Off-Road Lidar Simulation with Data-Driven Terrain Primitives

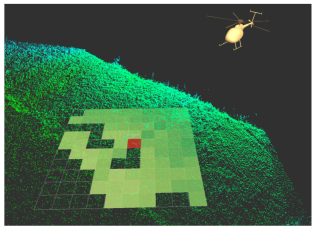
Abhijeet Tallavajhula[1] and Çetin Meriçli[1] and Alonzo Kelly[1]

***Abstract*— Developing software for large scale off-road robot applications is challenging and tedious due to cost, logistics, and rigor of field testing. High-fidelity sensor-realistic simulation can speed up the development process for perception and state estimation algorithms. We focus on Lidar simulation for robots operating in off-road environments. Lidars are integral sensors for robots, and Lidar simulation for off-road environments is particularly challenging due to the way Lidar rays interact with natural terrain such as vegetation. A hybrid geometric terrain representation has been shown to model Lidar observations well [1]. However, previous work has only been able to simulate a single, fixed scene, and the entire scene had to be precisely surveyed. In this work, we add semantic information to the hybrid geometric model. This allows us to extract terrain primitives, such as trees and shrubs, from data logs. Our approach uses these primitives to compose arbitrary scenes for Lidar simulation. We evaluate our simulator on a real-world environment of interest, and show that primitives derived using our approach generalize to new scenes.**

I. Introduction

Lidars are a key technology that have helped advance achievements in outdoor robots. As sensors, they provide robots with observations of the environment. A large amount of software is developed to process Lidar observations, for example, algorithms for state estimation and recognition. Such software should be able to deal with the noise and scale of Lidar data, among other requirements. The reliable and high-quality performance of perception software is essential for good decision making in the higher levels of autonomy. As the scale of robot operations grows, so do the challenges of software development. It may be difficult, unsafe, or expensive to develop software on enough real-world environments. Simulation has attracted attention as a solution to these problems.

(a) Perception for off-road mobile robots. Image from `http://bit.ly/2tkR2UC`.



(b) Perception for autonomous helicopters. Image from [2].

Fig. 1: The high costs of large scale software development for robots motivate high-fidelity off-road Lidar scene simulation.

[1]The Robotics Institute, Carnegie Mellon University, atallav1@andrew.cmu.edu, cetin@cmu.edu, alonzo@cmu.edu

In this work, we focus attention on the off-road case. We present some motivating examples for building high-fidelity off-road Lidar scene simulators. Consider developing software, for tasks such as terrain modeling [3] and virtualized reality [4], for off-road mobile robots (see Figure 1a). Real-world development can be very expensive, with potential delays due to unforeseen weather and hardware issues. Further, some tests amount to repeated robot runs under different software or hardware configurations. Such problems are an ideal fit for solution by simulation. To be useful, however, simulation must be high-fidelity: we would like simulated data to challenge software development to the same degree that real data does. Additionally, the simulator must be flexible enough to simulate different real scenes of interest. Another example is the use of Lidars in autonomous helicopters, see Figure 1b. In [2], a convolutional neural network (CNN) was trained to detect landing zones from Lidar observations. Real data was obtained from expensive flight tests. Synthetic data was obtained in [2] using a simple simulator. It was noted that *a priori* modeling of Lidar returns from vegetation was difficult, and that simulator evaluation was a challenge. In this work, we build realistic Lidar models from data. We also perform data-driven evaluation of simulators, comparing against real data from complex scenes. Synthetic data from the simple simulator in [2] was found to be useful in CNN architecture selection; a high-fidelity Lidar simulator can be of greater value in training learning algorithms.

For Lidar simulation in urban scenes, and for objects with planar structure, raycasting with additive noise may be good enough. Off-road scenes, on the other hand, consist of terrain such as uneven ground and vegetation. Effects such as pass-throughs and mixed pixels occur. Modeling the geometry of vegetation is an added challenge. We briefly review the current options for off-road Lidar scene simulation. The first is to use data logs as a simulator. In a sense, such a simulator has the highest fidelity, since real data is simply replayed. Of course, data logs do not generalize to new sensor poses, which is why they are useful only in the preliminary stages of software development. Data logs have low flexibility, as new scenes cannot be composed. A second option is to use a general-purpose robotics simulator, such as Gazebo [5]. The fidelity of such a simulator is at best, untested, and at worst, low: there exists little work in systematically comparing simulated Lidar observations to real data. On the other hand, such a simulator is highly flexible: new scenes may be composed using seasoned user interfaces. A third option is a simulator based on a hybrid geometric model, as in [1]. The simulator in [1] used a sensor model which took

into account the non-idealities of Lidar observations. Models were fit to real Lidar observations, and high-fidelity simulation was demonstrated. However, the simulator was fixed to a single scene (or 'geospecific'). Flexibility was low since new scenes could not be composed. In this work, we use the hybrid geometric simulator as a starting point, inheriting its fidelity. In addition, we add semantic information to Lidar observations. This allows us to extract scene primitives, such as trees and shrubs, from training data. New scenes can then be composed using the extracted scene primitives.

The outline of the paper is as follows. In Section II we discuss related work. We summarize work that led up to the hybrid geometric models for Lidar simulation. We make connections to recent research in robot and computer vision. Section III contains notation used in the rest of the paper. In Section IV, we present an overview of our approach to Lidar scene simulation, followed by details. In Section V, we evaluate our approach with real-world experiments. We conclude in Section VI with a discussion of the next steps being taken in this direction of research.

