Supplementary Material:
GeoSim: Realistic Video Simulation via Geometry-Aware Composition for Self-Driving

In this supplementary material, we first describe additional technical details on 3D reconstruction, semantic retrieval, depth completion, post-composition, video simulation, dataset breakdown as well as A/B human test design (Sec. 1). Furthermore, we provide an extensive collection of qualitative results on comparisons, qualitative ablation on different modules of post-synthesis, and failure case analysis (Sec. 2). Finally, we recommend the reviewers to watch our supplementary video ($supp\_video.mp4$), which contains an overview of our approach and video simulation results. More specifically, the 4K vi

1. Additional Technical Details

1.1. 3D Reconstruction Network

Our 3D reconstruction network takes cropped images and LiDAR sweeps from multiple viewpoints. All cropped images are padded to have the same height and width and are then resized to 256×256. A small fully convolution network (as seen in Fig. 1) is used to extract image features. Note that in the figure, $Conv(K, S, C)$ refers to a convolution layer with kernel size K, stride S and output channel C. Padding is adjusted to make sure the output size is the same as the input. GroupNorm [9] with 32 channels per group is used after each convolution. ReLU is used as the non-linear activation. The final output is flattened to a single feature vector for each image.

To fuse the image features from multiple views, we design a fuse block (as shown on the right of Fig 1). In the block, multiview features are maxpooled to a single vector, which is then concatenated with each input. The augmented feature vector for each view is processed by a two layer MLP. The dimensions of all hidden layers and output layers of the MLP are 1024. We adopt 4 fuse blocks and maxpool to generate the final image-based feature vector. We adopt a standard PointNet [7] as the LiDAR feature extractor, which produces a feature vector with size 1024. Two linear layers consume the final concatenated LiDAR and image features to produce the mean-shape mesh deformation δV. The mean-shape mesh is initialized from an icosasphere with 2562 vertices and 5120 triangle faces.

As defined in the paper, we supervise the mesh reconstruction pipeline using silhoutte consistency loss, LiDAR consistency loss and the mesh regularization terms. The *regularizers* are applied to enforce prior knowledge over the resultant 3D shape, namely local smoothness on the vertices as well as normals.

$$\ell_{\text{regularization}}(\mathbf{M}_i) = \alpha\ell_{\text{edge}}(\mathbf{M}_i) \ + \ \beta\ell_{\text{normal}}(\mathbf{M}_i) \ + \ \gamma\ell_{\text{laplacian}}(\mathbf{M}_i)$$

The *edge regularization* term penalizes long edges, thereby preventing isolated vertices. $\ell_{\text{edge}}(\mathbf{M}_i) = \sum_{\mathbf{v}\in\mathbf{V_i}}\sum_{\mathbf{v}'\in\mathbf{N_v}}\|\mathbf{v}-\mathbf{v}'\|_2^2$, with $\mathbf{N_v}$ the first ring neighbour vertices of a given vertex $\mathbf{v}$. The *Laplacian regularization* [5] preserves local geometry and prevents intersecting mesh faces by encouraging the centroid of the neighbouring vertices to be close to the vertex: $\ell_{\text{laplacian}}(\mathbf{M}_i) = \sum_{v\in\mathbf{V_i}}\|\sum_{\mathbf{v}'\in\mathbf{N_v}}(\mathbf{v}-\mathbf{v}')\|_2^2$. The *normal regularization* enforces smoothness of the local surface normals, i.e., neighbouring faces are expected to have similar normal direction: $\ell_{\text{normal}}(\mathbf{M}_i) = \sum_{(i,j)\in\mathbf{N_F}}(1-\langle\mathbf{n}_i,\mathbf{n}_j\rangle)$, with $\mathbf{N_F}$ the set of all the neighbouring faces indices, and $\mathbf{n}_i$ the surface normal of a given face $\mathbf{f}_i$.

