Learning to Simulate Realistic LiDARs

Benoît Guillard[1*], Sai Vemprala[2], Jayesh K. Gupta[2],
Ondrej Miksik[3], Vibhav Vineet[2], Pascal Fua[1], Ashish Kapoor[2]

***Abstract*— Simulating realistic sensors is a challenging part in data generation for autonomous systems, often involving carefully handcrafted sensor design, scene properties, and physics modeling. To alleviate this, we introduce a pipeline for data-driven simulation of a realistic LiDAR sensor. We propose a model that learns a mapping between RGB images and corresponding LiDAR features such as raydrop or per-point intensities directly from real datasets. We show that our model can learn to encode realistic effects such as dropped points on transparent surfaces or high intensity returns on reflective materials. When applied to naively raycasted point clouds provided by off-the-shelf simulator software, our model enhances the data by predicting intensities and removing points based on the scene's appearance to match a real LiDAR sensor. We use our technique to learn models of two distinct LiDAR sensors and use them to improve simulated LiDAR data accordingly. Through a sample task of vehicle segmentation, we show that enhancing simulated point clouds with our technique improves downstream task performance.**

I. Introduction

Simulation plays a major role today in the development of safe and robust autonomous systems. Compared to real world data collection and testing, which can be expensive and time consuming, simulation makes it possible to gather massive amounts of labeled data and environmental interactions easily. In the age of deep learning, it paves the way towards training models for autonomous systems by allowing generation of massive amounts of training data.

Simulators have become increasingly good at rendering color imagery; e.g. rasterization-based as well as advanced physics-based rendering methods deliver realistic optical properties. While simulators are thus able to synthesize images with high visual fidelity, when it comes to non-visual classes of sensors, simulations often rely on simplified models. This is a major limitation because today's autonomous systems rely on a multitude of sensors - and specifically those like LiDAR lie at the heart of a majority of them [1].

Accurate LiDAR modeling is challenging due to the dependence of the output on complex properties such as material reflectance, ray incidence angle, and distance. For example, when laser rays encounter glass objects, the returns are rarely detected. Basic LiDAR models that exist as part of robotics simulators [2], [3] often yield simple point clouds obtained by raycasting (as seen in the bottom row, middle column of Fig. 1) which do not account for such complex yet useful properties for training LiDAR based downstream tasks and closing the sim-to-real gap. Simulators such as CARLA [4] do offer simulated raydrop, but this involves merely dropping points from a LiDAR point cloud at random without any modeling of material properties or physical interactions. It is sometimes possible to handcraft more advanced sensor models, but it involves significant human effort to study individual sensor characteristics and to accurately capture environmental interactions. While the idea of simulating realistic LiDAR observations from real data has been investigated by Manivasagam *et al.* [5], it requires replicating large world scenes in simulation and is hard to adapt to new sensors and scenarios or bootstrap from existing data.

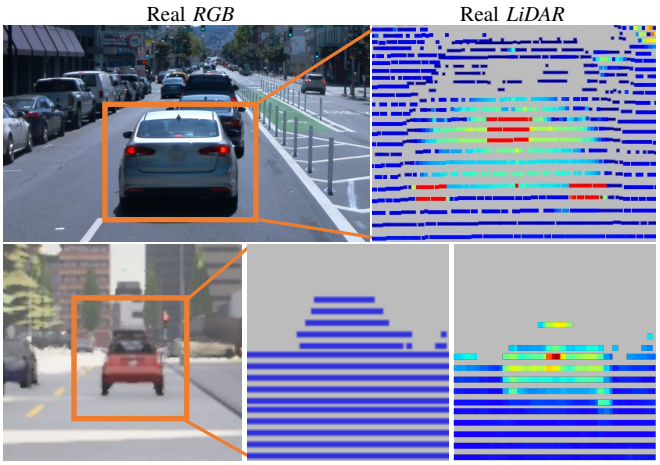
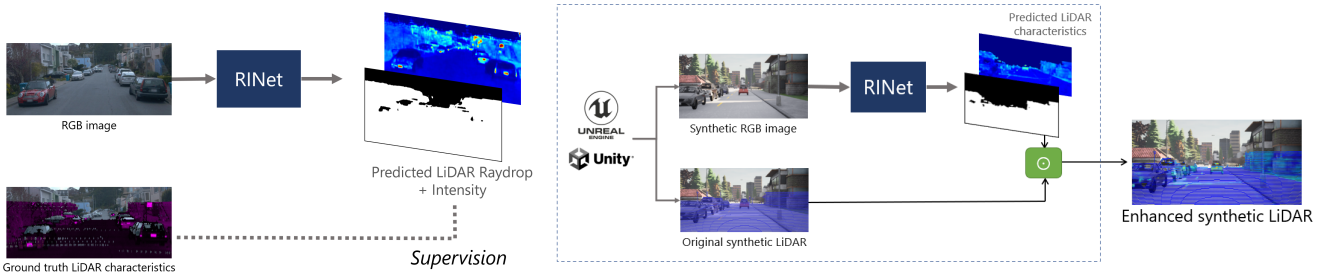


Fig. 1. **Real and synthetic LiDAR data. Top:** real data sample from the Waymo Perception dataset: RGB (left) and projected LiDAR colored by intensity (right). Raydrop is observed, i.e., returns are not recorded due to interactions with some surfaces such as glass. Similarly, intensity is often dependent on the type of material each ray interacts with. **Bottom:** Synthetic RGB (left) and projected LiDAR (middle) data from CARLA's simulator. The latter does not display realistic properties. Through our method, we enhance basic LiDAR simulations to show characteristics such as raydrop on glass surfaces and high intensity returns on license plates (right).

