

UrbanCAD: Towards Highly Controllable and Photorealistic 3D Vehicles for Urban Scene Simulation

Supplementary Material

This appendix details our method, implementation, experimental designs, additional experiment results, utilized resources, and broader implications. We first detail how to retrieve and optimize the CAD models in Section 8.1, Section 8.2, Section 8.3, Section 8.4, and Section 8.4, and then we show the process of urban lighting estimation in Section 8.6 and background reconstruction in Section 8.7. In Section 9, we provide details on experiment designs including baselines implementation (Section 9.1), synthetic data generation (Section 9.2), and perception systems implementation (Section 9.3). We also show more results and implementation details of our functionality in (Section 10). Finally, we report additional experiments and analysis in (Section 11).

8. UrbanCAD Implementation Details

8.1. CAD Model Filtering

Our method requires the CAD models to have correct material index assignment to support automatic coloring. However, we observe that in free CAD model libraries, there are small parts of handcrafted CAD models without proper material index designs. To this end, we design a script to filter the unqualified CAD models automatically or with a small amount of user interface based on the material design.

8.2. Pose Matching

Following [11, 14], we choose the CAD model rendering poses based on the DINO [4] feature similarity with the reference image. First, we crop the vehicles from both the reference image \mathbf{I}_{ref} and 360° retrieved CAD model renderings $\{\mathbf{I}_{cad}^k\}_{k=1}^M$, where $M = 360/A$ is the number of rendering views, and resize them to the same resolution. Then, we compute the DINO feature maps [4] for both vehicle image in reference view and CAD models rendering results using the DINO-ViT encoder \mathcal{E}_{DINO} : $\mathbf{F}_{ref} = \mathcal{E}_{DINO}(\mathbf{I}_{ref})$, $\{\mathbf{F}_{cad}^k\}_{k=1}^M = \mathcal{E}_{DINO}(\{\mathbf{I}_{cad}^k\}_{k=1}^M)$. Finally, we compute the L2 distances between the \mathbf{F}_{ref} and the $\{\mathbf{F}_{cad}^k\}_{k=1}^M$ and select the rendering that has the minimum L2 distance with the vehicle in the reference view. In our experiment, we find this approach can achieve accurate pose-matching results regardless of appearance and geometry differences between the retrieved CAD models and reference vehicles. The quality results of our pose-matching method are shown in Fig. 8.

8.3. Part-level Material Prior Retrieval

Since the retrieved CAD models usually have an unsatisfactory appearance, simply using Grounded SAM to segment the CAD model renderings will lead to many failure cases. ControlNet can translate primitives like edges into realistic pictures. Therefore, we propose to use ControlNet to augment the CAD model renderings and ensure the accurate segmentation of Grounded SAM. Specifically, we first extract edges from the material index maps rendered in 360°. Then, we input edges into a canny-based pre-trained ControlNet model and obtain the augmented multi-view images. Note that this canny-based ControlNet translation does not affect the position of the components. After that, we use Grounded SAM to segment the 360° augmented images with component text prompts like windows and wheels. Once we get the multi-view segmentation results, we first select the rendering with the highest mean mask confidence. Then, we calculate the material index masks that have an intersection with the segmented mask. We define them as active materials \mathbf{Mat}_{act} . We calculate the masks of each active material \mathbf{Mat}_{act} in material index map \mathbf{M}_{ind} and in the Grounded SAM segmentation map \mathbf{M}_{seg} . We then compute the IOU between \mathbf{M}_{ind} and \mathbf{M}_{seg} . If the IOU is larger than the IOU threshold (we set the IOU threshold as 0.5), the material will be classified into the corresponding component. We illustrate our method in Fig. 10.

8.4. Material Design Merging Using DINO Feature

Since the bodies of some vehicles are composed of many small components in the CAD models, only retrieving and optimizing materials for the largest part will lead to unsatisfactory results. However, Grounded SAM sometimes can't recognize tiny components like vehicle lights. Simply regarding all remaining parts after component recognition as car bodies will also lead to inaccurate material assignment. To this end, we utilize the DINO corresponding points proposed in [4] to merge the small components in the CAD models. Specifically, we first segment the known components in the input image using Grounded SAM. Then, we calculate the corresponding points between the remaining parts in the input images and the CAD model renderings. Since the remaining parts in the input images are the car body, the corresponding parts in the CAD model renderings are the car body as well. Besides, with a suitable setting of corresponding points' numbers, tiny components not belonging to car bodies will not be wrongly merged. During our experiment, this kind of merging produces good mate-



Figure 7. **More qualitative results** on KITTI-360 for novel view synthesis from reference (Ref.) and rotated (rot.) viewpoints.



