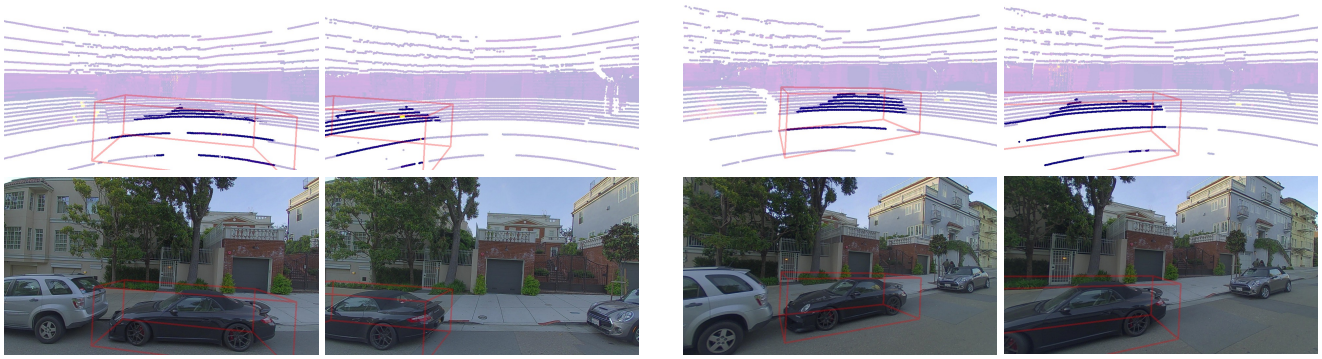
Reconstructing Objects in-the-wild for Realistic Sensor Simulation Supplementary Material

Ze Yang[1,2], Sivabalan Manivasagam[1,2], Yun Chen[1,2], Jingkang Wang[1,2], Rui Hu[1], Raquel Urtasun[1,2]
Waabi[1], University of Toronto[2]
{zeyang, manivasagam, yun, wangjk, urtasun}@cs.toronto.edu, rhu@waabi.ai



Training frames Testing frames

Fig. 1. Example training and testing frames of the PandaVehicle dataset.

In the supplementary material, we provide implementation details about our method and the experiments, additional qualitative visualizations, and limitations of our method. We first describe the details of our dataset in Sec. I. Then we describe the implementation details of our model in Sec. II. Next, we show additional details of the baseline models in Sec. III. After that, we showcase additional visualizations and results on downstream applications in Sec. IV. Finally, we analyze the limitations of our model in Sec. V.

I. Dataset and Experimental Settings

The PandaVehicle dataset contains 10 vehicles curated from PandaSet [1] with diverse shape and appearance under complex illumination and occlusion. The data was captured by a self-driving vehicle platform equipped with six cameras (front, front-left camera, left, front-right, right and back cameras) and two LiDARs (a 360° mechanical spinning LiDAR and a forward-facing LiDAR). All the sensors are calibrated. Each asset is captured when the self-driving vehicle (SDV) passes by. We employ the left camera for training and the front-left camera for evaluation, and we also use the 360° mechanical spinning LiDAR to train the model. Each asset has on average ∼24 views for training. Please refer to Fig. 1 and supplementary video for example frames. Since we focus on foreground vehicles, we use an off-the-shelf [2] algorithm to estimate the segmentation mask for each frame in the camera video. In addition to using the inferred segmentation masks as supervision for NeuSim, they are also used for evaluation to filter foreground pixels in the quantitative comparison. Please see Table I for detailed information for all 10 selected vehicles.

II. Implementation Details

Our network architecture is similar to IDR [3] and consists of three MLPs to encode the SDF, albedo, and material shininess. The SDF network f_{SDF} is a 8-layer MLP with a hidden size of 256 and Softplus activation. A skip connection is used to connect the input with the output of the fourth layer in the SDF MLP. We initialize the SDF MLP so it produces an approximate SDF of a sphere [4]. We initialize the learnable parameter $\beta = 20$ in the sigmoid-like function for conversion from SDF to α (Eqn. (3) in the main paper). The reflectance network $f_{\text{reflectance}}$ are implemented as 4-layer MLPs with a hidden size of 256 and ReLU activation. Both MLPs take as input the point location $\mathbf{x}$, normal $\mathbf{n}$ and the feature output of the last layer f_{SDF}. We use the Adam optimizer with learning rate of $5\text{e}{-4}$. The coefficients used for computing running averages are left at default values of $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 1\text{e}{-8}$. During training, the loss weights are $\lambda_{\text{lidar}} = 0.1$, $\lambda_{\text{mask}} = 0.1$, $\lambda_{\text{Eik}} = 0.1$ for SDF regularizer and $\lambda_{\text{sym}} = 1.0$ for structural symmetry prior, respectively.