II. Related work

A number of general-purpose robotics simulators [6], [7], [5] perform Lidar simulation. Most simulate observations by calculating the range to object models, and adding Gaussian random noise. Such a sensor model is not appropriate for the off-road case, where effects such as pass-throughs and mixed pixels [8], [9] occur. The sensor model presented in [10], and the corresponding simulator [11], account for mixed pixels by raycasting multiple rays for each Lidar beam. Unfortunately, this approach is expensive when simulating observations from thin structures. In addition to a sensor model, off-road Lidar simulation requires a terrain model. The graphics community has paid attention to modeling plants, such as the generative model in [12]. However, there was no comparison to real vegetation, nor was it obvious how the model could be tuned to fit real observations. A method to reconstruct trees from real data was presented in [13]. In a similar vein, [14] used domain knowledge of tree structure to guide an iterative reconstruction from data. Neither of these approaches considered modeling vegetation for the purpose of simulation.

An extensive evaluation of models for Lidar simulation was carried out in [1]. A hybrid geometric model was found to perform well for different types of terrain. The ground was modeled as a triangle mesh, and non-ground terrain as a cluster of ellipsoids. The sensor was modeled jointly with the terrain structure. Lidar effects such as pass-throughs were modeled by associating geometric elements with a hit probability. The models were fit to real data, and realistic simulation was demonstrated in [1]. We will refer to the resulting simulator as the hybrid geometric simulator. The hybrid geometric simulator is used as a base for our work, although, as noted in Section I, it only simulates a fixed scene.

Simulation for perception is a timely topic that has received interest in related communities. Simulators for testing robot perception algorithms were presented in [15], [16]. In both, synthetic scenes were created using Unreal Engine 4[1]. While the game engine allowed the flexible construction of simulated scenes, how closely they mapped to real scenes was unaddressed. Simulation has long been used in computer vision. Given the prevalence of learning-based methods, synthetic data is useful when obtaining real data is a challenge. This use of simulators is of greater relevance when considering deep learning methods [17]. A synthetic dataset for semantic segmentation was presented in [18], with diverse urban scenes generated using the Unity development engine[2]. How well they mapped to real scenes was unclear. For the task of multi-object tracking, a Virtual KITTI dataset was presented in [19]. For realism, virtual scenes were seeded with annotations from the real KITTI dataset. In our approach we evaluate simulation on real scenes. In this sense we are similar to [20], but different in domain. The work of [20] presented SceneNet, a synthetic dataset of annotated indoor scenes. The simulator used realistic sensor models[21], in combination with OpenGL[3]. Emphasis was laid on creating realistic scenes in simulation, using object co-occurrence statistics calculated from real data.

III. Notation

The world state is denoted by x. We consider all information in the world that affects sensor observations as part of x, not just the sensor pose. The sensor pose is q_{sensor}. Noting that the real world state x is never exactly recreated in simulation, we explicitly denote the world state in simulation by $\hat{x}$. The sensor observation is $z \in \mathbb{R}^{d_z}$. The true observation distribution is $p(z|x)$. Our model of this quantity, $\hat{p}(z|\hat{x})$, is called the sensor model. Note that the sensor model depends on the simulator world state. Samples drawn from the sensor model, $\hat{z} \sim \hat{p}(z|\hat{x})$, are simulated observations. For a Lidar, the observation z consists of range measurements of the environment, collected in some scanning pattern. A range measurement transformed to a point in the world frame, using the sensor pose q_{sensor} and intrinsics, is denoted by $y \in \mathbb{R}^3$. A set of points in the world frame , possibly derived from multiple observations, will be called a point cloud, $Y = \{y_j\}$. Simulated observations, similarly transformed to points in the world frame, constitute a simulated point cloud $\hat{Y} = \{\hat{y}_j\}$. Parameters of the simulator are denoted by θ. The quality of simulation with parameters θ is measured by comparing the real and simulated point clouds using a loss function $l(Y, \hat{Y}, \theta)$. Our approach is data-driven, and we assume the availability of datasets, such as $D_{\text{train}} = \{(q_{\text{sensor},i}, z_i)\}$ to train the simulator.

The building block of a simulated scene is a scene element $e = (\rho, \omega)$. We jointly model the terrain structure and the sensor. A scene element consists of the element geometry ρ, and sensor model parameters ω. The simulator world state $\hat{x} = (S, q_{\text{sensor}})$ consists of a scene S and the sensor pose q_{sensor} . A scene $S = \{o_k\}$ is a set of objects. An

[1] https://www.unrealengine.com/
[2] https://unity3d.com/
[3] https://www.opengl.org/

object $o = (c, M, q_{\text{object}})$ consists of an object class c, model M, and pose q_{object}. The object class c is a categorical variable, and functions as an identifier. The object model $M = \{e_i\}$ is a set of scene elements. An object with its pose set to identity is a scene primitive $\pi = (c, M)$. We omit writing the pose in the tuple of a primitive. The unique set of scene primitives is the primitive set, $\Pi = \{\pi_j\}$. The scene annotation $a = \{(c_k, q_{\text{object},k})\}$ is the just the set of object class and pose information. Intuitively, for a new scene, objects are constructed given the scene annotation and primitive set.

IV. Scene simulation

We first present an overview of our approach to simulation. Various steps, including implementation details, are discussed in subsections that follow. Our simulator is written in C++, and we use MATLAB for labeling and scene generation.

Approach overview

We assume the availability of a training dataset $D_{\text{train}} = \{(q_{\text{sensor},i}, z_i)\}$ consisting of sensor poses and observations. This may be collected, for example, by driving a robot with a Lidar in a real scene, and logging observations (see Figure 2). Using the sensor poses and intrinsics, we transform the range measurements to points in the world frame, to obtain a point cloud Y_{train}. The point cloud is segmented into ground and non-ground points. We then fit scene elements $\{e_j\}$ to each segment separately (hence the name, hybrid geometric simulator). To ground points, we fit a triangle mesh; the element geometry ρ is therefore a triangle. On the non-ground points, we perform a clustering; ρ in this case is an ellipsoid. Once the geometry of the point cloud has been obtained, we find the sensor model parameters ω for each element. These include hit probabilities and range variances. Having modeled the training scene, we can simulate observations by querying a Lidar pose q_{sensor}. We obtain a simulated point cloud $\hat{Y}_{\text{train}}$ by querying the sensor poses in the training data log D_{train}. A point cloud error metric is used to compare $\hat{Y}_{\text{train}}$ and Y_{train}. The simulator parameters θ are tuned to minimize this error metric.

