Supplementary Material: Neural Lighting Simulation for Urban Scenes

Ava Pun[1,3*†] **Gary Sun**[1,3*†] **Jingkang Wang**[1,2*] **Yun Chen**[1,2] **Ze Yang**[1,2]
Sivabalan Manivasagam[1,2] **Wei-Chiu Ma**[1,4] **Raquel Urtasun**[1,2]
Waabi[1] University of Toronto[2] University of Waterloo[3] MIT[4]

Abstract

In this supplementary material, we detail our method, implementation, experimental designs, additional quantitative and qualitative results, limitations, utilized resources, and broader implications. We first detail how we build the relightable digital twins from real world data (Sec. A.1), and then show the network architecture and training details for neural lighting simulation (Sec. A.2). In Sec. B, we provide details on baseline implementations and how we adapt them to our scene relighting setting. Next, we provide the experimental details for perceptual quality evaluation and downstream detection training in Sec. C. We then report perception quality evaluation at larger scale with more qualitative comparison with baselines (Sec. E.1) and detailed detection metrics for detection training (Sec. E.3), demonstrate our lighting estimation module yields more accurate sun prediction (Sec. E.4) and show additional scene relighting, shadow editing and controllable camera simulation results (Sec. E.5). Finally, we discuss limitations and future works (Sec. F), computation resources used in this work (Sec. G), license of assets (Sec. H) and broader impact (Sec. I).

A LightSim Implementation Details

A.1 Building Lighting-Aware Digital Twins of the Real-World

Neural Scene Reconstruction: We first perform neural rendering to reconstruct the driving scene using both front-facing camera images and 360° spinning LiDAR point clouds. We use a modified version of UniSim [26], a neural sensor simulator, to perform neural rendering to learn the asset representations. Unisim [26] incorporates the LiDAR information to perform efficient ray-marching and adopts a signed distance function (SDF) representation to accurately model the scene geometry. To ensure a smooth zero level set, the SDF representation is regularized using Eikonal loss [6] and occupancy loss. To capture the *view-independent* base color $\mathbf{k_d}$, we make adjustments to the appearance MLP by taking only grid features as input and omitting the view direction term; all other configurations remain consistent with [26].

After learning the neural representation, we employ Marching Cubes [12] to extract the mesh from the learned SDF volume. Finally, we obtain the vertex base color $\mathbf{k_d}$ by querying the appearance head at the vertex location. We adopt base Blender PBR materials [2] for simplicity. Specifically, we use a Principled BSDF with vertex color as the base color. We set materials for dynamic assets and background separately.

Neural Lighting Estimation: We leverage multi-camera data and our extracted geometry to estimate an incomplete panorama LDR image that captures scene context and the available observations

*Equal contributions.
†Work done while a research intern at Waabi.

37th Conference on Neural Information Processing Systems (NeurIPS 2023).

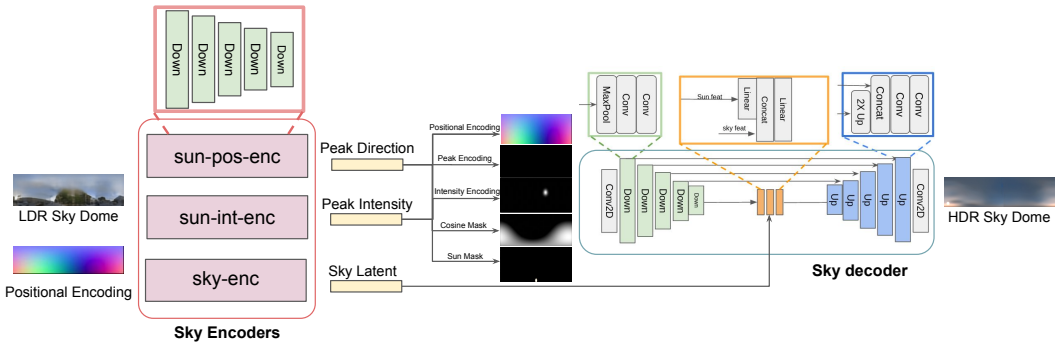

Figure A1: Network architecture of the neural lighting estimation module. Red and blue boxes denote the *sky encoders* and *HDR sky dome decoder* respectively.

of the sky. We then apply an inpainting network to fill in missing sky regions. Finally, we leverage a sky dome estimator network that lifts the LDR panorama image to an HDR sky dome and fuses it with GPS data to obtain accurate sun direction and intensity. To inpaint the panorama image from the stitched multi-camera image, we use the code of the inpainting network DeepFillv2.[1] The network is unchanged from its implementation on Github, and it is trained using the hinge loss. Holicity's LDR panoramas are used for training. Each panorama in the training set is first masked using an intrinsics mask – a random mask of observed pixels from a PandaSet log. Distortions are then applied to the mask, including random scaling and the addition of noise. The masked panorama is then fed into the network and supervised with the unedited Holicity panorama.