Figure 8. Pose matching results. It shows that the pose of retrieved CAD models (second row) can match accurately with the pose of the input vehicle images (first row) despite the large difference between appearance and geometry.



Figure 9. **Symmetric material design.** Different colors represent different material indexes.

rial assignment results on tiny components of CAD models.

8.5. Material Optimization

Since there is no exact correspondence between rendered and reference pixels, we use a part-level loss ℓ_{stat} by minimizing the difference between the mean and variance of the corresponding parts following [71]:

$$\ell_{mean} = |\mu(\mathbf{I}_{ref} \cdot \mathbf{S}_{ref}[\mathbf{c}]) - \mu(\hat{\mathbf{I}}_{render} \cdot \mathbf{S}_{cad}[\mathbf{c}])| \quad (4)$$

$$\ell_{var} = |\sigma^2(\mathbf{I}_{ref} \cdot \mathbf{S}_{ref}[\mathbf{c}]) - \sigma^2(\hat{\mathbf{I}}_{render} \cdot \mathbf{S}_{cad}[\mathbf{c}])| \quad (5)$$

$$\ell_{stat} = \ell_{mean} + \ell_{var} \quad (6)$$

where $\hat{\mathbf{I}}_{render}$ is the CAD model rendering after pose matching, $\mathbf{S}_{ref}[\mathbf{c}]$ and $\mathbf{S}_{cad}[\mathbf{c}]$ are the segmentation masks

of the component \mathbf{c} in the reference view and CAD model rendering.

To match the patterns of the reference view, we use a masked VGG loss ℓ_{vgg} using Gram matrices [20] to enhance visual similarity:

$$\ell_{vgg} = |Gram(\mathbf{I}_{ref}, \mathbf{S}_{ref}[\mathbf{c}]) - Gram(\hat{\mathbf{I}}_{render}, \mathbf{S}_{cad}[\mathbf{c}])| \quad (7)$$

To further match the color of the reference vehicles, we add a masked RGB loss ℓ_{rgb} on the overlap region between components in the reference view and CAD model rendering:

$$\ell_{rgb} = |\mathbf{I}_{ref} \cdot \mathbf{S}_{overlap} - \mathbf{I}_{cad} \cdot \mathbf{S}_{overlap}| \quad (8)$$

The total loss function is shown as below:

$$\ell_{total} = \lambda_{stat}\ell_{stat} + \lambda_{vgg}\ell_{vgg} + \lambda_{rgb}\ell_{rgb} \quad (9)$$

In our experiment, we set the λ_{stat} to 0.1, the λ_{vgg} to 1, the λ_{rgb} to 1. Note that spatially varying roughness parameters are difficult to optimize from single-view images due to limited highlight observations. Handcrafted procedural material graphs provide photorealistic spatially varying effects, so the roughness parameters of the retrieved material prior are fixed during optimization, as in [71]. Besides, we observe two types of materials with distinct spatially varying effects in car bodies depending on whether the vehicles are painted or not, as shown in Fig. 13. To best fit the observation, we recommend selecting the corresponding car body material prior via the user interface.

8.6. Spatially Varying Lighting Estimation Based on Fisheye Images

As shown in Fig. 14, to obtain spatially varying lighting, we first stitch 2 fisheye images into an LDR panorama. Then