In this paper, we introduce a data-driven pipeline for simulating realistic LiDAR sensors in an attempt to close the gap between real life and simulation. Specifically, we propose a model that can learn to drop rays where no return is expected and predict returned intensities from RGB appearance alone. To avoid the complexity of data acquisition and of high-fidelity physics-based modeling, we propose to use real data and leverage the shared statistical structure between RGB imagery and LiDAR point clouds to learn these characteristics.

Thus, as presented in Fig. 2, given an RGB image as input,

*Work done during a Microsoft Research internship
[1]École polytechnique fédérale de Lausanne, Lausanne, Switzerland benoit.guillard@epfl.ch
[2]Microsoft Research, Redmond, USA
[3]Microsoft Mixed Reality & AI Lab, Zurich, Switzerland



(a) Training Pipeline

(b) Simulation Pipeline

Fig. 2. **Proposed pipeline (a)** In the first phase of our pipeline, we learn to predict realistic LiDAR characteristics - specifically, a binary raydrop mask and intensities directly from an RGB image, using an RINet (*R*aydrop and *I*ntensity *Net*work). **(b)** We use this trained RINet to enhance synthetic LiDAR data by adding intensity values and removing points based on the appearance information.

our model outputs 1) a binary mask that specifies where rays will be dropped and 2) predicted LiDAR intensities over the scene. We call this model a *R*aydrop and *I*ntensity *Net*work (RINet). These outputs are then used to enhance a base point cloud, as displayed in Fig. 1(bottom right). Unlike other approaches to LiDAR sensor simulation [5], ours does not require any asset creation or replication of large real world scenes in simulation. Instead, we can train our model directly on existing real datasets, and predict both raydrop and intensities using a single model. To achieve this, we suggest a novel data representation for LiDAR as an improvement over conventional range images. Our representation densifies LiDAR points by aligning them with a corresponding RGB image space, allowing models to learn both ray drop and intensity from RGB data, whereas with range images, dropped points are already missing from the input. As raydrop can originate from several sources, we propose an objective function that is designed to isolate material-specific ray drop from random sensor noise, thus letting RINet focus on learning physical properties of surfaces from the appearance. We use our approach to enhance LiDAR point clouds obtained from the CARLA simulator [4], and show that the resulting point clouds yield better downstream task accuracy than those obtained from today's simulators.

We summarize our key contributions below.

1) We present a LiDAR simulation pipeline for generating realistic LiDAR data using a model that learns LiDAR properties such as raydrop and intensities directly from RGB images.
2) We learn to simulate the properties of two distinct LiDAR sensors, using the Waymo dataset [6] and the SemanticKITTI dataset [7]. We then apply these models to LiDAR point clouds generated by the CARLA simulator [4].
3) We use these enhanced point clouds for a downstream task of vehicle segmentation from LiDAR data, and show that our pipeline generates data that results in higher Intersection over Union (IoU) compared to existing methods.

II. Related Work

A. LiDAR sensor principles

The typical operating principle of a LiDAR involves scanning its field of view using one or more laser beams, and recording the returned signals through a photodetector. By processing the returned signal for each beam, LiDARs register the distance of the surface on which it was reflected, as well as an intensity value. The latter is a function of the type of material the beam interacted with, as well as the distance and the incidence angle. Furthermore, not all beams cast by the LiDAR are returned, a phenomenon called 'raydrop'. Raydrop can be of two kinds: random raydrop, which is an artifact of inherent sensor noise, and physical raydrop, which is due to the beams being cast on surfaces such as glass which do not generate returns.

Since LiDARs probe their environment with laser beams at discrete azimuth and elevation angles, a convenient representation of LiDAR data consists of using a 2D polar grid, with each row corresponding to an elevation value and each column to an azimuth. On such representations known as *range images*, each pixel has 2 channels: depth (or distance), and intensity. Rays that were dropped or out of range simply yield 0 values at the corresponding locations. An example from the Waymo Perception dataset [6] is shown in Fig. 3**(a)**.

B. Simulating LiDAR sensors

Simulating LiDAR sensors is possible through several simulation software geared towards autonomous systems such as CARLA [4], AirSim [2], or Gazebo [3]. Often built over game engines such as Unity or Unreal Engine, such simulations offer basic functionality for raycasting that results in simple, dense point clouds with no intensity data.

Prior research has also examined the idea of data-driven methods for LiDAR simulation. LIDARSim [5] involves a complex asset creation pipeline to gather digital twins of real objects that can then be composed into virtual scenes for simulations. It relies on aggregating multiple LiDAR captures and on meshing the environment to construct a digital copy of the world from which LiDAR data can be re-rendered. A neural network is trained to simulate raydrop on this digital copy of the world, but no intensity is generated.

The resulting simulator only works for synthesizing LiDAR data and no other modalities; and only with objects that were captured by the original sensor and does not generalize to unseen objects. Two other closely related approached by Fang *et al.* [8], [9] follow similar principles for LiDAR simulators, by placing captured or synthetic objects in virtual scenes and re-rendering them.

Nakashima *et al.* [10] and Caccia *et al.* [11] use the Generative Adversarial Network (GAN) learning framework to generate range images. The resulting networks account for raydrop but not for intensity, and provide no explicit control over the content of the newly sampled range images.

