UniSim: A Neural Closed-Loop Sensor Simulator
Supplementary Material

Ze Yang[1,2*] Yun Chen[1,2*] Jingkang Wang[1,2*] Sivabalan Manivasagam[1,2*]
Wei-Chiu Ma[1,3] Anqi Joyce Yang[1,2] Raquel Urtasun[1,2]
[1]Waabi [2]University of Toronto [3]Massachusetts Institute of Technology
{zyang, ychen, jwang, siva, weichiu, jyang, urtasun}@waabi.ai

In the supplementary material, we provide implementation details, experiment setting details, additional qualitative visualizations and metrics, and limitations of our method. We first provide details about UniSim in Sec. 1. Then in Sec. 2 we provide additional details on the baseline model implementations and how we adapt them to our sensor simulation setting. In Sec. 3 we explain in more detail our experimental setting for the experiments in the main paper. Next, we showcase additional visualizations and metrics for camera and LiDAR in Sec. 4. Finally, we analyze the limitations of our model in Sec. 5. Please refer to our project page https://waabi.ai/research/unisim/ for an overview of our methodology and video results on the various capabilities of UniSim and closed-loop autonomy simulations.

1. UniSim Details

Scene Representation Details: We begin by defining the region of interest for the static scene based on the trajectory of the self-driving vehicle (SDV). The scene extends 80 meters behind the SDV position at the first frame, and 80 meters ahead of the SDV position at the last frame. The scene width is 120 meters and the scene height is 40 meters. Please see Figure 1 for a birds-eye-view illustration of our region of interest.

Next, we generate an occupancy grid for the scene volume (see Section 3.2 in the main paper) and model it using multi-resolution feature grids. For distant regions outside the volume, we employ a background model similar to [1, 22]. For dynamic actors, each actor model is represented by an independent multi-resolution feature grids generated from a shared HyperNet. Following [10], we use a spatial hash function to map each feature grid to a fixed number of features.

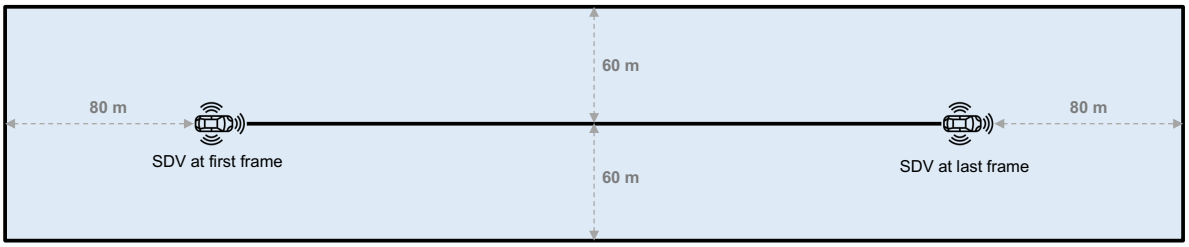

Figure 1. Region of interest of our scene representation.

Neural Feature Fields (NFFs) Details: To obtain the geometry s and feature descriptor $\mathbf{f}$ from the features grid for both the static scene and actors (Eq. 1 in main paper), we employ two networks, denoted as f_{bg} and $f_{\mathcal{A}}$. Each of these networks consists of two sub-networks. The first sub-network is an MLP that takes the interpolated feature $\{\texttt{interp}(\mathbf{x}, \mathcal{G}^l)\}_{l=1}^{L}$ and predicts the geometry s (signed distance value) and intermediate geometry feature, the second sub-network is an MLP takes the intermediate geometry feature and viewpoint encoding as input and predicts the neural feature descriptor $\mathbf{f}$. A convolutional neural network g_{rgb} is applied on top of the rendered feature map and produces the final image. We also have an additional decoder for lidar intensity (g_{int}).

*Indicates equal contribution.

Actor Model Details: Although we assume the dynamic actor tracklets are provided, they might be inaccurate, even when human-annotated, and lead to blurry results. We find it advantageous to refine the actor tracklets during training. For each dynamic actor $\mathcal{A}_i$ with trajectory initialized by a sequence of SE(3) poses $\{\mathbf{T}^t_{\mathcal{A}_i}\}_{t=1}^T$, we jointly optimize the rotation and translation components of the poses at each timestamp. We parameterize the rotation component using a 6D representation [25]. We also incorporate a symmetry prior along the longitudinal axis for vehicle objects, which are the most common actors in self-driving scenes. Specifically, we denote the query points and view direction in the canonical object coordinate (Front-Left-Up) as $\mathbf{x}_{\mathcal{A}_i}$ and $\mathbf{d}_{\mathcal{A}_i}$. During training, we randomly flip the input point and view direction when querying the neural feature fields:

$$\mathbf{x}'_{\mathcal{A}_i} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \mathbf{x}_{\mathcal{A}_i}, \quad \mathbf{d}'_{\mathcal{A}_i} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \mathbf{d}_{\mathcal{A}_i} \tag{1}$$

Actor Behavior Model for Closed-loop Simulation: For simplicity, we modelled the counterfactual actor behaviors with heuristics. While we only demonstrated human-specified trajectories in the paper, it is intuitive to to incorporate other models to control actors' behaviors, such as intelligent driver models [6, 17] or deep-learning based models [5, 16].