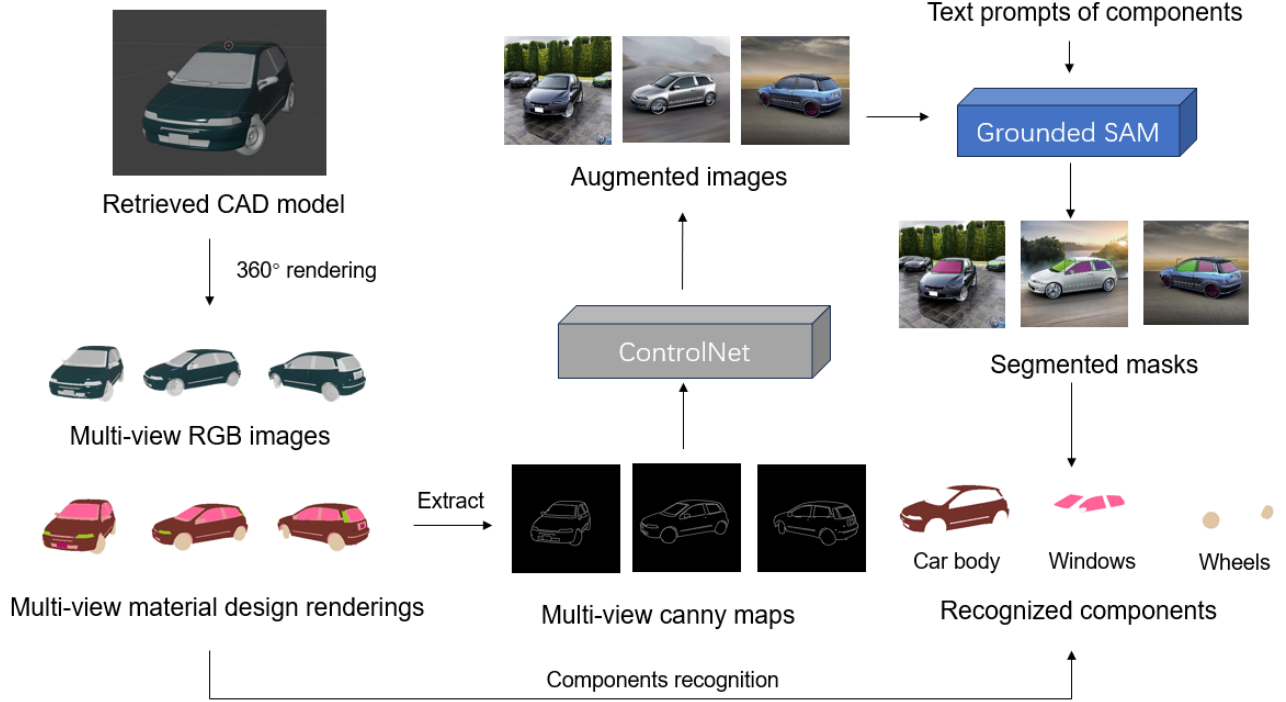


Figure 10. Illustration of Semantic-based Part-aware Material Prior Retrieval Module. To accurately recognize the semantic meaning of the retrieved CAD model for material prior retrieval, we first render the multi-view material designs and convert them to the canny maps. Subsequently, we use the canny-based ControlNet [78] to produce multi-view augmented images. Note that the components’ locations in augmented images are aligned with the corresponding material design renderings. After that, we use Grounded SAM [52] and components’ names (e.g. windows) to segment the components in the augmented images and obtain multi-view segmented masks with corresponding components’ meanings. Finally, we utilize these segmented masks to recognize the material indexes of corresponding components in the material designs.