Vacek *et al.* [12] train a Convolutional Neural Network (CNN) in a fully supervised manner on range images to learn to predict the intensity channel. As a consequence of using real range images from which rays are already missing as inputs to their network, they cannot learn raydrop. They apply their model to raycasted point clouds from a GTA game engine to produce synthetic LiDAR data with intensity. Elmadawi *et al.* [13] follow a similar approach and use machine learning techniques on range images to regress Echo Pulse Width (EPW) – a temporal equivalent to intensity.

To close the gap between real and synthetic LiDAR data, some methods rely on deep domain adaptation using GANs [14] with cycle consistency [15]–[17]; or on neural style transfer [18]. They successfully demonstrate improved downstream task performance.

In contrast to the above methods, ours involves minimal data pre-processing, and models both raydrop and intensity with a single network, owing to a new and simple data representation. Similar to Vacek *et al.* [12], we can augment already existing simulators such as [2], [4] and hence, one can continue to access other sensor outputs and maintain full control over scenarios and semantic content of the scenes. As opposed to domain adaptation methods which need to be retrained for each source domain-target domain pairs, our goal is to learn a sensor network which is agnostic to the synthetic data being used. We concentrate on predicting realistic raydrop and intensity, and rely on traditional simulators to provide depth via standard raycasting.

C. Learning from LiDAR data

LiDAR sensors are used as a data source for deep-learning based methods for semantic segmentation [19], [20], object detection [21]–[23] or SLAM [24]–[26]. These methods are trained on annotated real datasets such as [27]–[30] and all use LiDAR intensity data except for Yang *et al.* [23]. Realistic LiDAR simulation allows training these models with larger amounts of generated data at minimal costs.

III. Method

A. Task Description

From standard autonomous system simulation software [2]–[4] one can usually get the following data:

- RGB renderings of scenes;
- Clean point clouds simulating LiDAR data, but with no intensity, and no raydrop.

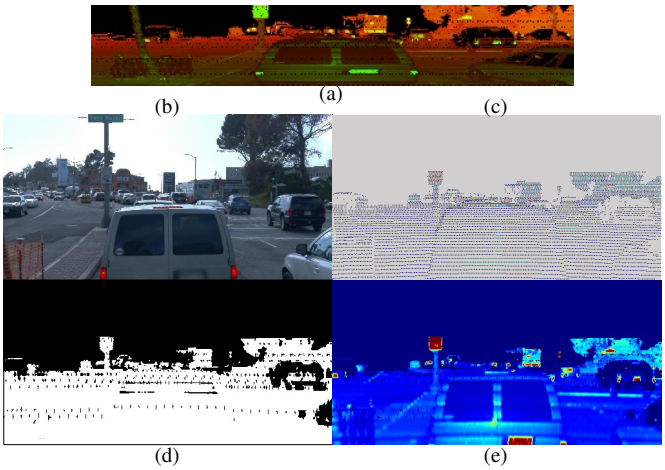


Fig. 3. **Data representations (a)** A LiDAR frame can be represented as a *range image* (red=distance, green=intensity). Points can be projected to the RGB image of the scene **(b)**, yielding a very sparse image seen in **(c)** that is ill suited for processing by CNNs. In **(d)** we propose to densify the projection by connecting neighboring triplets of range pixels, on which we interpolate intensities to get **(e)**. It yields a mask representing intensity and raydrop which is aligned with the RGB image and more suitable for CNNs.

Our goal is to enhance such synthetic LiDAR data by:

- Adding a realistic intensity channel;
- Removing points on surfaces that often don't return laser beams.

We propose achieving this in a data-driven fashion, using real world datasets which have pairs of RGB + LiDAR data and the relative poses of both sensors. Hence, given a real dataset, we train a network to predict which part of the scene would return a laser beam and the associated intensity value, based only on appearance from an RGB image. As in CARLA, sensor noise is simply emulated by randomly removing a fraction of points, uniformly sampled.

Thus, we define our main task as to learn the mapping from RGB appearance to LiDAR raydrop and intensity using a convolutional neural network (CNN). First we need to align both modalities to a common 2D spatial representation in which such a mapping can be learned, which we explain next.

B. Data representation

Previous approaches train CNNs with range images as inputs to learn LiDAR sensor characteristics such as raydrop [5] or intensity [12]. This representation however does not easily allow learning raydrop from appearance, because missing points are already absent from the input data. For example, windows on the right car in Fig. 3**(a)** do not provide any return. Learning to drop points on such surfaces (as in [5]) requires to inpaint them for creating a representation in which they are not missing. This amounts to performing 3D scene reconstruction or inpainting – pre-processing steps we wish to avoid. Moreover, the limited spatial resolution of range images (64 rows for a standard 64-channel LiDAR sensor) compared to RGB images constrains standard CNN architectures to extract coarse features only.

We therefore propose to learn typical LiDAR sensor responses in RGB camera space. Naïvely projecting LiDAR points to RGB space yields a sparse representation that is ill-suited for CNNs, as shown on Fig. 3**(c)**. Moreover, the precise locations of reprojected LiDAR points on the RGB image depend on their distance to the sensor (see Fig. 3**(c)**). A network predicting an overlay of sparse LiDAR points from RGB images is thus expected to infer the geometry of the scene. We want to avoid this, since the scene geometry can already be fully handled by the simulation software performing the raycasting operation.

For this reason, we propose to densify LiDAR points in RGB space. We do so by detecting triplets of points that provided a laser return, and that are neighbors in the range image. The camera space re-projections of all such triplets are then connected with a triangle – effectively corresponding to a meshing of LiDAR points that are neighbors on the range image. This yields a dense binary mask representing raydrop shown in Fig. 3**(d)**, in which an empty *(azimuth, elevation)* location in the range image translates into a hole. On this binary mask, we use bi-linear interpolation between projected points intensities to get a dense intensity channel, as shown in Fig. 3**(e)**. We refer to this combination of the binary mask and the intensity channel as a *dense intensity mask*.

