A LiDAR System Simulator Using Parallel Raytracing and Validated by Comparison with a Real Sensor

G F Gusmão¹ ², C R H Barbosa¹*, A B Raposo² and R C de Oliveira²

¹Postgraduate Program in Metrology, PUC-Rio, R. Marquês de São Vicente, 225, Gávea, Brazil
²Tecgraf/PUC-Rio, R. Marquês de São Vicente, 225, Gávea, Brazil

*Email: hall@puc-rio

Abstract. The advances in 3D imaging and Computer Vision are allowing for a massive acquisition of data, especially in the form of point clouds. However, scanning with these sensors can be costly and time consuming, reducing the efficiency of certain procedures, like deep learning dataset generation and forest canopy calculation. One trending solution is the creation of 3D imaging sensor simulators. This paper presents a simulator for light detection and ranging (LiDAR) systems with a parallel raytracing approach and a flexible scene creator. Finally, the simulator sensor is validated with a real LiDAR data.

Keywords: Sensor simulator; LiDAR; synthetic point cloud; remote sensing.

1. Introduction
There is a rising interest on the use of 3D imaging sensors by both academy and industry, especially now that computing power allows processing of massive datasets, even in mobile form factors. These type of sensor systems permit the gathering of geospatial information from a desired target, usually resulting in a dense point cloud [1-3].

However, the process of scanning is still time consuming and expensive, depending on the size of the target, with hours of field work required and hazards. This hinders applications that are data intensive, like training of deep learning models [4-7].

One solution seeing a lot of development is the creation of 3D imaging sensors simulators, especially for light detection and ranging (LiDAR) systems. These simulators are able to, given 3D models in a virtual scene, generate a synthetic point cloud. As the virtual world doesn’t have the same restrictions of the physical world, a user could generate a great volume of datasets in days, in contrast to weeks in field, with reduced costs and risks [4,5,7].

The automobile industry concentrates the most prominent researchers of this solution, focusing on deep learning models for autonomous cars [7-10]. One work [7] shows the benefits of using a ray tracing approach to simulate the LiDAR sensor behavior in a simulator for generating deep learning dataset for autonomous vehicles. Recently [10], a similar approach used a car simulator called Carla to generate synthetic point clouds with “semantic labels” of the detected obstacles.

As most of these researches focus on autonomous vehicles, there is a lack of more general simulators in the literature. In this paper, it is presented a LiDAR system based simulator with a parallel ray tracing approach and a scene builder.
2. Simulator

The simulator was developed with NVIDIA OptiX ray tracing engine, described as “a programmable system designed for NVIDIA GPUs and other highly parallel architectures” [11]. The engine enables a varied set of ray tracing algorithms and applications.

The simulator can be divided into two parts, the scene, and the virtual sensor. The used workflow is shown in Figure 1.

Figure 1: System workflow – parallelograms are data files and squares are applications.

2.1. Scene

The scene is constructed loading a scene description file, 3D models files of desired targets and a sensor path file into the application.

The scene description contains information about scale, rotation, position, and identification number of the 3D models in the virtual world. The identification number is a unique number assigned for each object in the virtual world. For example, if a forest area is created: the terrain, the trees and rocks could each be marked with an individual identification number. This information can be passed for each point in the output point clouds, as seen in Figure 2. Many procedures benefit from this type of data structure, especially segmentation [10,15].

The 3D model is a file that contains a 3D object including coordinates, texture maps, among other information. The used format is OBJ [17].

The sensor path is a file with coordinates of where to position the sensor for each scanning shot. The file contains 3D position and orientation.

This flexible virtual environment permits creating point clouds from simple objects, outdoors, indoors, or even hazardous environment, like subsea survey.

Figure 2: The synthetic point cloud of the forest area. The points are colored based on their objects’ identification number by a point cloud viewer.
2.2. Virtual Sensor
A virtual sensor is a mathematical representation, an algorithm, which emulates a real sensor’s behavior [12]. For the simulator described in this paper, it was used the definition “sensor measurement model”, described in [7] as “models based on the measurement process and generating low level data from the built scene”.

2.2.1. LiDAR. LiDAR system is a ranging measurement device that generates, after the data gathered is processed by a computer, massive sets of geospatial points called point clouds. This system components are, usually, a GPS unit, an inertial measurement unit and an active sensor based on a laser and a photodetector [1,2,3,13].

The LiDAR sensor’s working principle is based on time of flight, i.e., the time a laser beam shot from the sensor takes to return to the photodetector, as shown in Figure 3a. This time is used to calculate the distance of the beam travelling path as

\[
\text{distance} = \frac{\text{time of flight} \times \text{light speed}}{2}.
\]

The sensor has an operational scanning angle and distance range. The sensor shoots successive rays, separated by the step angle, within the operational range. Distance range is the distance interval where the LiDAR manufacturer guarantees the measurements will be within the expected measurement uncertainty.