Run-time and Resources: Table 1 reports training/inference time for UniSim and baselines on an A5000 GPU for a scenario with 31 actors (Figure 6 row 1). We believe the run-time can be further optimized with recent advancements [3, 20] and code optimization (*e.g.*, customized CUDA kernels). We can also balance run-time and realism by adjusting the image resolution, lidar rays sent, and the number of points sampled along the ray.

	Train (Hour) ↓	Camera (FPS) ↑	LiDAR (FPS) ↑
FVS[57]	≈ 24	0.33	-
NSG[51]	≈ 20	0.06	-
Instant-NGP[46]	0.11	2.60	-
Ours	1.67	1.25	11.76

Table 1. Comparison of training speed, inference speed for camera image (1080p) and LiDAR frame ($\approx 77k$ rays per frame).

2. Implementation Details for Baselines

2.1. Free View Synthesis (FVS)

One key component of FVS is that it requires a proxy geometry to perform warping. Instead of using COLMAP to perform structure-from-motion as in the original paper, we take advantage of the fact that the data collection platform records accurate LiDAR depth information for the scene. For the static scene, we aggregate LiDAR points across all frames with dynamic points removed and then create triangle surfels for them. We first estimate per-point normals from 200 nearest neighbors, and then we downsample the points into $4cm^3$ voxels [24]. After that, triangle surfels are created with a 5cm radius. This includes static vehicles. For dynamic objects, we create surfels from aggregated point clouds using human labeled 3D bounding boxes as correspondence. To enhance the photorealism of the rendered images, we also add the same adversarial loss as in UniSim, which helps produce higher quality results.

To render a target image, a few source images need to be selected for warping. During training, we randomly select from nearby frames (10 frames before and 10 frames after). To test the interpolation during inference, we select the 2 most nearby frames. To test the lane shift, we select every 4 frames starting from 12 frames before to 4 frames after current frame. We heuristically tried multiple settings and found this setting usually achieves better balance between performance and efficiency. Fewer source images will result in images with large unseen region, and the network may fail to inpaint it. More source images may produce blurry results because of geometry misalignment and runs out-of-memory on the GPU. The FVS is trained on 2 A5000 GPUs with learning rate 0.0001, batch size 8 for $50,000$ iterations. Images are cropped to 256×256 patches during training.

2.2. Instant-NGP

Instant-NGP [10] achieves state-of-the-art NeRF rendering by introducing an efficient hash encoding, accelerated ray sampling and fully fused MLPs. In our experiments, we scale the PandaSet scenes to fully occupy the unit cube and set

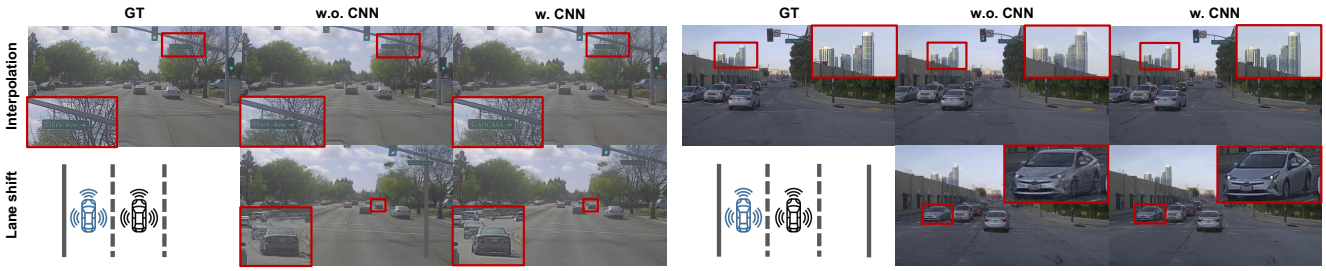

Figure 2. **Effects of CNN Decoder**. The convolutional RGB decoder helps to make the rendered image more clean and learn more details.

Figure 3. **Effects of actor shape prior**. We show the unobserved side of the vehicle after SDV lane change, incorporating actor shape prior helps to reconstruct unobserved side of the vehicle.

`aabb_scale` as 32 to handle the background regions (e.g., far-away buildings and sky) outside the unit cube. We tuned `aabb_scale` to be 1, 2, 4, 8, 16, 32, 64 and find 32 leads to the best performance. The model is trained for 20K iterations and converges on the training views. However, we find there are obvious artifacts including floating particles and large missing regions on the ground. This is due to the radiance-geometry ambiguity [22] and limited viewpoints that are sparse and far apart in the driving scenarios compared to dense indoor scenes. Therefore, to improve the geometry for better extrapolation (e.g., lane shift), we enhance the original method with LiDAR depth supervision. Specifically, we aggregate the recorded LiDAR data and create a surfel triangle representation. We follow the same surfel asset generation pipeline as stated in the implementation of FVS (see Sec. 2.1). We then use the surfel mesh representation to render a depth image at each camera training viewpont. We also tried to ignore the dynamic actors with rendered object masks but it leads to worse performance in terms of PSNR since all dynamic actors are removed. We use the official repository [1] (commit id: `ca21fd399d0eec4d85fd31ac353116bc088f3f75`, Oct 19, 2022) for our experiments.