In the following, we denote the RGB image of the scene as $I \in [0,1]^{256\times512\times3}$, and the dense intensity mask as $M \in [0,1]^{256\times512}$ where $M[u,v] = 0$ means no LiDAR point was recorded for the object shown in I near pixel coordinate (u,v). M is indeed aligned with I, which allows to easily learn correlations between RGB appearance and LiDAR sensor response. As detailed next, this data representation enables direct supervision of CNNs for predicting both raydrop and intensity, without involving a complex reconstruction pipeline.

C. Network supervision

We build a convolutional neural network model for predicting raydrop and intensities from RGB images, which we term RINet. From an RGB image I, RINet predicts a two channel array, where the channels correspond to two predictions: raydrop and intensities respectively.

The first channel $\widetilde{M_R} \in [0,1]^{256\times512}$ is used to supervise the network with an L1 loss on *R*aydrop:

$$\mathcal{L}_R = |\widetilde{M_R} - \mathbb{1}[M > 0]| \ , \tag{1}$$

where $\mathbb{1}[\cdot]$ is the element-wise indicator function.

According to Lehtinen *et al.* [31], applying an L1 loss on noisy data allows to recover a prior on median values, and enables robust training in the presence of noise. In our case, it allows to filter out the influence of random raydrop and only predict $\widetilde{M_R}[u,v] = 0$ for surfaces on which laser beams do not return to the sensor most of the time.

The second channel $\widetilde{M_I} \in [0,1]^{256\times512}$ is used to supervise the network with an L2 loss on regressed intensity values. We mask out pixels where raydrop is happening:

$$\mathcal{L}_I = \left\| \mathbb{1}[M > 0] \odot (\widetilde{M_I} - M) \right\|_2 \ , \tag{2}$$

where $\odot$ is the element-wise multiplication.

The total loss function is taken as the sum of Eqs. (1) and (2):

$$\mathcal{L} = \mathcal{L}_R + \mathcal{L}_I \ ,$$

and the final predicted mask as the product of the thresholded raydrop mask and the intensity one:

$$\widetilde{M} = \mathbb{1}[\widetilde{M_R} > 0.5] \odot \widetilde{M_I} \ . \tag{3}$$

D. Realistic LiDAR Simulation

Existing simulation engines based on Unreal Engine or Unity (e.g. CARLA [4], AirSim [2] etc.) provide a rendered RGB image of the scene as well as the LiDAR point cloud obtained via simple raycasting. RINet uses RGB images to predict realistic LiDAR phenomenon such as raydrop and intensities. The clean raycasted pointclouds are projected to the camera space, transformation from Eq. (3) is applied, and projected back to the pointcloud space. As desired, these points can then be assembled into range images. Thus, Fig. 2**(b)** describes how, with minimal invasion, we are able to leverage best parts of existing simulated data generation pipelines to enhance realism of LiDAR sensor simulations.

IV. Results and Discussion

In the following, we first present the implementation details of our pipeline such as datasets used and the training details. In Sec. IV-B we qualitatively show the ability of RINet to make relevant predictions of realistic LIDAR characteristics on synthetic data for which no ground truth is available. Sec. IV-C quantitatively validates that synthetic data enhanced by our method can be used to train neural networks on the downstream task of LiDAR semantic segmentation. For training the LiDAR segmentation networks, we also experiment with adding various amounts of real labels to replicate annotation scarcity, and examine the advantages of using synthetic data in the low-data regime. Finally in Sec. IV-D we assess the importance of LiDAR intensities for the downstream task, the relevance of our data representation compared to range images, and discuss the effects of different levels of random noise.

A. Implementation

We use two datasets for training the RINet model, each resulting in a distinct LiDAR model: the Waymo Perception dataset [6] and SemanticKITTI [7]. Both datasets provide sequences of paired RGB and LiDAR frames captured from a car driven in urban environments. We list some relevant details of the datasets in Tab. I. Training the RINet only requires temporally synchronized RGB and LiDAR frames. In our implementation based on code from Zhu *et al.* [32], RINet uses fully convolutional residual blocks [33] and instance normalization layers [34]. We use the *Adam* optimizer [35] and train for 30 epochs, starting from a learning rate of $2e{-}2$ and linearly decreasing it to 0 during the last 10 epochs.

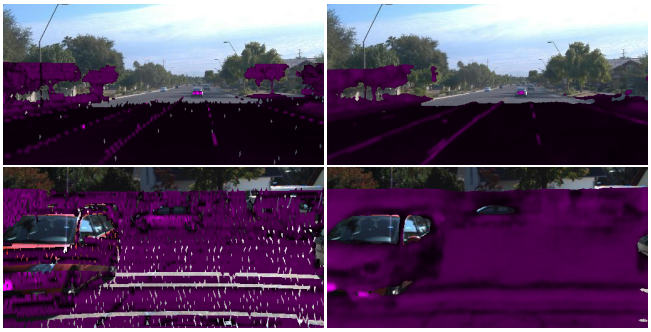


Fig. 4. **Learning to predict intensity masks** The left column shows real intensity masks from Waymo (top) and SemanticKITTI (bottom) test sets overlaid on the RGB picture. The right column shows the corresponding predictions by RINet.