We set α, β, γ to 10.0 in our experiments. The model is trained for 200 epoches with 4 input views and batch-size 64 on 16 GPUs. We train it using Adam optimizer with initial learning rate 0.001 and decaied by 0.1 at 150, 180 epoch respectively. It takes about 6 hours to train.

1.2. Segment Retrieval Details

Single-view Segment Retrieval: To retrieve object-views for rendering in target-view, we first eliminate candidates from significantly different viewpoints (larger than 10° view changes to the target view). Then we rank the existing objects by considering their similarity in relative view angle θ and distance d from the camera.

$$\text{score}(\text{object}_{\text{tgt}}, \text{object}_{\text{src}}) = |\theta_{\text{tgt}} - \theta_{\text{src}}| + 5 \cdot \max(d_{\text{tgt}} - d_{\text{src}}, 0)$$

Objects are then sampled (as opposed to a hard max) according to a categorical distribution weighted by their inverse score.

Multi-view Segment Retrieval: To retrieve object with multi views for rendering in videos and multi-cameras, we consider the view range of source object. For every object, we first calculate the view ranges overlap with the target object $\Delta\Theta$, and filter out source objects with small overlap ($\Delta\Theta < 20^\circ$). Then we rank the existing objects by considering their overlap and minimum distance d_{src} from the camera.

$$\text{score}(\text{object}_{\text{tgt}}, \text{object}_{\text{src}}) = 2 \cdot \Delta\Theta + 5 \cdot \max(\min d_{\text{tgt}} - \min d_{\text{src}}, 0)$$

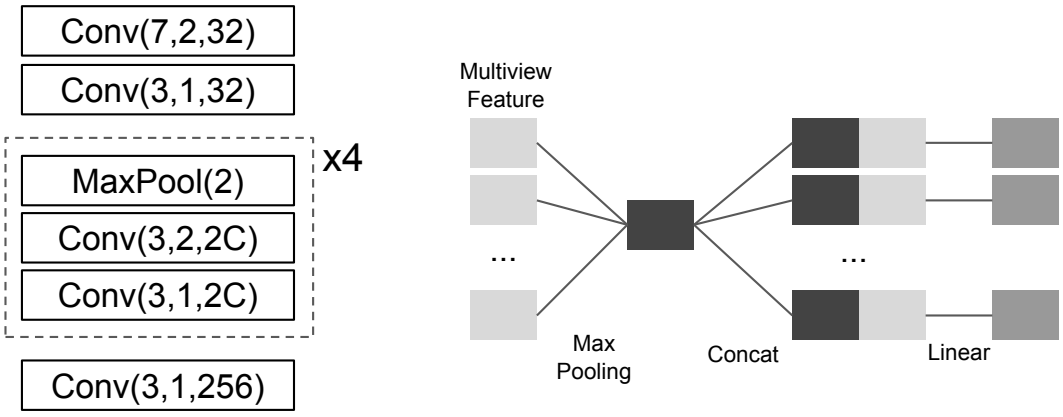

Figure 1: **3D reconstruction network architecture.** Left: Image feature extraction backbone; Right: Multi-view image fusion block.

Objects are then sampled (as opposed to a hard max) according to a categorical distribution weighted by their inverse score.

1.3. Depth Completion

To realistically place the simulated object in the new scene, we need to infer the occlusion relations between the simulated object and the existing scene elements. To do that, we compare the rendered depth of each simulated object's pixel with the corresponding pixel of the background scene. Since the partial LiDAR sweep belonging to the background scene is not dense enough, we first perform depth completion to generate a dense depth map for the corresponding scene. In this section, we first provide the ground-truth dense depth dataset preparation details, followed by the architectural details of the depth completion model.