1) Background Model: Outdoor scenes contain backgrounds (buildings, trees, sky, *etc*.) that are arbitrarily far away, which leads to resolution issues in volume rendering (Eqn. (1) in the main paper). NeRF [5] addresses this issue by utilizing Normalized Device Coordinates (NDC) parametrization, but it cannot cover the space outside the reference view's frustrum. To address this limitation, NeRF++ [6] propose to use an inverted sphere parameter-

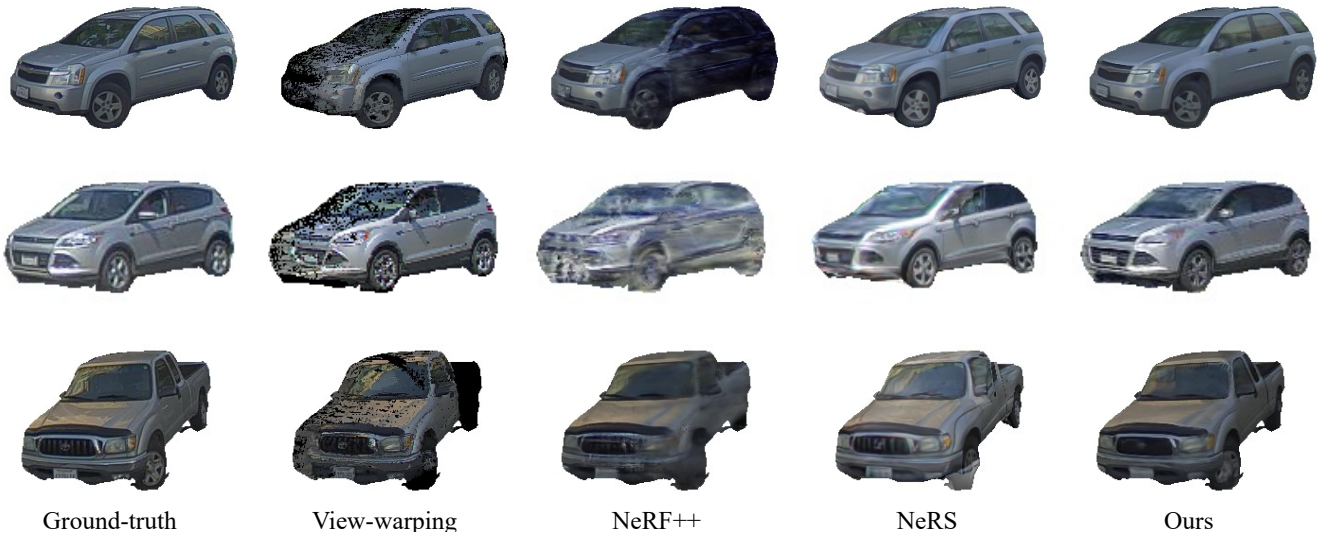


Fig. 2. Novel view synthesis results on challenging data.

ization [6] to participate space into an inner sphere volume and outer volume, where the foreground objects and all the cameras are normalized inside the inner sphere volume. This works well for 360° captures centered on close objects. However, this can be problematic when the foreground objects are far from the sensors or the sensors move along a long trajectory path. This formulation would require a large inner sphere volume to cover the space of foreground objects and all sensors, and this hurts the sampling resolution. In our scene representation, we assume the rendered ray $r(t)$ intersects with the object's region of interests (acquired from an annotation or an automatic detection output) at t_{near} and t_{far}. We divide the traversed space into foreground ($\{t_{\text{near}} < t < t_{\text{far}}\}$) and background. To compute the background scene radiance, we sample the ray's intersections with Multiple-Sphere Images (MPI) surrounding the object of interest. We generate the radii for MSI by linearly interpolating inverse depths.

III. Baseline Details

We compare our model with the state-of-the-art baselines: the volume rendering method NeRF++ [6], NeuS [7] and the inverse graphics model NeRS [8], NVDiffRec [9]. We choose these baselines as they model reflectance and work well in our outdoor setting. We also compare against geometry-based method LiDAR-guided view-warping [10] and SAMP [11]. Next, we next provide the implementation details of these baseline models.

A. Volume Rendering-based Baselines

1) NeRF++: For NeRF++ [6], we do scene normalization to move the camera's trajectory within the unit sphere. We adopt the same hyperparameters from the original code[1], except we train $100k$ iterations because the PandaVehicle dataset has sparser views and the model converges faster.

