Supplementary Material
CADSim: Robust and Scalable in-the-wild 3D Reconstruction for Controllable Sensor Simulation

Jingkang Wang[1,2] **Sivabalan Manivasagam**[1,2] **Yun Chen**[1,2] **Ze Yang**[1,2]
Ioan Andrei Bârsan[1,2] **Anqi Joyce Yang**[1,2] **Wei-Chiu Ma**[1,3] **Raquel Urtasun**[1,2]
Waabi[1] University of Toronto[2] Massachusetts Institute of Technology[3]
{wangjk,manivasagam,zeyang,yun,iab,ajyang,urtasun}@cs.toronto.edu weichium@mit.edu

Abstract: In this supplementary material, we provide additional details on our method and experiments, and then show additional application results using CADSim. For our method, we describe how we create a shared low-dimensional representation space for optimizing over a set of CAD models (Sec A.1), provide details on our selected appearance representation (Sec A.2), and the exact inference optimization procedure performed and hyperparameters used CADSim (Sec A.3). For our experiments, we first provide implementation details of the baselines compared against (Sec B), as well as dataset and metric details (Sec C). We then report thorough ablations on our choice of geometry and appearance (Sec D.1), demonstrate the robustness of CADSim to data noise (Sec D.2), show our approach applied to non-vehicle objects (Sec D.3), and show more experiment results of our model improving perception evaluation (Sec D.5). Finally, we show additional applications of CADSim. We show using CADSim for multi-sensor simulation examples for scenario replay (Sec E.1) and mixed reality (Sec E.2), as well as showing CADSim naturally supporting texture transfer for creating diverse assets (Sec E.3). Additionally, we include a supplementary video, **supplementary_56.mp4** providing an overview of our methodology, as well as video results on novel-view synthesis, and realistic multi-sensor simulation.

A CADSim Implementation Details

A.1 Learning a shared representation

As discussed in Sec 3, we create a shared representation space over a set of CAD models to handle a wide variety of vehicle shapes during optimization. Specifically, we apply principal component analysis (PCA) on the vertex coordinates of the CAD models to obtain a shared low-dimensional code $\mathbf{z}$. Formally, we have

$$\begin{aligned} \mathbf{z} &= \mathbf{W}^\top (\mathbf{V} - \boldsymbol{\mu}), \\ \widehat{\mathcal{M}} &= \{\widehat{\mathbf{V}}, F\}, \quad \text{where } \widehat{\mathbf{V}} = \mathbf{W}\mathbf{z} + \boldsymbol{\mu}. \end{aligned} \tag{1}$$

$\boldsymbol{\mu} \in \mathbb{R}^{|V|\times 3}$ is the mean vertices of the meshes, and $\mathbf{W}$ is the top K principle components. $\mathbf{z}$, $\widehat{\mathcal{M}}$ and $\widehat{\mathbf{V}}$ are the latent code, reconstructed mesh and vertices respectively. We note that since all the CAD models are parameterized with Eq. (1) from the main paper, the reconstructed mesh $\widehat{\mathcal{M}}$ by nature consists of parts and supports wheel articulation. As shown in Fig. A1 (first row), the deformed meshes maintain important geometry details of vehicles such as rear-view mirrors and car grilles.

CAD library alignment. In order to learn a low dimension code over a variety of vehicles, we must align the templates from different CAD models and establish a one-to-one dense correspondence among the vertices. The original CAD models are unaligned, as they have a varying number of vertices and the vertex ordering differs across models. Specifically, we select a single template mesh $\mathcal{M}_{\text{src}}$ as the source mesh and deform its vertices $\mathbf{V}$ such that it fits other meshes well. We exploit the

6th Conference on Robot Learning (CoRL 2022), Auckland, New Zealand.

vertices of the simplified target mesh (denoted as $\mathcal{P}_{\text{cad}}$) and minimize the following energy:

$$E_{\text{align}}(\mathbf{V}, \mathcal{P}_{\text{cad}}) = E_{\text{chamfer}}(\mathbf{V}, \mathcal{P}_{\text{cad}}) + \lambda_{\text{shape}} \cdot E_{\text{shape}}(\mathcal{M}_{\text{src}}). \tag{2}$$

Here, E_{chamfer} refers to the asymmetric Chamfer distance, and E_{shape} is the same as described in Sec. 3.2. We note that this is an offline procedure separate from the energy minimization in Sec 3.3.

A.2 Appearance Representation

In addition to geometry, our mesh representation must accurately capture how light interacts with its surface so that we can realistically reproduce sensor observations. Towards this goal, we parameterize the mesh appearance using a physically-based appearance representation. Specifically, we represent appearance using a micro-facet BRDF model with the differentiable split sum environment lighting [1] proposed by Munkberg et al. [2]. Since the topology of the mesh is fixed, we can build a one-to-one mapping relationship between each point on the mesh surface and each point in the 2D (u, v) space. We use a Physics-Based Rendering (PBR) material model from Disney [3], which contains a diffuse lobe $\mathbf{k}_d$ with an isotropic, specular GGX lobe [4]. Following the standard convention, we store them together with normals in three texture images $\mathbf{k}_d$, $\mathbf{k}_{\text{orm}}$ and $\mathbf{k}_n$. $\mathbf{k}_{\text{orm}} = (o, r, m)$ where o is left unused, r is the roughness value and m is the metalness factor that interpolates between plastic and metallic appearance by computing a specular highlight color: $\mathbf{k}_s = (1 - m) \cdot 0.04 + m \cdot \mathbf{k}_d$. For the lighting model, we use a differentiable version of the split sum shading model introduced by Munkberg et al. [2]. Specifically, we optimize a cube map ($6 \times 512 \times 512$), where the base level represents the pre-integrated lighting for the lowest supported roughness value and each smaller mip-level is reconstructed using [5]. We refer the readers to [2] for more details.

Intensity retrieval. For LiDAR simulation, the optimized meshes are enriched with per-vetex intensity by retrieving the top 10 closest points for aggregated point cloud and taking the average intensity value. Empirically we find the performance is similar to direct energy optimization on the vertex intensity.

A.3 Inference Details

Since all operations are differentiable, one straightforward way to conduct inference is to directly minimize the full energy with gradient-based methods. Unfortunately, due to the highly non-convex structure of the energy model as well as the noise in the observations, such an approach will often lead to sub-optimal solutions. We thus adopt the following curriculum strategy.

First, we optimize the latent code $\mathbf{z}$ initialized from $\mathbf{0}$ with Adam optimizer (learning rate 3e-2) for 200 iterations (Eq. 8 in the main paper), while keeping the other variables fixed. Given the initialization $\mathcal{M}_{\text{init}}$, we jointly optimize the vertices $\mathbf{V}$, appearance variables $\mathcal{A}$, and sensor poses Π as described in Eq. (3). At the beginning of the optimization (first 500 iterations), we do not add E_{color} term so that the model will focus on geometry (*e.g.*, E_{mask}, E_{lidar}, etc). The hyperparameters are set as $\lambda_{\text{mask}} = \lambda_{\text{lidar}} = 0.5, \lambda_{\text{shape}} = 0.1$. To enforce the learned geometry for vehicles to be symmetric, we flip the vertices of optimized mesh along the symmetry axis and add a symmetric Chamfer distance ($\lambda_{\text{sym}} = 0.5$) between the original and flipped meshes. We use AdamUniform [6] (learning rate 3e-2) to optimize the vertices and standard Adam (learning rate 1e-4) to optimize the intrinsics and the extrinsics of the sensors Π. Then in the next 500 iterations, we add E_{color} term to optimize the appearance variables $\mathcal{A}$ as well as refine the sensor poses and geometry through RGB information. We set the coefficients for appearance energy term as follows: $\lambda_{\text{app}} = 1, \lambda_{\text{mat}} =$ 1e-9, $\lambda_{\text{light}} =$ 1e-2. We use the Adam optimizer [7] with a learning rate scheduler with an exponential falloff from 0.03 to 0.01 over 500 iterations. Empirically we find this simple weighting strategy very effective.

B Implementation Details for Baselines

We compare our approach with a wide range of state-of-the-art 3D reconstruction methods, which can be roughly divided into: (1) *Neural radiance fields*: NeRF++ [10], and Instant-NGP [11]. (2) *Implicit surface representations*: NeRS [12], NVDiffRec [2], and NeuS [13]. (3) *Explicit geometry-based approaches*: single-image view-warping (SI-ViewWarp) [14], which warps the source image to the novel target view using depth estimated by LiDAR points, and multi-image view-warping (MI-ViewWarp), which blends multiple source images to the target view. We also compare against

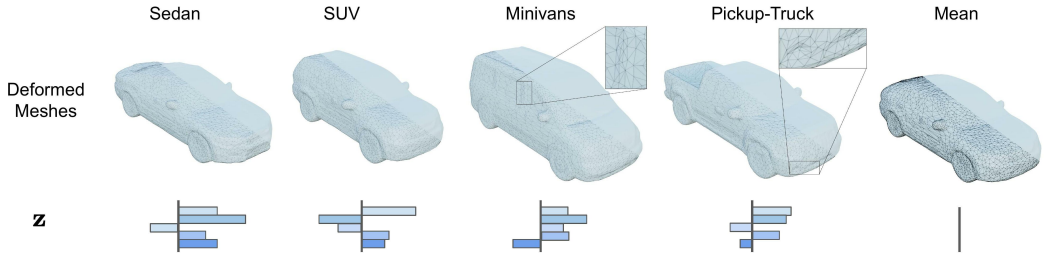

Figure A1: **Collect watertight and vertex-aligned vehicle meshes from CAD models**. Given a collection of CAD vehicles, we first use a robust algorithm [8, 9] to obtain watertight manifolds. Then, we pick one simplified mesh as the source mesh and deform it to the other meshes by minimizing the shape energy. Finally, we build the PCA basis on deformed meshes with latent codes stored for each object.