Sky dome estimator architecture: After receiving the full LDR output from the inpainting network, the sky dome estimator network converts the incomplete LDR panorama to HDR. This is achieved by employing GPS and time of day to improve the accuracy of sun direction estimation. To produce an LDR sky dome, only the top half of the panorama is used, as the bottom half does not contribute to global lighting from the sky. The network uses an encoder-decoder architecture, as shown in Fig. A1. The input consists of the LR panorama and the positional encoding map, which is an equirectangular projection that stores a unit vector pointing towards each pixel direction. The positional encoding is concatenated channel-wise with the LDR panorama. Three separate encoders with the same architecture are utilized to estimate the peak direction $\mathbf{f}_{\text{dir}}$ and intensity of the sun $\mathbf{f}_{\text{int}}$, as well as the latent of the sky $\mathbf{z}_{\text{sky}} \in \mathbb{R}^{64}$. To encode the predicted sun peak intensity $\mathbf{f}_{\text{int}}$ and peak direction $\mathbf{f}_{\text{dir}}$, five intermediate representations with the same size as the input LDR are explicitly encoded. The sky decoder is an U-Net which takes the encoded predicted sun peak intensity $\mathbf{f}_{\text{int}}$ and peak direction $\mathbf{f}_{\text{dir}}$ and fuses them with the sky latent vector to produce the HDR sky dome.

Training details: We train the sky dome estimator using 400 HDRs sourced from HDRMaps [7]. Accurately predicting the peak intensity $\mathbf{f}_{\text{int}}$ and direction $\mathbf{f}_{\text{dir}}$ is of utmost importance for achieving realistic rendering. Nevertheless, due to the inherently ill-posed nature of this problem, it is challenging to predict these parameters precisely, especially for cloudy skies. Therefore, we propose a dual-encoder architecture, with one encoder trained on HDRs with a clearly visible sun and another trained on all HDR images to capture peak intensity and sky latent more robustly. We empirically find that this works better than using a single encoder trained on all HDR images. The model takes around 12 hours to train with a single RTX A5000.

A.2 Neural Lighting Simulation of Dynamic Urban Scenes

Building augmented reality representation: Given our compositional neural radiance fields, we can remove actors (Fig. 7), add actors (Fig. A14), modify actor trajectories (Fig. 1 and Fig. A14), and perform neural rendering on the modified scene to generate new camera video in a spatially- and temporally-consistent manner. Using our estimated lighting representation, we can also insert synthetic assets such as CAD models [22] into the scenes and composite them with real images in a 3D- and lighting-aware manner (Fig. 1, 7, A14). These scene edits lead to an "augmented reality

[1] `https://github.com/zhaoyuzhi/deepfillv2`

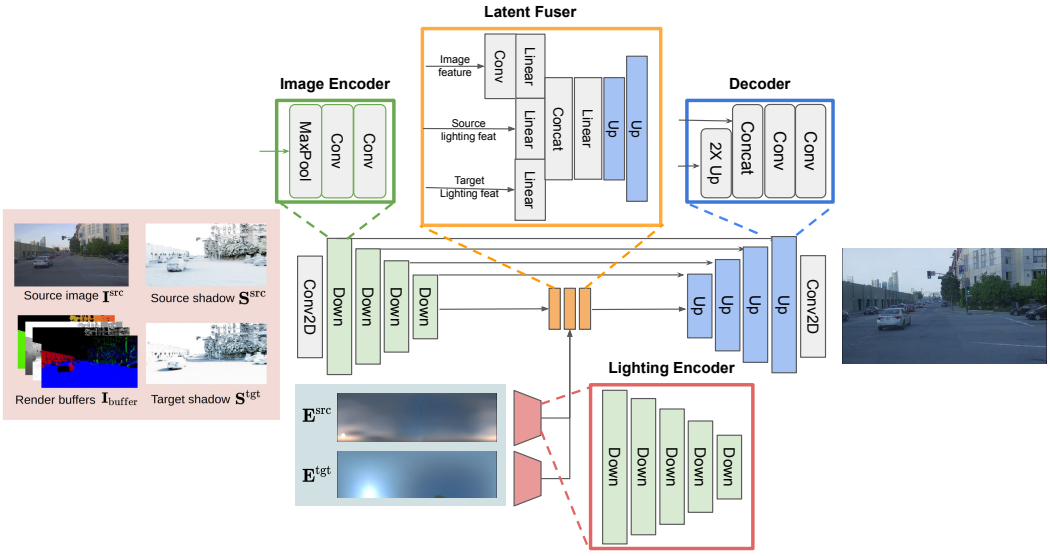

Figure A2: Network architecture of the neural deferred rendering module. Green, red, orange, and blue boxes denote the *image encoder*, *lighting encoder*, *latent fuser* and *rendering decoder*, respectively.

representation" $\mathcal{M}'$, $\mathbf{E}^{\text{src}}$ and source image $\mathbf{I}'_{\text{src}}$. We further explain the details of creating augmented reality representations for all the examples we have created in this paper. In Fig. 1, we insert a new CAD vehicle, barriers, and traffic cones and modify the trajectory of the white car. In Fig. 7, we insert three new CAD vehicles (black, white, gray) and construction items and remove all existing dynamic vehicles. In Fig. A14 (top example), we modify the trajectory of the original white vehicle and insert construction items. In Fig. A14 (bottom example), we remove all existing dynamic actors (pedestrians and vehicles), insert three dynamic actors from another sequence `016` with new trajectories, and insert construction items.