2.2.2. Sensor Model. The literature shows that the ray tracing algorithm is a good starting point to simulate the LiDAR sensor behavior [4-10]. The idea behind the algorithm [11] is tracing the ray’s path from an eye (or camera) to a pixel in the screen, returning the color and distance information of the object hit by the ray, as shown in Figure 3b.

![Figure 3](image)

Figure 3: (a) Time of flight principle of a LiDAR. (b) Ray tracing.

For the purposes of this work, the rays shot in the OptiX engine were modeled considering the LiDAR characteristics highlighted in the previous subsection. The ray tracing now follows an angle range, shooting rays with a defined step angle and within a distance range. This yielded a realistic generation of synthetic point clouds, as the simulator virtual sensor is modeled by real LiDAR specifications.

2.2.3. Synthetic point cloud generation. The point cloud data is obtained from the collision of each ray with an object in the virtual world. When this occurs, the program calculates the intersection distance, position, normal vector and color of the surface hit point, plus the identification number of the object.
3D coordinates are the position of the point in the world. It’s the basic information of a point cloud. However, these coordinates are not geospatial, only corresponding to the scene built in the simulation.

Normal vectors are vectors perpendicular to the surface of an object. Like the identification numbers, previously defined in section 2.1, they are very useful for many approaches in training deep learning models and segmentation routines [15].

Color, obtained in the form of RGB vectors, helps verifying if the point cloud corresponds to its target 3D model, although LiDAR systems usually don’t have this kind of information.

Intersection distance is the raw information from a LiDAR sensor. Combined with rays’ directions, the LiDAR system calculates the 3D coordinates and builds the point cloud [3,7,10].

As described in section 2.1, these synthetic point clouds are taken from a series of shots following the instructions from the sensor path file. The application generates as many point clouds as there are scanning positions.

Nevertheless, some procedures [4,5,15], need a point cloud from the entire or a big part of target model. Using Geogram [14], a programming library of geometric algorithms, an application was developed to merge the single viewpoint clouds in a combined point cloud. It removes redundant points (points which 3D coordinates coincide or are too close to one another) based on a defined proximity threshold. The combined point cloud can also be normalized to be limited by a cube with an edge 2 units. The combined point cloud, as shown in Figure 4, can then be saved as a XYZ or PLY file.

![Figure 4: On the left, a 3D model of a Rubik Cube. On the middle, the single viewpoint cloud. On the right, the combined point cloud.](image)

3. Validation

The measured intersection distance of the virtual sensor model was compared to the intersection distance measured by a LiDAR sensor from a target object to verify if the virtual sensor behaviour was coherent with the LiDAR specifications. The LiDAR sensor URG-04LX-UG01 from Hokuyo was used for acquisition of data. It has an infrared laser (wavelength 785 nm) with a horizontal scan area of 240° (angle resolution approx. 0.3516°) and no vertical scan area. It has two operational distance ranges: from 60 mm to 1000 mm with measurement uncertainty of 30 mm; and from 1000 mm to 4000 mm with measurement uncertainty of 3% of detected distance.

For the target object a soda can was chosen, shown in Figure 5a, as it has a good reflective material and simple geometry. The middle of the can was positioned at 650 mm and 1300 mm from the LiDAR sensor in front of an infrared non-reflective wall, so no background information was gathered. For the simulator, the scene was based on the real target area and the virtual sensor configured with the Hokuyo LiDAR sensor specifications. The 3D model used was a can with its body covered by a cylinder mathematically generated with the same diameter as the soda can, as shown in Figure 5b. Both virtual sensor and LiDAR sensor were positioned to hit only the middle height of their respective cans.
These scanning experiments resulted in four range maps plotted with the measured intersection distances and shooting angle, two from the LiDAR and two from the virtual sensor. Figure 6 shows the range maps at distance 1300 mm and Figure 7 shows the range maps at distance 650 mm. LiDAR measured distances are marked as ‘o’ and virtual sensor measured distance are marked as ‘x’.

To calculate the root mean square error (RMSE), the difference between the virtual and real sensors data was used, shown in Figure 8. For the can positioned at 650 mm, the RMSE was 33.36 mm. For the can positioned at 1300 mm, the RMSE was calculated only for successful rays returned to the LiDAR, as the sensor missed information from two rays. The RMSE at position 1300 mm was 7.16 mm.

These are good indications of the virtual sensor behavior, as the simulator scene has no noise and background information from the environment was reduced for the real target, so the error between them is expected to be within the measurement uncertainty of the Hokuyo’ LiDAR. In fact, both RMSEs calculated are within the uncertainty range.
Figure 7: Range maps from the Hokuyo URG-04LX-UG01 LiDAR ('o') and the virtual sensor ('x') at position 650 mm (distance vs ray shooting angle).