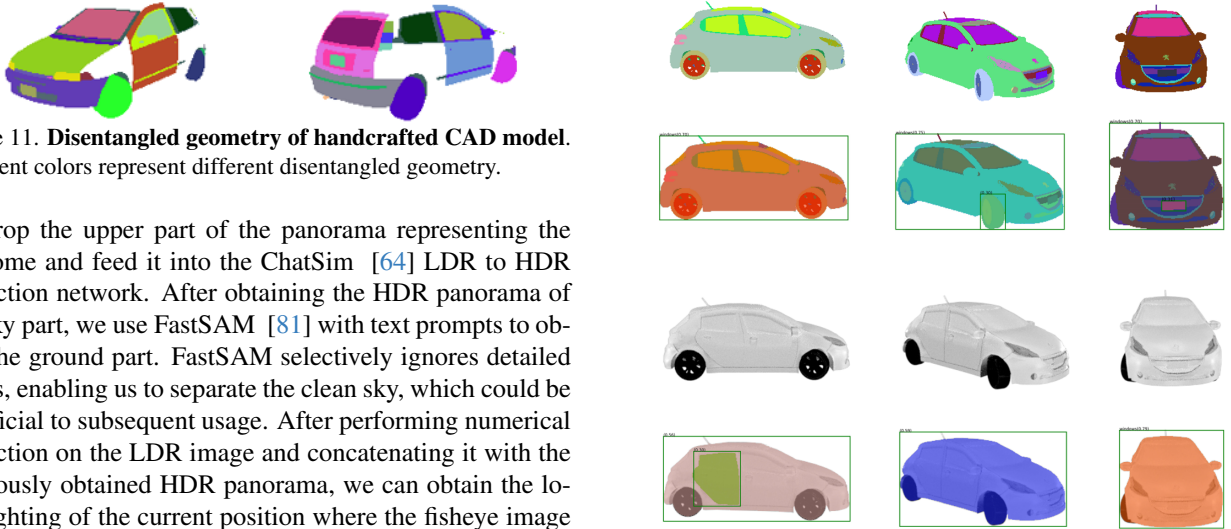


Figure 11. **Disentangled geometry of handcrafted CAD model.** Different colors represent different disentangled geometry.

we crop the upper part of the panorama representing the skydome and feed it into the ChatSim [64] LDR to HDR prediction network. After obtaining the HDR panorama of the sky part, we use FastSAM [81] with text prompts to obtain the ground part. FastSAM selectively ignores detailed pixels, enabling us to separate the clean sky, which could be beneficial to subsequent usage. After performing numerical correction on the LDR image and concatenating it with the previously obtained HDR panorama, we can obtain the local lighting of the current position where the fisheye image is captured.

8.7. Background Reconstruction using 3DGS

We employ the HUGS [82] to reconstruct the background of urban scenes. This process involves utilizing multi-view ap-

Figure 12. **Quality results** of part-recognition based on randomly colored material design (top) and retrieved CAD renderings without ControlNet augmentation (bottom) with the text prompt of "windows".



Figure 13. Two types of car body materials with different roughness. Vehicles in the left column are painted and vehicles in the right column are not painted.

pearance observations and pseudo-semantic labels obtained from InverseForm [9]. The HUGS is trained for a total of 30,000 iterations, using two front-perspective cameras and two side-look fisheye cameras in each sequence. Each sequence encompasses 40 frames both prior to and following the target frame. For this reconstruction process, we adhere to the configurations defined by the HUGS. Notably, when converting a static car into our optimized CAD model, we can use inpainting methods [74] for background inpainting to animate the car without leaving holes in the ground.

9. Implementation Details of Experiments

9.1. Baselines Implementation

During appearance comparison, we evaluate 1800 images of 30 models rendered from 360° views and report FID/KID scores comparing with 1800 reference images collected from [67]. Besides, we report the LIPIS scores by comparing the difference between input reference vehicle images and CAD renderings under matched poses.

PixelNeRF. PixelNeRF [73] is an image-based reconstruction method using a conditional implicit function. It supports single-view reconstruction tasks on real-world images. We use the official model pre-trained on ShapeNet [10] to evaluate the performance. We input our single-view images to the PixelNeRF and rendered the reconstructed neural radiance field in 360° with 180 frames.

Wonder3D. Wonder3D [42] is a image-based single view 3D generation method using diffusion priors. We use the official pretrained model to evaluate its single-view generation quality on our input images.

LRM. LRM [26] is a conditional implicit function based single view 3D reconstruction method with large scale training. Since the official LRM implementation hasn't been open-sourced, we use the open-sourced implementation OpenLRM [24]. When inferring on single view image, we simply use its open-sourced pre-trained model.

HUGS. As described in 8.7, we employ HUGS to reconstruct the urban scene, including the target vehicle. The extraction of the target vehicle requires identifying the specific Gaussians that constitute the vehicle. Fortunately, our approach achieved the 3D semantic reconstruction facilitated

Labor Cost	Method	FID↓	KID↓	LIPIS↓
High	OpenShape [39]	73.10	0.0453	0.5761
Middle	UrbanCAD (w/o opt.)	81.05	0.0567	0.6174
Low	OpenShape* [39]	116.36	0.0990	0.6676
Middle	UrbanCAD (Ours)	62.80	0.0479	0.5242

Table 4. **Quantitative Comparison** on the photorealism of retrieved CAD models with different kinds of materials.

by HUGS, where every 3D Gaussian possesses a semantic label. This allows for extracting the target vehicle by selecting 3D Gaussians that lie within the bounding box and carry car semantic labels. By manipulating the position and orientation of the 3D Gaussians with a transformation matrix, we can easily manipulate the vehicle representation.

