2021 NDIA GROUND VEHICLE SYSTEM ENGINEERING AND TECHNOLOGY SYMPOSIUM
MODELING SIMULATION AND SOFTWARE TECHNICAL SESSION
AUGUST 10-12, 2021 - NOVI, MICHIGAN

FAST LIDAR VEGETATION RESPONSE MODELING IN SIMULATION

Rich Mattes, James Pace

Neya Systems

ABSTRACT

Off-road autonomy development is increasingly leveraging simulation for its ability to rapidly test and train new algorithms as well as simulate a wide variety of terrains and environmental conditions. Unstructured off-road environments require modeling complex environmental phenomena, such as LIDAR responses from vegetation. Neya has developed an approach to characterize the variability of measurements of vegetation and approximate the variability of vegetation measurements using that characterization. This method adds a small overhead to existing LIDAR models, works with many types of LIDAR sensor models, and simply requires objects to be tagged in the environment as vegetation for the sensor models to respond appropriately.

Citation: R. Mattes, J. Pace, "Fast LIDAR Vegetation Response Modeling in Simulation", In *Proceedings of the Ground Vehicle Systems Engineering and Technology Symposium* (GVSETS), NDIA, Novi, MI, Aug. 10-12, 2021.

1. INTRODUCTION

Off-road environments consist of complex and unstructured features, posing challenges for Autonomous Ground Vehicles (AGVs) and their off-road autonomy software. Accurately representing complex features in a simulation environment, such that simulated AGV sensor measurements provide enough fidelity to stand in for real sensors, poses a significant challenge. Simulation engineers are constantly making trade-offs between model fidelity and execution speed in the context of the simulation's interaction with an autonomy system. Models that approximate relevant dynamics of a system without introducing a significant amount of computational complexity greatly enhance the utility of simulation tools.

One such trade off is how accurately to model a LIDAR's response to vegetation. Vegetation in off-road environments contains fine geometric details at a large scale, representing those fine details in a simulation environment requires significant computational resources. This paper describes a methodology developed for LIDAR-based modeling of vegetation which approximates the impact of vegetation's complex geometry on LIDAR measurements without requiring detailed vegetation geometry models. The methodology was implemented in a simulation environment, and results of our implementation are compared with data collected from field experiments in

off-road environments. The execution time of our implementation is measured to capture the overhead of the algorithm compared to an unmodified LIDAR model.

2. LIDAR MODELING IN SIMULATION

Scanning 3D LIDAR sensors are a popular sensor in on-road and off-road vehicle autonomy. They provide a 360 degree 3D point cloud of sampled ranges in the environment, measured with centimeter-level accuracy, several times per second. These high bandwidth (up to millions of points per second), continuous measurements are used to generate detailed models of the autonomous vehicle's operating environment.

LIDAR models in 3D simulators must provide a realistic representation of range measurements originating from a real sensor to stand in for real LIDAR sensors. The required fidelity of the representation varies depending on the sensitivity of the algorithms being exercised with the sensor data. LIDAR sensor models must also execute quickly in the simulation environment, to enable use cases such as software-in-the-loop (SIL) or hardware-in-the-loop (HIL) testing requiring real-time execution speed. Based on those constraints, simulated LIDAR range calculations are typically implemented in 2 ways in 3D simulation environments: collision based raytracing and sampling of depth rasterization.

2.1. Ray Trace Based Methods

A common way to simulate LIDAR range measurements is to approximate each LIDAR beam with a ray cast from the sensor through the environment. This ray tracing method uses collision detection algorithms to propagate the ray through the environment until it collides with an object, or reaches the maximum detection range of the LIDAR without a collision. Physics engines, such as Unreal's PhysX, provide implementations of this type of ray-based collision detection[1], making the model implementation relatively straightforward.

Efficiently detecting collisions with ray casts is a complicated problem, and the complexity increases with the number and detail of objects in an environment. The PhysX raycast API implements collision detection in several phases, filtering the objects in the environment to the ones most likely to result in a collision before performing expensive, detailed collision checks. Increasing the number of objects in the environment, and the detail of those objects, slows down the collision checking algorithms. With LIDAR sensor models requiring potentially millions of collision checks per second, even small slow-downs in collision checking have large effects in the overall speed of the sensor simulation.