Figure 8: Difference between real and virtual range maps at position 650 mm (dashed line) and at position 1300 mm (solid line).

The results from the validation experiments verifies that the virtual sensor follows the scanning behaviour of the LiDAR sensor it was modelled after, as the rays hit the surface of the can at the same ray shooting angle and the difference between measurements are within the uncertainty of the real sensor. However, even though all the affirmations above are true, some interesting phenomena appear in Figure 6 and Figure 7.

In Figure 6, the LiDAR didn’t receive information from two rays, meaning that the rays that hit the surface of the can were reflected away from the photodetector. Also, one of the successful rays hit was detected nearer than it truly is. Both issues show that the system has more difficult measuring the distance close to the curvature of the can. This can be related with known LiDAR problems, like backscattering and diffuse reflection, as they can cause the system to register a false data position, called artifacts, or no information at all, called holes [1-3,13].
In Figure 7, the LiDAR sensor measurements show an intensification of the problems observed in Figure 6, as the distances measured from the can’s curvature were even smaller than the real distances. This is expected from time of flight devices when measuring distances below 1000 mm [3,13], as the light returns too fast for the device photodetector to register the information correctly.

4. Conclusion and Future works
This manuscript presented a LiDAR system based simulator able to consistently generate synthetic point clouds, as the virtual sensor is modeled by real LiDAR specifications, but it could be improved initially in two different aspects:

- Adding inherent documented sensor measurements errors, like those shown in the Hokuyo LiDAR sensor, during the virtual sensor scanning procedures [4,5,7,10]. An error distribution model is already being developed for future uses of this simulator;
- Adding environmental noises, like scattering particles, and target media or object material characteristics, like refractive and reflexive index [4,5,7-9]. This would allow for creating more accurate hazardous environment scenarios for the simulator. A Monte Carlo Mie Scattering approach is being researched to develop filters that disturb the rays travelling path like in the mentioned issues.

Acknowledgements
The authors thank for the financial support provided by the Brazilian funding agencies CNPq, FINEP and FAPERJ. This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (Capes) - Finance Code 001.

References
[1] Blais F 2004 Review of 20 years of range sensor development J. Electron. Imaging 13 231-240
[2] Chow J C K 2014 Multi-Sensor Integration for Indoor 3D Reconstruction. Alberta, 239 p. PhD Dissertation - Department of Geomatics Engineering. University of Calgary
[3] Oliver-Codina G and Massot-Campos M 2015 Optical sensors and methods for underwater 3D reconstruction Sensors 15 31525-31557
[4] Tallavajhula A 2018 Lidar simulation for robotic application development: modeling and evaluation. Pittsburgh, 107 p. PhD Dissertation - The Robotics Institute. Carnegie Mellon University
[5] Öhman N 2018 Simulation of LiDAR data for forestry applications. Umeå, 29 p. MSc Thesis - Department of Physics. Umeå University
[6] Majek K and Bedkowski J 2015 Range Sensors Simulation Using GPU Ray Tracing Proc. the 9th International Conference on Computer Recognition Systems CORES, Cham
[7] Hanke T et al 2017 Generation and validation of virtual point cloud data for automated driving systems Proc. IEEE 20th International Conference on Intelligent Transportation Systems
[8] Hanke T et al 2015 Generic Architecture for Simulation of ADAS Sensors 16th International Radar Symposium (IRS) p 125-130 Dresden
[9] Hirsenkorn N et al 2017 Learning Sensor Models for Virtual Test and Development In Workshop Fahrerassistenz und automatisiertes Fahren
[10] Wang F et al 2019 Automatic Generation of Synthetic LiDAR Point Clouds for 3-D Data Analysis IEEE Trans. Instrum. Meas 68 2671-2673
[11] Parker S G et al 2010 OptiX: a general purpose ray tracing engine ACM Trans. Graph. 29 66
[12] Constantini R and Sussstrunk S 2004 Virtual sensor design Proc. Electronic Imaging, San Jose, California, United States
[13] Gokturk S B et al A time-of-flight depth sensor – system description, issues and solutions Proc. Conference on Computer Vision and Pattern Recognition Workshop, Wanshigton
[14] ALICE project-team, [Online]. Available: http://alice.loria.fr/index.php/home.html [Acessed 23 06 2019]
[15] Qi C R et al 2017 Pointnet: Deep learning on point sets for 3d classification and segmentation Proc. IEEE Conference on Computer Vision and Pattern Recognition 652-660
[16] Salmon J K et al 2011 Parallel random numbers: as easy as 1, 2, 3 Proc. International Conference for High Performance Computing, Networking, Storage and Analysis p 16, ACM
[17] The Graphics File Formats Page, [Online]. Available: http://www.martinreddy.net/gfx/ http://alice.loria.fr/index.php/home.html [Accessed 23 06 2019]