[1] https://github.com/Kai-46/nerfplusplus

2) NeuS: For NeuS [7], we follow the official code repository[2] to perform scene normalization to make the asset's region of interest fall inside the unit sphere, and model the background by a separate model similar to NeRF++ [6]. We train each asset for 200k iterations.

B. Inverse Graphics-based Baselines

1) NeRS: For NeRS[3] baseline, we downscale the input images $4\times$ (in contrast to $2\times$ in NeRSDF and NeRF++) due to GPU memory limitations. We initialized the cuboid template with the assets' coarse 3D dimensions and set the level of unit ico-sphere as 6. As the precise camera poses are given, 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 the PandaVehicle dataset and it takes longer to converge. To reconstruct missing parts and obtain better evaluation results on extrapolated views, we also applied symmetry constraints to the deformed textured meshes along the heading axis. The learning rate of Adam optimizer in the three stages is $1e^{-4}$.

2) NVDiffRec: NVDiffRec [9] is an efficient differentiable rendering-based 3D reconstruction approach that combines differentiable marching tetrahedrons and split-sum environment lighting. It achieves SotA performance on a wide variety of synthetic datasets with dense camera views. We follow the official code repository[4] and 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

[2] https://github.com/Totoro97/NeuS
[3] https://github.com/jasonyzhang/ners
[4] https://github.com/NVlabs/nvdiffrec