2.2. Image Based Methods

Another way to simulate LIDAR sensors in a 3D simulation is to take advantage of a Graphics Processing Unit (GPU) that can render the 3D objects in a simulated scene. Using a GPU, the objects in a 3D environment can be rendered such that each pixel is colored by its distance from the camera, as opposed to its RGB color from the object's textures and the environment's lighting. The pixel values in the depth image are then sampled computing the propagation distances for each LIDAR beam.

This method requires that the 3D objects in the environment to be transferred to and rendered by a GPU. Simulation engines that are capable of 3D rendering, through a game or other visualization engine, implement much of the functionality required for GPU-based LIDAR modeling. New LIDAR instances add a similar amount of overhead as adding a new camera to the scene for image capture.

2.3. Vegetation Modeling

While both methods provide for rapid generation of LIDAR range returns, they typically lack fidelity to represent the interaction of LIDAR beams with

dense vegetation. Increasing the fidelity of each type of sensor model involves trade-offs for execution time and computational resource usage.

The ray trace methods rely on 3D collision algorithms, which are highly dependent on geometry for performance. As the polygon count for each object rises, it becomes more computationally intensive to test for collisions against an object in the environment. This is a problem for vegetation, where individual leaves, branches, and blades of grass are numerous, small, and complex. Adding many unique objects to the scene with high detail also increases memory consumption, as the physical make up of each object must be kept in memory to be referenced for collision detection.

Image based LIDAR methods have a similar scaling problem. Depth values are rendered at a fixed resolution, so as complex objects get farther away from the sensor, the level of detail captured by the sensor is reduced. Increasing the resolution of the depth image captures more detail at longer ranges, at the expense of greater GPU processing time per scan.

3. VEGETATION RESPONSE LIDAR MODEL

We have developed a methodology to represent detailed LIDAR measurements of geometrically complex vegetation without requiring detailed 3D models. It adds two key features to returns from simulated vegetation. The first is a variance in the range measurements to emulate sampling complex vegetation structure. Second, it contains a dual-return model that simulates second return paths, emulating partial reflections off of complex vegetation.

3.1. Range Variance

The primary feature of our LIDAR model is the selective addition of range variance for vegetation measurements. The model first detects which LIDAR returns originate from objects that are classified in the simulation as vegetation. For each of these returns, the model adds a Gaussian random amount of range error to the return.

Naively adding large random noise values to the range measurements occasionally causes measurements to appear below the ground plane, or to penetrate an otherwise solid obstacle behind the vegetation model. To prevent this phenomena, we perform an additional check to detect the range to any objects behind the detected vegetation. If a collision with an object behind the vegetation is found, that collision range is used to limit the range error that can be added to the vegetation measurement.

To allow our simulation to accurately simulate the noise added by the vegetation we needed to determine the standard deviation and mean of that noise. As mentioned previously, our model of the noise is that the measured range value is the combination of the actual distance to the plant plus some Gaussian random noise. To determine the Gaussian noise parameters from real data, we calculate the average and standard deviation of the difference between the expected and measured ranges.

The first step to measure these parameters with real LIDAR data is to use a box filter with manually set bounds that select only points that correspond to vegetation. We then loop through each point that is vegetation and calculate the difference between its measured range the expected range at that point.

To determine the expected range value for a point from the measured data, we calculate the average range of its nearest neighbors. A point is considered another point's neighbor when that point is also considered vegetation and is in the same LIDAR scan ring as the original point. A neighbor point is considered near another point when the angle from the LIDAR to that point is close to the angle from the LIDAR to the original point. Splitting the point cloud into neighborhoods based on the ring and angle from the LIDAR is effectively splitting the point cloud based on the input to the model of the LIDAR system. It is tempting to limit the neighborhoods based on the

local coordinates of the returns, except the noise in the range from that is what we are ultimately trying to measure.

3.2. Dual Returns

The second important feature of our LIDAR model is support for dual returns. Newer commercial LIDAR sensors support measurements of multiple ranges from a single LIDAR emission. This allows for measuring multiple partial reflections of the laser beam, which often occurs when measuring small features like leaves or grass that do not totally reflect the LIDAR emission.