TABLE I

CHARACTERISTICS OF DATASETS USED FOR TRAINING RINET

	LiDAR type	# of channels	VFoV (deg)	Max. range (m)	Training set size	Test set size
Waymo	Proprietary	64	50	75	158081	39987
SemanticKITTI	Velodyne HDL-64E	64	27	80	20409	2702

The trained RINet models are applied on top of point clouds from CARLA using the pipeline described in Section III-D to provide LiDAR training data for downstream tasks. We generate 34k pairs of RGB and LiDAR frames for Waymo's sensors' configuration, and 40k for SemanticKITTI, corresponding to between 4-5k frames for 7 or 8 different CARLA environments. We note here that CARLA allows setting parameters such as sensor pose and field of view, as desired by the user. Thus, to minimize the domain gap between real and synthetic data, when generating synthetic data corresponding to each dataset's LiDAR, we adjust the camera and LiDAR's positions and fields of view in CARLA to approximately match their real counterparts' settings.

B. Simulating multiple LiDAR sensors

Given that our RINet model can be trained to simulate different LiDAR sensors, we train one sensor simulator for each real dataset: Waymo and SemanticKITTI. As shown in Fig. 4, the simulator network picks highly reflective surfaces such as license plates, and predicts the absence of returns on transparent surfaces such as car windows. Notice that by design there is no sensor noise on predicted masks, thanks to the L1 supervision of Eq. (1). Real masks instead display randomly distributed holes.

In Fig. 5, we show the qualitative results of the effect of RINet's transformation of original synthetic LiDAR data based on the corresponding RGB input. Despite the domain gap introduced in input RGB images, the main features of LiDAR point clouds are preserved. Following CARLA [4], we additionally corrupt these range images by randomly masking 45% of the points to simulate sensor noise.

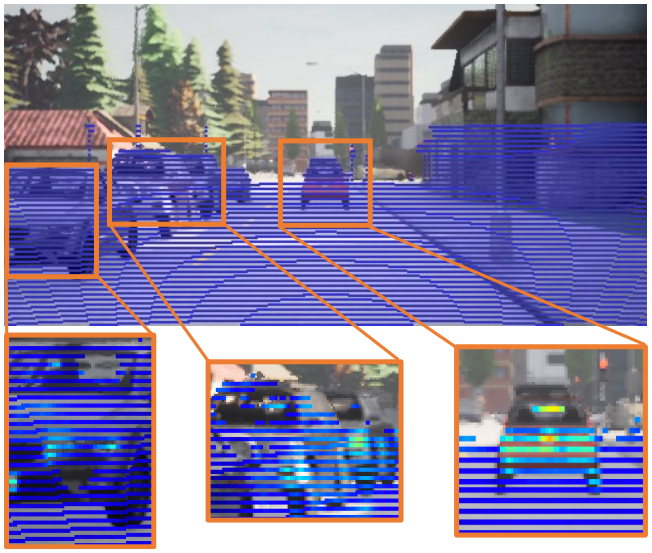


Fig. 5. **Enhancing synthetic LiDAR point clouds** CARLA pointclouds (top) have a correct geometry, but no intensity value and no raydrop. Using our method (bottom insets) we can provide the intensity information (high on car lights and plates) and remove points on surfaces that usually do not return laser beams (car windows).

C. Relevance for downstream task

We now turn to using enhanced synthetic data generated in the manner presented above for training LiDAR segmentation networks. Labels can be obtained for free in simulated environments, thus cutting annotation costs.

We use segmentation networks on range images from Milioto *et al.* [19] with a RangeNet21 backbone. We follow the original training procedure and test them on real data. We report their vehicle IoU performance on the test sets of Waymo and SemanticKITTI which include semantic annotations of LiDAR data points. This metric is used as a proxy to quantify the similarity between synthetic LiDAR samples and real ones, and as a direct measure of their usefulness for training purposes. Note that since we train on cropped range images and with supervision for a single object class, the metrics we report are lower than on public benchmarks.

TABLE II

VEHICLE IOU COMPARISON: TRAINING WITH OUR PIPELINE AND BASELINE SYNTHETIC LIDARS, TESTING ON TWO REAL DATASETS.

	Waymo	SemanticKITTI
Vanilla	45.7%	25.9%
Noise only	53.1%	27.5%
Ours	**59.4%**	**31.3%**

1) Training on synthetic data only: For each one of the two sensors (Waymo and SemanticKITTI), we train a vehicle segmentation network on CARLA synthetic data only and test it on real range images. Synthetic range images are either enhanced by our method (*Ours*), by simply dropping random points with a probability of 0.45 (*Noise only*), or kept as clean raycasted pointclouds (*Vanilla*). Tab. II shows the superiority of our approach compared to randomly dropping points. We train vehicle segmentation networks only with