Neural deferred rendering architecture: The neural deferred rendering architecture is depicted in Fig. A2. It is an image-to-image network adapted from U-Net [18] that consists of four components: *image encoder*, *lighting encoder*, *latent fuser*, and *rendering decoder*. The inputs to the *image encoder* comprise a source RGB image $\mathbf{I}_{\text{src}}$, rendering buffers $\mathbf{I}_{\text{buffer}}$ (ambient occlusion, normal, depth, and position), and source/target shadow ratio maps $\{\mathbf{S}_{\text{src}}, \mathbf{S}_{\text{tgt}}\}$; each of these components has 3 channels. In the *latent fuser*, the output of the image encoder is run through a 2D convolution layer, then a linear layer that compresses the input into a latent vector. The image latent and two lighting latent vectors (source and target) are concatenated and upsampled. Finally, the *rendering decoder* upsamples the fused latent vectors and produces the final relit image $\hat{\mathbf{I}}^{\text{tgt}} \in \mathbb{R}^{H \times W \times 3}$.

Learning details: We train the relighting network using a weighted sum of the photometric loss ($\mathcal{L}_{\text{color}}$), perceptual loss ($\mathcal{L}_{\text{lpips}}$), and edge-based content preserving loss ($\mathcal{L}_{\text{edge}}$): $\mathcal{L}_{\text{relight}} = \frac{1}{N}\sum_{i=1}^{N}\left(\mathcal{L}_{\text{color}} + \lambda_{\text{lpips}}\mathcal{L}_{\text{lpips}} + \lambda_{\text{edge}}\mathcal{L}_{\text{edge}}\right)$, where λ_{lpips} is set to 1 and λ_{edge} to 400. Our model is trained on pairs of PandaSet [25] scenes, lit with $N_{\text{HDR}} = 20$ HDRs: ten estimated from PandaSet scenes using our neural lighting estimation module, and ten obtained from the HDRMaps dataset [7]. We apply random intensity and offset changes to the HDRs as data augmentation.

B Implementation Details for Baselines

For each approach, we evaluate on 1,380 images, applying 23 lighting variations to 15 PandaSet scenes with 4 frames per scene, and report FID/KID scores. To test the diversity and controllability of each method, 11 of the target lighting variations are estimated from real PandaSet data, while the remaining 12 are outdoor HDRs sourced from HDRMaps [7].

B.1 Self-supervised Outdoor Scene Relighting (Self-OSR)

Self-OSR [27] is an image-based inverse-rendering method that utilizes InverseRenderNet [28] to decompose an image into albedo, normal, shadow, and lighting. Subsequently, it employs two generative adversarial networks (GANs) for neural rendering and sky hallucination, based on the predicted lighting. We use the official pre-trained models[2] and perform inference under novel lighting conditions. If the target lighting originates from PandaSet logs, the model is first applied to a single front-camera image (first frame) to recover the lighting parameters $\in \mathbb{R}^{9\times 3}$, using an order-2 spherical harmonics model. For outdoor HDRs sourced from HDRMaps, the HDR environment maps are converted into spherical harmonics. The sky masks for PandaSet images are estimated using PSPNet [29].

B.2 NeRF for Outdoor Scene Relighting (NeRF-OSR)

We use the official code from NeRF-OSR[3] and make several modifications to improve its performance on the PandaSet dataset. Since the self-driving scenes in the PandaSet dataset are usually larger than the front-facing scenarios on the NeRF-OSR dataset, more space needs to be sampled, which presents a challenge. To overcome this, we use LiDAR points as a guide to sample the points for camera rays by aggregating LiDaR points and creating a surfel mesh. Then, we sample only the points close to the surface $\pm 50cm$ for each camera ray. By doing so, we significantly reduce the number of sampled points to eight in the coarse and fine stages. Without these modifications, NeRF-OSR demonstrates slower convergence rates and struggles to learn reasonable geometry for relighting. Since our inference HDRs (from PandaSet or HDRMaps) differ greatly from NeRF-OSR environment maps (indicating inaccurate scene decomposition by the model), directly applying the environment maps to the scene leads to significant artifacts. Therefore, we retrieve the nearest HDR in the NeRF-OSR dataset during inference for better perceptual quality. For training, we follow the original code base for all other settings. Training one log on NeRF-OSR usually takes 15 hours on 4 T4 GPUs.

B.3 Color Transfer

Color Transfer [15] uses image statistics in (L^*, a^*, b^*) space to adjust the color appearance between images. We adopt the public Python implementation[4] for our experiments. If the target lighting originates from PandaSet logs, we use the single front-camera image as the target image for color transfer. For outdoor HDRs sourced from HDRMaps, we use the rendered synthetic images (using LightSim digital twins) $\mathbf{I}_{\text{render}|\mathbf{E}^{\text{tgt}}}$ as the target image, as it produces much better performance compared to transferring with the LDR environment map. The latter results in worse performance since the source environment map and target limited-FoV images are in different content spaces.

B.4 Enhancing Photorealism Enhancement (EPE)

EPE [16] was designed to enhance the realism of synthetic images (*e.g.,* GTA-5 [17] to CityScapes [5]) using intermediate rendering buffers and GANs. We adapt EPE to handle lighting simulation with our established lighting-aware digital twins. Specifically, EPE uses the rendered image $\mathbf{I}_{\text{render}|\mathbf{E}^{\text{tgt}}}$ and rendering buffers $\mathbf{I}_{\text{buffer}}$ generated by our digital twins to predict the relit image. Note that EPE uses the same training data as LightSim, with 20 HDR variations. It also takes all PandaSet front-camera images (103 logs) as the referenced real data. We adopt the official implementation[5] and follow the instructions to compute robust label maps, crop the images, match the crops (5 nearest neighbours) and obtain 459k sim-real pairs. We train the EPE model until convergence for 60M iterations on one single RTX A5000 for around six days.