2.3. Neural Scene Graph (NSG)

We use the codebase on GitHub[2] and adopt the default setting for the model and training parameters. For data preprocessing, only dynamic vehicles are considered as object nodes, static vehicles are considered as part of the background. Also, we only select five dynamic actors that are the most present in the log due to memory limitations. We find with a larger number of actors the model runs out-of-memory during data loading on a 128GB memory machine because our image data is higher resolution (1920×1080) than the KITTI data (1242×375). NSG only works well on front-facing scenarios with only small movements along the camera viewing direction. We tried to improve the background modeling by increasing the background points sampling and increasing model size. However, we did not see significant improvement in the visual quality, while also making the computation cost much more expensive. Therefore, we stick to the default setting.

2.4. LiDARsim

We re-implement LiDARsim based on OptiX ray tracing engine [12]. We first create the asset bank following [9]: Specifically, for background assets, we aggregate the LiDAR points across all the frames and remove all actors using 3D bounding box annotations. For dynamic actors, we aggregate the LiDAR points inside the bounding boxes in the object-centric coordinate. We then estimate per-point normals from 200 nearest neighbors with a radius of $20cm$ and orient the normals upwards

[1] https://github.com/NVlabs/instant-ngp
[2] https://github.com/princeton-computational-imaging/neural-scene-graphs

for flat ground reconstruction. We downsample the LiDAR points into 8cm voxels and create per-point triangle faces (radius $10cm$) according to the estimated normals. Note that we also track the per-point intensity values during the above post-processing. Given the asset bank, we place the background and all actors in its original locations and transform the scene to the LiDAR coordinate for ray-triangle intersections computation. For LiDAR realism experiments, we created surfel assets using training frames $(0, 2, 4, \cdots, 78)$ and performed raycasting using real LiDAR rays on held-out frames $(1, 3, 5, \cdots, 79)$. For perception evaluation and training, the assets assets are created and raycasted with all 80 frames. Instead of using real LiDAR rays which are infeasible to obtain in real-world simulation, we set the sensor intrinsics (*e.g.*, beam angle, azimuth resolution, etc) based on the public documents and raw LiDAR information [3].

3. UniSim Experiment Details

3.1. Image Similarity Metrics

Due to limited computation resources available to run NSG on all the logs in PandaSet, 10 scenes were selected for image similarity evaluation: `001,011,016,028,053,063,084,106,123,158`. These scenes include a variety of both city and suburban areas, includes one night log (`063`), and one log containing no dynamic actors (`028`). These 10 scenes are of a subset of the 24 selected for perception validation, described in the following section.

3.2. Perception Evaluation and Training

To study whether the sensor simulation is realistic with respect how autonomy perceives it, we first analyze the domain gap for perception tasks. Specifically, we leveraged the state-of-the-art camera-based brids-eye-view (BEV) detection model BEVFormer [7] and evaluate on the full PandaSet. Since there are no official train/val splits available, we split PandaSet into 79 and 24 sequences as training and validation sets by considering geographic locations, time span and day-night variety. Specifically, we select the sequences `001,011,013,016,017,028,029,032,053,054,063,065,068,072,084,086,090,103,106,110,112,115,123,158` as validation set and leave other 79 sequences as training.

We consider two setups (a) **Real2Sim**: evaluating the perception model trained on real data on simulated data; (b) **Sim2Real**: training perception models with simulated data data and testing on real data. Specifically, we evaluate the real model on 24 simulated validation logs for Real2Sim and train perception models with 79 simulated training logs for Sim2Real. For data augmentation experiments, we first use all simulation variations (log-replay, lane shift to the left for 0.5 and 2.0 meters) to train the detector (Sim). Then we combine those simulation data with real data (Sim + Real) to retrain the detector. Note that for all experiments, we train UniSim and other baselines with all image frames (*i.e.*, frames 0 - 79). As shown in Table 5 (main paper), incorporating UniSim lane shift variations help autonomy achieve superior detection performance compared to training with real data alone. More rigorous study on how to sufficiently leverage the simulated data (*e.g.*, advanced actor behavior simulation, actor insertion, camera rotations, etc) is left as future works.

BEVFormer Implementation Details: We adapt the official repository [4] for the training and evaluation on PandaSet. Specifically, we focus on single-frame vehicle detection with the front camera. We ignore the actors that are out of the camera field-of-view during training and evaluation. The models are trained in the vehicle frames with FLU convention (x: forward, y: left, z: up). The region of interest is set as $x \in [0, 80m], y \in [-40m, 40m], z \in [-2m, 6m]$. Due to memory limit, we adopt the BEVFormer-small architecture [5] with a batch size of 2 per GPU. We train the models using AdamW optimizer [8] for 20 epochs with the cosine learning rate schedule [6]. Each model takes around 12 hours to train with 2$\times$ RTX A5000. For data augmentation experiments, we generate the simulated data offline and train the models for 10 epochs as the datasets become three to four times larger compared to original PandaSet.