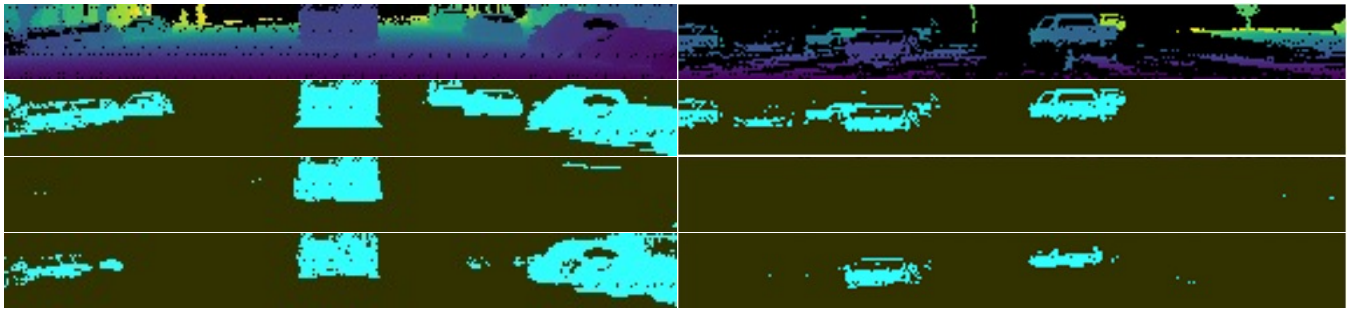


Fig. 6. **Waymo range image segmentations** From top to bottom: **(1)** input, **(2)** ground truth vehicle segmentation masks, **(3)** predictions from a network trained with synthetic data directly from CARLA, or **(4)** from our pipeline.

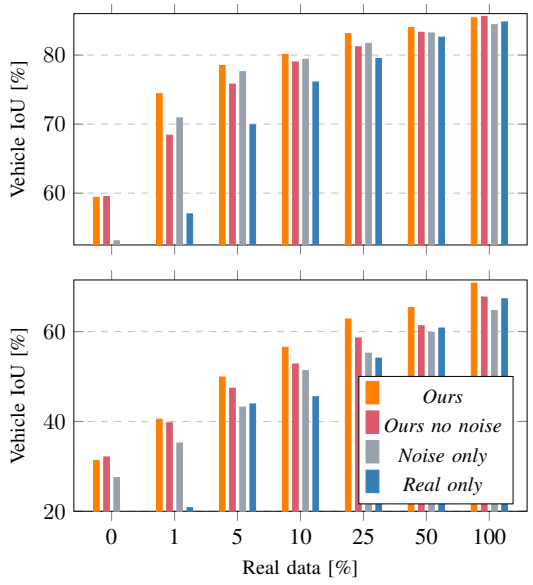


Fig. 7. Training on a mixture of synthetic and real data on Waymo (top) and SemanticKITTI (bottom), for different synthetic data enhancement pipelines.

synthetic LiDAR data and test them on 2 real datasets. Enhancing synthetic training data with our pipeline (*Ours*) improves performance compared to the default CARLA corruption method of random raydrop (*Noise only*), which itself is better than original uncorrupted data (*Vanilla*). Sample segmentation results from *Vanilla* and *Ours* are shown in Fig. 6.

2) Training on synthetic and real data: We then turn to training with a mix of synthetic and real data, for both sensors. In addition to CARLA synthetic range images, we add 1, 5, 10, 25, 50 or 100% of the real training dataset. Note that these subsets imply different ratios of synthetic vs. real data for Waymo and SemanticKITTI. Fig. 7 shows results for our full enhancement method (*Ours*), our method without random raydrop (*Ours no noise*), CARLA's default raydrop (*Noise only*) and real data only (*Real only*). Our method consistently outperforms CARLA's default corruption, with an average of $+\mathbf{2.1\%}$ **IoU** on Waymo and $+\mathbf{5.7\%}$ **IoU** on SemanticKITTI.

Our method allows for higher IoU compared to training with real data only, especially in the lower data regime.

TABLE III

EFFECT OF INTENSITY ON VEHICLE IOU, REAL DATA

Real	*Real, no intensity*
84.8%	83.4%

TABLE IV

EFFECT OF INTENSITY ON VEHICLE IOU, SYNTHETIC DATA

Ours	*Ours, no intensity*	*Ours, no intensity, no* $\mathcal{L}_I$
59.4%	55.9%	52.8%

For instance, augmenting 25% of real training data with synthetically produced range images yields higher IoUs than training solely with 50% of real data. This demonstrates that synthetically produced range images can help maintain high performance levels while reducing annotation costs and labelling efforts. Performance is also improved when using 100% of the real training data and complementing it with synthetic range images, further proving the relevance of the enhancement process.

Fig. 7 also highlights that both components of the enhancement process are useful, since networks trained with our full data pipeline (*Ours*) perform overall better than with each of its constituents. Indeed, solely using the data driven enhancement (*Ours no noise*) or the random noise (*Noise only*) separately yields lower IoUs in the majority of the cases.

D. Ablation study

1) Is intensity useful? Tab. III shows that CNNs trained on real range images benefit from having access to the intensity channel. A segmentation network trained on Waymo range images with the intensity channel (*Real*) performs better than without it (*Real, no intensity*).

Tab. IV shows that this is also the case for networks that are trained on synthetic range images enhanced by our pipeline. A segmentation network trained on synthetic range images with the intensity channel (*Ours*) performs better than without it (*Ours, no intensity*) when tested on Waymo data. IoU degrades even more when the sensor simulator is not supervised for intensity by Eq. (2) (*Ours, no intensity, no* $\mathcal{L}_I$). This shows that the intensity channel predicted by the

TABLE V

VEHICLE IOU WITH OUR REPRESENTATION VS. RANGE IMAGES

Ours, no noise	*Range image, no noise*
59.5%	53.5%

TABLE VI

EFFECT OF VARYING SYNTHETIC DATA NOISE ON VEHICLE IOU

CARLA to Waymo								
Noise probability $p =$		0	0.1	0.2	0.3	0.4	0.45	0.5
+0% real data	*Ours*	59.5	62.6	61.5	59.9	59.4	59.4	56.4
	Noise only	40.8	50.3	54.2	54.1	54.9	53.1	54.2
+1% real data	*Ours*	68.4	71.9	72.1	73.2	75.1	74.4	74.2
CARLA to SemanticKITTI								
Noise probability $p =$		0	0.1	0.2	0.3	0.4	0.45	0.5
+0% real data	*Ours*	32.1	33.7	33.0	29.4	28.8	31.3	31.8
	Noise only	25.9	29.5	31.1	30.0	29.3	27.5	28.6
+1% real data	*Ours*	39.7	40.1	41.8	39.3	40.3	40.5	39.6