[2] `https://github.com/YeeU/relightingNet`

[3] `https://github.com/r00tman/NeRF-OSR`

[4] `https://github.com/jrosebr1/color_transfer`

[5] `https://github.com/isl-org/PhotorealismEnhancement`

C LightSim Experiment Details

C.1 Perceptual Quality Evaluation

Following [16, 4, 26], we report Fréchet Inception Distance [8] (FID) and Kernel Inception Distance [8] (KID) to measure perceptual quality since ground truth data are not available. Due to NeRF-OSR's large computational cost, we select 15 sequences {001, 002, 011, 021, 023, 024, 027, 028, 029, 030, 032, 033, 035, 040, 053} for quantitative evaluation in the main paper. We also provide Table A1 for larger-scale evaluation (NeRF-OSR excluded), in which 47 sequences (all city logs in PandaSet excluding the night logs and 004 where the SDV is stationary) are used for evaluation. The 47 sequences are {001, 002, 003, 004, 005, 006, 008, 011, 012, 013, 014, 015, 016, 017, 018, 019, 020, 021, 023, 024, 027, 028, 029, 030, 032, 033, 034, 035, 037, 038, 039, 040, 041, 042, 043, 044, 045, 046, 047, 048, 050, 051, 052, 053, 054, 055, 056, 139}.

For each sequence, we select four frames {6, 12, 18, 24} and simulate 23 lighting variations (see Fig. A3). Note that in 23 lighting variations, 20 HDRs (10 estimated HDRs from PandaSet, 10 real HDRs sourced from HDRMaps) are used for data generation to train the LightSim models, while 3 HDRs are unseen and only used during inference. Unless stated otherwise, we use all 8240 real PandaSet images as the reference dataset.

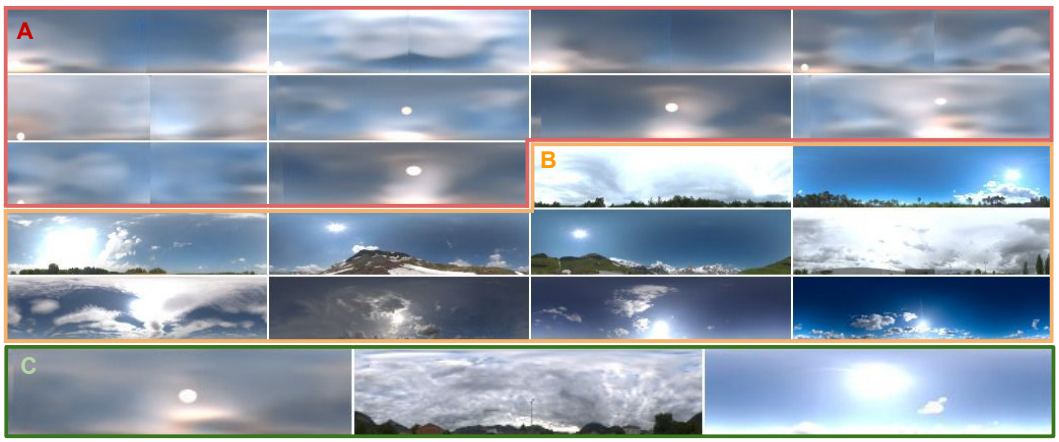

Figure A3: 23 HDR sky domes used for perceptual quality evaluation.

C.2 Downstream Perception Training

To investigate if realistic lighting simulation can improve the performance of downstream perception tasks under unseen lighting conditions, we conduct experiments on PandaSet using the SoTA camera-based 3D vehicle detection model BEVFormer [10]. We use train on 68 snippets collected in the city and evaluate on 35 snippets in a suburban area, since these two collections are independent and exposed to different lighting conditions. Specifically, the sequences {080, 084, 085, 086, 088, 089, 090, 091, 092, 093, 094, 095, 097, 098, 099, 100, 101, 102, 103, 104, 105, 106, 109, 110, 112, 113, 115, 116, 117, 119, 120, 122, 123, 124, 158} are selected for the validation set, and the remaining sequences are used for training. For the experiments, we use all 80 frames for training and evaluation. We report the average precision (AP) at different IoU thresholds: 0.1, 0.3, and 0.5. The mean average precision (mAP) is calculated as $\mathrm{mAP} = (\mathrm{AP@0.1} + \mathrm{AP@0.3} + \mathrm{AP@0.5})/3.0$.

As shown in Table 2, the integration of LightSim synthetic simulation significantly enhances the performance of monocular detection compared to training with other basic augmentation methods. Further exploration on sufficiently utilizing the simulated data, such as actor behavior simulation or actor insertion, is left to future work.

BEVFormer Implementation Details: We use the official repository[6] for training and evaluating our model on PandaSet. We focus on single-frame monocular vehicle detection using the front camera, disregarding actors outside the camera's field of view. The models are trained within vehicle frames using the FLU convention (x: forward, y: left, z: up), with the region of interest defined as $x \in [0, 80\,\text{m}], y \in [-40\,\text{m}, 40\,\text{m}], z \in [-2\,\text{m}, 6\,\text{m}]$. Given memory constraints, we adopt the BEVFormer-small architecture[7] with a batch size of two per GPU. Models were trained for five epochs using the AdamW optimizer [13], coupled with the cosine learning rate schedule.[8] Training each model took approximately six hours on $2\times$ RTX A5000 GPUs. We report the best validation performance across all data augmentation approaches, as models can begin to overfit in the final training stage.