SAMP [15], a CAD model mesh optimization approach which leverages SDF-aligned CAD models for joint pose and shape optimization. In the following, we provide the overview and implementation details of these reconstruction baselines.

B.1 NeRF-based Approaches

NeRF++ [10]. NeRF++ introduced an inverse sphere parameterization to extending NeRF to large-scale and unbounded 3D scene. To train NeRF++ properly on PandaVehicle, we first scale the scene to normalize all cameras' position within a unit sphere. Then we adopt the same hyperparameters from the official code repository[1] except that we train for 100k iterations because PandaVehicle has fewer views and the model converges with fewer iterations.

Instant-NGP [11]. Instant-NGP achieved the state-of-the-art performance by introducing the efficient hash encoding and fully fused MLPs. In our experiments, we properly scale the objects to be reconstructed in the unit cube and set `aabb_scale` as 4 to handle the background visible outside the unit cube. We tuned `aabb_scale` to be 1, 2, 4, 8 and 16 and find 4 leads to the best performance. The model is trained for 5k iterations and converges on the training views. During inference, we render with 4 samples per pixel for better results (effectively doing 4x superresolution for anti aliasing).

B.2 Implicit Surface Representations.

NeRS [12]. Zhang et al. proposed a neural surface representation combined with differentiable rendering to generalize with sparse in the wild data. We followed the official NeRS[2] implementation. On MVMC, we use the official evaluation code and focus on the "fixed optimized camera" setting. For PandaVehicle, we initialized the cuboid template with the assets' coarse 3D dimensions and set the level of unit ico-sphere as 6. We employ a three-stage training process: sequentially optimizing the shape, texture, and illumination parameters. To ensure better visual quality and semantic metrics, we increased the weights of the chamfer loss and perceptual loss to 0.04 and 1.0, respectively. We also removed off-screen loss as not all input views contain the complete vehicle shapes. Moreover, we increased the training iterations on the three stages to $3k, 12k, 3k$ since more input views are provided in PandaVehicle and it takes longer to converge. We also applied symmetry constraints to the deformed textured meshes along the heading axis.

NVDiffRec [2]. Munkberg et al. proposed an efficient differentiable rendering-based reconstruction approach that combines differentiable marching tetrahedrons and split-sum environment lighting. It achieves the state-of-the-art performance on a wide variety of synthetic datasets with dense camera views. We follow the official code implementation[3]. We set the tetrahedron grid resolution as 64 and the mesh scale as 5.0 (real vehicle scale). The model is trained for 5k iterations (batch size 8) with a learning rate exponentially decayed from 0.03 to 0.003.

[1] https://github.com/Kai-46/nerfplusplus
[2] https://github.com/jasonyzhang/ners
[3] https://github.com/NVlabs/nvdiffrec

NeuS [13]. We follow the official code repository[4] and train each asset for 200k iterations. We perform scene normalization to make the asset's region of interest fall inside a unit sphere, and model the background by NeRF++ [10]. To train the NeuS with LiDAR supervision, we render the LiDAR depth map, and use it to supervise the volume rendered depth similar to DS-NeRF [16].

B.3 Geometry-based Approaches.

SI-ViewWarp [17]. For warping based methods, we follow the implementation in GeoSim [17]. Give an asset mesh, we first render the mesh at the target viewpoint to generate the target depth map. Then, we un-project the target depth map to source images and get the corresponding pixel color using the inverse warping operations. We compare un-projected depth in source view and asset-rendering depth in source view to filter invisible region. This approach doesn't need training for rendering. To get the best rendering results, we warp using all source images as candidates, and heuristically pick the one with minimum unseen region in the target view.

MI-ViewWarp. Since there are often unobserved regions at the target view when warping from a single source view, we further extend SI-ViewWarp to warp from multiple source images progressively (MI-ViewWarp). Specifically, we sort the source images in an increasing order according to the distance of the viewpoints (or camera matrices) between source and target images. Then we iterate on the sorted source images, conduct SI-ViewWarp and fill in the pixels progressively only if not occupied. We find this heuristic warping strategy works well in practice as it takes the "confidence" (according to distance) of each source image into account. Compared with simple averaging of all warped single images, our strategy produces non-blurry results and better metrics.

SAMP [15]. Given the processed CAD library where each mesh is watertight and simplified, we compute volumetric SDFs for each vehicle in metric space (volume dimension $100 \times 100 \times 100$). Following [15, 18], we apply PCA on the SDF volumes and set the embedding dimension as 25. During inference, we jointly optimize the shape latent code, a scaling factor on the SDF (handle diverse shapes) and relative vehicle pose (rotation, translation) to fit the LiDAR points. We adopt the $\mathcal{L}_1$ loss on the SDF difference and a total variation loss on the scale factor to penalize abrupt local SDF changes. The weights of data and regularization terms are 1 and 0.1. We use the Adam optimizer with a learning rate of 0.01. The mesh is extracted from the SDF volume via marching cubes [19]. Given the optimized shape, we then use a similar CADSim-like energy minimization procedure to optimize a 2D UV texture image using a differentiable renderer. Compared with CADSim, optimized SAMP geometries usually have fewer fine-grained details and have worse image alignments.

C Dataset and Metric Details

C.1 PandaVehicle Details

We derive PandaVehicle from the PandaSet [20] dataset. PandaSet is a dataset captured by a self-driving vehicle platform equipped with six cameras (left, front left, front, front right, right and back cameras) and two LiDARs (a top 360° mechanical spinning LiDAR and a forward-facing LiDAR). In PandaVehicle, we select vehicles that are observed by the main top Pandar64 sensor, as well as the left, front left and front cameras. We observe that there is poorer calibration for the front right, right and back cameras and therefore do not use them for quantitative evaluation. We train or reconstruct vehicle meshes with observations from the top LiDAR and the left camera images only. In the novel view synthesis task, we test by rendering the mesh and comparing with images from the front left and front cameras. We generate all semantic segmentation masks with PointRend [21].

We selected 10 vehicles that have high-quality camera-LiDAR alignment for evaluation. Fig. A19 shows the aggregated LiDAR points and images used for one of the selected vehicles. Table A10 includes detailed information for all 10 selected vehicles.

[4] https://github.com/Totoro97/NeuS

C.2 LiDAR Rendering Metrics

We provide additional details on how we compute LiDAR rendering metrics used in the main paper Table 3. Given the aggregated point clouds $\mathcal{P}$, we apply voxel downsampling with a resolution of 5cm to obtain the input point clouds $\mathcal{P}_{\text{input}}$ for reconstruction. Then the held-out real LiDAR points can be written as $\mathcal{P}_{\text{held}} = \mathcal{P} \setminus \mathcal{P}_{\text{input}}$. We evaluate the average per-ray ℓ_2 error and what fraction of real held-out LiDAR points have a corresponding simulated point (*i.e.*, Hit rate). Lastly, we place the reconstructed vehicle mesh in its original location and perform ray-casting to generate a simulated point cloud and report the Chamfer and Hausdorff distance with original aggregated point clouds $\mathcal{P}$.

D Additional Experiments and Analysis

In this section, we show additional experiments on MVMC and PandaVehicle, including geometry comparison with NeRS on MVMC, NVS with larger extrapolation, and ablation studies on LiDAR supervision, mesh initialization and appearance representation. We also show additional downstream perception results.

D.1 Ablation Experiments on MVMC and PandaVehicle

Method	Geometry	MSE ↓	PSNR ↑	SSIM ↑	LPIPS ↓	FIDS ↓
SI-ViewWarp [14]	NeRS	0.0683	12.3	0.659	0.288	93.5
	Ours	**0.0587**	**12.9**	**0.674**	**0.257**	**79.1**
MI-ViewWarp	NeRS	0.0430	14.4	0.665	0.225	67.2
	Ours	**0.0311**	**15.6**	**0.687**	**0.185**	**51.4**
UV Map Opt.	NeRS	0.0371	15.1	0.690	0.215	74.6
	Ours	**0.0215**	**17.1**	**0.725**	**0.160**	**51.6**

Table A1: Comparison of geometry quality with NeRS [12] on MVMC.

Method	MSE ↓	PSNR ↑	SSIM ↑	LPIPS ↓
Instant-NGP [Müller et al., 2022]	0.0302	15.68	0.417	0.479
NeuS [Wang et al., 2021]	0.0280	16.17	0.403	0.439
SAMP [Engelmann et al., 2017]	0.0258	16.42	0.401	0.362
CADSim (ours)	**0.0218**	**16.87**	**0.424**	**0.334**

Table A2: Evaluation of novel-view synthesis on extreme views (left → front camera) on PandaSet.

Geometry Comparison on MVMC. To further demonstrate CADSim is able to reconstruct higher-quality geometries, we report the NVS results comparing our reconstructed mesh compared to NeRS [12] mesh on various appearance representations. We follow the same evaluation setting as Table 1 in the main paper. We only change how the texture for the mesh is created, while keeping the optimized geometry fixed. As shown in Table A1, our geometry consistently leads to better NVS performance due to better alignment with images and improved optimization with CAD priors.

Additional Qualitative Results. We provide additional qualitative results on PandaVehicle in Figure A2. CADSim is able to outperform all baselines consistently on large extrapolation setting. This is because we leverage CAD priors during shape and appearance optimization and produce higher quality geometry. In contrast, NeRF-based and implicit surface approaches usually suffer from obvious artifacts due to shape-radiance ambiguity [10]. The other geometry-based approaches either relies on non-manifold surfel meshes or coarse volumetric SDFs thus leading to missing pixels, blurry results and large image-geometry misalignment.

