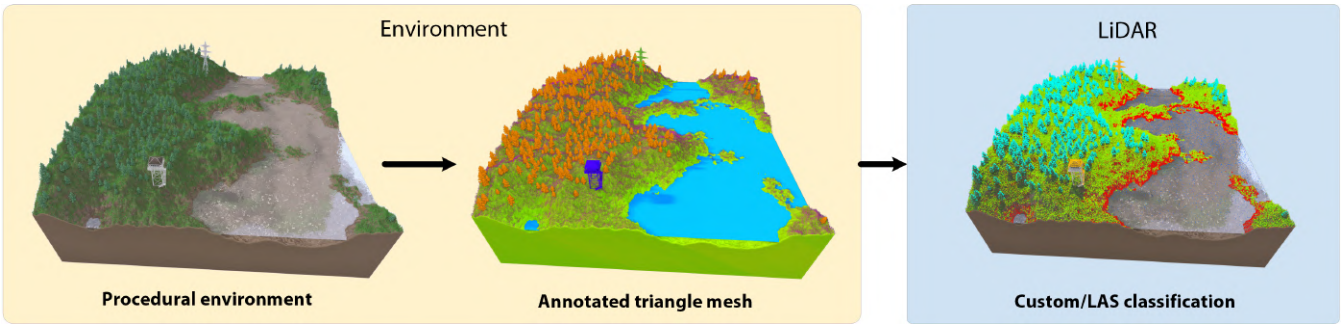

Figure 1: *Workflow of our LiDAR application. A procedural or static environment is first loaded and classified using tags defined before execution. Then, a non-bathymetric LiDAR simulation is launched over the previous scene.*

3. Synthetic environments

The proposed LiDAR sensor is applied to synthetic environments similar to world scenes, as the resulting points aim to serve as training datasets for DL algorithms. Accordingly, we propose the use of environments composed of CAD models. However, static scenes harden the efficient generation of several point clouds, since models must be labelled previously in order to obtain classified points. To solve this problem, we introduce procedural modelling of environments. More specifically, we include forest environments for simulating the airborne LiDAR. As opposed to large-scale systems, terrestrial LiDAR is mainly tested with indoor scenes.

The realism of our procedural scene is enhanced by applying a hydraulic erosion algorithm to a steep terrain representation. Then, vegetation is included as low-polygonal trees and grass models that are randomly spread across the terrain, whereas water is modelled as a planar surface. To instantiate vegetation, we calculate a weighted sum of terrain slope and distance to water and use a threshold factor to determine whether an instance is added to the environment. Some additional features are introduced considering the LAS 1.4 (LASer) standard file format, which has been extensively used to represent LiDAR point clouds. This file format defines 18 encoded categories. Thus, two additional classes, such as transmission tower or building, are also instantiated to extend the number of active labels during the airborne simulation.

Before LiDAR simulation, synthetic environments are labelled by associating classes with models within the scene. Therefore, this is a time-consuming task that is much more efficient for procedural environments. Scenes can be annotated by either using the LAS standard or a custom classification with any level of detail, thus increasing significantly the number of objects that can be distinguished. Figure 1 shows a forest labelled with custom semantic tags, whereas the resulting point cloud is annotated with LAS labels.

4. Data structure for spatial indexing

The described environments are represented by triangle meshes ranging from a few thousand primitives to several million triangles. Therefore, solving millions of collisions in a reduced response time is a cumbersome task. However, the main challenge is to index a scene in an efficient spatial data structure to speed up the search process. A conventional data structure for ray-tracing applications is the Boundary Volume Hierarchy (BVH), a binary tree of axis-aligned bounding boxes (AABB), where its leaves represent the scene primitives. Therefore, nodes are merged up until the root is reached (Figure 2). Given the complexity of building a BVH, we implement the massively parallel methodology described in [MB18]. For traversing the BVH, we start by the root node and descend into the tree, discarding large parts of the scene while searching for the nearest collision.