UrbanCAD (w/o opt.). UrbanCAD (w/o opt.) is implemented by directly using the official pre-trained checkpoint of OpenShape [39], a multi-modality joint representation method, to retrieve the CAD models from Objaverse [15] dataset according to the input single-view images. Note that while Objaverse includes vehicle CAD models with high-quality texture maps, these require significant manual labor and cannot be optimized to fit observation data. In contrast, our method only requires CAD models with base colors as input, reducing the need for human effort. We further evaluate the quality of these labor-intensive handcrafted textures in Table 4. OpenShape [39] refers to the retrieved CAD models with external handcrafted texture maps, while UrbanCAD (w/o opt.) refers to the CAD models with base colors. OpenShape* [39] denotes the retrieved CAD models without any materials. UrbanCAD (Ours) refers to CAD models with our optimized materials. The results show that our method generates materials that better fit the observations, achieving comparable or superior quality to the labor-intensive handcrafted texture maps.

LatentPaint. LatentPaint [45] is a mesh texturing method using a generative model. When implementing LatentPaint, we found its open-sourced code doesn't support textual inversion. Therefore, we use ChatGPT4 [3] to implement textual inversion by asking ChatGPT4 to estimate the colors of the input vehicles. After we get the colors described in the text, we use the official implementation of LatentPaint to accomplish the mesh texturing task.

Paint3D. Paint3D [75] is a SOTA mesh texturing method using diffusion model. It generates high-resolution textures in a coarse-to-fine manner and supports texture transfer from a single view image using IP-Adapter [70]. In our implementation, we directly use its open-source code and checkpoints to do the inference.

PhotoScene. Since the procedural graph library used in PhotoScene is different from our method, which may lead to unfairness, we implement PhotoScene on our pre-defined procedural graph library. Specifically, we directly assign the metal material used in our method and further optimize the

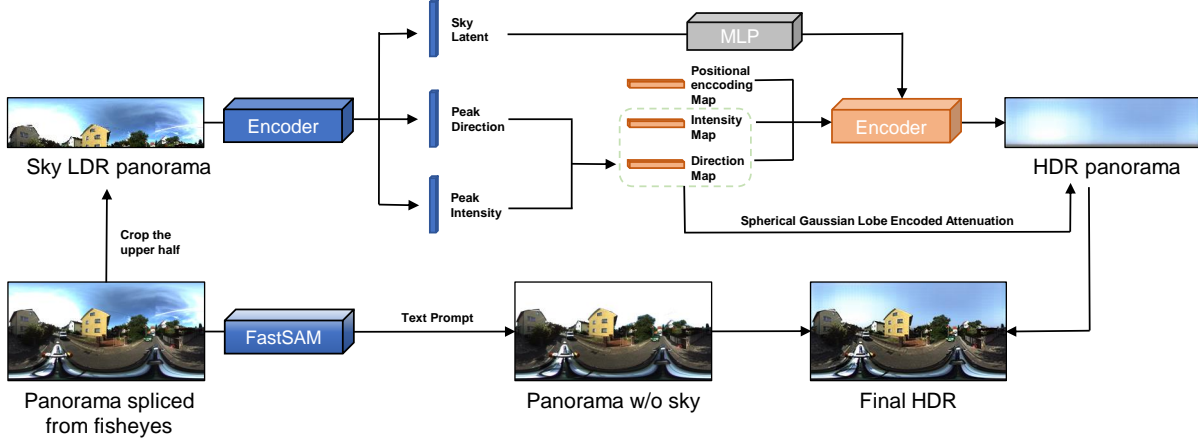


Figure 14. LDR to HDR reconstruction pipeline. The upper half obtains the HDR panorama from the LDR input. The other half stitches the origin panorama with the predicted HDR sky to get the spatially varying lighting

material, since retrieving materials based on visual similarity proposed in Photoscene will lead to severe degradation of appearance.

9.2. Synthetic Data Generation

We utilize a series of 3d bounding boxes to control the movement of vehicles. We construct our synthetic data for self-driving perception system testing in 3 different trajectories as illustrated in Fig. 15. Specifically, trajectory 1 involves vehicles moving normally on the road. Trajectory 2 includes scenarios of vehicles rotating 360°. Trajectory 3 involves vehicles moving in near and partially obscured views, which are typically more challenging for perception models. For each group of synthetic data, there are 60 images for Trajectory 1, 90 images for Trajectory 2, and 120 images for Trajectory 3. When constructing scenarios using UrbanCAD without lighting estimation, we position six uniform point lights along the positive and negative x, y, and z axes.