NVS for Extreme Views on PandaVehicle. To test the robustness of CADSim in the self-driving context, we conduct a more challenging NVS task: left camera → front camera. Fig. A19 (left) visualizes the significant change in viewpoint. Since there is no overlap in the field-of-view between

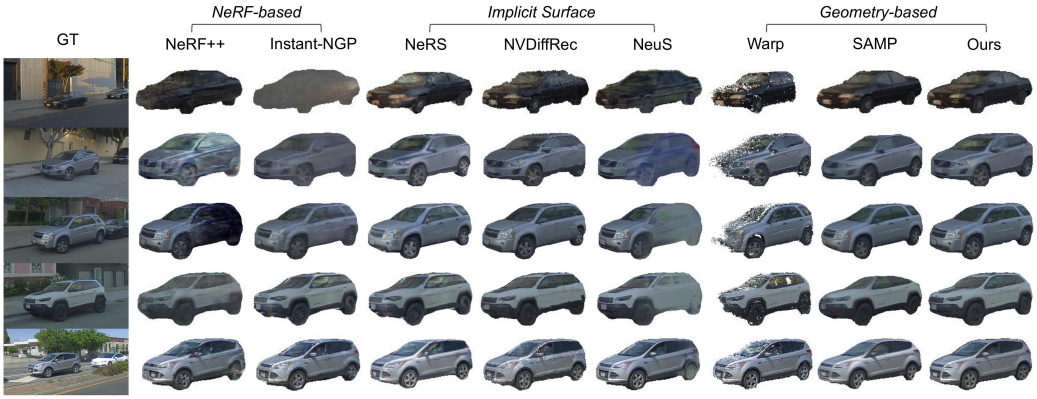

Figure A2: **Additional qualitative results on PandaVehicle for novel view synthesis.** Compared to existing reconstruction approaches, CADSim produces more robust and realistic results on large extrapolation.

left and front cameras on Pandaset[5], it requires better geometry and appearance for large extrapolation. We pick the baselines with the best performance on each category in Sec B. As shown in Table A2, our approach results in the best performance, especially on LPIPS which measures how well the machine learning models perceive the simulated images.

Investigation of LiDAR Supervision. Compared to previous reconstruction approaches, CADSim can leverage the LiDAR points that are usually accessible in the self-driving applications. While it is not our focus to extend existing baselines to leverage LiDAR properly, we choose one of the best performing baselines, NeuS, and use the rendered LiDAR depth map to supervise the volume rendered depth, similar to DS-NeRF [16]. Moreover, we also remove the LiDAR branch E_{lidar} of CADSim for comparison. As shown in Table A3 and Table A4, adding depth supervision for NeuS helps improve the NVS performance (especially on LPIPS) and LiDAR rendering results by reducing the ambiguity caused by sparse views. We note that although the LiDAR rendering metrics have improved, the geometry obtained by NeuS is still not complete. This may lead to a dramatic performance decrease when we place actors in completely new locations and viewpoints for simulation. In contrast, CADSim is less dependent on LiDAR supervision and our assets are always complete with the help of CAD shape priors.

Method	MSE ↓	PSNR ↑	SSIM ↑	LPIPS
NeuS [Wang et al., 2021]	0.0115	21.37	0.640	0.247
NeuS + LiDAR	0.0103	21.42	0.649	0.228
CADSim - LiDAR	0.0096	21.59	0.665	0.231
CADSim (ours)	**0.0087**	**21.72**	**0.674**	**0.220**

Table A3: Evaluation of novel-view synthesis (left → front-left camera) on PandaSet.

Geometry	L_2 error ↓	Hit rate ↑	Chamfer↓	Hausdorff↓
NeuS [13]	0.367	90.3%	0.424	1.151
NeuS + LiDAR	0.191	95.3%	0.261	1.017
CADSim - LiDAR	0.155	95.2%	0.249	0.988
CADSim (ours)	**0.151**	**96.3%**	**0.245**	**0.972**

Table A4: Comparison of LiDAR rendering metrics.

Geometry and Reflectance Model Ablation Study. We rigorously evaluate that our method's use of a CAD model improves performance in Table A5, where we replace the proposed CAD

[5]https://scale.com/open-datasets/pandaset

initialization with the following alternatives: a unit sphere, a rescaled ellipsoid, and geometries from either SAMP [15] or NeRS [12]. Our CAD mesh initialization has the highest performance. We also evaluate our reflectance model choice [4] in Table A6. Our physics-based material and lighting model performs the best and generalizes well to novel views.

Initialization	PSNR ↑	SSIM ↑	LPIPS ↓
Sphere	20.61	0.631	0.243
Ellipsoid	21.10	0.653	0.233
SAMP [15]	21.32	0.667	0.230
NeRS [12]	21.58	0.665	0.235
CAD (ours)	**21.72**	**0.674**	**0.220**

Table A5: Ablation on the mesh initialization.

Texture Model	PSNR ↑	SSIM ↑	LPIPS ↓
SI-ViewWarp [22]	17.81	0.540	0.318
MI-ViewWarp	19.22	0.570	0.277
Per-vertex Color	19.37	0.595	0.270
Texture UV Map	19.51	0.618	0.263
Phong Model [23]	20.13	0.645	0.248
Cook-Torrance Model [4]	**21.72**	**0.674**	**0.220**

Table A6: Ablation on varied texture models.

D.2 Robustness of CADSim to Data Noise

CADSim is designed to handle real-world driving data captured by SDVs that inevitably contains noise. Sources of noise include: incomplete LiDAR scans, corrupted object masks, noisy actor poses and imperfect lidar-camera calibration. Examples of noise in PandaVehicle data are shown in Figure A3. Existing approaches do not perform as well on sparse and noisy in-the-wild data as shown in Figure A3 and Table 1. Our approach achieves stronger results in this setting qualitatively and quantitatively.

To further demonstrate the robustness of CADSim, we conduct the following additional experiments.

Robustness to actor poses. To investigate the robustness of our approach on the noise in the actor poses (3D bounding boxes), we add synthetic noise to the bounding box annotations in PandaVehicle. Specifically, we apply zero-mean Gaussian noise to the center location of the bounding boxes (x, y, z) with variance increasing from 0m to 0.5m. We also insert zero-mean Gaussian noise with variance of 5 degree to the yaw angle. The results for actor `1d79eded-2fb0-4f89-ba35-323926f45ade` are presented in Table A7. Compared to Instant-NGP [11], our reconstruction performance only drops slightly as actor pose noise increases. This is because our framework also jointly optimizes the actor pose (*i.e.*, ξ in Eqn. (4)). Note that for Instant-NGP, we also enable the camera extrinsics optimization. Finally, we disable the actor pose optimization component and re-run the experiment. Qualitative results in Figure A4 show the effectiveness of pose optimization in obtaining non-blurry rendering results.

Noise ($\mu = 0$)	Method	PSNR ↑	SSIM ↑	LPIPS ↓
$\sigma_{xyz} = 0$m	CADSim	23.32	0.761	0.181
	Instant-NGP [11]	22.66	0.759	0.223
$\sigma_{xyz} = 0.1$m	CADSim	23.30 (-0.02)	0.757 (-0.004)	0.188 (+0.007)
	Instant-NGP [11]	20.26 (-2.40)	0.661 (-0.098)	0.439 (+0.216)
$\sigma_{xyz} = 0.2$m	CADSim	23.12 (-0.20)	0.744 (-0.017)	0.195 (+0.014)
	Instant-NGP [11]	19.77 (-2.89)	0.655 (-0.104)	0.461 (+0.238)
$\sigma_{xyz} = 0.5$m	CADSim	22.91 (-0.41)	0.725 (-0.036)	0.230 (+0.049)
	Instant-NGP [11]	19.19 (-3.47)	0.649 (-0.110)	0.471 (+0.248)

Table A7: Robustness to noisy actor poses.

Robustness to corrupted object mask. To evaluate the impact of corrupted masks, following [2], we add synthetic noise into the predicted object masks. Specifically, we find the contour of the object and perturb each contour point pixel location with a zero-mean Gaussian noise. The variance is set from 0px to 50px. We train the model using the front camera and corrupted object masks and test on the front-left camera for the actor `1d79eded-2fb0-4f89-ba35-323926f45ade`. Experiments in Table A8 show that our approach is quite robust to the corrupted masks even in a high-noise regime (e.g., $\sigma = 50$px) thanks to the robust CAD initialization and our carefully designed energy formulation.