3.3. Open-loop Autonomy Evaluation

Evaluation Set: In main-paper Sec. 4.5, we ran open-loop autonomy metrics on PandaSet scenes. The prediction [4] and planning modules [14] require map information to operate, which PandaSet does not have. Maps were annotated for 9 scenes in PandaSet that are in the autonomy validation set. These are a subset of the 10 scenes (excludes `063`) used for quantitative

[3] https://github.com/scaleapi/pandaset-devkit/issues/67

[4] https://github.com/fundamentalvision/BEVFormer

[5] https://github.com/fundamentalvision/BEVFormer/blob/master/projects/configs/bevformer/bevformer_small.py

[6] https://pytorch.org/docs/stable/generated/torch.optim.lr_scheduler.CosineAnnealingLR.html

Figure 4. **Effects of tracklet refinment**. Tracklet refinement helps to learn sharp details for dynamic actors.

evaluation in main-paper Sec. 4.3 that also have LiDAR semantic segmentation from PandaSet to make map annotation possible.

Open-loop Autonomy Metrics: We now describe in more detail the autonomy metrics computed for each module.

For detection, we compute a metric called detection agreement. For detection agreement, we evaluate the real-trained BEVformer model on the real image and the simulated image, obtaining real-image and simulated-image detection outputs. We assign the real-image detections as the labels and compute the average precision (AP) between the simulated-image detections and the real-image detections (set as labels) to report the final result. We compute the detection agreement at intersection-over-union (IoU) 0.3. This metric measures whether the simulated sensor data generates the same true-positives and false-positives as the real data at each frame.

For prediction, we compute the prediction average displacement error, where the motion forecasted trajectory is for a 6 second horizon at 10Hz frequency. The prediction module selected for the autonomy is from Cui et. al. [4], which takes a rasterized map and 5 seconds past history tracks for each detected actor as input and outputs 6 second multi-modal trajectory horizons (number of modes = 6) for each actor. The most-likely prediction mode for each actor generated in the real image by the prediction model is set as the label. We then evaluate prediction outputs generated by simulated images by identifying matched actors between the label and predicted, and computing the minimum average displacement error (minADE) across the 6 predictions for that actor. We report minADE at the maximum F1 score for each simulation method, where a true positive is defined as a matched detected actor from the simulated sensor data and real sensor data with IoU=0.3.

For planning, we compute the plan consistency metric. At each timestep, the autonomy model outputs a 5 second plan it would execute given the prediction output and map information. We compute the final displacement error between the 5 second end point when running on real images and when running on simulated images at each timestep.

4. Additional Experiments and Analysis

4.1. Ablation Study

Convolutional RGB Decoder: The Convolutional RGB decoder improves the overall image quality by spatial relation reasoning and increases model capacity. Figure 2 shows a visual comparison on interpolation and lane shift settings. It can be seen from the figure that the convolutional RGB decoder helps to make the rendered image more clean and perserve more details.

Actor Shape Prior: We incorporate hypernetwork and symmetry prior to learn better assets. Figure 3 shows that incorporating these actor priors helps to reconstruct unobserved regions of the vehicle when SDV performs a lane shift.

Tracklet Refinement: We show a visual comparison training a model with and without tracklets optimization in Figure 4. Optimizing tracklets improves reconstruction of the dynamic actors and makes sharpens their detail.

4.2. Additional Camera Simulation Results

Realism Evaluation: Please see Figure 5 for more examples on qualitative comparisons with the baseline models. We show rendered results in both the interpolation and lane-shift test settings. We also show the comparison to baselines on actor removal and manipulation in Figure 6 left. Additionally, we visualize the rendered normal (normalized gradient of SDF s) and depth map in Figure 7.