9.3. Perception Systems Implementation

For YOLOv8 instance segmentation method, we use the official model yolov8n pre-trained on COCO dataset [35]. For the Mask2Former instance segmentation method, we use the official pretrained models with different backbones on the cityscapes dataset [13].

9.4. Computing Resource

We use a single RTX3090 GPU to perform material optimization. Optimizing a material takes about 35 seconds for 300 optimization epochs.

10. Functionality

Since our created vehicle models are fully controllable, we showcase more editing results including component editing, relighting, material transfer, 360° rotation, and novel view rendering.

10.1. Component Editing

Our produced 3D vehicle models support easy component editing mainly due to the handcrafted disentangled geometry as shown in Fig. 11. Note that complete component editing requires human effort for animation, such as setting joint types and parameters in Blender. Additionally, some retrieved handcrafted CAD models may have merged geometry, for example, the four wheels are merged in one mesh. For these cases, simply hiding other vehicle components and entering the edit mode to separate the wheels by selection in the Blender can solve the problem with small manual efforts. However, we notice that some vehicle CAD models have been post-processed by geometry merging, which means the loss of part controllability. Fortunately, most handcrafted vehicle CAD models in the Objaverse still preserve part controllability without being post-processed, and many post-processed CAD models still have disconnected geometry, which can be manually separated by Blender “Separate Selection” operation after selecting connected geometry (“Select Linked” function in “Select” menu). Besides, more corner case results are displayed in Figure 16 thanks to the representation of CAD models. In addition to the editing results mentioned earlier, we can generate more scenes, using the powerful physical simulation effects in Blender. By assigning physics properties to the vehicle model, we can create collision scenes or even simulate car accidents in Blender.



Figure 15. Illustration of our synthetic data during self-driving system testing.

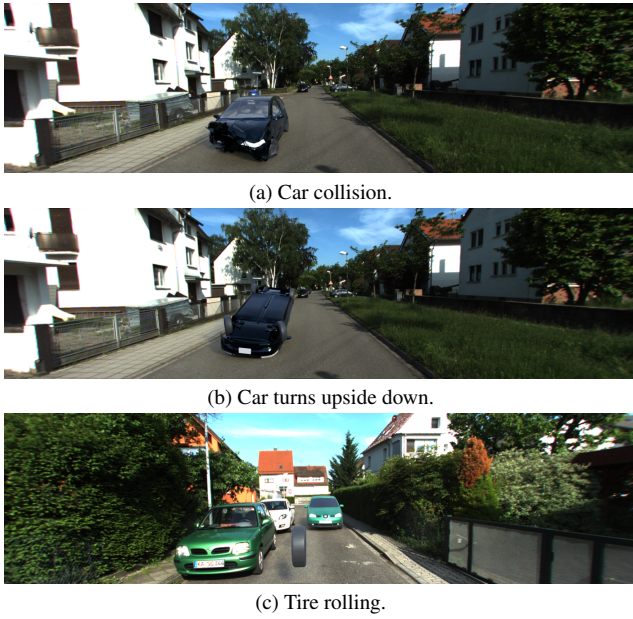


Figure 16. More corner cases.

10.2. Relighting

Realistic insertion results are shown in Figure 17. We utilize the LDR and HDR pairs from online databases to perform the relighting.

10.3. Material transfer

Material transfer results are shown in Figure 18. Since we have obtained the semantic meaning of CAD model material designs, we can easily transfer the part-aware material from one to another.



Figure 17. Realistic Insertion.

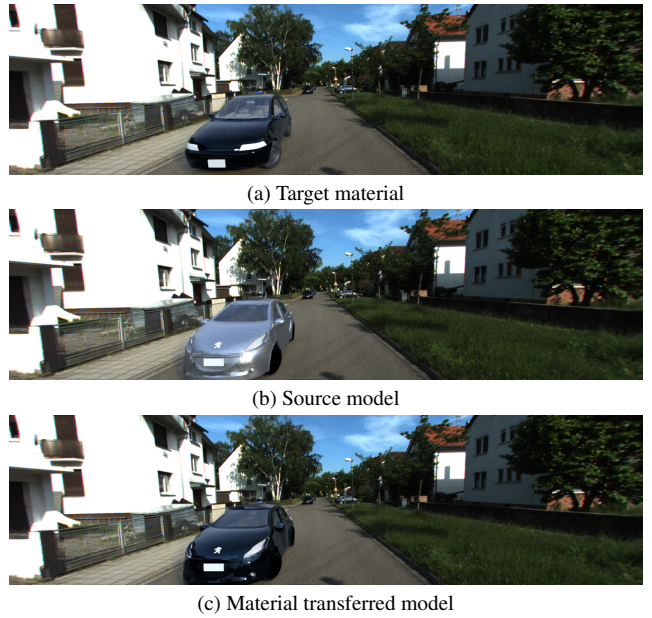


Figure 18. Material Transfer.