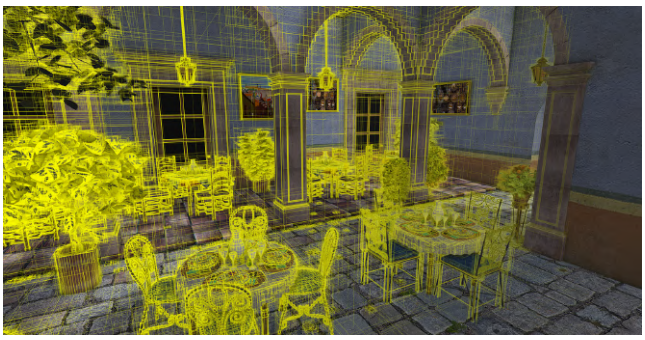

Figure 2: *Boundary Volume Hierarchy for lower-polygon San Miguel scene (5,6 million triangles and 16,8 million vertices).*

5. LiDAR simulation

Once we have built several environments, we proceed to describe a LiDAR simulation focused on terrestrial and airborne laser scanning, i.e. TLS and ALS. The behaviour of both LiDAR sensors is simulated with different parameters based on their positioning and movement. However, here we aim to describe a generic simulation by avoiding in-depth details. Despite the different behaviour of each simulation, they can be implemented using a generic workflow.

5.1. Laser emission

A LiDAR simulation starts by propagating LiDAR beams/pulses in the 3D world. Although emitted pulses are physically represented

as cylinders or cones for collimated or diverging beams respectively, the discretization of a pulse into several rays allows obtaining faster results. Thus, the simulation is more precise and time-consuming if the number of discrete rays increases. For a TLS, we use a reference position from where rays are originated, acting as both emitter and receiver of our LiDAR sensor. The number of rays to be instantiated depends on the horizontal and vertical resolution, i.e. the number of subdivisions of a spherical projection. However, the whole space is not frequently covered. Accordingly, the surveyed space is bounded by the horizontal and vertical aperture angle. We have assumed that the number of channels was one ($n_c \leftarrow 1$), whereas high-resolution requirements are better handled with multiple channels, represented by a set of interleaved positions. Thus, the vertical aperture angle is equally split into n_c, although a small overlap can be included through simulation parameters. To avoid some unwanted visual effects due to the overlapping of a large number of points with repeating patterns, the rays are jittered with a magnitude of δ through a random uniform function. However, note that we can exchange the distribution of randomized buffers used along this process.

For airborne LiDAR, the scanning procedure changes mainly due to the mobile platform and the projection domain. We implement several scanning patterns, from parallel to elliptical or zigzag surveys, whereas custom paths obtained from the user input are also enabled in our simulation. Regarding the projection domain, a planar projection is enough for a single viewpoint instead of a spherical projection. Accordingly, the field of view is limited, and the scanning session is defined by its scanning frequency, given by the number of scans and pulses per second as well as the movement speed of the mobile platform.

Despite the simplicity of the methodology, generating the buffer of rays in the CPU is a time-consuming task. Note that the number of rays significantly increases when a pulse is discretized as multiple rays. Therefore, we generate the emitted rays in GPU so that each thread handles a different pulse. Randomness is introduced by using a large buffer of noise for a random distribution, which represents a circular buffer for the significant number of threads. On the other hand, we cope with the limited memory in GPU by splitting the generation methodology into several batches, and the same applies to the behaviour described in the next section.

5.2. LiDAR behaviour

This section aims to describe a generic interaction with object surfaces, both for TLS and ALS sensors. For the sake of simplicity, the methodology here proposed is represented by different stages (i.e. shaders) to induce synchronous memory accesses. Also, note that several batches are needed to dispatch the same routine whether the number of rays exceeds a threshold. The following list enumerates the steps used for each iteration:

1. **Find collisions**. Given the aforementioned BVH traversal, each ray is handled by a different thread. Thus, the goal is to find the collision with the scene.
2. **Reduce rays**. Although pulses are discretized, they generate a single return at most per iteration. Hence, the nearest collision is identified, whereas adjacent intersections are checked to determine if they collided with the same chunk of surface. Whether they do not, rays continue their path during multiple iterations as long as they do not overcome LiDAR capabilities. Multiple returns are especially relevant for our procedural environment, as they allow penetrating the forest canopy to reach the ground (Figure 3). Furthermore, multiple errors are simulated in this stage by distorting the position. First, mirror-like surfaces simulate the 'time-walk' effect, which makes objects appear more distant than they are. For the procedural environment, terrain-induced translations may also appear due to its slope.
3. **Intensity evaluation**. Beyond their 3D position, collisions are further completed with their intensity data using the LiDAR equation. Note that surface materials are a key factor in this equation. For that reason, objects are associated with a material and its corresponding BRDF, thus defining its backscattering behaviour as well as its diffuse and specular colours.
4. **Generation of outlier points**. Whether we consider the point cloud is not virtually noise-free, it is relevant to substitute some intersections with noise whose source is the scanning technique or environmental conditions. With the aforementioned buffer of random values, a threshold and the set of colliding rays in the last iteration, outliers are defined using the ray equation. Therefore, outliers are visible along the pulse path.
5. **Update of returns**. Once the pulses have stopped their propagation, collisions are updated with the total number of returns generated by a pulse. Thus, the canopy can be filtered out by retrieving points that are the last collision of each pulse. However, that does not guarantee retrieving only ground points due to dense forest canopies.

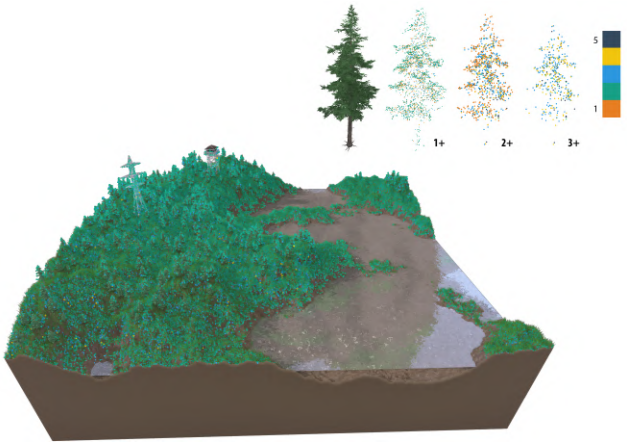

Figure 3: *Returns generated after the first collision over a procedural scene. The same configuration is applied to a single tree (upper right image) to show how pulses advance through the canopy.*

6. Evaluation

We have evaluated our solution with two scenes of different complexity, ranging from 330 thousand points (Conference scene) to nearly 10 million points (procedural forest). Both environments are tested against several LiDAR configurations, represented by their number of pulses (n_p) and discrete rays (n_r). Results are reported both for CPU and GPU implementations, as the main objective of

Table 1: *Response time of a TLS scanning session for multiple spatial resolutions. The massively parallel method (GPU) is compared to a sequential implementation (referred to as CPU method).*

		Stage	
n_p	**Method**	Ray Instancing	LiDAR Behaviour
1M	**GPU**	**0,068s**	**0,177s**
	CPU	3,345s	8,663s
5M	**GPU**	**0,208s**	**0,675s**
	CPU	15,396s	39,152s
10M	**GPU**	**0,412s**	**1,39s**
	CPU	32,349s	83,223s
20M	**GPU**	**0,835s**	**3,034s**
	CPU	64,607s	165,775s

Table 2: *Response time of an airborne LiDAR with parallel scanning pattern using several configurations with a different number of pulses and rays within a pulse.*

			Stage	
n_p	Method	n_r	Ray Instancing	LiDAR Behaviour
1M	**GPU**	5	**0,355s**	**1,155s**
	CPU	5	1,533s	160,530s
	GPU	15	**0,984s**	**2,226s**
	CPU	15	4,258s	354,640s
5M	**GPU**	5	**1,724s**	**4,44s**
	CPU	5	10,739s	797,615s
	GPU	15	**4,159s**	**9,711s**
	CPU	15	27,004s	1.783,889s

this work is to show the GPU speedup. Furthermore, the response time is split into **Ray generation** and **LiDAR behaviour**. Measurements were performed on a PC with Intel Core i9-9900 3.1 GHz, 48 GB RAM, RTX 2080 Ti GPU with 11 GB RAM. The proposed methodology is implemented in C++ along with OpenGL (Open Graphics Library) both for rendering and developing massively parallel algorithms through general-purpose compute shaders.