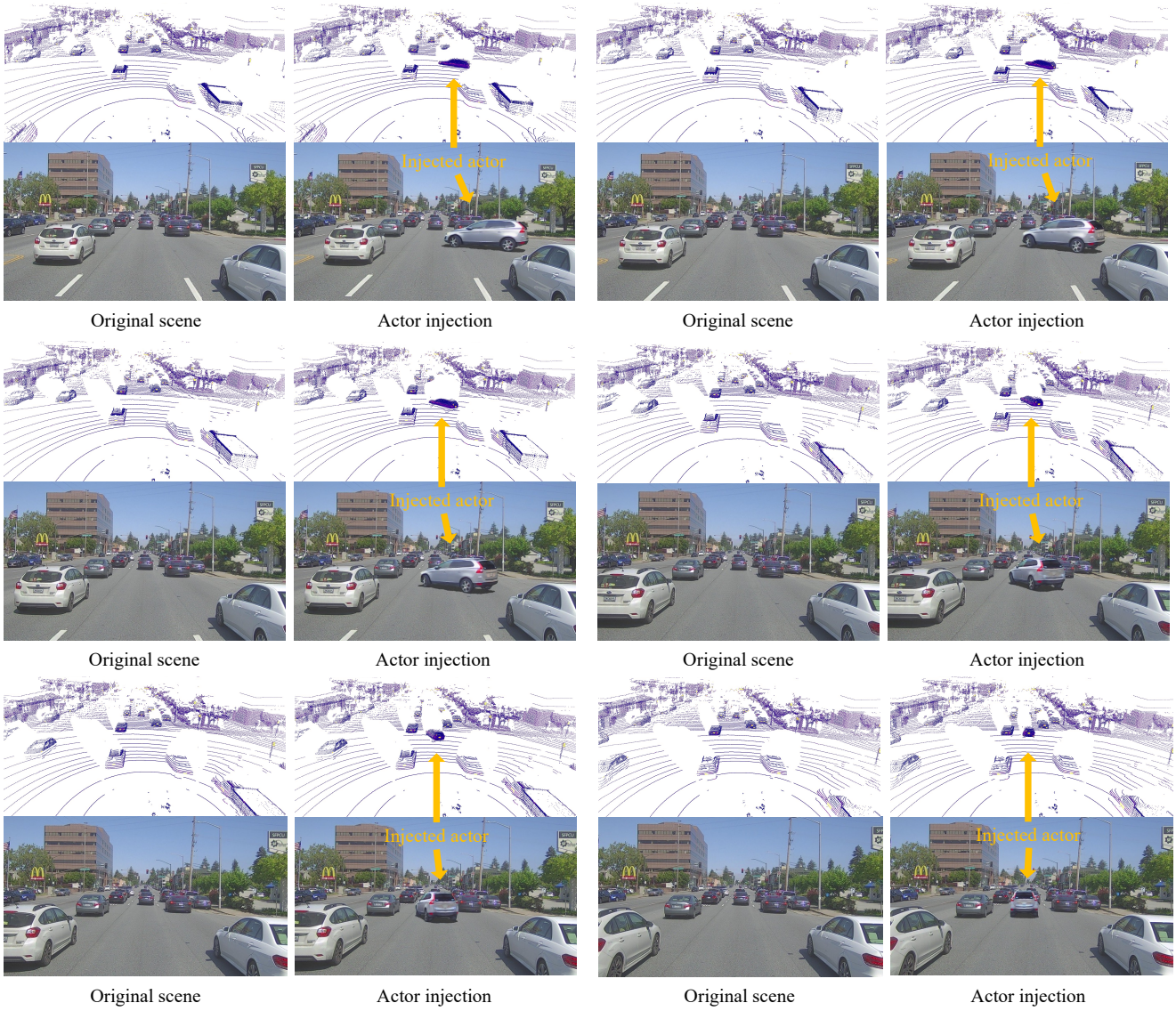


Fig. 3. We can generate consistent multi-sensor simulation using our reconstructed assets. The reconstructed asset performs a right turn and merges into the main road. We visualized LiDAR and camera data for sampled frames with/without the added actor. Time increases going from left to right and from top to bottom.

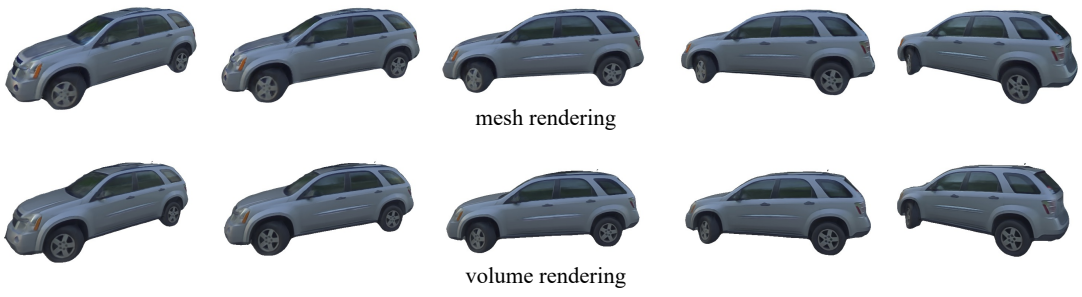


Fig. 4. Visual comparison of mesh rendering and volume rendering. From left to right we show the rotation of the asset from different viewpoints.

8) with a learning rate exponentially decayed from 0.03 to 0.003.

C. Geometry-based Baselines

1) LiDAR-guided View-warping: We first aggregate LiDAR points with the Iterative Closest Points (ICP) algorithm

Fig. 5. Visualization of shadows generated by different approaches. We render shadows for the added actor at two different poses. For neural rendered shadow (left image in each pair), we extract the shadow weight from NeuSim's rendered results with the ground. For rasterization-based shadow (right image in each pair), we use a rasterization engine to generate the shadow based on the geometry of the inserted actor, assuming a top-down light.

Fig. 6. Relighting results when rotating the environmental lighting map $\mathcal{E}$. Top: Asset example 1. Bottom: Asset example 2. From left to right we show the change in environmental lighting rotation.

to create a surfel representation for the asset. Given a testing viewpoint, we render the surfel asset to this viewpoint and generate the corresponding depth map. Using the rendered depth map, the source camera image and the camera calibration, we generate the object's texture using the inverse warping operation as in [12]. To choose which source image to warp to the target frame, we warp all available source images to the target view and choose the one with the highest overlap with the rendered surfels.

2) SAMP: For SAMP [11], we first processe the CAD library and make each mesh watertight and simplified, we compute volumetric SDFs for each vehicle in metric space (volume dimension $100 \times 100 \times 100$). Following [11], we apply PCA on the SDF volumes and set the embedding dimension as 25. In the inference time, we jointly optimize the shape latent code, a scaling factor on the SDF (handle difference shapes) and relative vehicle pose (handle rotation, translation) to fit the LiDAR points. We adopt a L1 loss on the SDF value 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. We use marching cubes to extract the mesh from the optimized SDF volume. We then use a differentiable renderer to optimize the 2D UV texture for each asset.

IV. Additional Results

We now provide more results and details of our model on in-the-wild data.

A. Novel View Synthesis

We show additional novel view synthesis results in Fig. 2. Since the recorded vehicles are far away from the cameras, we enlarge the images. Our model captures more fine-grained details and generalizes better.