The approach to mitigate creating unrealistically long range measurements by searching for objects behind the vegetation return naturally lends itself to simulating a dual return LIDAR. The range from the ray trace to a second object behind the vegetation is added as a second LIDAR return originating from the same LIDAR emission. Dual returns are not computed for objects that are not detected as vegetation.

4. IMPLEMENTATION IN UNREAL

We have implemented the vegetation aware LIDAR model as an Unreal Engine plugin working in collaboration with the Applied Research Associates (ARA) Virtual Heroes Division. That plugin was integrated with VISE, an Unreal-based off-road autonomy simulator, for testing and evaluation. The implementation extends a custom GPU-based LIDAR model, which relies on a custom GPU shader that uses the values from the scene depth buffer to determine the depth values for the pixels in the scene. We chose the GPU LIDAR model as a basis for the model, over a ray cast model, due to the difficulty of consistently capturing detailed vegetation geometry in the Unreal physics subsystem.

Unreal supports two different physics colliders, complex and simple, which are used for different applications depending on the required resolution of the collider. To use ray casts to simulate LIDAR, the colliders must have a small resolution, and thus use complex colliders. Our experience is that assets that can be acquired from the Unreal Marketplace do not consistently have accurate complex colliders. While complex colliders can be added after the assets have been acquired, this slows down the process of creating new environments. In contrast, our shader based method just requires that the asset renders correctly in the scene.

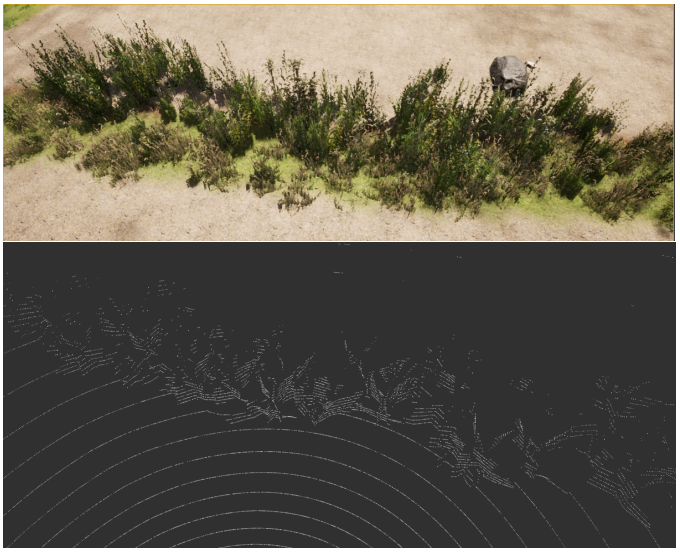

Figure 1: Top: Unreal scene with vegetation models. Bottom: Corresponding output from ray cast LIDAR model.

Figure 1 shows an example of physics colliders that do not represent the complexity of the vegetation mesh. The top image shows a collection of vegetation rendered in Unreal. The bottom image shows the corresponding point cloud from a ray cast LIDAR sensor model. In that point cloud, the vegetation objects are represented by simplified planar colliders, as can be seen by their flat, continuous structure. Note that the smaller vegetation in the bottom left portion of the cluster of vegetation is not captured at all by the ray cast LIDAR model.

4.1. Vegetation Model Implementation

The vegetation aware LIDAR model uses the custom depth distance and custom stencil buffers to determine if a pixel is considered vegetation. It assumes that all assets in the scene that are vegetation have been tagged with a particular custom stencil value either when the environment is created or when the environment is first loaded. The custom shader encodes this information into the RGB pixel value for the image which is sent from the shader to the model implementation. The most significant bit of the pixel is set to 1 if the pixel is vegetation and 0 if it is not. The rest of the bits are used to encode the depth of the image as an unsigned integer.

The output of the shader is then processed to generate 3D points. The image's pixels are sampled at points representing the 3D LIDAR scan points based on the angular sample rate, and the number and vertical offset of lasers in the LIDAR. At each sample point, the depth value is used to encode a range, and the vegetation flag is used to determine whether to apply the vegetation model discussed in Section 3.