LiDAR simulator is relevant and realistic. Moreover, jointly supervising the LiDAR simulator for raydrop ($\mathcal{L}_R$ in Eq. (1)) and intensity ($\mathcal{L}_I$ in Eq. (2)) seems to improve the overall plausibility of the synthetic range images, further suggesting that these two aspects are entangled.

2) Is our data representation better? To test the relevance of our upscaling of LiDAR data to camera space, we train vehicle segmentation models on two kinds of synthetic range images and test them on real Waymo data. In the first case, LiDAR data predicted by our simulator pipeline is used to supervise the downstream model. In the second case, the simulator network is a CNN trained directly on range images as in Vacek *et al.* [12], without densification. To isolate the networks' contributions to generating realistic data, we do not add random raydrop noise.

Tab. V shows that our data representation allows a simulator network to provide more useful synthetic data, yielding a higher vehicle IoU of the downstream model. We hypothesize two possible causes: the larger resolution which better preserves image features, and the ability to model physical raydrop on surfaces generating no bounces. Note that the pre-processing needed to get our representation has a very light computational overhead compared to raw range images. However, while training RINet, since the network is processing larger arrays when using our densified masks, from our experiments we estimate it results in a 60% increase in computation time compared to range images, which are smaller.

3) What is the ideal amount of noise? In the above experiments we used CARLA's default value of $p = 0.45$ for the probability of randomly dropping points on synthetic range images to simulate sensor noise. In Tab. VI we show results of the vehicle segmentation networks trained on synthetic data from *Our* pipeline and from *Noise only* corruption, with different values of p, for both Waymo and SemanticKitti.

On both datasets, training with 1% of real data in addition to *Our* synthetic sample shows a shift in the optimal value of p: $p = 0.1$ is optimal when not using real data for both Waymo and SemanticKITTI. With 1% of real data, the optimal values are 0.4 and 0.2 respectively, suggesting a sensor or dataset-specific behavior. Also note that on Waymo, *Ours* always perform better that *Noise only*, regardless of the value of p. This is only partially true on SemanticKitti, where *Ours* is better for a majority of values of p, and fairly close to *Noise only* in other cases.

Random raydrop probability is thus a hyperparameter that must be tuned depending on the application. This tuning does not involve retraining RINet, which was specifically trained to output noise-free raydrop masks.

V. CONCLUSIONS

We introduced a pipeline for simulating realistic LiDAR properties such as raydrop and intensities given monocular vision input. Specifically, we proposed a model RINet that learned to model LiDAR raydrop and predict intensities from RGB appearance alone. We also proposed a novel data representation that densifies LiDAR points in the RGB image space, allowing for simultaneous predictions of raydrop and intensity, while relying on basic simulators to provide depth information through simple point clouds. We evaluated our approach by modeling two different LiDAR sensors used in the Waymo and SemanticKITTI datasets to enhance LiDAR point clouds generated by the CARLA simulator. We showed that through our pipeline, we can model LiDAR raydrop and intensities using any existing real dataset without having to re-model real world scenarios such as in Manivasagam *et al.* [5]. Through a sample downstream task of vehicle segmentation from LiDAR data, we showed that the data enhanced through our pipeline result in greater segmentation IoUs compared to vanilla point clouds generated by current simulators.

We identify several avenues for future work. Some extensions to our current method include using 360 degree imagery (when available from real datasets) to be able to simulate properties for the entire point cloud as opposed to just the forward facing region. In parallel, we plan to study the effect of scalinp up the amount of generated synthetic data. Similarly, we also wish to investigate the effect of learning from a sequence of RGB images to better model any temporal effects that could be part of the LiDAR output; as well as more advanced ways of modeling sensor noise as opposed to a uniformly sampled one. Currently, our method only models the RGB to LiDAR mapping, and through future work, this can be extended to a more multimodal setting where other sensor data could be leveraged in case RGB data is insufficient (for instance, night time scenarios). Furthermore, we also envisage creating a single LiDAR simulation model that can encode properties from several distinct sensors, and which can be conditioned at inference time to generate a particular kind of LiDAR data. Various neural network inference optimization techniques [36], [37] could be leveraged to reduce RINet's computational expense in the simulation pipeline. Finally, since RINet is trained on real data and deployed in a simulator, its accuracy could be improved with domain adaptation training techniques or careful data augmentation.

REFERENCES

[1] E. Yurtsever, J. Lambert, A. Carballo, and K. Takeda, "A survey of autonomous driving: Common practices and emerging technologies," *IEEE Access*, vol. 8, pp. 58 443–58 469, 2020.

[2] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "Airsim: High-fidelity visual and physical simulation for autonomous vehicles," in *Field and service robotics*, Springer, 2018, pp. 621–635.

[3] 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 (IROS)*, Sendai, Japan, 2004, pp. 2149–2154.

[4] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator," in *Conference on Robot Learning (CoRL)*, 2017, pp. 1–16.

[5] S. Manivasagam *et al.*, "LiDARsim: Realistic LiDAR simulation by leveraging the real world," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 11 167–11 176.