The procedure outlined above leads to a Lidar simulator which has been optimized to simulate observations in the fixed training scene. This is exactly where the work of [1] terminates. At this point, the scene elements $\{e_j\}$ are blind to objects semantics. Therefore, we perform object segmentation and labeling in the point cloud Y_{train}, from which the annotation $a = \{(c_k, q_{\text{object},k})\}$ is derived. The segmentation also allows us to group elements into an object model, $M_k = \{e_j \mid e_j \in \text{segment}_k\}$. These pieces of information are combined into scene objects $\{o_k\}$, $o_k = (c_k, M_k, q_{\text{object},k})$. Finally, we transform objects to identity (as a reference pose), thereby extracting a set of primitives $\Pi = \{\pi_i\}$, $\pi_i = (c_i, M_i)$ from the training scene. The approach is summarized in Figure 2.

For a test (or new) scene, we collect data in a similar manner as for the training scene, resulting in the dataset D_{test}. In this case, however, we do not fit scene elements to the point cloud Y_{test}, since we want to evaluate the generalization of the scene primitives obtained from the training data. The cloud Y_{test} is segmented and labeled, resulting in an annotation $a = \{(c_k, q_{\text{object},k})\}$. This annotation for the test scene is combined with the scene primitives, from the training scene, to result in objects. Having constructed a simulated test scene, we can simulate Lidar observations at query sensor poses q_{sensor}. We evaluate our scene simulator by comparing the simulated point cloud $\hat{Y}_{\text{test}}$ with Y_{test}.

Given our approach, note that it is possible to construct arbitrary scenes, which is a common use case for a simulator. For example, we may want test perception software in a densely forested environment. All that is needed is to supply the framework with an annotation for the desired scene. However, an important aspect of our work is that the annotations are derived from real data. This allows us to make claims about the fidelity of simulation for new, real-world scenes of interest.

Ground segmentation

Ground segmentation is performed on the point cloud $Y = \{y_j\}$ obtained from the training data, and is a preprocessing step before scene elements are fit to Y. Ground segmentation is a routine step in point cloud processing, and we use a simple procedure based on geometric features, as in [1]. The spherical variance $\phi(y)$ of a point y is calculated as follows. For points in a ball of radius d_ϕ, centered at y, the covariance is computed. If the eigenvalues of the covariance matrix are $\lambda_1 \leq \lambda_2 \leq \lambda_3$, then $\phi(y) = \frac{\lambda_1}{\lambda_1+\lambda_2+\lambda_3}$. If there are less than $N_{\text{min},\phi}$ neighbors, a default value is assigned. The spherical variation is a measure of the 'flatness' of the local neighborhood. Classification of a point is ground if $\phi(y) < \phi_{\text{thresh}}$, and non-ground otherwise. Before classification, the features may optionally be smoothed in some neighborhood of y. The parameters used in ground segmentation are tuned on a manually segmented point cloud.

Scene elements

Once an input cloud is segmented into ground and non-ground, we jointly model the terrain and sensor by fitting scene elements to the point cloud. A lesson for Lidar simulation, from the work of [1], was that surface elements are appropriate for terrain with planar structure, while volumetric elements are appropriate for terrain with irregular structure. Working with this insight, we model ground points by a triangle mesh (see Figure 3). We fit a regularized surface to the points using the fast RBF interpolation utility in ALGLIB [22]. We then Delaunay triangulate the surface using CGAL [23]. Ray-triangle intersections are also queried using CGAL. Therefore, each ground element $e_j = (\rho_j, \omega_j)$ has a triangle for its geometry ρ_j. The sensor model parameter $\omega_j = (\sigma^2_{\text{ground}}, p_{\text{hit},j})$ consists of the range variance σ^2_{ground}, and a hit probability $p_{\text{hit},j}$. Suppose that a ray hits a triangle element, and the nominal range is r. The simulator adds noise independently sampled from the Gaussian distribution $\mathcal{N}(0, \sigma^2_{\text{ground}})$ to r. Note that we use the same variance for

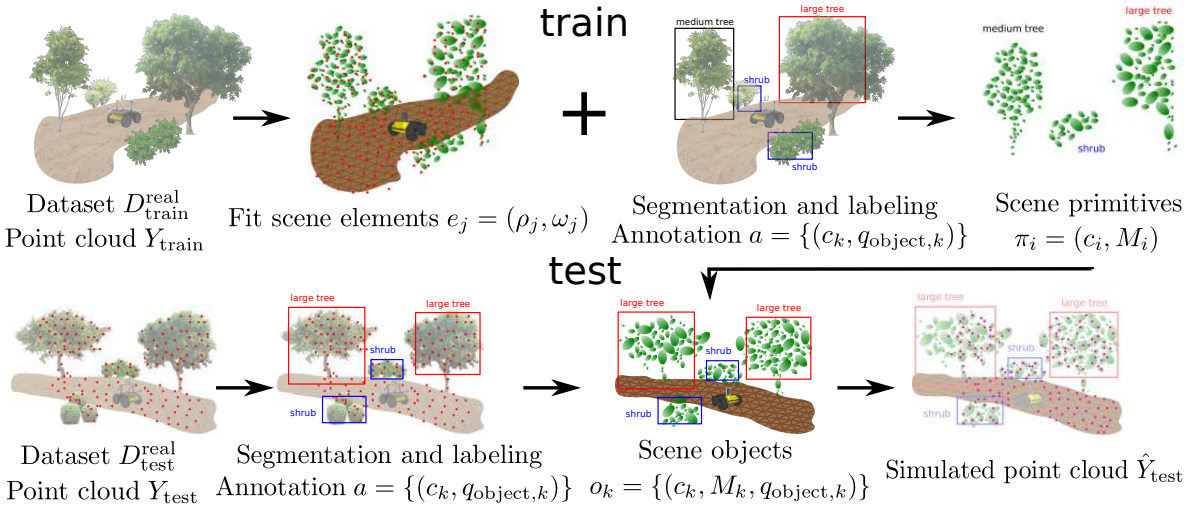


Fig. 2: An overview of the approach, in pictures.