LIDAR Type	Number of Points	Mean	Variance
Real	63383	-0.099250	1.594310
Simulated	63409	-0.0912173	0.587409

Table 1: Difference between measured and expected range in real and simulated measurements, in meters.

Table 1 shows the results from analyzing real measurements from vegetation, as well as from measurements taken in a representative simulated environment. The difference in mean and variance in the real and simulated data was used to set the amount of noise to add to simulated vegetation measurements to approximate real measurements. Using these measurements, the vegetation model is configured to apply zero-mean Gaussian noise with a variance of approximately $1m^2$, or standard deviation of $1m$.

Unreal's built in ray tracing capability is used to determine the second LIDAR response range. It is not possible to see through an object with the shader without using very specific custom depth tags, which would conflict with how we are using the tags to label vegetation objects. For the sake of computational efficiency, a ray trace is only performed for points that have already been determined to be vegetation, and the ray trace is only computed from the last impact site (as opposed to from the sensor position). For points that are not vegetation, the second return is placed in the same position as the first, based on our observations from data collected on vehicle. The potential lack of complex physics colliders discussed in Section 4 is less of an issue when simulating second returns, as the algorithms Neya tests are generally less sensitive to errors in the position of the second returns than to errors in the position of the first returns.

On top of generating the second return data, the distance from the ray cast is also used to provide a cap on the amount of noise we add when generating the first return. Without using the second point as a cap, the additional noise could send the first return point past a hard object behind the vegetation, which is unrealistic. For example, when the LIDAR is looking down at some tall grass, one would expect the first point to be noisy, but there would be a hard stop where the lasers intersect with ground.

5. RESULTS

This section analyzes the output of the LIDAR model implementation described in Section 4, compared to the output from a a depth-based sensor model as described in Section 2.2. It also examines the relative computational speed of these models for a scanning 3D LIDAR generating 10 scans a second.

5.1. Range Variance Comparison

This section looks at two different scenarios, a dense line of bushes and a less dense lower cluster of vegetation, using data from both simulation and

a real sensor. A third scenario with data from a wooded area shows dual return results with data from the simulation and a real sensor. The real sensor data was generated by a Velodyne VLP32MR, a 32 beam scanning 3D LIDAR capable of dual return measurements. For simulation, two different sensors are compared to the real data, both using parameters to match the VLP32MR. The first sensor is the vegetation aware LIDAR discussed in this paper; the second sensor is not vegetation aware, but otherwise uses the same shader-based method to generate the point cloud as our method.

Dense Line of Bushes

The first scenario is of a dense line of bushes. The image on the left of Figure 2 shows a picture taken with a camera attached to the front of the vehicle facing a line of bushes with a road behind them. The image on the right shows the point cloud generated by the VLP32MR of this scene. Notice the lack of structure in the portion of the point cloud that corresponds to the bushes. The noise is even more pronounced when looking at how the point cloud changes over time. The vegetation points change much more rapidly than the points that do not correspond to vegetation.

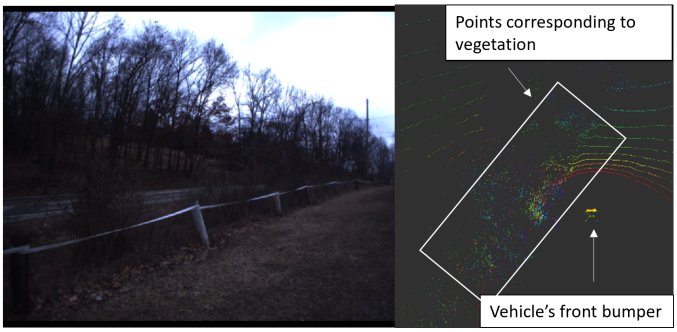

Figure 2: Sample LIDAR scan from dense bushes.

Figure 3 shows a similar scene built in simulation. The image on the left shows a capture of a camera attached to the front of the vehicle. The image on the right shows the point cloud generated from the shader-based sensor that is not vegetation aware. Much of the structure of the vegetation that is washed out by the noise in the real point cloud is visible in the simulated point cloud. The difference is more pronounced when looking at how the point cloud changes over time. The points in the point cloud from the physical sensor change position over time, whereas the simulated points are completely static from scan to scan.