Training Data Preparation: The UrbanData dataset has long trajectories of LiDAR sensor and camera sensor data with manually annotated detection labels. We first generate a dense LiDAR point cloud by aggregate the multi-sweep LiDAR sensor data. For the static background scene, we aggregate the multi-sweep data by compensating the ego-motion of the SDV. For all the objects (filtered out using the detection labels), we aggregate the multi-sweep data by transforming each of them to the corresponding object-coordinate system. For the dynamic objects, we additionally perform color-based ICP to better register the multi-sweep data. We further densify all the aggregated "vehicle" point clouds by converting each of them to a dense watertight mesh using a pre-trained implicit surface reconstruction model, DeepSDF [6]. The "non-vehilce" categories were densified by splatting each LiDAR point onto a triangular surfel disk. The aggregated point cloud is then rendered to the corresponding camera images to generate the dense depth maps. We first render the background scene aggregated points, followed by (instance-segmentation map aware) rendering of all the detected objects in ascending order of the objects median depth. We used manually annotated ground-truth instance-segmentation maps for depth map refinement.

Architecture Details: We use the generated dense-depth dataset to train a depth completion model. For the model, we use same network architecture as DeepLabV3 [1], except the first and the last convolution layers. The input to the model is a concatenated array of camera image, projected sparse depth image, projected sparse depth mask, and the dilated (dilated to 9-neighbouring pixels) sparse depth map. We intialize the DeepLabV3 architecture with the pre-trained COCO weights.

1.4. Shadow Generation Details

As we discussed in Sec. 4.2 in the paper, Fig. 2 shows the procedure that we applied to generate shadows. More qualitative comparison results on the difference between whether applied shadow generations are shown in Fig. 5.

1.5. Post-composition Synthesis Details

Training Data Preparation: Our synthesis network is trained on dynamic object images with per-pixel instance labels inferred by [3] in the target scene. Given a scene image I, we first sample a vehicle binary mask M in that scene, as well as its corresponding RGB segment $S = I \cdot M$. Then we apply data augmentation on the segment and mask to mimic the noisy input

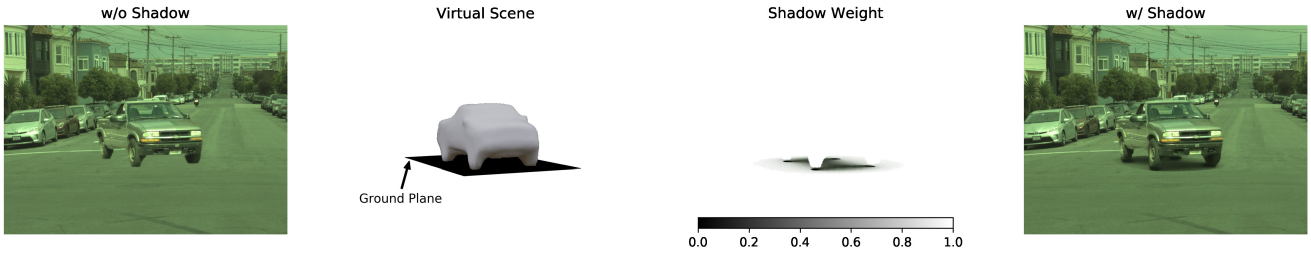

Figure 2: **Schematics of shadow generation**. (left to right): result without shadow, schematics of virtual scene, shadow weight (ratio of intensity between rendered image with inserted object and without inserted object), result with shadow

Figure 3: **Input data preparation for training the synthesis network**. From left to right: scene image I, object segment S and mask M and three random data augmentation including color-jitter, segment boundaries erosion-expansion and random mask in the boundary.

at the inference stage, with color inconsistency, missing texture and imperfect boundary. Specifically, we applied 1) random color jitter on the scene and segment separately, which randomly changes the brightness, contrast and saturation to mimic color inconsistency between foreground and target scene; 2) random erosion on the segment boundary from 3 to 20 pixels and a random dilation on the mask M from 3 to 40 pixels to blend object boundary naturally; 3) a random drop on the segment S with 0.1%–1% of total boundary pixels and applied random dilation on those samples to mimic missing textures of the inserted virtual object. Please refer to Fig. 3 for illustration of such data augmentation process.