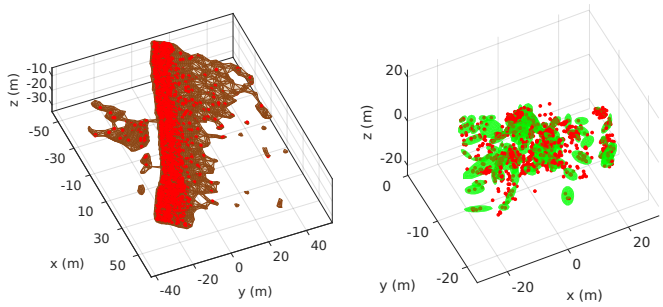


Fig. 3: Hybrid geometric scene elements. Ground points are modeled by a triangle mesh. Non-ground points are clustered into ellipsoids. The red markers are real Lidar range data.

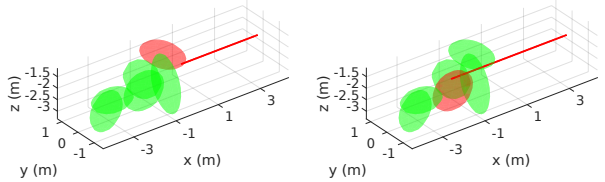


Fig. 4: Elements are associated with a hit probability, according to which a ray intersecting an element may pass through. On the left, the ray first strikes the ellipsoid element shaded in red. It may pass through, and intersect the element shaded in red, on the right.

all ground elements. It is calculated from the training data once the triangle mesh has been fit.

Non-ground points for off-road terrain in our case correspond to Lidar observations from trees, shrubs, and other vegetation. To model their irregular structure, we fit ellipsoids to these points (see Figure 3). We first run FLANN's [24] hierarchical clustering on the points. Each cluster is a scene element. The geometry $\rho_j = (\mu_j, \Sigma_j)$ of the element is an ellipsoid, whose mean and covariance are calculated as $\mu_j = \frac{1}{n}\sum_k y_k$, $\Sigma_j = \frac{1}{n-1}\sum_k (y_k - \mu_j)(y_k - \mu_j)^T$. The index k runs over points y_k belonging to cluster j. The sensor model parameters in this case are $\omega_j = (\mu_j, \Sigma_j, p_{\text{hit},j})$. Note that the same mean and covariance are used for a Gaussian distribution. If a ray hits an ellipsoid element e_j, then the simulated observation is a sample from the distribution $\mathcal{N}(\mu_j, \Sigma_j)$. Sampling in this manner smears observations, taking into account some of the mixed pixel effects. The hit probability is a parameter that models pass-throughs in Lidar data. When a ray intersects an element e_j, a Bernoulli random variable is sampled with probability $p_{\text{hit},j}$. If the value is 0, the ray passes through. The ray may intersect another element e_k, in which case $p_{\text{hit},k}$ is sampled (see Figure 4). The hit probability is calculated from data as $p_{\text{hit}} = \frac{\#\text{hits}}{\#\text{hits}+\#\text{misses}}$, where #hits is the number of rays which hit an element, and #misses is the number which missed the element, conditioned on the fact that the rays intersected the element. Ray-ellipsoid intersections can be calculated in closed-form [1]. The hit probability for surface triangle elements is used, and is calculated in, exactly the same manner.

The algorithms used for fitting scene elements, such as those for finding a regularized surface, clustering, etc., in turn have parameters. These are part of the overall simulator parameters θ.

Scene segmentation and labeling

In this work, segmentation and labeling is manual. We first segment the Lidar data corresponding to a scene using the tools in CloudCompare [25]. The segments are then labeled using a custom MATLAB tool, visualized in Figure 7. We consider the following label classes: small shrub, medium shrub, large shrub, small tree, medium tree, large tree. While the spatial extent of trees are often distinct, shrubs may be spread over a large, irregular region, which we refer to as a patch. We deal with patches by dividing them into cells, see Figure 5. During training, primitives in each cell are stored as primitives. During test, each cell in a patch is populated using a cell primitive.

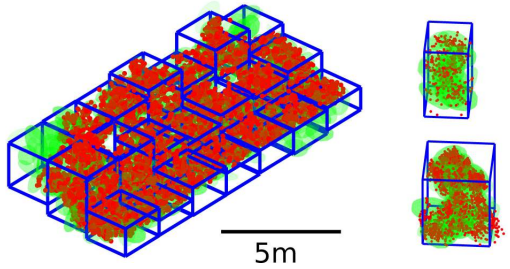


Fig. 5: In off-road terrain, shrubs often occur in extended sections. We deal with such patches by dividing them into cells, as shown on the left. Like trees, shrubs also occur in isolated instances, with examples on the right.