C.3 Generalization on nuScenes

To evaluate the generalizability of our model, we train the model on PandaSet [25] and evaluate the pre-trained model on nuScenes [3]. The nuScenes [3] dataset contains 1000 driving scenes collected in Boston and Singapore, each with a duration of $\approx$ 20 seconds ($\approx$ 40 frames, sampled at 2 Hz) acquired by six cameras (Basler acA1600-60gc), one spinning LiDAR (Velodyne HDL32E), and five long-range RADAR (Continental ARS 408-21) sensors. We curate 10 urban scenes from nuScenes [3] characterized by dense traffic and interesting scenarios.

We incorporate front-facing camera and spinning LiDAR and run the neural scene reconstruction module (Section 3.1) to extract the manipulable digital twins for each scene. Then, we utilize the neural lighting estimation module (Section 3.2) to recover the HDR sky domes. This enables us to generate new scenarios and produce the rendering buffers $\mathbf{I}_{\text{buffer}}$ (Eqn. 4 in the main paper) for scene relighting.

D Additional Discussions

Challenges of inverse rendering on urban scenes: LightSim assumes several simplifications when building lighting-aware digital twins, including approximate diffuse-only reconstruction, separate lighting prediction, and fixed base materials. Those result in imperfect intrinsic decomposition and sim/real discrepancies. Recent concurrent works such as FEGR [24] and UrbanIR [11] make steps towards better decomposition, but it is still a challenging open problem to recover perfect decomposition of materials and light sources for large urban scenes. As shown in Fig. A4 (top right), the recovered materials bear little semblance to semantic regions in the original scene. These recent relighting works [16, 19, 24, 11] also have shadows baked into the recovered albedo (Fig. A4 left).

We remark that our novelty lies in leveraging neural deferred rendering to overcome the limitations of purely physically-based rendering when the decomposition is imperfect. This allows us to generate better relighting results than prior works that have imperfect decompositions. It is an exciting future direction to incorporate better intrinsic decomposition along with neural deferred rendering for improved relighting.

Prediction-based vs. optimization-based lighting: We explain our design choices in the following two aspects. (a) We use a feed-forward network for lighting estimation, which is more efficient and can benefit from learning on a larger dataset. In contrast, the optimization paradigm is more expensive, requiring per-scene optimization, but has the potential to recover more accurate scene lighting from partial observations. (b) The ill-posed nature of lighting estimation and extreme intensity range make inverse rendering challenging for outdoor scenes [24]. Optimization of the environment map requires a differentiable renderer and high-quality geometry/material to achieve good results. The existing/concurrent state-of-the-art works [19, 24] cannot solve the problem accurately, as shown in Fig. A4 bottom right.

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

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

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

Figure A4: Challenges of inverse rendering on urban scenes for existing works.

Temporal consistency for neural deferred shading: While we do not guarantee temporal consistency, LightSim can produce temporally-consistent lighting simulation videos in most cases. We believe this temporal consistency comes from temporally- and multi-view-consistent inputs during inference (real image, G-buffers), as well as our combination of simulation and real paired relighting data during training. Explicitly enforcing temporal consistency is an interesting direction for future work.

Random shadowed triangles when relighting due to mesh artifacts: We noticed that random shadowed triangles are common in the $\mathbf{I}_{\text{render}|\mathbf{E}^{\text{src}}}$ due to non-smooth extracted meshes. This is more obvious for the nuScenes dataset where the LiDAR is sparser (32-beam) and the capture frequency is lower (2Hz); for this dataset, we notice many holes and random shadowed triangles. However, thanks to our image-based neural deferred rendering pass trained with mixed sim-real data, our relighting network takes the original image and modifies the lighting, which removes many of those artifacts in the final relit images. We show two nuScenes examples in Fig. A5.