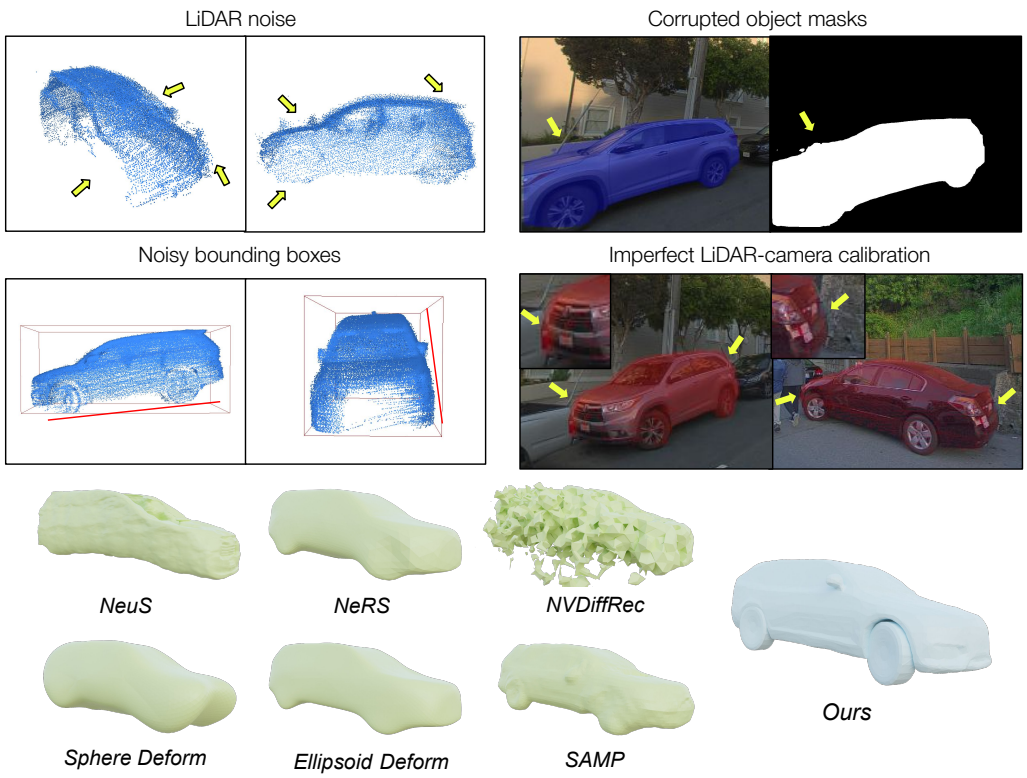

Figure A3: **Visualizations of data noise on PandaVehicle.** Sources of noise include (clockwise from top-left): incomplete and noisy LiDAR scans, corrupted object masks, noisy actor bounding boxes and imperfect lidar-camera calibration. Reconstruction with this in-the-wild data is challenging as (a:left) Sparse and incomplete LiDAR points can result in collapsed shapes on the invisible side. (a:right) Noisy LiDAR aggregation will lead to inaccurate or non-smooth geometry surface; (b) The corrupted object masks due to inaccurate segmentation predictions can result in inaccurate geometry and appearance; (c) The lidar-camera calibration errors will result in imperfect geometry that does not align with the images well and potentially has blurry appearance. (d) The noise in bounding boxes will lead to blurry appearance and imperfect initialization and alignment. Compared to existing approaches, CADSim is robust to those data noise meanwhile maintaining fine-grained geometry details and editable parts.

Noise ($\mu = 0$)	PSNR ↑	SSIM ↑	LPIPS ↓
$\sigma = 0$px	23.32	0.761	0.181
$\sigma = 5$px	23.29 (-0.03)	0.756 (-0.005)	0.192 (+0.011)
$\sigma = 10$px	23.22 (-0.10)	0.753 (-0.008)	0.198 (+0.017)
$\sigma = 20$px	22.99 (-0.34)	0.743 (-0.018)	0.214 (+0.023)
$\sigma = 50$px	22.56 (-0.66)	0.737 (-0.024)	0.223 (+0.032)

Table A8: Robustness to corrupted object masks.

In summary, we believe that our approach is more robust to data noise and recovers more accurate geometries from sparse/noisy data compared to existing approaches.

D.3 Reconstruction of Non-Vehicle Classes

While we mainly focus on the vehicle reconstruction as vehicles are the primary traffic participants, we note that CAD models are readily available for most object classes, and that our approach can be extended to other classes. In this section, we showcase the reconstruction of motorcycles and cones on PandaSet using CADSim.

Figure A4: **Ablation study on the pose optimization for CADSim.** From left to right, we show (a) noisy actor bounding box ξ (red) by inserting synthetic Gaussian noise; (b) CADSim learning using perturbed actor poses but with the actor pose optimization module disabled; (c) CADSim learning with noisy poses; (d) CADSim learning using clean actor poses. We can observe the effectiveness of pose optimization in CADSim for obtaining non-blurry rendering results.

Challenges: Motorcycles are more challenging to reconstruct in-the-wild. Since they are smaller than vehicles, the scanned lidar points are sparser and noisier, and there are fewer image pixels corresponding to the motorcycles. Moreover, due to complex topology and occlusion, the object masks predicted by PointRend are more corrupted (see Figure A5) and there are larger relative camera/lidar misalignments for small objects. Please refer to Sec D.4 for potential future directions.

We encode the semantic priors for a single CAD asset and obtain three parts (handlebar, front wheel, and the rest body). We optimize the per vertex offset for the body and handlebar parts. To make motorcycle behave in a physically plausible manner, for front wheel and handlebar, we also optimize the relative pose to the motorcycle origin, scale factors, relative rotation, and translation offset. As shown in Figure A5, CADSim is able to reconstruct motorcycles with reasonable geometry and appearance in spite of large data noise (*i.e.*, large LiDAR image misalignment shown in the leftmost column, coarse and noisy segmentation mask) and very sparse observations. In Figure A6, we use CADSim to reconstruct a traffic cone, demonstrating that CAD priors enable efficient and accurate mesh reconstruction for very small objects as well.

D.4 Additional Limitations

In addition to the limitations described in the main paper, we describe additional limitations and possible extensions. Our method does not allow for arbitrary changes in mesh topology and it might lead to some unexpected behavior when the topology changes within one class (see the windshield in Figure A5), which is an exciting future direction to address. Futhermore, we use aggregated LiDAR points instead of doing per-frame registration. This may make the approach less robust to LiDAR image misalignment, especially for small or far-away objects. While CADSim is more robust than other reconstruction approaches, it might still produce unexpected results on extremely noisy data (*e.g.*, large calibration misalignment, highly corrupted object masks).

With regards to potential extensions, rigging additional parts (*e.g.*, vehicle doors) or even automatic rigging may be conducted in future works. Finally, our experiments also focus on sensor simulation for perception models, and we may investigate the performance for downstream planning tasks for future works.

D.5 Additional Downstream Evaluation on Camera Simulation

To verify if CADSim helps reduce domain gap for downstream perception tasks consistently, we evaluate another perception model on simulated camera images at novel views. Specifically, we follow the same setting as Table 4 in the main paper except replacing Mask R-CNN [24] with PointRend [21] algorithms. As shown in Table A9, using CADSim assets usually leads to the largest agreement with real images under different settings. This indicates the effectiveness of reconstructed CADSim assets for end-to-end autonomy testing.

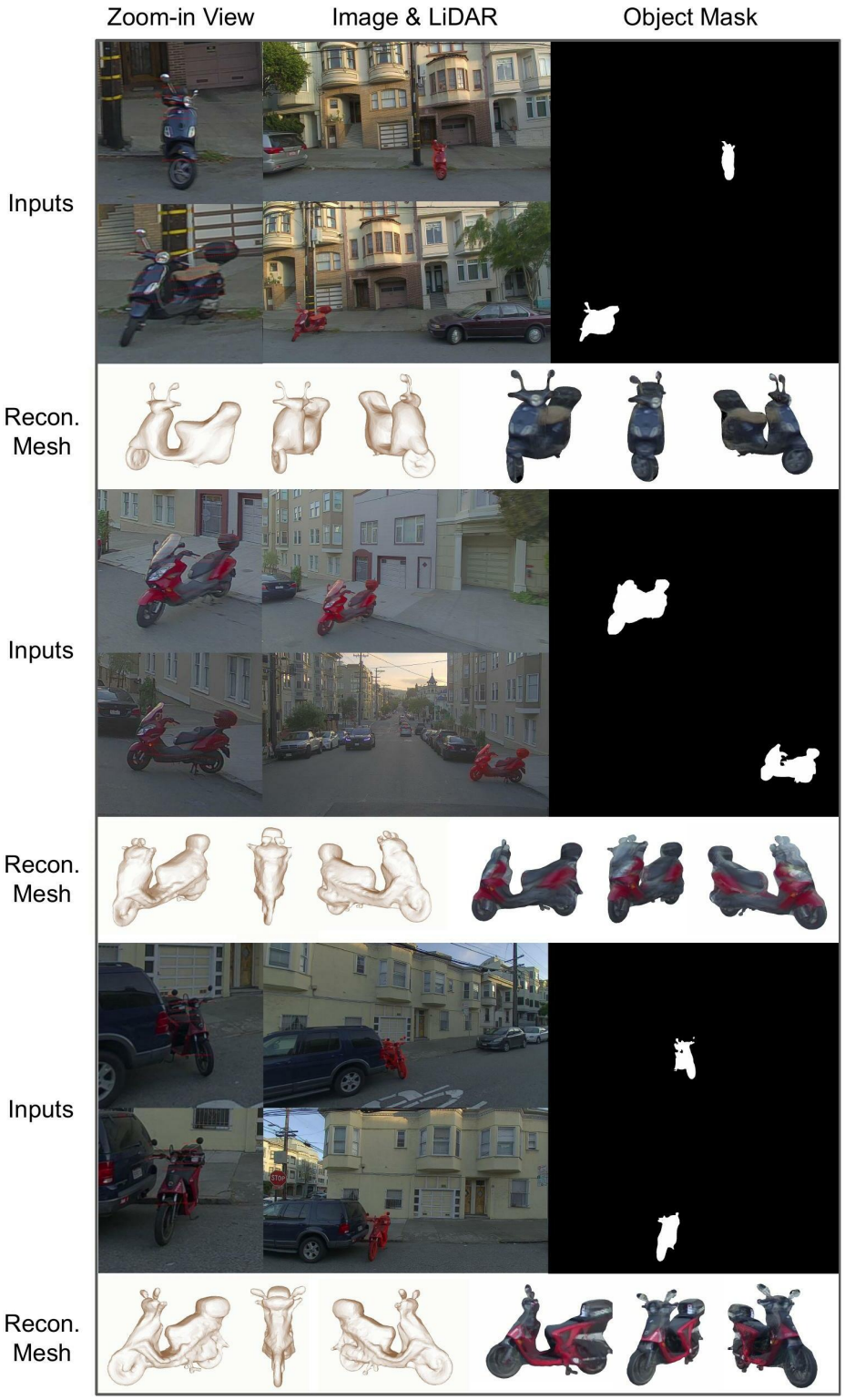

Figure A5: Reconstruction of motorcycles on PandaSet.