Scene objects and primitives

Having obtaining scene elements and labeled segments, we combine the pieces of information to obtain scene objects. For each segment with label c containing points $\{y_j\}$, we find the pose q_{object}. The translation is the centroid of the points. We assume rotation along the z-axis only, and find the orientation using the principal axes in the x-y plane of the segment points. For the object model, a bounding box oriented along the 2D principal axes is fit to the segment points. The object model then consists of scene elements in the box interior, $M = \{e_j \mid e_j \in \text{object bounding box}\}$. Using the pose q_{object}, we then transform the object model M to identity, and store the result as a scene primitive $\pi = (c, M)$. The primitives together constitute the primitive set Π.

For a new scene, we use an annotation $a = \{(c_k, q_{\text{object},k})\}$, consisting of object classes and poses, to populate the simulated scene. For a pair $(c_k, q_{\text{object},k})$, we pick a primitive $\pi_i = (c_i, M_i)$ such that $c_i = c_k$. In our implementation, the annotation for a new scene is also derived from a semantic segmentation. In that case, the annotation for a scene object includes a bounding box. We use the heuristic of selecting the primitive whose bounding box most closely matches the bounding box of the scene object. For the metric between bounding boxes, we use the euclidean norm of the box extents. The primitive model M_i is then transformed to the pose $q_{\text{object},k}$, resulting in object o_k. The annotation a can correspond to arbitrary scenes. In this work, when evaluating simulation for a real scene, the annotation is derived from a segmentation and labeling step.

Simulator baseline and evaluation

We implemented a baseline simulator in which object models were open-source 3D mesh models. The sensor model consisted of raycasting with the object models, followed by adding Gaussian noise to the nominal range. This baseline is representative of what can be implemented in current general-purpose robotics simulators. We created a primitive set for the baseline simulator, using mesh models from TurboSquid[4]. We briefly note that the number of models we worked with was limited due to: scarcity of freely available mesh models of natural terrain; use of proprietary file types; large-size ($>$ 10MB) mesh models. The raw mesh models were transformed, scaled and sorted into appropriate classes. Example baseline primitives are shown in Figure 8. Given an annotation, scene objects are constructed from mesh model primitives for the baseline in the same way as for our hybrid geometric scene simulator.

In our case, a sensor observation z is a packet of Lidar data. Each packet consists of returns from rays fired at different directions, from a common sensor pose q_{sensor}. Some of the rays result in hits, and others in misses. For a simulated observation $\hat{z}$, we compute the packet error,

$$l^{\text{pack}}(z, \hat{z}) = \frac{1}{\#\text{ true hits}} \sum_{j \in \text{ true hits}} \|y_j - \hat{y}_j\|. \quad (1)$$

Where y is the 3D point corresponding to a range return. Given n packets, the mean packet error is $\frac{1}{n}\sum_i^n l^{\text{pack}}(z_i, \hat{z}_i)$. We also compute the F1 score over packets, $\frac{2*\text{precision}*\text{recall}}{\text{precision}+\text{recall}}$, using

$$\text{precision} = \frac{\#\text{true hits}}{\#\text{true hits} + \#\text{false hits}} \quad (2)$$

$$\text{recall} = \frac{\#\text{true hits}}{\#\text{true hits} + \#\text{false misses}} \quad (3)$$

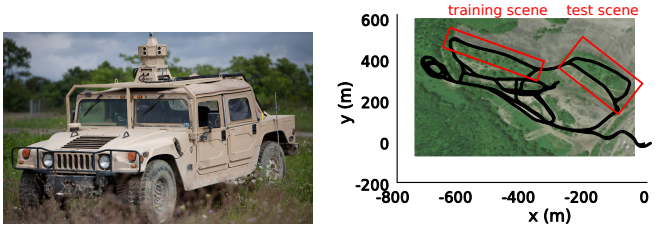
Apart from an observation-level error, we also compare the real point cloud Y with the simulated point cloud $\hat{Y}$. The asymmetric point cloud error is

$$\tilde{l}^{\text{pcd}}(Y, \hat{Y}) = \frac{1}{n'} \sum_{j=1}^{n'} \|y_{\text{nearest}(j)} - \hat{y}_j\|, \quad (4)$$

where, for each point $\hat{y}_j$ in cloud $\hat{Y}$, $y_{\text{nearest}(j)}$ is the closest point in cloud Y. Using the above, we compute the symmetric point cloud error,

$$l^{\text{pcd}}(Y, \hat{Y}, \theta) = \frac{1}{2}(\tilde{l}^{\text{pcd}}(Y, \hat{Y}) + \tilde{l}^{\text{pcd}}(\hat{Y}, Y)). \quad (5)$$

V. Experiments



(a) Data collection platform. The Lidar is at the top of the sensor head.

(b) Data collection site. The vehicle path is marked in black. Test and training scenes are marked in red.

Fig. 6

In our experiments, data was collected using a custom ground vehicle mounted with a Velodyne-32 Lidar[5], see

[4] https://www.turbosquid.com/

[5] http://www.velodynelidar.com/hdl-32e.html

Parameter name	Selection method	Value
Modeling ellipsoid elements		
(hit count prior, miss count prior)	hand	$(0, 1)$
clusters per point	optimization	0.042
maximum Mahalanobis distance for hit	optimization	3.5
Modeling triangle elements		
(hit count prior, miss count prior)	hand	$(1, 2)$
range variance	fit to data	0.07m^2
maximum residual for hit	optimization	0.75m

TABLE I: Hybrid geometric simulator parameter values

Figure 6a. Lidar packets, each consisting of hit and miss returns, were received at 20 Hz. Each packet constituted an observation z. Vehicle pose was provided by a Novatel SPAN[6] pose system. After post-processing, pose information was available at sub-centimeter position accuracy, at a frequency of 2E2 Hz. The sensor pose q_{sensor} was calculated from the vehicle pose, and the fixed relative pose of the Lidar with respect to the vehicle. Data was collected in an off-road site nearby Pittsburgh, depicted in Figure 6. Lidar packets were logged as the vehicle was driven manually, at an average speed of 0.4 m/s. Modeling and evaluation steps of the simulator were conducted offline. We selected one scene for training and one for test, as marked in Figure 6. We ran the hybrid geometric modeling steps on the training scene, resulting in a representation of the scene as a combination of surface triangles and volumetric ellipsoids. Values for simulator parameters θ are summarized in Table I. Parameters were optimized (using NLopt [26]) on a 10 sec slice of the training scene. The objective was minimization of the mean packet error+$2(1 - \text{F1 score})$. Parameters for certain other steps, such as filtering points to remove outliers, are not listed due to space constraints.