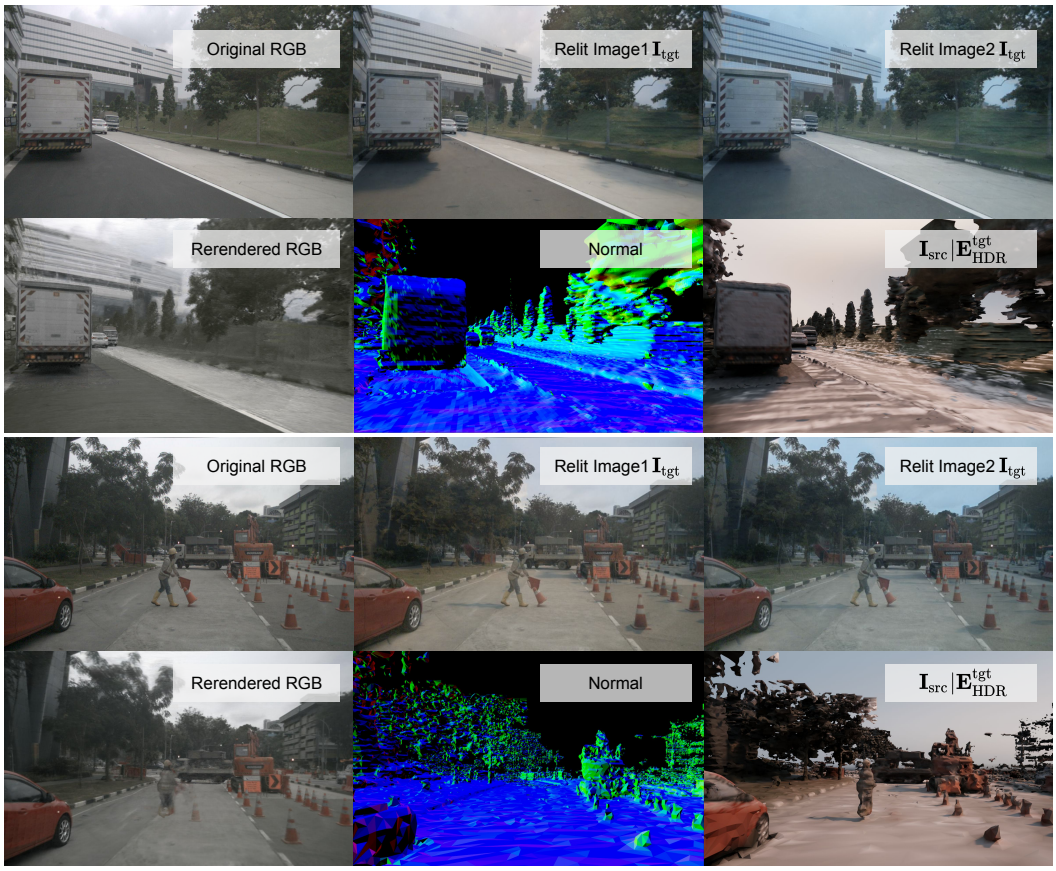

Figure A5: Random shadowed triangles are removed after neural deferred rendering.

E Additional Experiments and Analysis

We provide additional results and analysis for scene relighting, ablation studies, downstream training, and lighting estimation. We then showcase more simulation examples using LightSim.

E.1 Additional Perception Quality Evaluation

Due to the large computational cost of NeRF-OSR [19], we select 15 PandaSet sequences for perceptual quality evaluation in Table 1. Here, we supplement the evaluation at larger scale (47 sequences in total) in Table A1. LightSim achieves perceptual quality (FID and KID) on par with Color Transfer, while the latter approach only adjusts the color histogram and cannot simulate intricate lighting effects properly. Self-OSR and EPE suffer from noticeable artifacts, resulting in significantly worse perception quality and a larger sim-real domain gap.

Method	FID ↓	KID ($\times 10^3$) ↓
Self-OSR [27]	97.3	89.7 ± 11.5
Color Transfer [15]	50.1	18.0 ± 4.5
EPE [16]	79.6	50.0 ± 9.1
Ours	52.3	16.0 ± 4.6

Table A1: Additional perceptual quality evaluation on 47 sequences.