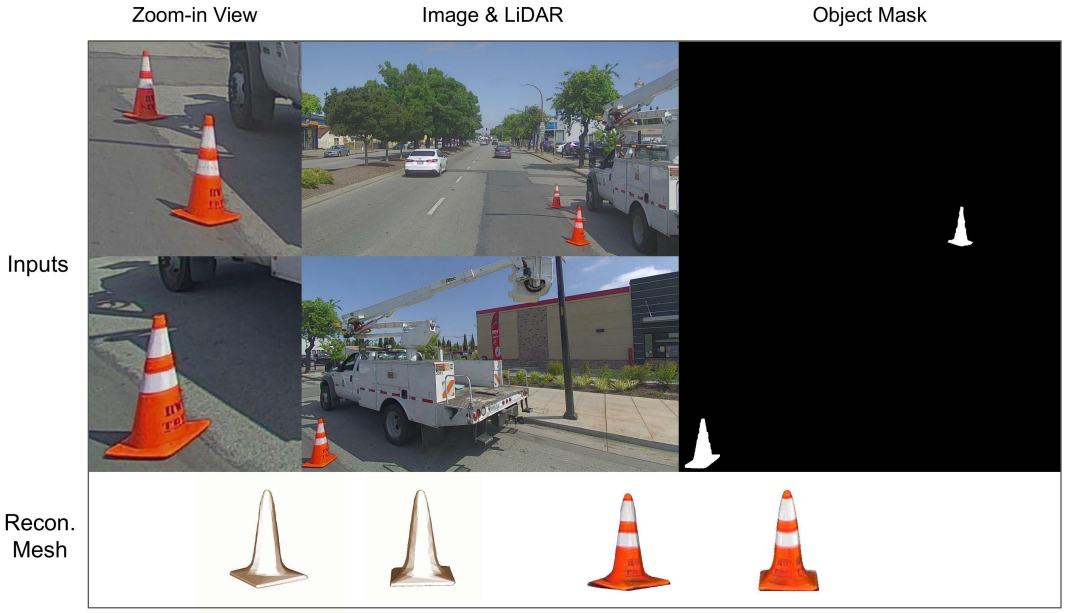

Figure A6: Reconstruction of traffic cones on PandaSet.

	Left camera → *Front-left camera*				*Left camera* → *Front camera*			
	Blending [17]		Copy-Paste		Blending [17]		Copy-Paste	
	Det.	Segm.	Det.	Segm.	Det.	Segm.	Det.	Segm.
Instant-NGP	86.18	87.03	79.84	80.61	71.74	71.22	50.08	49.05
NeuS	94.59	**95.24**	92.83	93.72	75.03	74.65	69.47	68.91
SAMP	90.82	90.01	90.88	90.85	81.79	78.00	**82.39**	**82.01**
CADSim	**94.71**	94.43	**94.68**	**94.66**	**82.68**	**80.81**	82.33	81.32

Table A9: Evaluation of downstream perception tasks (*i.e.*, object detection, instance segmentation) on camera simulation. We report the metric agreement (instance-level IoU) with the model evaluated on real data.

E Realistic and Controllable Simulation

E.1 Log-Replay Simulation with CADSim Assets

We now show that CADsim can reconstruct the static vehicles at scale from sequences of sensor data. In Figures A7-A12 we show the original sensor data for both camera and LiDAR, the reconstructed meshes with CADsim rendered by normal and RGB appearance, and then show the simulated sensor data generated when placing the assets in their original views. For camera simulation, we follow GeoSim [17], where given a scenario configuration, we first render the asset to the target view and then apply a post composition network and conduct occlusion reasoning to seamlessly blend the actor to real backgrounds. For LiDAR simulation, similar to [25, 26], we use a high-fidelity Pandar64 [20] simulator that conducts LiDAR ray-casting on the added actor according to the LiDAR calibration and remove points in the real LiDAR sweep that are occluded by the added actor. CADsim reliably constructs the nearby vehicles with high quality geometry and appearance, and have high fidelity with the original sensor data. These assets thereby provide a more accurate way of generating sensor data for simulation scenarios.

E.2 Mixed Reality with Realistic Animated Vehicle Insertion

The previous section demonstrated CADSim generates assets that match with the original sensor data. We can now insert these assets into existing scenarios and create new variations. As we have accurate 3D meshes, we can also render shadows for the inserted actor. We use the rasterization

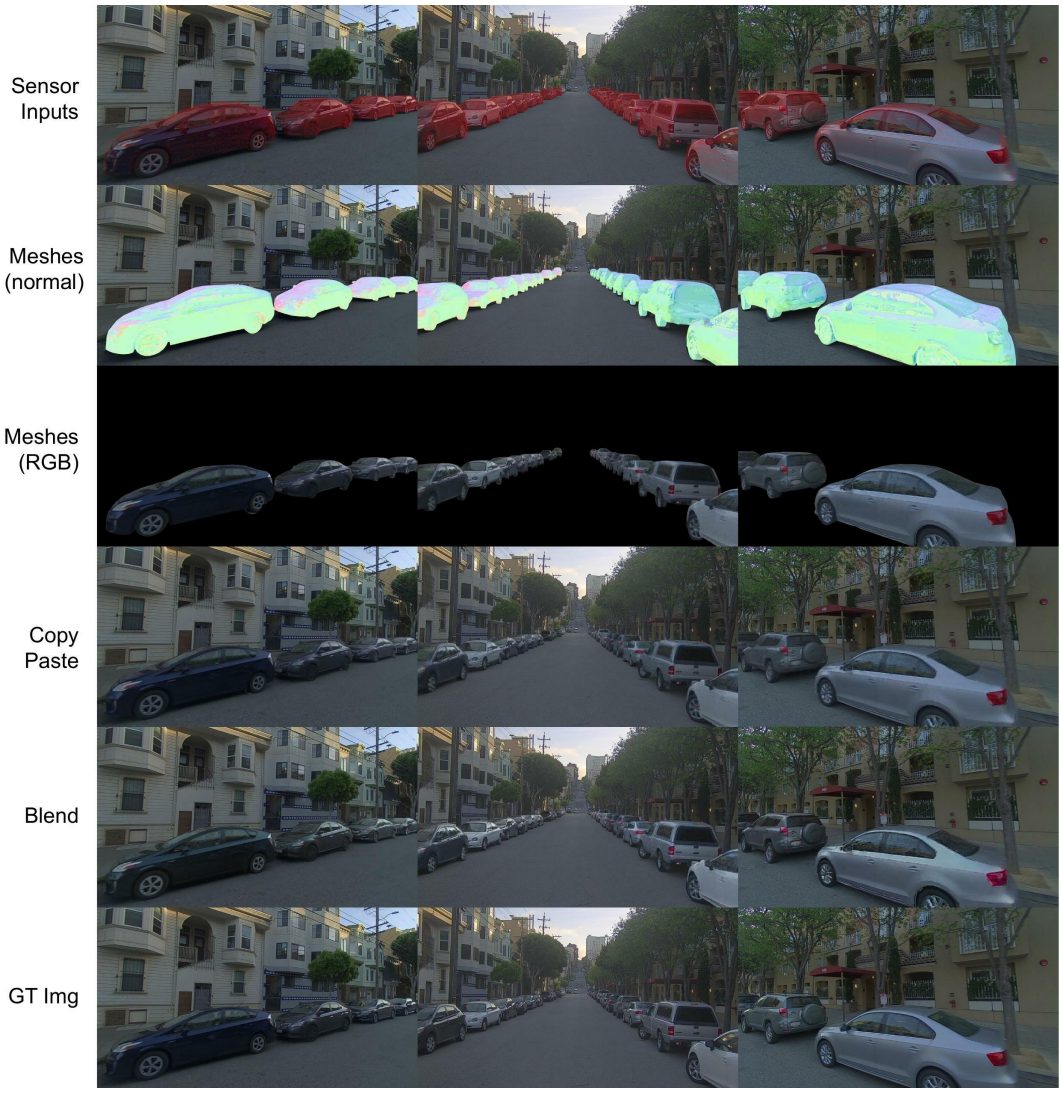

Figure A7: Reconstruction of all nearby vehicles and camera re-simulation on PandaSet Log028.

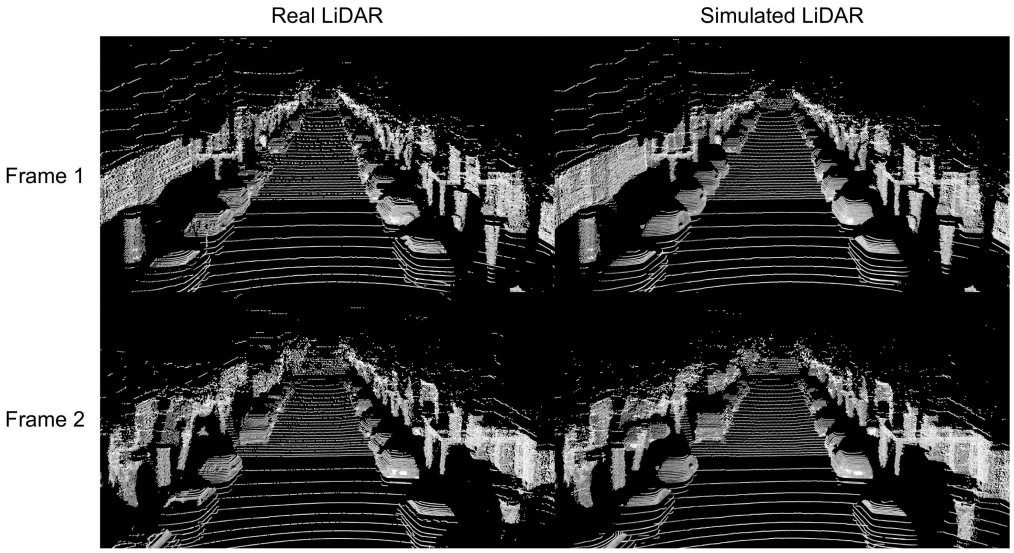

Figure A8: LiDAR re-simulation using CADSim assets on PandaSet Log028.