Architecture Details: Our synthesis network architecture is inspired by [10]. One difference is that our network takes the instance segment mask as an additional input. Thus, the network takes as input the object segment, target scene and mask region. We crop a 512×512 region centered around the object center from the full scene.

Loss Functions: We apply perceptual loss [2] on the generated image I' and I, using the `conv3_3` feature activations in a pretrained VGG16 network F_v.

$$L_G^{\text{perc}} = \sum \|F_v(I) - F_v(G(I \cdot (1 - M^A), M^A, S^A))\|_1$$

A GAN loss is also applied to optimize the synthesis network.

$$L_G^{\text{gan}} = -\mathbb{E}_{z \sim P_z(z)}[D(G(I \cdot (1 - M^A), M^A, S^A))]$$

For the discriminator, we adopt the same loss as [10].

Inference: At the inference stage during simulation, our network takes the raw composition image from novel view warping and occlusion reasoning as input and produces natural blended results. Specifically, we crop a square region with the visible rendered segment in the centre, which is twice the size of the larger side of rendered vehicles ($size$). The cropped size is set to be 128×128 at least and 1024×1024 at most. Invisible pixels due to inverse warping are filled with zeros. Then we c erode

the rendered segment by $\frac{size}{64}$ pixels and dilate the mask by $\frac{size}{32}$ pixels. These inputs are fed into the synthesis network and the final outputs paste back to the original location.

1.6. Video Simulation Details

In order to simulate a realistic video footage with new dynamic objects inserted, we first select a subset of snippets from our dataset of real-world logs, each consisting of 50 consecutive frames sampled at 10Hz (for a total duration of 5s) to augment with up to 5 simulated objects using our approach.

For these relatively short sequences, we sample the object placement within the initial camera field-of-view in the first frame of the sequence and equip the object with a realistic path using the local lane graph, as illustrated in Figure 3 of the main text.

To retrieve an object for insertion from the 3D asset bank, we consider the view-range of each source object. For every object, we first calculate the view-range overlap with the target object, $\Delta\Theta$, and filter out source objects with small overlap ($\Delta\Theta < 20°$). Then we rank the existing objects by considering their overlap and minimum distance d_{src} from the camera.

$$\text{score}(\text{object}_{\text{tgt}}, \text{object}_{\text{src}}) = 2 \cdot \Delta\Theta + 5 \cdot \max(\min d_{\text{tgt}} - \min d_{\text{src}}, 0)$$

Objects are then sampled (as opposed to a hard max) according to a categorical distribution weighted by their inverse score.

We now describe how we use a set of heuristics-based behavior and lateral/longitudinal driving models to refine the path into a timestep-by-timestep *trajectory* of *kinematic states* comprising of location, orientation, velocity, and acceleration. This realistic trajectory simulation helps the added vehicle in the simulated video achieve realistic motion such as braking and acceleration. After executing segment retrieval and performing a final collision check, the object and its refined trajectory is then inserted into the log for inclusion. Additional objects can be added in a similar fashion.

We use a cost-based heuristic behavior model for determining lane change actions, a heuristic lateral model for the side-to-side motion of the object, and the Intelligent Driver Model [8]for the longitudinal movement. We define the following notation:

$$\begin{aligned} p_x &= \text{longitudinal position} \\ v_x &= \text{longitudinal velocity} \\ p_y &= \text{lateral position} \\ v_y &= \text{lateral velocity}. \end{aligned}$$

All models use a frequency, or reaction time, of 10Hz.