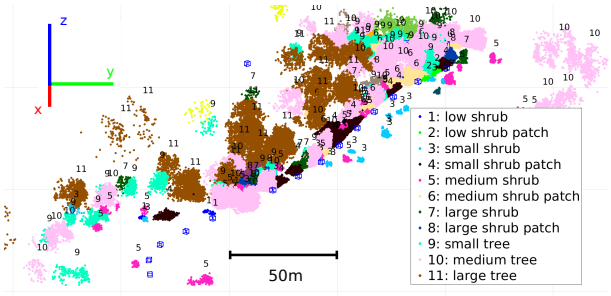


Fig. 7: Segmentation and labeling for the training scene. The test scene is processed in the same way.

The semantic segmentation of the training scene is shown in Figure 7, and example primitives obtained for each class are shown in Figure 8. Clearer visualizations of the semantic segmentation can be found in the accompanying video. The test scene, which was longer than the training scene, was then constructed in simulation, see Figure 9a. Lidar data from the test scene was used to obtain the annotation, but the simulated objects were constructed entirely from the training

[6]https://www.novatel.com/products/span-gnss-inertial-systems/

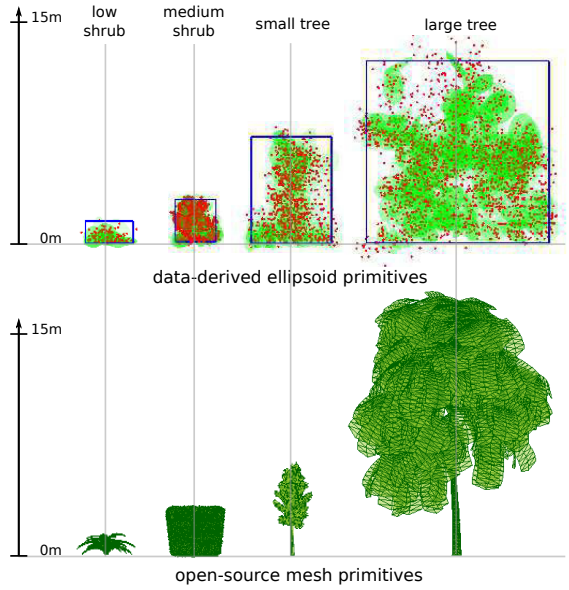
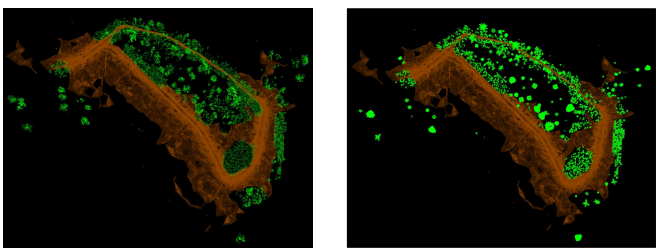


Fig. 8: Examples from the primitive set. The top figure shows primitives obtained from the training data. The bottom figure shows primitives obtained from freely available mesh models.

scene primitives. A simulated scene was also constructed using the baseline simulator, see Figure 9b. More visualizations of the simulated test scenes are in the accompanying video. From the data collection step, we obtained the test dataset of sensor poses and observations, $D_{\text{test}} = \{(q_{\text{sensor},i}, z_i)\}$. Given the dataset, we could directly model the test scene for simulation. Our aim, however, was to evaluate simulation of complex, new scenes using primitives from training scenes. By querying the simulated scenes at the sensor poses in the test dataset, we obtained the simulated sensor observations, $\{(q_{\text{sensor},i}, \hat{z}_i)\}$. Evaluation results of simulation are summarized in Table II. Our simulation approach, based on data-driven terrain primitives, is quantitatively better than the baseline, based on open-source mesh primitives.

Simulator	Point cloud error	Mean packet error	Precision	Recall	F1
Our approach	0.48m	2.9m	0.57	0.85	0.68
Baseline	0.55m	3.11m	0.58	0.75	0.65

TABLE II



(a) Simulated objects with the hybrid geometric elements.

(b) Simulated objects with the mesh model simulator.

Fig. 9: Simulated test scenes.

There are also qualitative ways in which our approach is better. For example, compared to the mesh model primitives, the data-driven primitives better approximate the shape of vegetation. Lidar returns from trees at the off-road site tended to be distributed over the entire volume. This is reflected in the data-driven tree primitives, see Figure 8. The mesh model trees, on the other hand, often have a well-defined stem. Other qualitative instances, with explanations, are shown in Figure 10. On the other hand, a failure mode of our approach was that the clusters were not fine-grained enough. This led to large ellipsoids in some cases, with corresponding Gaussian distributions of high variance. Intersection with these ellipsoids, in turn, resulted in false hits. This is why the precision of our approach is lower than that of the baseline, see Table II. This failure mode suggests that we include the FLANN clustering parameters in the simulator optimization, a step not currently performed.

VI. Conclusion

For the purpose of Lidar simulation, we have shown that data-driven object models derived from our approach generalize from a training to a new test scene better than open-source mesh models. Our approach can also benefit efforts to create point cloud maps of off-road geographical sites. A detailed data log can be gathered in a representative section, and then extrapolated to other sections. To conclude, we mention steps which we believe will further improve simulation. First, we used a fixed clustering density to obtain ellipsoids. A different density for each class might be more appropriate. Second, we are working on automating scene generation, using well-studied tools for semantic segmentation of point cloud data [27], [28]. In this work, segmentation and labeling was manual, which serves as ground truth for future work.