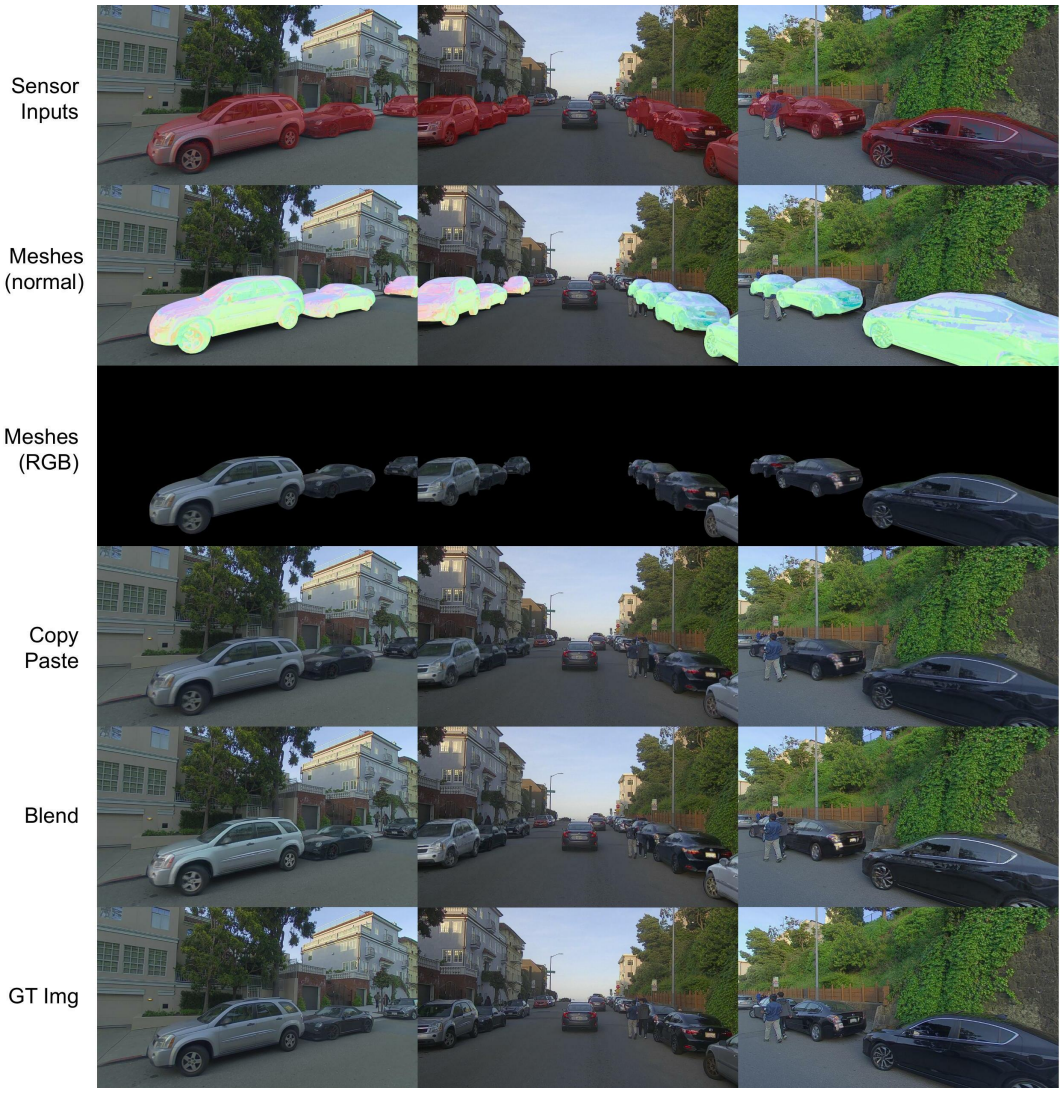

Figure A9: Reconstruction of all nearby vehicles and camera re-simulation on PandaSet Log030.

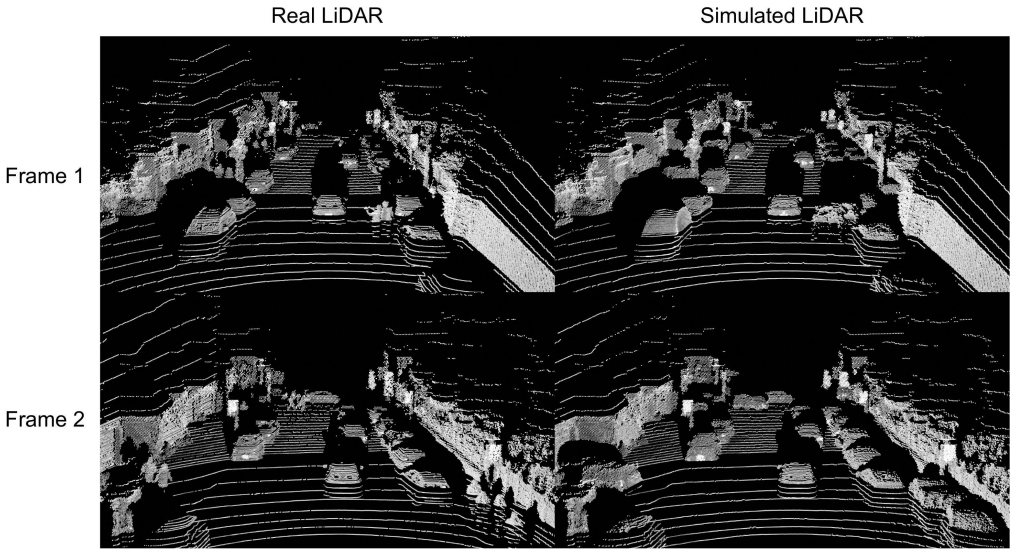

Figure A10: LiDAR re-simulation using CADSim assets on PandaSet Log030.

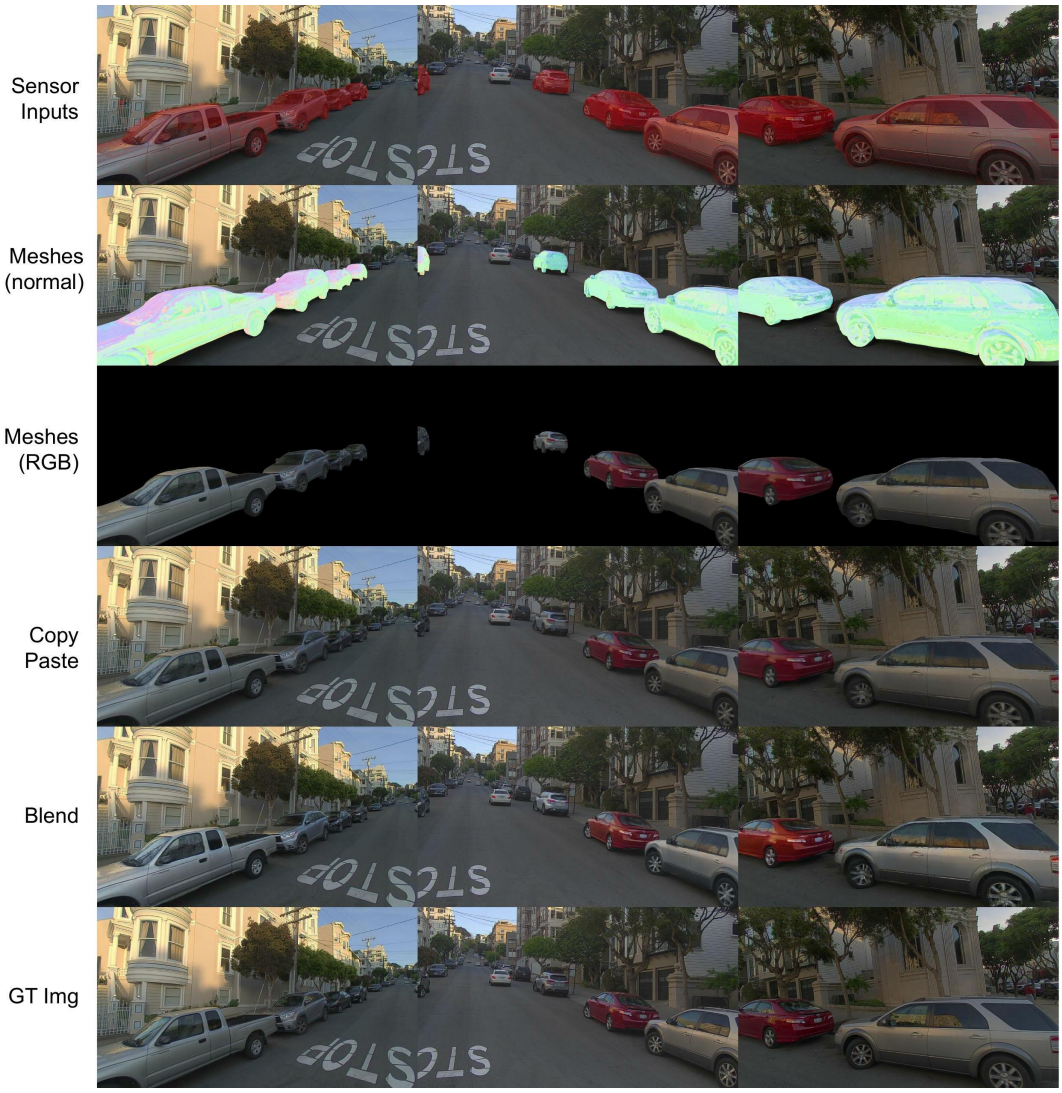

Figure A11: Reconstruction of all nearby vehicles and camera re-simulation on PandaSet Log139.