Behavior Model Details: Two actors are considered as colliding if the inter-vehicle distance between them is less than 2m. Let d_h, d_f denote the distances between the given vehicle and the headway (nearest front) vehicle or following (nearest back) vehicle respectively. Similarly, define c_h, c_f to be the headway and following costs. The cost functions we use are:

$$c_h = \begin{cases} 10^8 & d_h < 2 \\ \frac{10^4}{d_h} & \text{otherwise} \end{cases}$$

$$c_f = \begin{cases} 10^8 & d_f < 2 \\ \frac{10^2}{d_f} & \text{otherwise}. \end{cases}$$

The cost of making a lane change is 10^3, and the cost c_l for being close to the end of a lane when distance d_l from the lane end is

$$c_l = \tfrac{10^5}{d_l}.$$

Lateral Model Details: We use a simple heuristic for the target lateral speed that seeks to return to the lane centerline, bounded by a function of the longitudinal (forward-backward) movement. Specifically,

$$v_x = \min(-p_x, 0.1 v_y),$$

clipped so that the maximum acceleration magnitude is 3ms^{-2}.

Dataset Split	#Logs	#Frames used
Split A	5K	1.2M
Split B	7K	7K
Split C	2.8K	2.8K
Split D	2K	2K

Table 1: **UrbanData dataset splits.**

Task	SubTask	Dataset Split	Label Used
GeoSim	Mesh Reconstruction training	Split A	3D Box and mask from [3]
GeoSim	Post-Compostion training	Split A	mask from [3]
GeoSim	Depth Completion training	Split B	Aggregated Lidar
GeoSim	Whole pipeline	Split C	3D Box from [4]
Baselines	Training	Split B	GT Semantic Mask
Baselines	Test	Split C	GT Semantic Mask
Downstream	Sim2Real Training	Split C	GT Semantic mask and GeoSim Mask
Downstream	Sim2Real Evaluation	Split D	GT Semantic mask

Table 2: **UrbanData experimental setting.**

Longitudinal Model Details: We use the following parameters: minimum and maximum target speeds of $15\mathrm{ms}^{-1}$ and $25\mathrm{ms}^{-1}$ respectively, acceleration exponent of $4\mathrm{ms}^{-2}$, maximum acceleration and deceleration magnitudes of $5\mathrm{ms}^{-2}$, minimum gap of $2\mathrm{m}$, headway time of $1.5\mathrm{s}$, and default vehicle length of $4.5\mathrm{m}$.

1.7. UrbanData Dataset breakdown

In the UrbanData, we have roughly 16.5K labelled snippets. Out of these, 12K snippets have manually annotated 2D segmentation maps (one frame per snippet). As shown in Tab. 1, we divide these 16.5K snippets into multiple splits for training and testing various components of the GeoSim pipeline as well as the baslines. Tab. 2 maps each task to its corresponding training/ testing dataset split. Note that GeoSim does not need any gt labeled data. All labeled data are for training/evaluation purposes.

1.8. A/B interface and instructions

As discussed in Section 5.2 of the main paper, we performed a human study to demonstrate that GeoSim images appear more realistic compared to other baseline methods. For interface simplicity and ease of annotation, we performed pairwise comparisons instead of ranking. We also perform pair-wise comparison over ranking to mitigate user-bias, as all the baselines we compare against use the same object proposal method. An example interface is shown in Fig. 4. Each user would be provided two images, one generated with GeoSim and one generated with one of the baseline methods. Each image would be assigned with equal probability to the top or bottom location. Both the baseline and GeoSim would receive the same input real image and semantic segmentation. Based on the method and placement procedure, an object would be added to the scene, not necessarily in the same location (as seen in Fig. 4). We asked 16 users who are familiar with self-driving but who had limited or no knowledge of image simulation or of our method to select which image they prefer. On average, users annotated approximately 120 images, for a total of $\sim$1900 images. Here are the detailed instructions we provided along with each query: Detailed instructions:

> For each pair of images, please select the more realistic of the two by selecting either TOP or BOTTOM as the image label. Make sure to consider all relevant aspects of realism including but not limited to: visual appearance, lighting, shadows, relationship between elements within the image (ie occlusions), consistency in color, weather conditions, positioning on the road, and traffic regulations. We recommend clicking either on the images or on the top left blue arrow button to resize both images to fit the window. Click "next task" to move on. Note you will not be able to go back to a previous image, and the "Finish" button has no effect until all examples have been labeled, so there is no risk in accidentally clicking it. You can use the keyboard shortcuts 1,2 for TOP, BOTTOM, respectively.