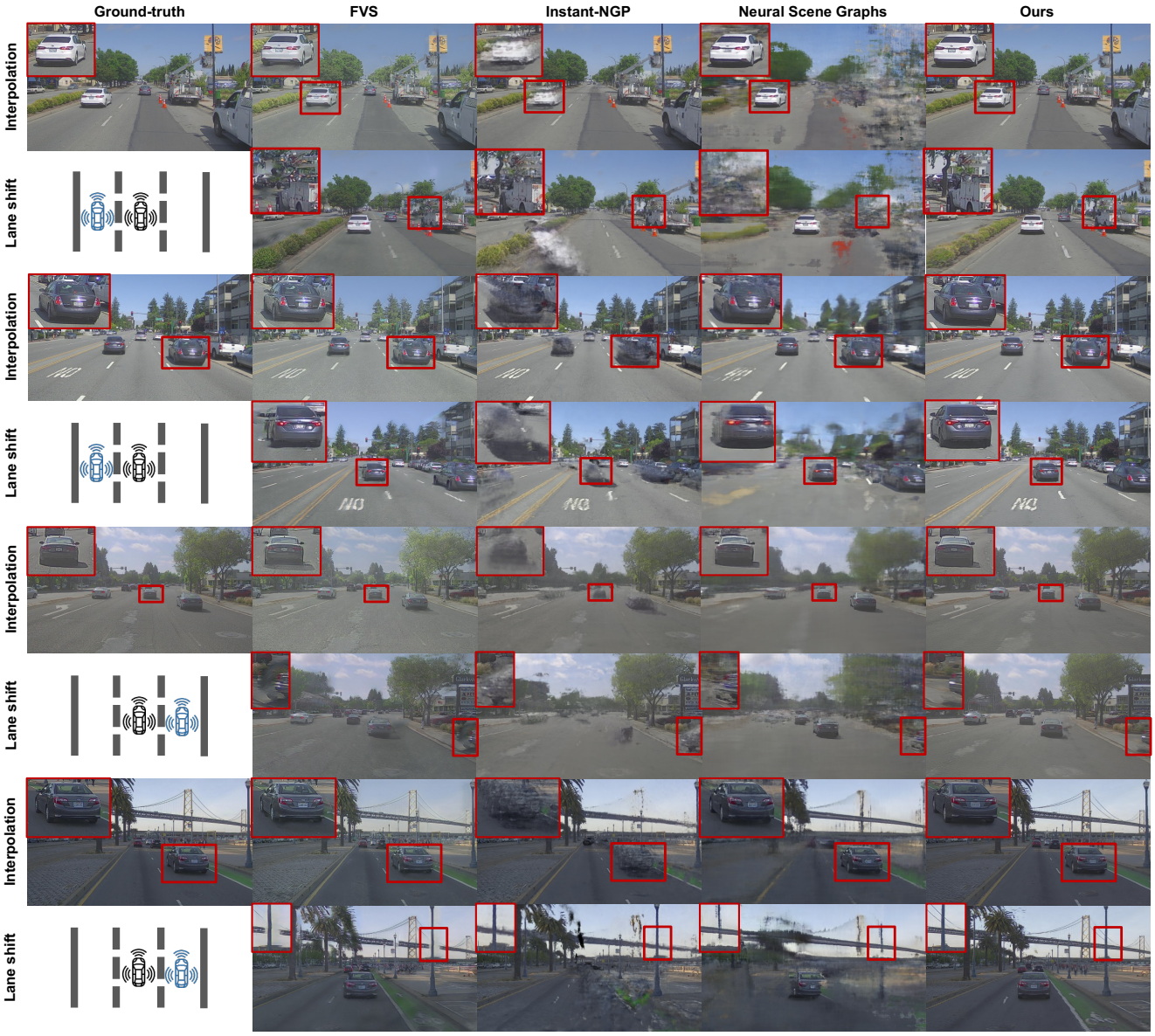

Figure 5. **Qualitative comparison on more examples.** We show rendering results in both the interpolation and lane-shift test settings.

Editing Scene Elements: Similar to dynamic actors, we can represent and edit static elements using bounding box annotations. Figure 6 right shows we copy the cones to new placements and create new scenarios.

4.3. Additional LiDAR Simulation Results

To investigate the realism of LiDAR simulation, we compare UniSim with the state-of-the-art approach LiDARsim [9] in the *log-replay* setting. Figure 8 show qualitative examples of real LiDAR scans (left), and simulated LiDAR scans using LiDARsim and UniSim. LiDARsim conducts raycasting with the aggregated surfel meshes and produces incomplete beam rings and noisy scans. This is because the surfel meshes created with the original LiDAR points can have noisy normal estimates and incomplete shape. Compared to LiDARsim, the LiDAR sweeps simulated by UniSim is more accurate and smoother. This is because our neural representations capture the underlying geometry better. Moreover, UniSim can simulate the intensity values more smoothly and accurately (See the third row in Figure 8).

Figure 6. **Left:** Comparison to baselines on actor removal (top) and manipulation (bottom). **Right:** Edit static elements by copying cones.

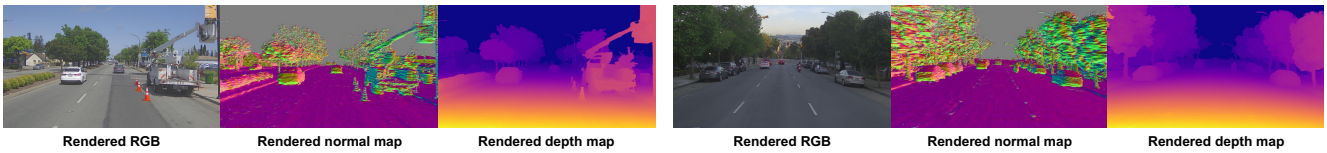

Figure 7. **Rendered Scene Geometry**. For each example, from left to right: rendered RGB image, rendered normal map, and rendered depth map.

Real2Sim (*replay*)	**mAP**	**AP@0.1**	**AP@0.3**	**AP@0.5**	**AP@0.7**
FVS	37.2	63.4	46.7	27.1	11.5
Instant-NGP	22.6	40.9	28.8	15.4	5.4
UniSim	**40.2**	**67.2**	**50.7**	**30.8**	**12.2**
Real	40.9	68.1	51.4	31.3	12.7

Table 2. Real2Sim domain gap in *replay* setting.

Real2Sim (*lane-shift*)	**mAP**	**AP@0.1**	**AP@0.3**	**AP@0.5**	**AP@0.7**
FVS	30.3	54.1	38.9	21.7	6.6
Instant-NGP	18.1	34.3	22.2	11.9	3.8
UniSim	**37.0**	**63.6**	**47.8**	**27.2**	**9.4**

Table 3. Real2sim domain gap in *lane-shift* setting.

Sim2Real (*replay*)	**mAP**	**AP@0.1**	**AP@0.3**	**AP@0.5**	**AP@0.7**
FVS	38.7	66.2	48.8	28.7	11.1
Instant-NGP	34.0	62.3	43.9	22.9	6.7
UniSim	**39.9**	**67.6**	**51.2**	**29.4**	**11.4**
Real	40.9	68.1	51.4	31.3	12.7

Table 4. Sim2Real domain gap in *replay* setting.