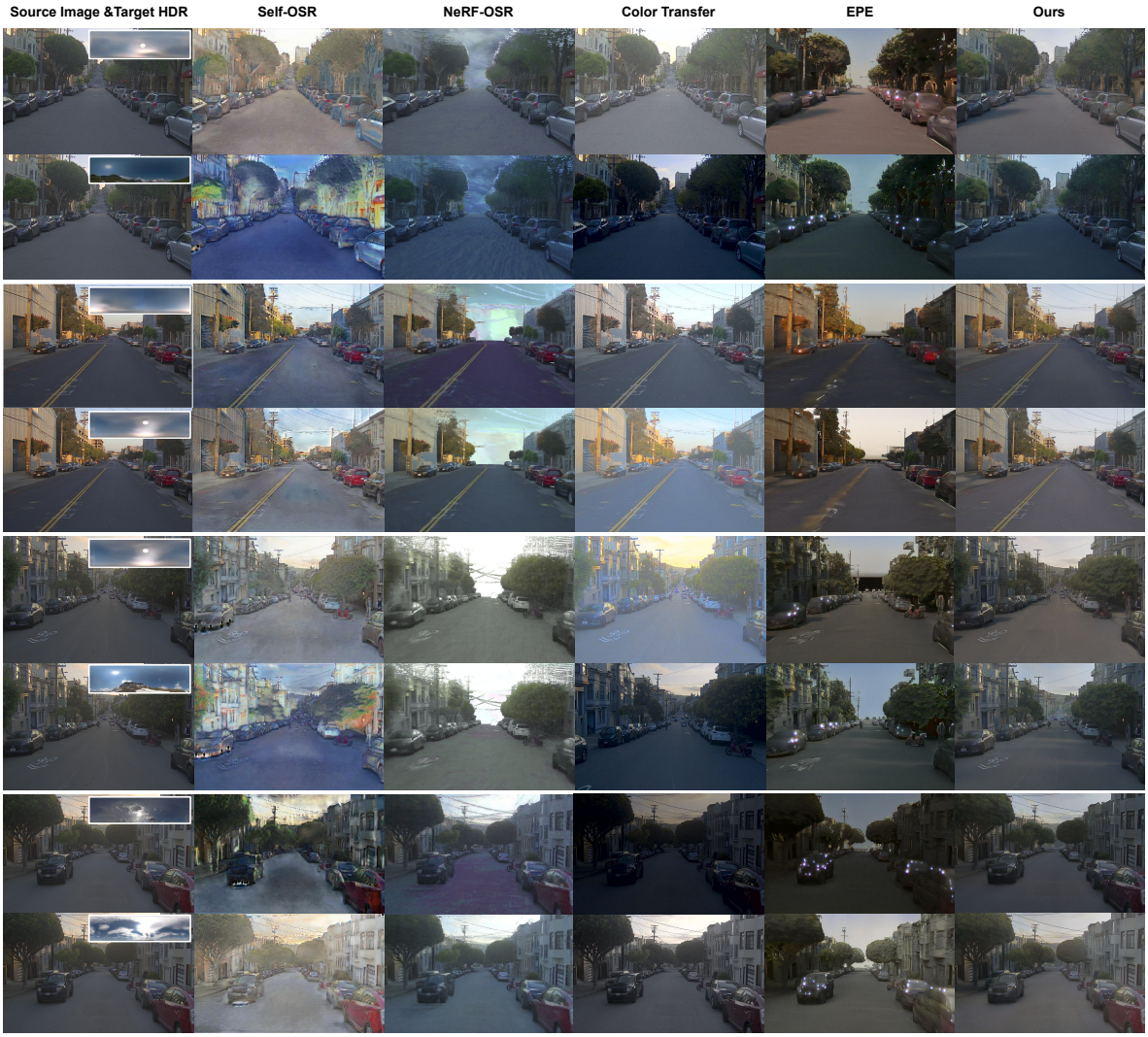

Figure A6: Qualitative comparison against SoTA approaches in scene relighting.

We also provide more qualitative comparisons against SoTA scene relighting approaches in Fig. A6. For each scene, we showcase two different lighting conditions with the inset target HDRs in the leftmost column. We show more scene relighting results of LightSim in Fig. A7 and Fig. A8. We also show results in Fig. A9, where we rotate the HDR skydome and render the shadows at different sun locations, demonstrating controllable outdoor illumination and realistic simulated results. Please refer to the project page for video examples.

Figure A7: Qualitative examples of scene relighting for LightSim (Part 1).

Figure A8: Qualitative examples of scene relighting for LightSim (Part 2).

Figure A9: Qualitative examples of scene relighting with shadows edited.

E.2 Additional Ablation Study

We provide a more thorough ablation study on the important components of the neural deferred rendering module. The perception quality metrics and additional qualitative examples are provided in Table A2 and Fig. A10.

#ID	Data Pairs *sim-real*	Data Pairs *identity*	Rendering Buffers $\mathbf{I}_{\text{buffer}}$	Rendering Buffers $\{\mathbf{S}^{\text{src}}, \mathbf{S}^{\text{tgt}}\}$	Edge loss $\mathcal{L}_{\text{edge}}$	FID ↓	KID ($\times 10^3$) ↓
0	✗					60.9	31.1 ± 4.0
1		✗				62.5	32.7 ± 4.0
2			✗			50.5	21.4 ± 4.1
3				✗		49.8	23.1 ± 5.0
4					✗	109.8	88.7 ± 7.3
5					200	67.1	40.9 ± 4.9
6					800	57.3	31.8 ± 4.3
Ours	✓	✓	✓	✓	400	55.4	27.6 ± 3.7

Table A2: Ablation studies on LightSim components. For clarity, we only mark the differences between our final model and other configurations. Blank components indicate that the setting is identical to our final model.

Figure A10: Additional ablation study on neural deferred rendering. LightSim can simulate intricate lighting effects (highlights, shadows) while maintaining realism.

We choose sequence `001` for the quantitative evaluation (FID/KID scores), and all 80 front-camera images are used as the reference dataset. To sufficiently measure rendering quality, we generate 3×80 relit images using the three unseen HDRs in Fig. A3. We also generate 72 relit images (first frame) with shadows edited using the first unseen HDR. Then, to test generalization to unknown lighting conditions, we pick the first frame and generate 92 relit images using other real HDR maps. Therefore, for each model configuration, there are 404 simulated images used in total for perceptual quality evaluation. We set `kid_subset_size=10` for this experiment.

As shown in Table A2 and Fig. A10, *sim-real* and *identity* data pairs provide useful regularization for the neural deferred rendering module by reducing visual artifacts caused by imperfect geometry. Removing those data pairs leads to less realistic simulation results and worse FID/KID scores. On the other hand, the rendering buffers and shadow maps play an important role in realistically simulating intricate lighting effects such as highlights and shadows. We observe unrealistic color and missing cast shadows if pre-computed buffers $\mathbf{I}_{\text{buffer}}$ and shadow maps $\{\mathbf{S}^{\text{src}}, \mathbf{S}^{\text{tgt}}\}$ are removed. Note that the KID/FID metrics are lower when removing rendering buffers since the reference dataset does not include the real data under new lighting conditions; this cannot be interpreted as better visual quality. Finally, we ablate the content-preserving loss $\mathcal{L}_{\text{edge}}$ and find that a proper loss weight helps the model reduce synthetic mesh-like artifacts (compared to $\lambda_{\text{edge}} = 0$) while properly simulating new lighting effects (compared to $\lambda_{\text{edge}} = 800$).

E.3 Additional Object Detection Metrics

We report detailed detection metrics for perception training with different data augmentation approaches for better reference. Specifically, we report the average precision (AP) at different IoU thresholds: 0.1, 0.3, and 0.5. As shown in Table A3, using LightSim-simulated data yields the best performance improvements. Color Transfer and standard color augmentation [10] are also effective ways to promote the performance of autonomy model under novel lighting conditions. In contrast, Self-OSR and EPE either harm the detection performance or bring marginal gains due to noticeable visual artifacts that cause sim-real domain gap between training and validation.