We compute the success rate, i.e. whether our method is preferred over a specific baseline as: $\frac{\text{\# of times GeoSim selected}}{\text{\# of pairs with GeoSim and baseline}}$. We perform one-tailed binomial testing with the null hypothesis that GeoSim is not better than the baseline in over 50% of cases. The maximum p-value across all human evaluations is 1.64e-18 (when GeoSim was preferred in 211/279 pairs in the 2D synthesis ablation), indicating statistical significance.

2. Qualitative Results

2.1. Visual Comparisons

In addition to the qualitative results shown in Fig. 5 in the paper, we further showcase more qualitative comparisons among the previously-discussed image simulation baselines in Fig. 5. As can be seen from Fig. 5, GeoSim produces much more realistic and 3D aware simulated images, compared to the visually significant failures (e.g., blurred textures, implausible

Figure 4: **A/B test user interface**. Users must select which image ("TOP" or "BOTTOM") they consider most realistic.

placements, distorted shapes, boundary artifacts) produced by the other simulation methods. We also showcase more qualitative comparisons on public dataset Argoverse in Fig. 6.

2.2. Ablation Analysis

We also conduct qualitative ablation on three key components in post processing: occlusion reasoning, shadow and synnet. We compare GeoSim results against the one with the selected component removed. As seen in Fig. 8, with any one of the component being removed, the realism of the synthesis results drop significantly. Removing occlusion reasoning, the synthesized vehicles aren't able to conform with the existing scene elements. Removing Shadow, the synthesized vehicles seems to be off-the ground. Removing synnet, the synthesized vehicles show inconsistent illumination and color-balancing w.r.t the target scene and discrepancies at the boundaries. Please zoom in to see the detail.

In addition, we also show lane map and synthesis network ablations. Specifically, Fig. 9 shows results of GeoSim using random sampling instead of the lane map, where we uniformly sample empty ground locations according to the LiDAR sensor data and draw uniform orientations. Random sampling generates some interesting and useful edge cases, but it is not temporally consistent, making it not amenable to video simulation. The A/B test result shows that humans prefer our GeoSim at 95.5% of the time. As for human scores for GeoSim without the image synthesis network, users overwhelmingly (95.0%) prefer the full GeoSim.

2.3. Failure Cases

While GeoSim simulated images manages to produces realistic results in many cases, there is still space for potential improvements. In Fig. 7, we highlight four major failure cases: (1) Incorrect Occlusion Relationships in complicated scene, (2) Irregular reconstructed mesh, (3) Inaccurate object poses, usually caused by Map error and (4) Illumination failure by distinct illumination difference between rendered segment and target scene.