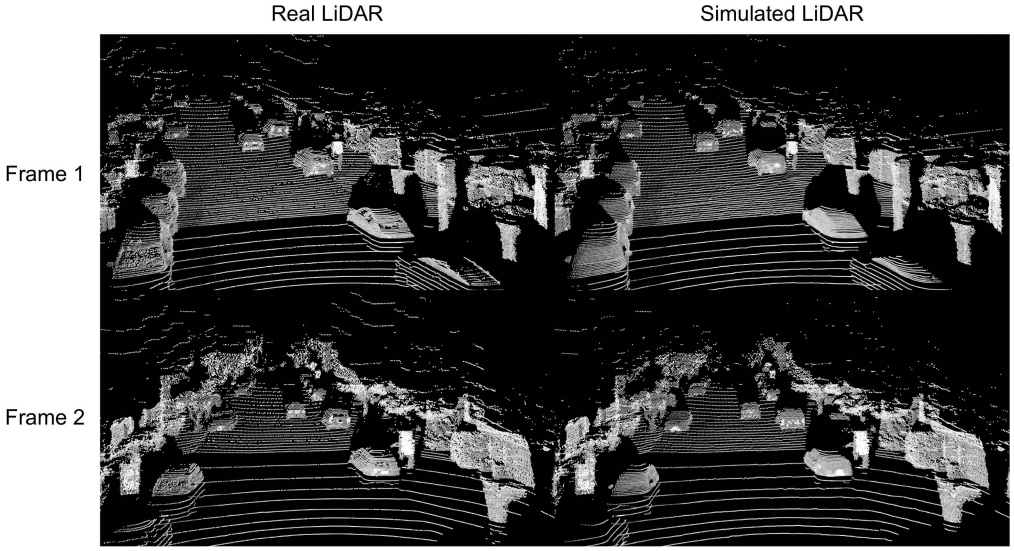

Figure A12: LiDAR re-simulation using CADSim assets on PandaSet Log139.

engine PyRender [27] to generate the shadow based on the geometry of the inserted actor, assuming a top-down light.

We now show how CADsim assets can be inserted into new scenes and can be manipulated to create variations. As shown in Figure A13, we render the CADsim asset at a new pose and blend the rendered image segment into the new scene. The position and rotation of the added car are manipulated. Since our assets are multi-sensor consistent, we can generate the LiDAR point cloud (left) and the camera image (right) for the modified scene. Both simulated sensor data look realistic. We notice that the wheels spin when moving forward and backward, and the wheels rotating when model turning.

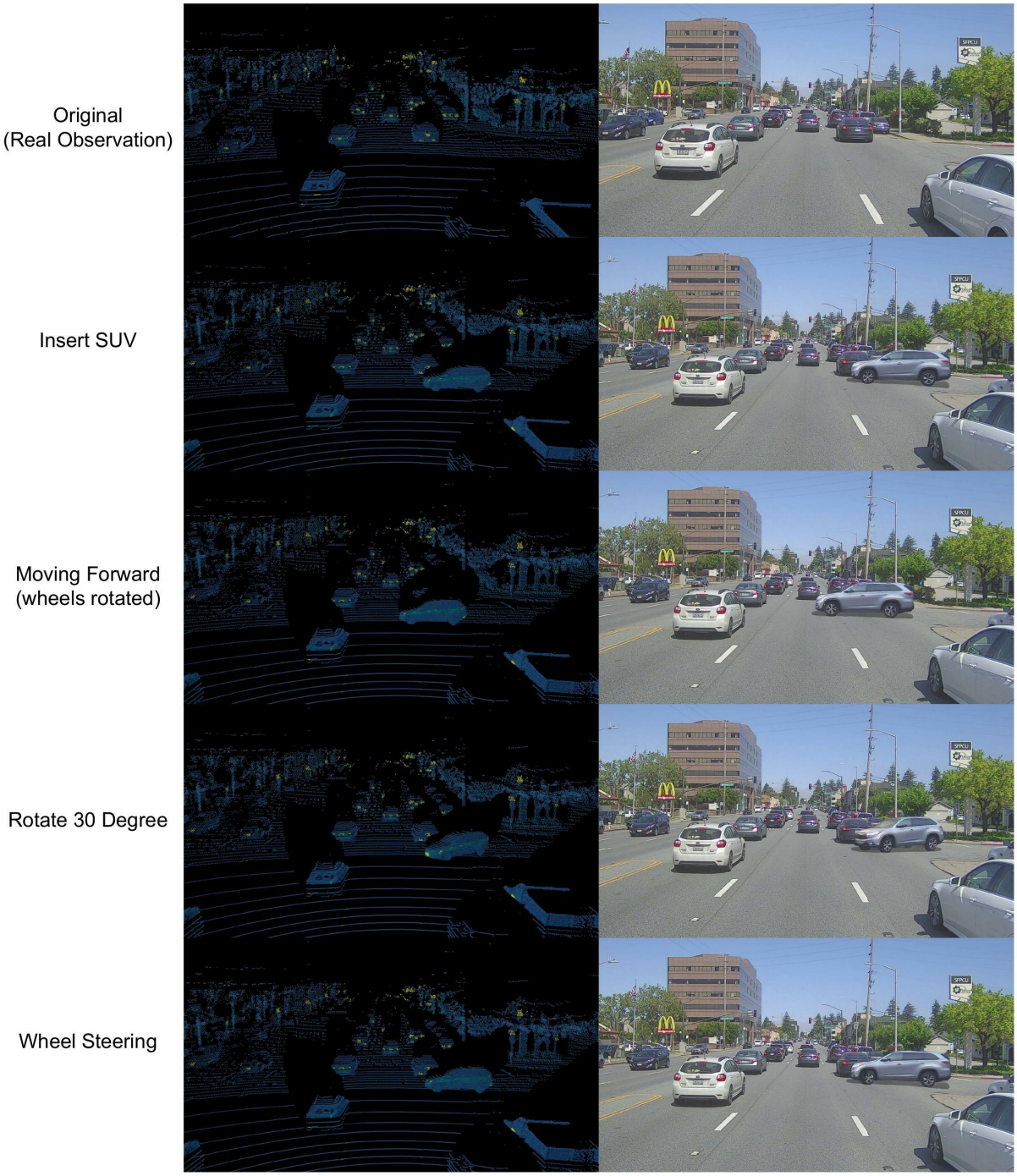

Figure A13: Mixed reality actor manipulation with articulated CADSim vehicle.

We can also generate interesting new scenarios with CADSim assets to test our autonomy systems. In Figure A14, we generate two safety-critical scenarios (left: an actor aggressively turning right into our lane; right: a moving vehicle aggressively changing two lanes at once) and show realistic image and LiDAR simulation. The simulated camera and LiDAR data are blended seamlessly into the original scenario, creating more interesting long-tail scenarios. Note that the occlusion and actor movement are physically plausible. We provide more examples of actor insertion in Figure A15.

Figure A14: Realistic multi-sensor simulation for safety-critical mixed reality scenarios.

Finally, we compared to directly inserting a CAD model into the original scenario (Figure A16), our inserted CADsim assets have more realistic appearance and harder to distinguish as simulated.

E.3 Texture Transfer and Synthesis

Finally, we show that our approach can align textures across different vehicle shapes, enabling texture transfer to create new asset variations. Unlike recent work (*e.g.*, AUV-Net [28]) that focuses on synthetic objects with unrealistic textures, we demonstrate the texture transfer across multiple actors in the real world and use it for realistic camera simulation. Figure A17 shows in each column the texture trasferred across different vehicle types, while each row shows the same vehicle shape having different textures. In Figure A18, we choose three nearby actors and transfer the texture from one actor to the other ones. Our assets are vertex-aligned with high-quality part correspondence across different shapes, allowing us for realistic and seamless simulation. This technique can also be used to generate diverse textured assets that have never been observed.

Figure A15: Realistic multi-sensor actor insertion simulation.

		Training view frame id	Testing view frame id	
Actor UUID	**Seq id**	**Left camera**	**Front camera**	**Front-left camera**
1d79eded-2fb0-4f89-ba35-323926f45ade	139	46-63	-	44-55
f7bd1486-1fbe-4f33-ba28-f00dae3e0298	139	57-77	38-53	54-69
526e2f5e-e294-415c-aad6-578d27921465	030	38-78	0-28	35-55
56e10a51-35ed-43b0-837c-cea8aff216cc	139	26-52	-	25-46
ba222d39-2f13-4849-8ff4-91e247d5cedf	120	12-37	0-6	0-25
2160d735-3fda-49f8-9bd9-e2cba3b51faa	038	34-47	0-30	27-41
1be68ce6-68c5-467f-abb1-fa5e03d1db7a	053	33-36, 40-49	0-29	25-41
2ee4d8f8-af0a-48f3-bb6c-ed479a7829e7	039	47-67	0-44	28, 31-59
94c06b25-d17a-4ee7-a2df-7faa619bee89	035	49-58, 60-61	4, 7, 9-42	47-51
5ce5fb69-038d-4f82-8c64-90b73c6f6681	030	17-62	0-17	0-45

Table A10: Actors and camera image frame ids used in PandaVehicle. '-' means no images are available for the associated camera.