B. Efficient Rendering

We can also render the baked asset efficiently using off-the shelf rasterization engines. Mesh rendering is on average three orders of magnitude faster than volume rendering (76.79 frames per second (FPS) vs 0.03 FPS for a 960 x 540 image resolution) and still provides good visual quality. Table II shows an image metrics comparison between the volume rendering result and mesh rendering result. The quantitative metrics show that the performance is competitive with other neural volume rendering baseline methods, while being much faster to render. Fig. 4 shows a visual comparison between volume rendering and mesh rendering. They look close to indistinguishable.

C. Sensor Simulation

Importantly, we can use our reconstructed assets to generate consistent multi-sensor simulation for self-driving. In Fig. 3, the injected NeRSDF vehicle asset performs a right turn and merges into the main road. We render the image segment using the NeuSim baked asset and a rasterization engine. Additionally, we render the asset's shadow to ensure the inserted actor region looks realistic. We experiment with two approaches. In the "neural rendered" approach, we extract the shadow weight from NeuSim's rendered results with the ground, and apply it to the background in the target scene. In the "rasterization-based approach," we use a rasterization engine and performed shadow mapping to generate the shadow mask for the inserted actor, assuming a top-down light (Fig. 3). Finally, a neural network is applied to blend the rendered actor and shadow with the background [12]. Fig. 5 shows a comparison between shadows generated by the neural-rendered approach and the rasterization-based approach. To render the simulated LiDAR we use the asset's mesh geometry and perform raycasting

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

TABLE I

ASSET ID, LOG ID AND CAMERA IMAGE ID FOR THE 10 SELECTED ASSETS FROM PANDASET [1].

Rendering	FPS ↑	PSNR↑	SSIM↑	LPIPS↓
Volume Rendering	0.03	**22.44**	**0.692**	**0.202**
Mesh Rendering	**76.79**	19.87	0.609	0.239

TABLE II

COMPARISONS OF VOLUME RENDERING AND MESH RENDERING.

according to the LiDAR sensor's extrinsics and intrinsics. We then convert the simulated LiDAR point cloud and real point cloud into a spherical depth-image representation and merge the point clouds to ensure realistic occlusion and LiDAR shadows [13], [14]. By using the same baked asset for both camera and LiDAR simulation, we ensure that the simulated data for both sensors match.

1) Control over appearance and illumination: We rotate the environmental lighting map $\mathcal{E}$ and show the rendering results under novel lighting conditions for two different assets in Fig. 6. Please see our video for additional visualizations. With the asset's texture and material properties constant, we can generate realistic variations of the asset views.

V. LIMITATIONS

Our model has difficulties reconstructing certain glass materials that reflect camera rays while LiDAR rays penetrate. This inconsistency of the sensor observations will cause artifacts on the surface. Additionally, our Phong-based reflectance model cannot handle complex reflection and refraction well, or infer shadows casted on the observed view. Given sparse sensor data, we plan to explore how to leverage more complex reflectance and lighting models that better handle these effects in future work. Our model also require good camera and LiDAR calibration to ensure alignment of observations across sensors.

REFERENCES

[1] 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. 1, 5

[2] A. Kirillov, Y. Wu, K. He, and R. Girshick, "Pointrend: Image segmentation as rendering," in *CVPR*, 2020. 1

[3] L. Yariv, Y. Kasten, D. Moran, M. Galun, M. Atzmon, R. Basri, and Y. Lipman, "Multiview neural surface reconstruction by disentangling geometry and appearance," in *NeurIPS*, 2020. 1

[4] M. Atzmon and Y. Lipman, "SAL: Sign agnostic learning of shapes from raw data," in *CVPR*, 2020. 1

[5] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, "NeRF: Representing scenes as neural radiance fields for view synthesis," in *ECCV*, 2020. 1

[6] K. Zhang, G. Riegler, N. Snavely, and V. Koltun, "NeRF++: Analyzing and improving neural radiance fields," *arXiv*, 2020. 1, 2

[7] 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," 2021. 2

[8] 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. 2

[9] 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," in *CVPR*, 2022, pp. 8280–8290. 2

[10] S. Tulsiani, R. Tucker, and N. Snavely, "Layer-structured 3d scene inference via view synthesis," in *ECCV*, 2018, pp. 302–317. 2

[11] F. Engelmann, J. Stückler, and B. Leibe, "Samp: shape and motion priors for 4d vehicle reconstruction," in *WACV*. IEEE, 2017, pp. 400–408. 2, 4

[12] 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. 4

[13] 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. 5

[14] J. Fang, X. Zuo, D. Zhou, S. Jin, S. Wang, and L. Zhang, "Lidar-aug: A general rendering-based augmentation framework for 3d object detection," in *CVPR*, 2021. 5