One could conceive of entirely different approaches to Lidar scene simulation. For example, we could divide the full scene into voxels, and use graphical models for learning and inference, as in [3]. Or we could use deep learning methods, given their success in learning to generate objects, as in [29]. While these directions may have merit, we believe that the most valuable next step is software-level simulator evaluation. In this work, we evaluated simulation at the level of sensor observations. However, simulation is closely tied to software developed on it. We are working on a general method to quantify the utility of simulation for software development, agnostic to the simulation approach.

Acknowledgement

Abhijeet Tallavajhula was supported by the National Science Foundation under Grant Number 1317803.

References

[1] B. Browning, J.-E. Deschaud, D. Prasser, and P. Rander, "3D Mapping for high-fidelity unmanned ground vehicle lidar simulation," *The International Journal of Robotics Research*, vol. 31, no. 12, pp. 1349–1376, 2012.

[2] D. Maturana and S. Scherer, "3d convolutional neural networks for landing zone detection from lidar," in *Robotics and Automation (ICRA), 2015 IEEE International Conference on*. IEEE, 2015, pp. 3471–3478.

[3] C. Wellington, A. Courville, and A. Stentz, "A generative model of terrain for autonomous navigation in vegetation," *The International Journal of Robotics Research*, vol. 25, no. 12, pp. 1287–1304, 2006.

[4] A. Kelly, N. Chan, H. Herman, D. Huber, R. Meyers, P. Rander, R. Warner, J. Ziglar, and E. Capstick, "Real-time photorealistic virtualized reality interface for remote mobile robot control," *The International Journal of Robotics Research*, vol. 30, no. 3, pp. 384–404, 2011.

[5] N. Koenig and A. Howard, "Design and Use Paradigms for Gazebo, An Open-Source Multi-Robot Simulator," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sendai, Japan, Sep 2004, pp. 2149–2154.

[6] S. Carpin, M. Lewis, J. Wang, S. Balakirsky, and C. Scrapper, "USARSim: a robot simulator for research and education," in *Robotics and Automation, 2007 IEEE International Conference on*. IEEE, 2007, pp. 1400–1405.

[7] B. Gerkey, R. T. Vaughan, and A. Howard, "The player/stage project: Tools for multi-robot and distributed sensor systems," in *Proceedings of the 11th international conference on advanced robotics*, vol. 1, 2003, pp. 317–323.

[8] D. Anderson, H. Herman, and A. Kelly, "Experimental characterization of commercial flash ladar devices," in *International Conference of Sensing and Technology*, vol. 2, 2005.

[9] J. Tuley, N. Vandapel, and M. Hebert, "Analysis and Removal of artifacts in 3-D LADAR Data," in *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*. IEEE, 2005, pp. 2203–2210.

[10] C. Goodin, R. Kala, A. Carrrillo, and L. Y. Liu, "Sensor modeling for the virtual autonomous navigation environment," in *Sensors, 2009 IEEE*. IEEE, 2009, pp. 1588–1592.

[11] C. Goodin, P. J. Durst, B. Gates, C. Cummins, and J. Priddy, "High fidelity sensor simulations for the virtual autonomous navigation environment," in *International Conference on Simulation, Modeling, and Programming for Autonomous Robots*. Springer, 2010, pp. 75–86.

[12] O. Deussen, P. Hanrahan, B. Lintermann, R. Měch, M. Pharr, and P. Prusinkiewicz, "Realistic modeling and rendering of plant ecosystems," in *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*. ACM, 1998, pp. 275–286.

[13] Y. Livny, F. Yan, M. Olson, B. Chen, H. Zhang, and J. El-Sana, "Automatic reconstruction of tree skeletal structures from point clouds," *ACM Transactions on Graphics (TOG)*, vol. 29, no. 6, p. 151, 2010.

[14] J. Binney and G. S. Sukhatme, "3D tree reconstruction from laser range data," in *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*. IEEE, 2009, pp. 1321–1326.

[15] J. Skinner, S. Garg, N. Sünderhauf, P. Corke, B. Upcroft, and M. Milford, "High-fidelity simulation for evaluating robotic vision performance," in *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*. IEEE, 2016, pp. 2737–2744.

[16] M. Mueller, N. Smith, and B. Ghanem, "A benchmark and simulator for uav tracking," in *European Conference on Computer Vision*. Springer, 2016, pp. 445–461.

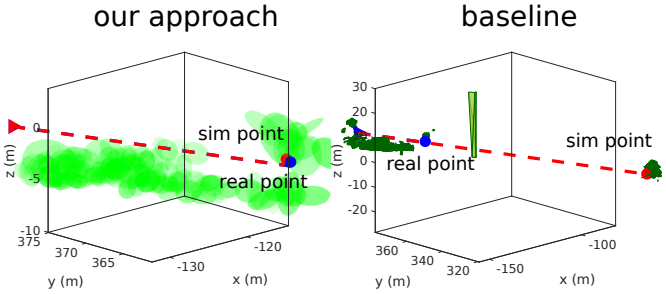
[17] X. Peng, B. Sun, K. Ali, and K. Saenko, "Learning deep object detectors from 3D models," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 1278–1286.

[18] G. Ros, L. Sellart, J. Materzynska, D. Vazquez, and A. M. Lopez, "The synthia dataset: A large collection of synthetic images for semantic segmentation of urban scenes," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 3234–3243.