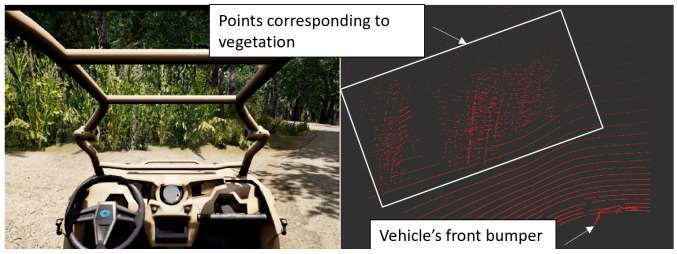

Figure 3: Simulated LIDAR scan from dense bushes.

Figure 4 shows the same scene, but captured by the vegetation aware LIDAR. Like the real sensor data, but unlike the previous simulated LIDAR data, there is a significant amount of noise in the point cloud. When looking at the cloud over time, unlike the previous simulated sensor, but like the real data, the points that are considered vegetation move and are not completely static.

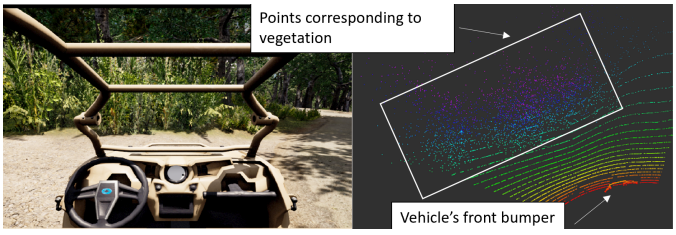

Figure 4: Simulated Vegetation LIDAR scan from dense bushes.

Low Cluster of Bushes

The second scenario looks at a region with slightly less dense brush. The left image in Figure 5 shows an image from a camera attached to a vehicle facing down a path with brush on either side of the path. The image on the right shows the point cloud

generated by the VLP32MR when looking at the same scene.

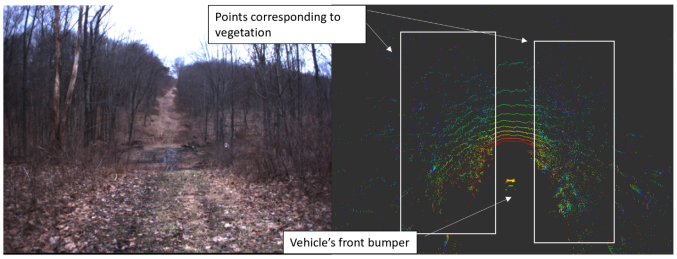

Figure 5: Sample LIDAR scan from a cluster of low bushes.

Figure 6 shows the a similar scene generated in simulation with the shader-based but not vegetation aware sensor. Note the points in the point cloud on the image on the right are slightly less noisy than in the point cloud for the real data. The difference is even more pronounced when looking at how the point cloud changes over time. The points for the real vary from scan to scan, whereas the points for the simulated data do not.

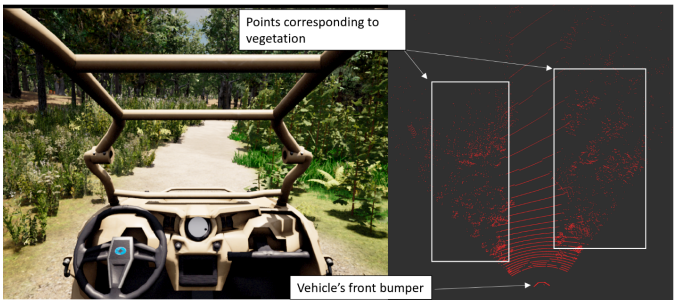

Figure 6: Simulated LIDAR scan from a cluster of low bushes.

Figure 7 shows the same scenario but generates the point cloud with our vegetation aware LIDAR. Note there is significantly more noise in the point cloud, closer to the real data. The improvement is even more pronounced when looking at how the point cloud changes over a period time. The points from our vegetation aware LIDAR vary over time, like the real points, whereas the points from the non vegetation aware sensor do not.