Table 1 shows the reported response time of a terrestrial LiDAR for a scene of 330k triangles, using only one ray and covering 360°x150°. The resolution varies depending on the aimed number of pulses. On the other hand, Table 2 shows the evaluation of an ALS sensor for an environment of 10M triangles. The number of ALS beams increases significantly, as each pulse is composed of 5 or 15 rays, and so does the algorithmic complexity since we simulate up to 5 returns to reach the forest ground.

From the results, we can observe that our GPU methodology significantly improves the sequential implementation. Note that the response time in the CPU is considerably higher for **LiDAR behaviour**, since solving the BVH-ray collision is a well-known time-bottleneck. To avoid this, our GPU method solves the collision of each ray in a different thread. Hence, the response time increases linearly as a function of the number of rays. Accordingly, the delay induced by threads managing multiple rays (related to pulse concept) is irrelevant in the reported response time.

7. Conclusions and future work

We have explored a generic LiDAR simulator that is massively accelerated through GPU hardware, thus allowing us to obtain labelled datasets in a reduced response time. The benefits of using procedural scenes for generating annotated datasets were proved throughout this work. Furthermore, we also developed a sequential method to evaluate the performance enhancement of the parallel methodology. Accordingly, scanning sessions solved in several minutes by the sequential method were simulated in a few seconds by our GPU methodology. Nevertheless, in future work, we would like to further optimize the procedure and reduce the use of memory to simulate more pulses per batch.

Acknowledgements

This work has been partially supported by the Spanish Ministry of Science, Innovation and Universities and the European Union (via ERDF funds), through the research projects TIN2017-84968-R and RTI2018-099638-B-I00.

References

[LCL20] LEE G., CHEON J., LEE I.: Validation of LIDAR calibration using a LIDAR simulator. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences XLIII-B1-2020* (Aug. 2020), 39–44. doi:10.5194/isprs-archives-xliii-b1-2020-39-2020. 1

[LM07] LOHANI B., MISHRA R.: Generating lidar data in laboratory: Lidar simulator. *Int. Arch. Photogramm. Remote Sens. 52* (01 2007). 1

[LMZ*20] LI Y., MA L., ZHONG Z., LIU F., CHAPMAN M. A., CAO D., LI J.: Deep Learning for LiDAR Point Clouds in Autonomous Driving: A Review. *IEEE Transactions on Neural Networks and Learning Systems* (2020), 1–21. doi:10.1109/TNNLS.2020.3015992. 1

[LSL*19] LIU W., SUN J., LI W., HU T., WANG P.: Deep learning on point clouds and its application: A survey. *Sensors 19*, 19 (Sept. 2019), 4188. doi:10.3390/s19194188. 1

[MB18] MEISTER D., BITTNER J.: Parallel locally-ordered clustering for bounding volume hierarchy construction. *IEEE Transactions on Visualization and Computer Graphics 24*, 3 (Mar. 2018), 1345–1353. doi:10.1109/tvcg.2017.2669983. 2

[PLK08] PEINECKE N., LUEKEN T., KORN B. R.: Lidar simulation using graphics hardware acceleration. In *2008 IEEE/AIAA 27th Digital Avionics Systems Conference* (Oct. 2008), IEEE. doi:10.1109/dasc.2008.4702838. 1

[WBU20] WESTLING F., BRYSON M., UNDERWOOD J.: SimTreeLS: Simulating aerial and terrestrial laser scans of trees. *arXiv:2011.11954 [cs, eess]* (Nov. 2020). arXiv: 2011.11954. 1

[Zoh20] ZOHDI T.: Rapid simulation-based uncertainty quantification of flash-type time-of-flight and lidar-based body-scanning processes. *Computer Methods in Applied Mechanics and Engineering 359* (Feb. 2020), 112386. doi:10.1016/j.cma.2019.03.056. 1