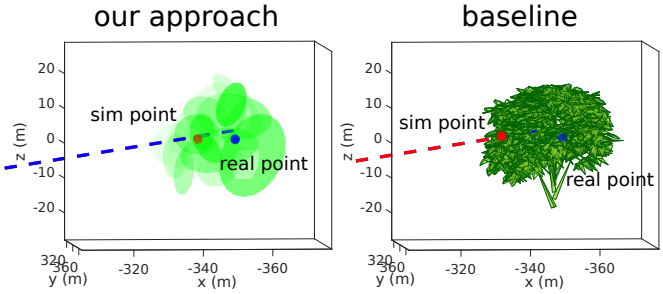
[19] A. Gaidon, Q. Wang, Y. Cabon, and E. Vig, "Virtual worlds as proxy for multi-object tracking analysis," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 4340–4349.

[20] A. Handa, V. Pătrăucean, S. Stent, and R. Cipolla, "Scenenet: An annotated model generator for indoor scene understanding," in *Robotics and Automation (ICRA), 2016 IEEE International Conference on*. IEEE, 2016, pp. 5737–5743.

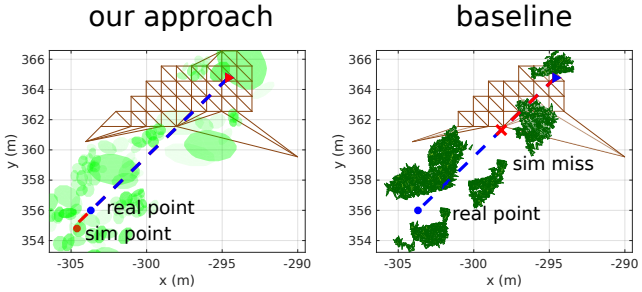
[21] A. Handa, R. A. Newcombe, A. Angeli, and A. J. Davison, "Real-time camera tracking: When is high frame-rate best?" in *European Conference on Computer Vision*. Springer, 2012, pp. 222–235.



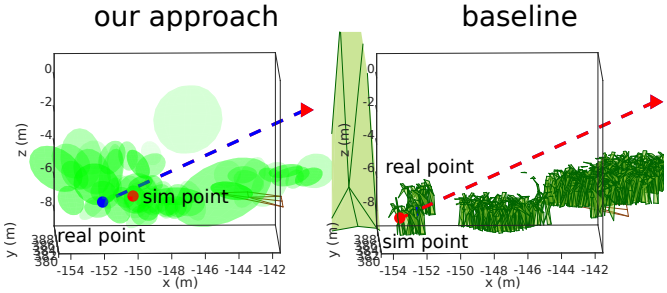
(a) This example illustrates the benefit of the volumetric ellipsoids of our approach, over the surface triangles of the baseline. The ray origin is on the left. In our approach, the ray intersects ellipsoids, and the simulated point is close to the real point. In the baseline, the ray misses the mesh triangles close to the real point. The simulated point instead is much farther down the ray, where it intersects another mesh. Note the difference in length scale in the two figures.



(b) This example illustrates the benefit of using permeable ellipsoids over opaque surface triangles. The ray origin is on the left, and the ray strikes a tree. The real point is in the interior of the tree. The simulated point from our approach is also in the interior, as the ray can pass through ellipsoids. The simulated point from the baseline, however, terminates at the surface of the tree. The baseline tree is visually realistic, but the hybrid geometric tree is sensor-realistic.



(c) This example illustrates another benefit of data-driven primitives over open-source primitives. The ray origin is at the top right. A number of shrubs are in the path of the ray. The shape of data-driven primitives better match the scene objects, compared to the baseline primitives. In our approach, the simulated point is close to the real point. In the baseline, however, the simulation result is a false miss, depicted as a red cross.



(d) This example illustrates a weakness of our approach. The ray origin is at the top right. Low shrubs are in the path of the ray. The clustering in this case is coarse relative to the size of the shrubs, visible as some large ellipsoids. This causes the simulated point in our approach to have large variance. This problem is not present in the baseline, however, and the simulated point is close to the real point. The real point is behind meshes, and therefore not in view.

Fig. 10: Examples of Lidar simulations from the test scene. Results from our approach and the baseline are shown. The real ray is depicted as a dashed blue line, and the simulated ray as a dashed red line. The real observation is a blue point, and the simulated observation is a red point. Simulated scene objects in the neighborhood of the rays are also shown. In our simulator, these are green ellipsoids. In the baseline, these are green meshes. Ground triangles, shown in brown, are common to both simulators.

[22] "ALGLIB (www.alglib.net), Sergey Bochkanov." [Online]. Available: www.alglib.net

[23] The CGAL Project, *CGAL User and Reference Manual*, 4.9 ed. CGAL Editorial Board, 2016. [Online]. Available: http://doc.cgal.org/4.9/Manual/packages.html

[24] M. Muja and D. G. Lowe, "Fast approximate nearest neighbors with automatic algorithm configuration," in *International Conference on Computer Vision Theory and Application VISSAPP'09)*. INSTICC Press, 2009, pp. 331–340.

[25] "CloudCompare (version 2.9.alpha) [GPL software]. (2017). Retrieved from http://www.cloudcompare.org/."

[26] "Steven G. Johnson, The NLopt nonlinear-optimization package." [Online]. Available: http://ab-initio.mit.edu/nlopt

[27] J.-F. Lalonde, N. Vandapel, D. F. Huber, and M. Hebert, "Natural terrain classification using three-dimensional ladar data for ground robot mobility," *Journal of field robotics*, vol. 23, no. 10, pp. 839–861, 2006.

[28] D. Munoz, N. Vandapel, and M. Hebert, "Onboard contextual classification of 3-d point clouds with learned high-order markov random fields," in *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*. IEEE, 2009.

[29] A. Dosovitskiy, J. Tobias Springenberg, and T. Brox, "Learning to generate chairs with convolutional neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 1538–1546.