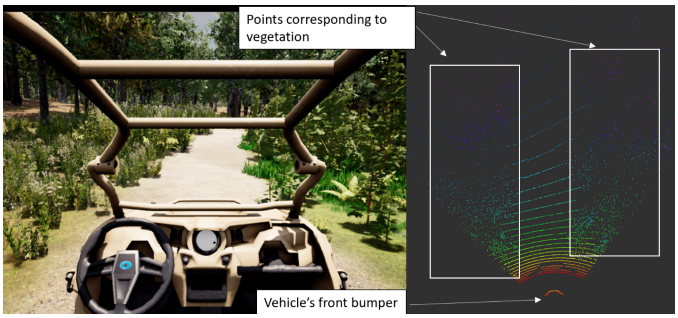

Figure 7: Simulated Vegetation LIDAR scan from a cluster of low bushes.

Dual Return from Wooded Area

The third scenario looks at how the simulated vegetation aware LIDAR handles dual return data. This scenario looks at a region with light vegetation in a wooded area off of a trail. Figure 8 shows the LIDAR response from the physical LIDAR. The blue points are from the first return and the red points are from the second. Note that there is noise in both clouds in the region of vegetation, but the first and second points overlap in the regions without vegetation.

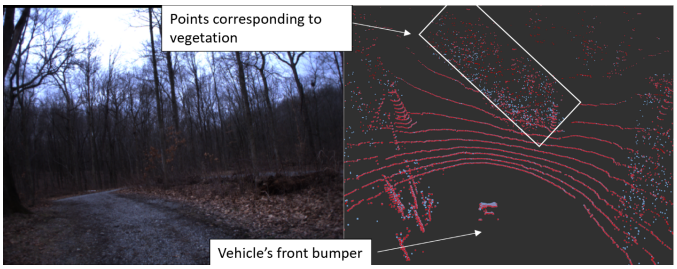

Figure 8: Sample Dual Return LIDAR scan.

Figure 9 shows a simulated scene in simulation using our vegetation aware simulated LIDAR. Notice that as with the real data in region without vegetation both returns overlap, but in the region with vegetation the position of both returns is noisy.

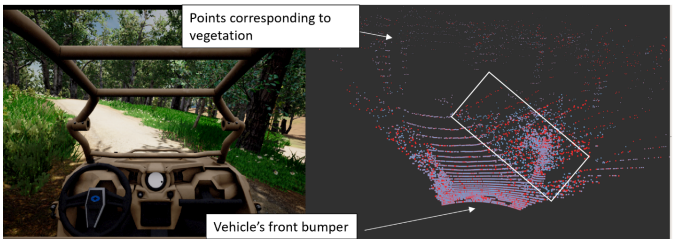

Figure 9: Simulated Dual Return Vegetation LIDAR scan.

5.2. Computational Overhead

To evaluate the computational overhead, we spawn a single LIDAR sensor instance in a simulated environment. Each LIDAR model has 64 simulated beams, and a 90 degree horizontal field of view. The simulation is configured to update 10 times a second, in lock step with the LIDAR sensor's update rate. We measure the wall-clock time required to execute the 100ms simulation steps, on a PC with an Intel i7-6700 processor, and an NVIDIA GeForce GTX TITAN X GPU. Results are shown in Table 2.

LIDAR Model	Average Execution Time (sec)
Depth LIDAR	0.038
Vegetation LIDAR (Single return)	0.123
Vegetation LIDAR (Dual return)	0.158

Table 2: Wall-clock execution timing for GPU based LIDAR models.

6. CONCLUSION

Neya has developed a method for capturing the variation of vegetation measurements in LIDAR models without requiring complex geometric modeling of the vegetation in an environment. An implementation in VISE is shown to provide range variation for vegetation, as well as dual LIDAR return support, with only a moderate performance penalty.

7. REFERENCES

[1] NVIDIA Corporation, "PhysX User's Guide: Scene Queries", 2017. [Online]. Available: `https://docs.nvidia.com/gameworks/content/gameworkslibrary/physx/guide/Manual/SceneQueries.html`. [Accessed: 20-May-2021]