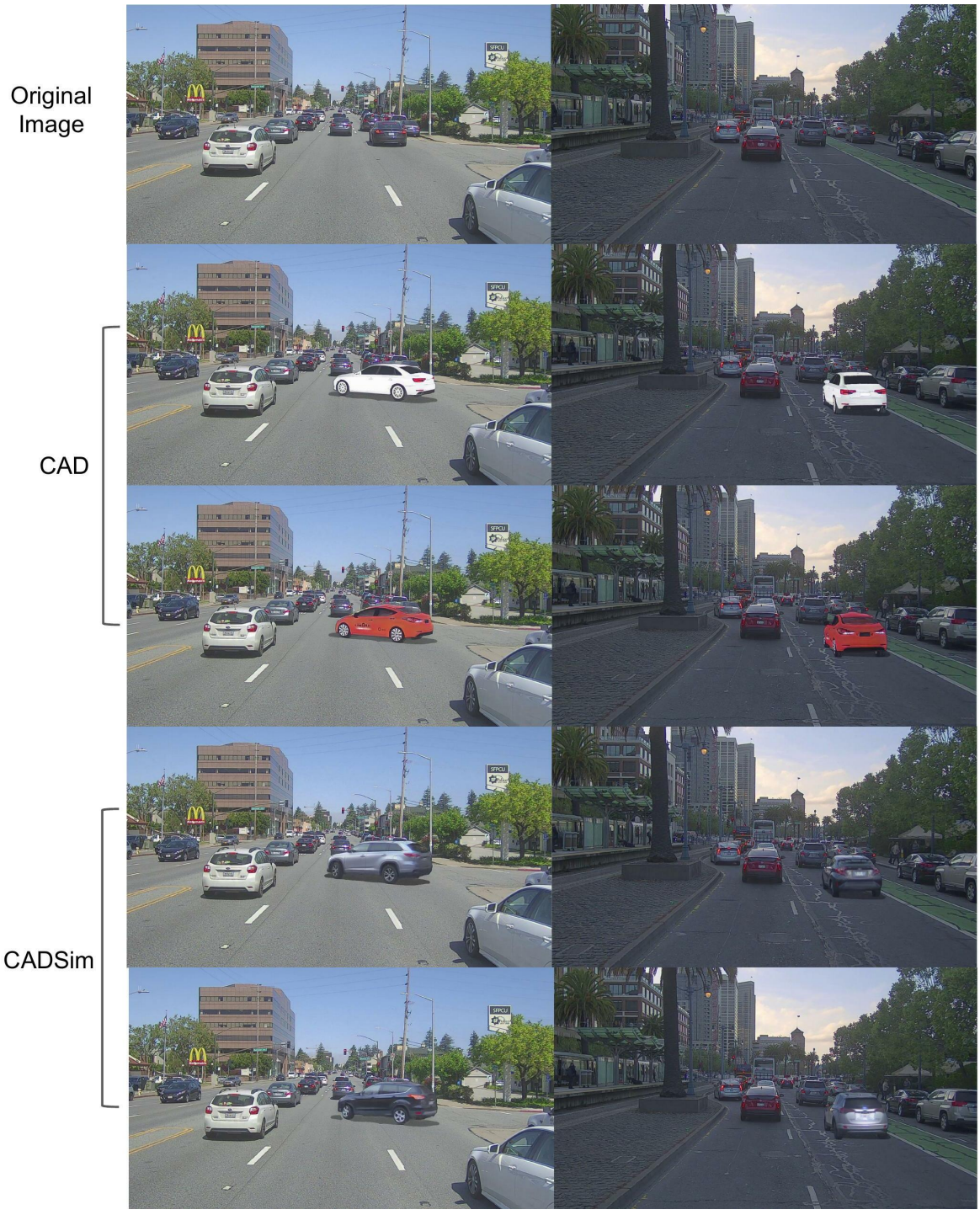

Figure A16: Comparison of camera actor insertion with CAD or CADSim assets. CADSim assets have more realistic textures and blend in better with the real image. CADSim enables generation of sensor data for completely new scenarios.

Figure A17: Texture transfer for reconstructed real-world assets.

Figure A18: Swapping vehicle textures in the real world.

Figure A19: Illustration of vehicle `526e2f5e-e294-415c-aad6-578d27921465` (in the red bounding box) in Pandaset sequence `030`. (Top) Image and LiDAR points from the left camera view used during mesh reconstruction. (Bottom) Front camera and front left camera views used during testing for the novel view synthesis task.

References

[1] B. Karis. Real shading in unreal engine 4. *Proc. Physically Based Shading Theory Practice*, 4 (3):1, 2013.

[2] J. Munkberg, J. Hasselgren, T. Shen, J. Gao, W. Chen, A. Evans, T. Müller, and S. Fidler. Extracting triangular 3D models, materials, and lighting from images. *arXiv preprint arXiv:2111.12503*, 2021.

[3] S. McAuley, S. Hill, N. Hoffman, Y. Gotanda, B. Smits, B. Burley, and A. Martinez. Practical physically-based shading in film and game production. In *ACM SIGGRAPH 2012 Courses*, pages 1–7. 2012.

[4] B. Walter, S. R. Marschner, H. Li, and K. E. Torrance. Microfacet models for refraction through rough surfaces. *Rendering techniques*, 2007:18th, 2007.

[5] S. Hill, S. McAuley, L. Belcour, W. Earl, N. Harrysson, S. Hillaire, N. Hoffman, L. Kerley, J. Patry, R. Pieké, et al. Physically based shading in theory and practice. In *ACM SIGGRAPH 2020 Courses*, pages 1–12. 2020.

[6] B. Nicolet, A. Jacobson, and W. Jakob. Large steps in inverse rendering of geometry. *ACM TOG*, 40(6):1–13, 2021.

[7] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. 2015.

[8] J. Huang, H. Su, and L. Guibas. Robust watertight manifold surface generation method for shapenet models. *arXiv preprint arXiv:1802.01698*, 2018.

[9] J. Huang, Y. Zhou, and L. Guibas. Manifoldplus: A robust and scalable watertight manifold surface generation method for triangle soups. *arXiv preprint arXiv:2005.11621*, 2020.

[10] K. Zhang, G. Riegler, N. Snavely, and V. Koltun. NeRF++: Analyzing and improving neural radiance fields. *arXiv*, 2020.

[11] T. Müller, A. Evans, C. Schied, and A. Keller. Instant neural graphics primitives with a multiresolution hash encoding. *arXiv preprint arXiv:2201.05989*, 2022.

[12] J. Y. Zhang, G. Yang, S. Tulsiani, and D. Ramanan. NeRS: Neural reflectance surfaces for sparse-view 3d reconstruction in the wild. In *NeurIPS*, 2021.

[13] P. Wang, L. Liu, Y. Liu, C. Theobalt, T. Komura, and W. Wang. NeuS: Learning neural implicit surfaces by volume rendering for multi-view reconstruction. In *NeurIPS*, 2021.

[14] S. Tulsiani, R. Tucker, and N. Snavely. Layer-structured 3d scene inference via view synthesis. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 302–317, 2018.

[15] F. Engelmann, J. Stückler, and B. Leibe. SAMP: shape and motion priors for 4d vehicle reconstruction. In *WACV*, 2017.

[16] K. Deng, A. Liu, J.-Y. Zhu, and D. Ramanan. Depth-supervised NeRF: Fewer views and faster training for free. *arXiv*, 2021.

[17] Y. Chen, F. Rong, S. Duggal, S. Wang, X. Yan, S. Manivasagam, S. Xue, E. Yumer, and R. Urtasun. Geosim: Realistic video simulation via geometry-aware composition for self-driving. In *CVPR*, 2021.

[18] S. Duggal, Z. Wang, W.-C. Ma, S. Manivasagam, J. Liang, S. Wang, and R. Urtasun. Mending neural implicit modeling for 3d vehicle reconstruction in the wild. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 2022.

[19] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. *ACM siggraph computer graphics*, 21(4):163–169, 1987.

[20] P. Xiao, Z. Shao, S. Hao, Z. Zhang, X. Chai, J. Jiao, Z. Li, J. Wu, K. Sun, K. Jiang, et al. Pandaset: Advanced sensor suite dataset for autonomous driving. In *ITSC*, 2021.

[21] A. Kirillov, Y. Wu, K. He, and R. Girshick. PointRend: Image segmentation as rendering. In *CVPR*, 2020.

[22] S. Tulsiani, N. Kulkarni, and A. Gupta. Implicit mesh reconstruction from unannotated image collections. *arXiv preprint arXiv:2007.08504*, 2020.

[23] B. T. Phong. Illumination for computer generated pictures. *Communications of the ACM*, 1975.

[24] K. He, G. Gkioxari, P. Dollár, and R. Girshick. Mask r-cnn. In *ICCV*, pages 2961–2969, 2017.

[25] S. Manivasagam, S. Wang, K. Wong, W. Zeng, M. Sazanovich, S. Tan, B. Yang, W.-C. Ma, and R. Urtasun. LiDARsim: Realistic lidar simulation by leveraging the real world. In *CVPR*, 2020.

[26] J. Wang, A. Pun, J. Tu, S. Manivasagam, A. Sadat, S. Casas, M. Ren, and R. Urtasun. Advsim: Generating safety-critical scenarios for self-driving vehicles. In *CVPR*, 2021.

[27] M. Matl et al. Pyrender, 2019.

[28] Z. Chen, K. Yin, and S. Fidler. AUV-Net: Learning aligned UV maps for texture transfer and synthesis. In *CVPR*, 2022.