[6] P. Sun *et al.*, "Scalability in perception for autonomous driving: Waymo open dataset," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 2446–2454.

[7] J. Behley *et al.*, "SemanticKITTI: A Dataset for Semantic Scene Understanding of LiDAR Sequences," in *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019.

[8] J. Fang *et al.*, "Augmented LiDAR simulator for autonomous driving," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 1931–1938, 2020.

[9] J. Fang *et al.*, "LiDAR-Aug: A General Rendering-based Augmentation Framework for 3D Object Detection," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021, pp. 4710–4720.

[10] K. Nakashima and R. Kurazume, "Learning to drop points for LiDAR scan synthesis," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2021, pp. 222–229.

[11] L. Caccia, H. Van Hoof, A. Courville, and J. Pineau, "Deep generative modeling of lidar data," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2019, pp. 5034–5040.

[12] P. Vacek, O. Jašek, K. Zimmermann, and T. Svoboda, "Learning to predict LiDAR intensities," *IEEE Transactions on Intelligent Transportation Systems*, pp. 1–9, 2021. DOI: 10.1109/TITS.2020.3037980.

[13] K. Elmadawi, M. Abdelrazek, M. Elsobky, H. M. Eraqi, and M. Zahran, "End-to-end sensor modeling for LiDAR Point Cloud," in *IEEE Intelligent Transportation Systems Conference (ITSC)*, IEEE, 2019, pp. 1619–1624.

[14] E. R. Corral-Soto *et al.*, "HYLDA: End-to-end Hybrid Learning Domain Adaptation for LiDAR Semantic Segmentation," in *arXiv preprint (2201.05585)*, 2022.

[15] S.-c. Mok and G.-w. Kim, "Simulated Intensity Rendering of 3D LiDAR using Generative Adversarial Network," in *IEEE International Conference on Big Data and Smart Computing (BigComp)*, IEEE, 2021, pp. 295–297.

[16] A. Barrera, J. Beltrán, C. Guindel, J. A. Iglesias, and F. García, "Cycle and Semantic Consistent Adversarial Domain Adaptation for Reducing Simulation-to-Real Domain Shift in LiDAR Bird's Eye View," in *IEEE International Intelligent Transportation Systems Conference (ITSC)*, IEEE, 2021, pp. 3081–3086.

[17] A. E. Sallab, I. Sobh, M. Zahran, and N. Essam, "LiDAR sensor modeling and data augmentation with GANs for autonomous driving," *ICML Workshop on AI for Autonomous Driving*, 2019.

[18] A. E. Sallab, I. Sobh, M. Zahran, and M. Shawky, "Unsupervised neural sensor models for synthetic lidar data augmentation," in *Machine Learning for Autonomous Driving Workshop, NeurIPS*, 2019.

[19] A. Milioto, I. Vizzo, J. Behley, and C. Stachniss, "RangeNet++: Fast and Accurate LiDAR Semantic Segmentation," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019.

[20] Y. Zhang *et al.*, "PolarNet: An improved grid representation for online LiDAR point clouds semantic segmentation," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[21] X. Chen, H. Ma, J. Wan, B. Li, and T. Xia, "Multi-view 3D object detection network for autonomous driving," in *IEEE conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[22] G. P. Meyer, J. Charland, D. Hegde, A. Laddha, and C. Vallespi-Gonzalez, "Sensor fusion for joint 3D object detection and semantic segmentation," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, 2019.

[23] Z. Yang, Y. Sun, S. Liu, and J. Jia, "3DSSD: Point-based 3D single stage object detector," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[24] X. Chen *et al.*, "SuMa++: Efficient LiDAR-based semantic SLAM," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019.

[25] X. Chen *et al.*, "OverlapNet: Loop Closing for LiDAR-based SLAM," in *Robotics: Science and Systems (RSS)*, 2020.

[26] P. Sun *et al.*, "RSN: Range sparse net for efficient, accurate LiDAR 3D object detection," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021, pp. 5725–5734.

[27] G. Pandey, J. R. McBride, and R. M. Eustice, "Ford campus vision and lidar data set," *International Journal of Robotics Research*, 2011.

[28] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[29] H. Caesar *et al.*, "nuScenes: A multimodal dataset for autonomous driving," *arXiv preprint arXiv:1903.11027*, 2019.

[30] J. Geyer *et al.*, "A2D2: Audi autonomous driving dataset," *arXiv preprint arXiv:2004.06320*, 2020.

[31] J. Lehtinen *et al.*, "Noise2Noise: Learning Image Restoration without Clean Data," in *International Conference on Machine Learning (ICML)*, 2018.

[32] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks," in *IEEE International Conference on Computer Vision (ICCV)*, 2017.

[33] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[34] D. Ulyanov, A. Vedaldi, and V. Lempitsky, "Instance normalization: The missing ingredient for fast stylization," *arXiv preprint arXiv:1607.08022*, 2016.

[35] D. P. Kingma and J. L. Ba, "Adam: A method for stochastic gradient descent," in *International Conference on Learning Representations (ICLR)*, 2015, pp. 1–15.

[36] Y. Cheng, D. Wang, P. Zhou, and T. Zhang, "A survey of model compression and acceleration for deep neural networks," *arXiv preprint arXiv:1710.09282*, 2017.

[37] T. Liang, J. Glossner, L. Wang, S. Shi, and X. Zhang, "Pruning and quantization for deep neural network acceleration: A survey," *Neurocomputing*, vol. 461, pp. 370–403, 2021.