Figure 19. Novel View Synthesis

10.4. Novel view rendering

We showcase our novel view rendering results after reconstructing the background using the implicit function and inserting our produced vehicle model, as shown in Fig. 19. Our method can produce high-fidelity rendering results of both background scenes and foreground vehicles under novel viewpoints.

11. Additional Experiments and Analysis

11.1. Lighting Estimation Comparison

We conduct lighting estimation comparison experiments with three baselines as shown in Fig. 22. (1) lighting estimation method using the generative model: Diffusion-Light [47]. (2) lighting estimation method with the autoregressive network: SOLD-Net [58]. (3) lighting estimation method using ray-tracing: ChatSim [64]. For the DiffusionLight, we use the open-sourced official code and checkpoints and take the single-view perspective image as input. For the SOLD-Net, we manually select two points on the ground to mark the area where the network estimates the lighting. After obtaining the output results, we selected the HDR image that closely matched the lighting of the real scene for testing. For the ChatSim [64], we used the view directly ahead of the vehicle as the network input. We also present the quality results of UrbanCAD without lighting estimation in Table 2, where we use six point lights positioned in the positive and negative x, y, and z axis. As demonstrated in the Fig. 22, our method performs better than the baselines, especially in sunny weather where the sun is absent from the perspective images. This is because our fisheye-based method has a 360° view of the environment to accurately capture the location and existence of the



(a) Scenarios with CAD model without optimization



(b) Scenarios with CAD model with optimization

Figure 20. Quality results on self-driving perception system.



Figure 21. Failure Cases

sun. However, our method may have limitations in estimating the lighting for objects in the shadow. This is due to the presence of overexposed areas in the fisheye camera’s captured image. When these overexposed areas are combined into a panorama, they are given higher brightness, resulting in artifacts when lightening the vehicles in shadow in the final rendering.



Figure 22. Lighting estimation comparison between ours, DiffusionLight [47], ChatSim [64], SOLD-Net [58], and ours without lighting estimation. In the setup of ours (w/o lighting estimation), the vehicles are illuminated by six point lights positioned along the positive and negative x, y, and z axes. The results show that our method estimates environmental lighting more accurately, particularly in sunny weather.

Method	FID↓	KID↓
NeRS [76] (Surrounding)	110.55	0.0780
NeRS [76] (Partial)	206.46	0.1685
UrbanCAD (Ours)	79.50	0.0530

Table 5. **Quantitative comparison** to NeRS on MVMC dataset in both surrounding and partial observation. Note that our method uses only a single-view image as input.

	Chamfer Dist.↓	Volume IOU↑
Ours	0.052	0.636
Wonder3D	0.058	0.588

Table 6. Geometry quality

11.2. Quality results of perception system

In Fig. 20, we show the quality result of different perception results on synthetic data created by UrbanCAD (Ours) and UrbanCAD without material optimization. We find the perception system may fail to work in the synthetic data constructed with the vehicle models with unrealistic materials.

11.3. Failure Cases

we provide failure cases in Fig. 21. Our method may provide unsatisfactory results when the retrieved CAD model is defective (e.g., missing wheels), when the reference vehicle has multiple colors in one component (e.g., ambulance), or when the vehicles in the reference view are rarely seen (e.g., heavy-duty truck).

11.4. Geometry quality.

We randomly select 30 vehicles from the ShapeNet dataset, encompassing various types, and retrieve their corresponding models from the Objaverse dataset. We report the Chamfer Distance and Volume IOU in Table 6. Our retrieved models’ geometry quality surpasses the reconstruction baseline, Wonder3D [42], as our retrieved CAD models often exhibit better geometry quality in unobservable regions.

12. Broader Impact

UrbanCAD may help the development of self-driving simulation technology, which can further encourage the development of the self-driving industry. However, our method may be used to create some false urban scenes, leading to some social problems.