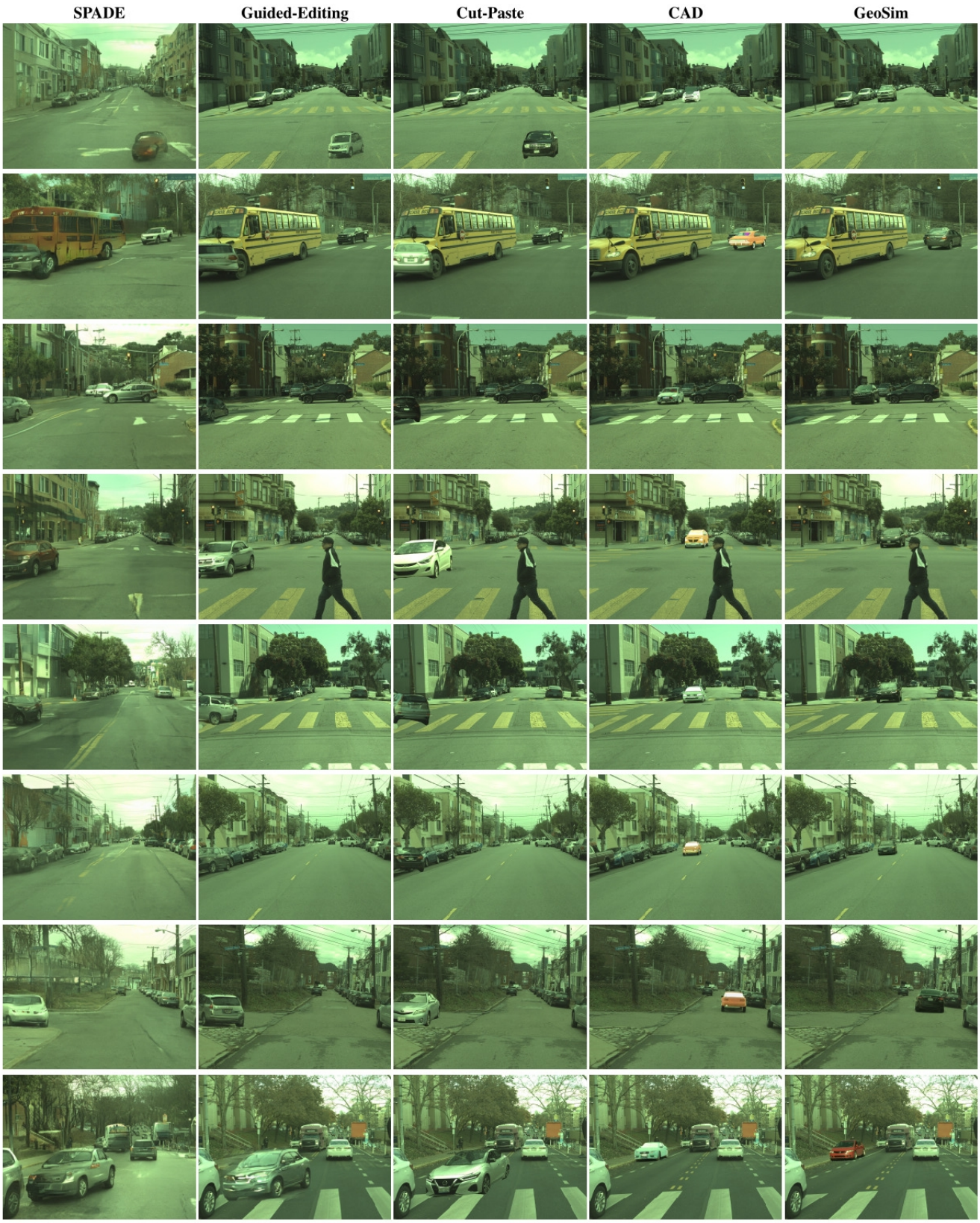

Figure 5: **Qualitative Comparison of Image Simulation approaches.**

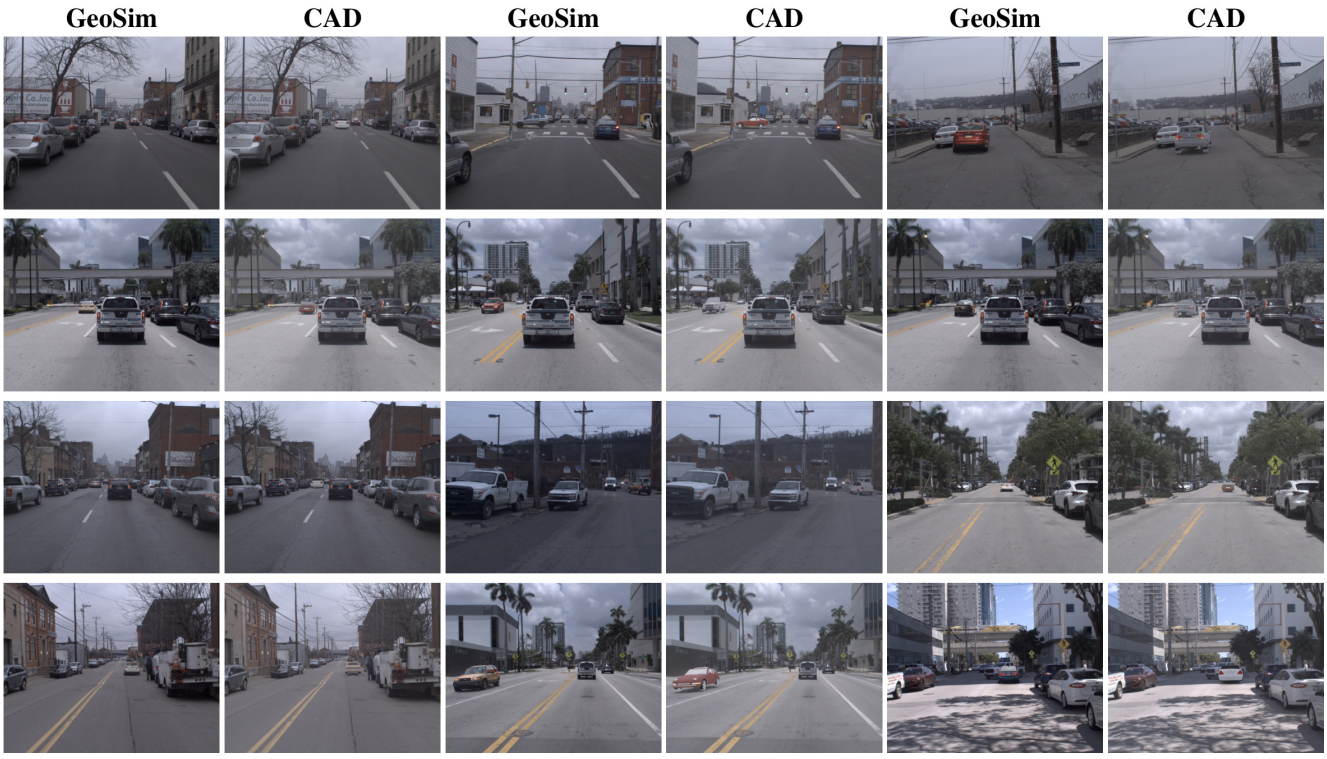

Figure 6: **Qualitative Comparison of Image Simulation approaches on Argoverse.**

Figure 7: **Qualitative visualization of the failure cases.**

Figure 8: **Qualitative ablation on the composition.**

Figure 9: **GeoSim results without lane map.**

References

[1] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking atrous convolution for semantic image segmentation. *arXiv preprint arXiv:1706.05587*, 2017. 2

[2] Justin Johnson, Alexandre Alahi, and Fei-Fei Li. Perceptual losses for real-time style transfer and super-resolution. *CoRR*, 2016. 3

[3] Alexander Kirillov, Yuxin Wu, Kaiming He, and Ross Girshick. Pointrend: Image segmentation as rendering. In *CVPR*, 2020. 2, 5

[4] Ming Liang, Bin Yang, Wenyuan Zeng, Yun Chen, Rui Hu, Sergio Casas, and Raquel Urtasun. Pnpnet: End-to-end perception and prediction with tracking in the loop. In *CVPR*, 2020. 5

[5] Andrew Nealen, Takeo Igarashi, Olga Sorkine, and Marc Alexa. Laplacian mesh optimization. 2006. 1

[6] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 165–174, 2019. 2

[7] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *arXiv*, 2016. 1

[8] Jens Schulz, Constantin Hubmann, Julian Löchner, and Darius Burschka. Interaction-Aware Probabilistic Behavior Prediction in Urban Environments. *arXiv*, 2018. 4

[9] Yuxin Wu and Kaiming He. Group normalization. In *ECCV*, 2018. 1

[10] Jiahui Yu, Zhe Lin, Jimei Yang, Xiaohui Shen, Xin Lu, and Thomas Huang. Free-Form Image Inpainting with Gated Convolution. *arXiv*, 2019. 3