4.4. Additional Perception Evaluation and Training for Camera Simulation

We report detailed detection metrics for perception evaluation and training for better reference. Specifically, we report the average precision (AP) at different IoU thresholds: 0.1, 0.3, 0.5, 0.7. The mean average precision (mAP) is calculated by $\text{mAP} = (\text{AP@0.1} + \text{AP@0.3} + \text{AP@0.5} + \text{AP@0.7})/4.0$. The Real2Sim results are shown in Table 2 (replay) and Table 3

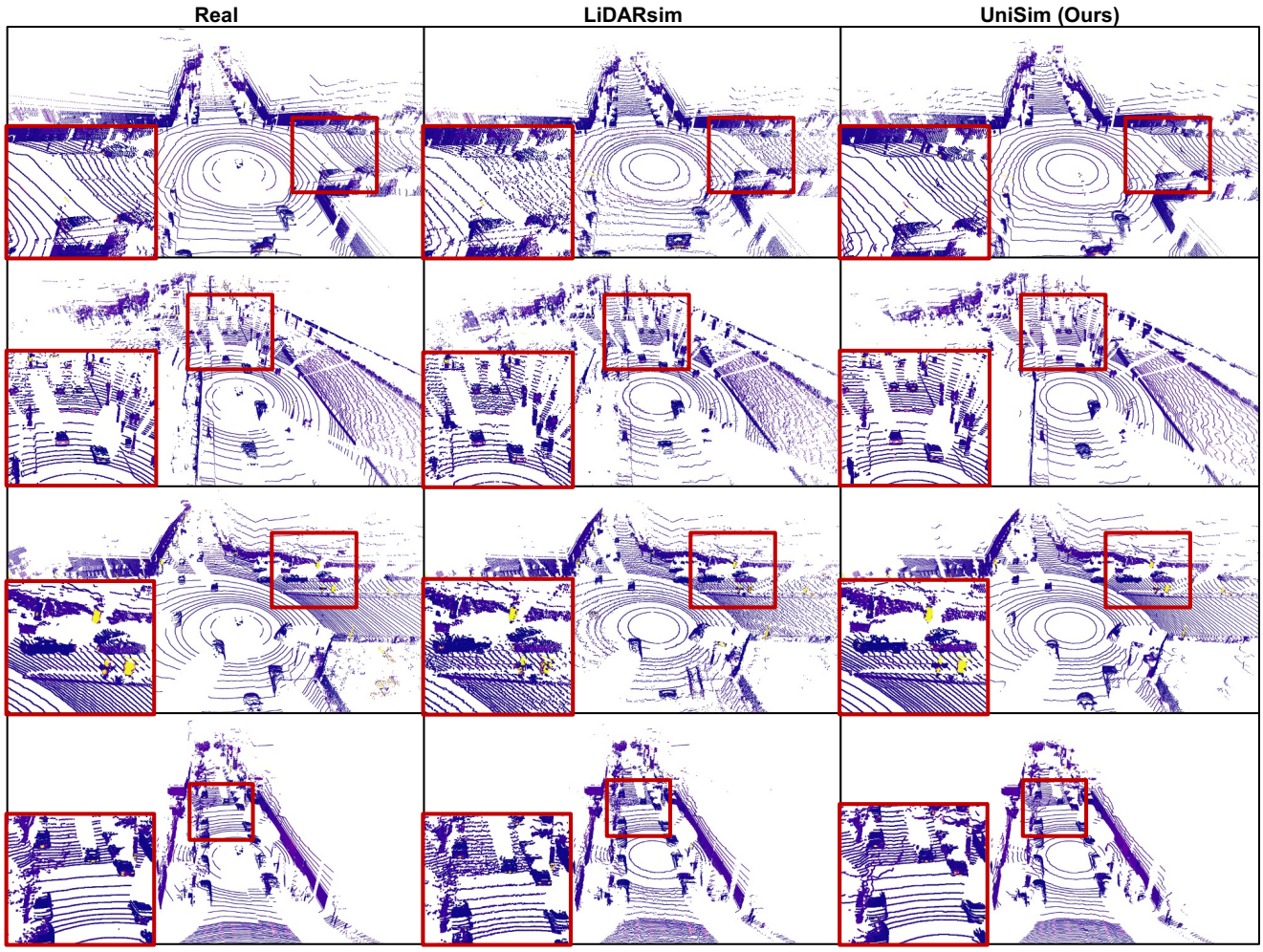

Figure 8. **Comparison of LiDAR simulation on PandaSet.** Left: real lidar sweeps; Middle: simulated point clouds using LiDARsim [9]; Right: simulated point clouds using UniSim. Compared to LiDARsim, our neural rendering approach produces higher-fidelity LiDAR simulation with less noise and more continuous beam rings that are closer to real LiDAR scans. Note that the UniSim simulation results are more realistic with respect to how the autonomy perceives it (*i.e.*, smaller Real2Sim and Sim2Real domain gap).

Figure 9. **Limitations of UniSim.** Left: Our method learns blurry results for animatable objects since we do not model the animation. Right: Our method has difficult rendering completely unseen objects after performing SDV lane change.

(lane-shift). The Sim2Real results are shown in Table 4 (replay) and Table 5 (lane-shift). The data augmentation experiments are presented in Table 6.

Sim2Real (*lane-shift*)	**mAP**	**AP@0.1**	**AP@0.3**	**AP@0.5**	**AP@0.7**
FVS	32.2	59.9	41.5	21.0	6.5
Instant-NGP	26.5	54.3	34.8	13.9	2.9
UniSim	**37.1**	**64.9**	**47.5**	**26.4**	**9.5**

Table 5. Sim2Real detection performance in *lane-shift* setting.

Data Augmentation	**mAP**	**AP@0.1**	**AP@0.3**	**AP@0.5**	**AP@0.7**
Real	40.9	68.1	51.4	31.3	12.7
Sim (FVS)	39.2	68.0	50.1	28.5	10.1
Sim (Instant-NGP)	32.4	60.7	41.1	21.3	6.6
Sim (UniSim)	**41.4**	**71.4**	**54.7**	**29.5**	**10.1**
Real + Sim (FVS)	41.1	71.6	53.6	29.6	9.6
Real + Sim (Instant-NGP)	40.1	70.1	51.5	29.0	9.9
Real + Sim (UniSim)	**42.9**	**71.9**	**54.3**	**32.7**	**12.8**

Table 6. **Augmenting real data with simulation.** Using UniSim data only to train the perception model is even better than training with all real data. UniSim augmentation yields a significant performance gain. In contrast, baseline data augmentation brings marginal gain or harms performance

Real2Sim	**mAP**	**AP@0.1**	**AP@0.3**	**AP@0.5**	**AP@0.7**	**AP@0.8**
LiDARsim	74.9	88.3	86.3	82.7	69.1	48.3
UniSim	**77.0**	**90.1**	**88.4**	**84.4**	**71.5**	**50.8**
Real	80.8	93.6	92.0	88.0	75.2	55.2

Table 7. Real2Sim detection performance for LiDAR simulation.

4.5. Perception Evaluation and Training for LiDAR Simulation

Besides camera simulation, we also analyze if the simulated LiDAR reduces the domain gap for perception tasks compared to an existing SoTA LiDAR simulator LiDARsim [9]. We leveraged the LiDAR-based birds-eye-view detection model PIXOR [19] and tested Real2Sim and Sim2Real setups. We used the same training validation splits as the camera perception model and considered the replay setting. Specifically, we use all frames to train the models (creating assets for LiDARsim) and test with our approximated sensor configurations. We focus on multi-sweep (5 frame) vehicle detection with a region of interest (ROI) is set as $x \in [0, 80m], y \in [-40m, 40m], z \in [-2m, 6m]$. We report the average precision (AP) at different IoU thresholds: 0.1, 0.3, 0.5, 0.7. 0.8. The mean average precision (mAP) is calculated by $\text{mAP} = (\text{AP@0.1} + \text{AP@0.3} + \text{AP@0.5} + \text{AP@0.7} + \text{AP@0.8})/5.0$.

As shown in Table 7 and Table 8, UniSim results in smaller domain gap in both Real2Sim and Sim2Real compared to LiDARsim especially when the IoU threshold is strict (e.g. AP@0.7, AP@0.8). This indicates our approach further bridges the gap between simulation and real world and can help better evaluate and train autonomy. We have performed two analyses on perception model performance on camera and LiDAR sensors separately with the same UniSim models generating both synthetic data. Future investigation would include performing multi-sensor perception experiments to further understand the domain gap of UniSim.

4.6. Other Explorations

We have observed that certain commonly-used techniques in novel-view-synthesis did not offer significant improvements with our architecture and data setting on our initial attempts. Specifically, we found that hierarchical or fine-grained sampling did not significantly enhance realism but resulted in a considerable decrease in processing speed, possibly due to the majority of the scene being distant from the self-driving vehicle. Additionally, optimizing exposure per frame did not improve realism as the exposure does not change within a single log snippet for a single camera. We also attempted multi-camera training, but

Sim2Real	mAP	AP@0.1	AP@0.3	AP@0.5	AP@0.7	AP@0.8
LIDARsim	75.9	91.6	**89.9**	**85.6**	70.6	42.0
UniSim	**77.9**	**92.0**	89.8	85.4	**72.0**	**50.3**
Real	80.8	93.6	92.0	88.0	75.2	55.2

Table 8. Sim2Real detection performance for LiDAR simulation.

found that several of the side cameras in PandaSet had mis-aligned calibration and slightly different exposures. Optimizing the camera poses and exposure in this setting did not offer improvements, potentially due to limited data from a single 8 second log. Additionally, more training iterations and deeper architecture did not provide significant improvements, but rather increased the complexity of the model.

5. Limitations and Future Works

UniSim has several limitations. Our neural features grid cannot render lighting variations, as the modeled radiance bakes the lighting into the representation. Future work involves explicitly modelling and manipulating scene lighting [2, 15, 23] and weather. We also do not model animation (*e.g.*, turn signals, traffic lights) or actor deformation (*e.g.* walking) and hope to incorporate ideas from works [11, 13, 18, 21] that tackle this. We also have artifacts when rendering areas that were previously outside the field-of-view. Figure 9 shows qualitative visualizations of these phenomena. We hope future work in this direction can further enhance sensor simulation realism.

References

[1] Jonathan T Barron, Ben Mildenhall, Dor Verbin, Pratul P Srinivasan, and Peter Hedman. Mip-NeRF 360: Unbounded anti-aliased neural radiance fields. In *CVPR*, 2022. 1

[2] Mark Boss, Varun Jampani, Raphael Braun, Ce Liu, Jonathan Barron, and Hendrik Lensch. Neural-PIL: Neural pre-integrated lighting for reflectance decomposition. 2021. 10

[3] Zhiqin Chen, Thomas Funkhouser, Peter Hedman, and Andrea Tagliasacchi. Mobilenerf: Exploiting the polygon rasterization pipeline for efficient neural field rendering on mobile architectures. *arXiv*, 2022. 2

[4] Henggang Cui, Vladan Radosavljevic, Fang-Chieh Chou, Tsung-Han Lin, Thi Nguyen, Tzu-Kuo Huang, Jeff Schneider, and Nemanja Djuric. Multimodal trajectory predictions for autonomous driving using deep convolutional networks. *ICRA*, 2019. 4, 5

[5] Maximilian Igl, Daewoo Kim, Alex Kuefler, Paul Mougin, Punit Shah, Kyriacos Shiarlis, Dragomir Anguelov, Mark Palatucci, Brandyn White, and Shimon Whiteson. Symphony: Learning realistic and diverse agents for autonomous driving simulation. *arXiv*, 2022. 2

[6] Karsten Kreutz and Julian Eggert. Analysis of the generalized intelligent driver model (gidm) for uncontrolled intersections. In *ITSC*, 2021. 2

[7] Zhiqi Li, Wenhai Wang, Hongyang Li, Enze Xie, Chonghao Sima, Tong Lu, Qiao Yu, and Jifeng Dai. Bevformer: Learning bird's-eye-view representation from multi-camera images via spatiotemporal transformers. In *ECCV*, 2022. 4

[8] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv*, 2017. 4

[9] Sivabalan Manivasagam, Shenlong Wang, Kelvin Wong, Wenyuan Zeng, Mikita Sazanovich, Shuhan Tan, Bin Yang, Wei-Chiu Ma, and Raquel Urtasun. Lidarsim: Realistic lidar simulation by leveraging the real world. In *CVPR*, 2020. 3, 6, 8, 9

[10] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. 2022. 1, 2

[11] Keunhong Park, Utkarsh Sinha, Jonathan T Barron, Sofien Bouaziz, Dan B Goldman, Steven M Seitz, and Ricardo Martin-Brualla. Nerfies: Deformable neural radiance fields. In *ICCV*, 2021. 10

[12] Steven G Parker, James Bigler, Andreas Dietrich, Heiko Friedrich, Jared Hoberock, David Luebke, David McAllister, Morgan McGuire, Keith Morley, Austin Robison, et al. Optix: a general purpose ray tracing engine. *TOG*, 2010. 3

[13] Sida Peng, Yuanqing Zhang, Yinghao Xu, Qianqian Wang, Qing Shuai, Hujun Bao, and Xiaowei Zhou. Neural body: Implicit neural representations with structured latent codes for novel view synthesis of dynamic humans. In *CVPR*, 2021. 10

[14] Abbas Sadat, Mengye Ren, Andrei Pokrovsky, Yen-Chen Lin, Ersin Yumer, and Raquel Urtasun. Jointly learnable behavior and trajectory planning for self-driving vehicles. *IROS*, 2019. 4

[15] Pratul P. Srinivasan, Boyang Deng, Xiuming Zhang, Matthew Tancik, Ben Mildenhall, and Jonathan T. Barron. NeRV: Neural reflectance and visibility fields for relighting and view synthesis. In *CVPR*, 2021. 10

[16] Simon Suo, Sebastian Regalado, Sergio Casas, and Raquel Urtasun. Trafficsim: Learning to simulate realistic multi-agent behaviors. In *CVPR*, 2021. 2

[17] Martin Treiber, Ansgar Hennecke, and Dirk Helbing. Congested traffic states in empirical observations and microscopic simulations. *Physical Review E*, 2000. 2

[18] Tianhan Xu and Tatsuya Harada. Deforming radiance fields with cages. In *ECCV*, 2022. 10

[19] Bin Yang, Wenjie Luo, and Raquel Urtasun. Pixor: Real-time 3d object detection from point clouds. In *CVPR*, 2018. 9

[20] Alex Yu, Ruilong Li, Matthew Tancik, Hao Li, Ren Ng, and Angjoo Kanazawa. Plenoctrees for real-time rendering of neural radiance fields. In *ICCV*, 2021. 2

[21] Yu-Jie Yuan, Yang-Tian Sun, Yu-Kun Lai, Yuewen Ma, Rongfei Jia, and Lin Gao. Nerf-editing: geometry editing of neural radiance fields. In *CVPR*, 2022. 10

[22] Kai Zhang, Gernot Riegler, Noah Snavely, and Vladlen Koltun. NeRF++: Analyzing and improving neural radiance fields. *arXiv*, 2020. 1, 3

[23] Xiuming Zhang, Pratul P Srinivasan, Boyang Deng, Paul Debevec, William T Freeman, and Jonathan T Barron. Nerfactor: Neural factorization of shape and reflectance under an unknown illumination. *TOG*, 2021. 10

[24] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. Open3d: A modern library for 3d data processing. *arXiv*, 2018. 2

[25] Yi Zhou, Connelly Barnes, Jingwan Lu, Jimei Yang, and Hao Li. On the continuity of rotation representations in neural networks. In *CVPR*, 2019. 2