Model	mAP (%)	AP@0.1	AP@0.3	AP@0.5
Real	32.1	51.2	29.5	15.7
Real + Color aug. [10]	33.8 (+1.7)	53.9	31.0	16.4
Real + Sim (Self-OSR)	30.3 (−1.8)	45.6	29.4	16.0
Real + Sim (EPE)	32.5 (+0.4)	50.2	30.6	16.7
Real + Sim (Color Transfer)	35.1 (+3.0)	55.3	32.3	17.6
Real + Sim (Ours)	**36.6** (+**4.5**)	**57.1**	**33.8**	**19.0**

Table A3: Data augmentation with simulated lighting variations.

E.4 Comparison with SoTA Lighting Estimation works

We further compare our neural lighting estimation module with the SoTA lighting estimation approaches SOLDNet [21] and NLFE [23]. Table A4 shows the lighting estimation results on PandaSet, where the GPS-calculated sun position is used as reference in error computation. For SOLDNet, we use the official pre-trained model to run inference on limited field-of-view (FoV) front-camera images. For NLFE, we re-implement the sky dome estimation branch without differentiable actor insertion and local volume lighting since the public implementation is unavailable. We also compare a variation of NLFE (named NLFE*) that takes our completed LDR panorama image $\mathbf{L}$ as input. For a fair comparison, LightSim uses the predicted sun position during the encoding procedure. For NLFE and LightSim, the sky dome estimators take the sun intensity and direction explicitly to enable more human-interpretable lighting control. Therefore, we also evaluate the decoding consistency error (log-scale for sun intensity and degree for sun direction). The average metrics are reported on all PandaSet sequences with night logs excluded.

As shown in Table A4, LightSim recovers more accurate HDR sky domes compared to prior SoTA works, with the lowest angular error. It also produces lower decoding error compared to NLFE. Interestingly, we also find that using more camera data (panorama vs limited-FoV image) significantly enhances NLFE's estimation performance and reduces decoding errors. This verifies our idea of leveraging real-world data sufficiently to build the lighting-aware digital twins.

Method	Input	Angular Error ↓	Decoding Error ↓	
			Intensity	Angle
SOLDNet [21]	Limited-FoV	69.98°	–	–
NLFE [23]	Limited-FoV	78.29°	2.68	12.15°
NLFE* [23]	Panorama	47.39°	2.27	8.53°
Ours (no GPS)	Panorama	**20.01°**	**1.25**	**1.78°**

Table A4: Comparisons of sky dome estimation on PandaSet. As reference, our model with GPS leads to 3.78° angular error in sun direction prediction and 1.64° decoding error.

Fig. A11 shows more lighting estimation examples on PandaSet, including stitched partial panorama images $\mathbf{I}_{\text{pano}}$, completed LDR panorama $\mathbf{L}$, and HDR sky domes $\mathbf{E}$. LightSim leverages GPS and time data to get the approximate sun location, enabling recovery of the sun in predicted HDRs even if not observed in partial panorama images $\mathbf{I}_{\text{pano}}$, and the surrounding observed sky and scene context can still be used to approximately estimate the sun intensity.

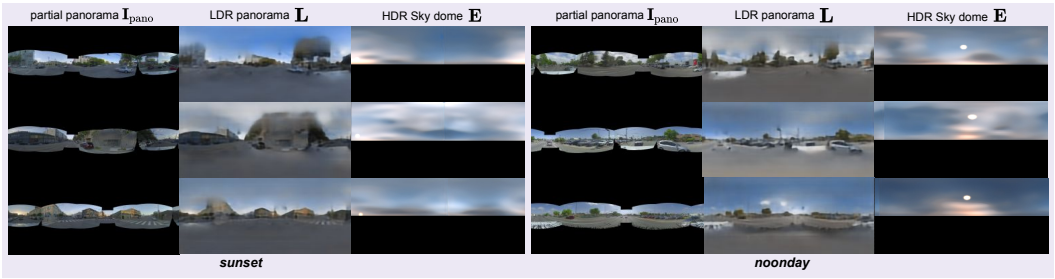

Figure A11: More lighting estimation results on PandaSet.

We further compare virtual actor insertion against SOLDNet and NLFE in Figure A12 on PandaSet sequence `001`. We highlight two regions $\{A, B\}$ for comparison to showcase the importance of accurate sun intensity and location prediction, as well as the capability to model inter-object lighting effects. For SOLD-Net and NLFE, we use Poisson surface reconstruction (PSR) [9] to obtain the ground mesh as the plane for virtual actor insertion. Specifically, we first only keep the ground points using semantic segmentation labels, estimate per-point normals from the 200 nearest neighbors within 20cm, and orientate the normals upwards. Then, we conduct PSR with octree depth set as 12 and remove the 2% lowest density vertices.

Figure A12: Qualitative comparison of lighting-aware virtual object insertion.

For SOLD-Net, the inserted vehicle looks too bright, with hard shadows cast differently from the other actors since the predicted sun intensity is too strong and the sun direction is not correctly inferred. NLFE estimates the sun intensity and direction more reasonably by consuming our completed LDR panorama image. However, it cannot simulate the shadow cast by the original actor onto the

inserted green vehicle due to the lack of 3D digital twins. In contrast, LightSim can perform virtual actor insertion with inter-object lighting effects simulated accurately thanks to the accurate lighting estimation and 3D world modelling.

In Fig. A13, we show additional lighting-aware actor insertion examples on another PandaSet sequence 024, where the sun is visible in the front camera. LightSim inserts the new actors seamlessly and can model lighting effects such as inter-object shadow effects (between real and virtual objects).

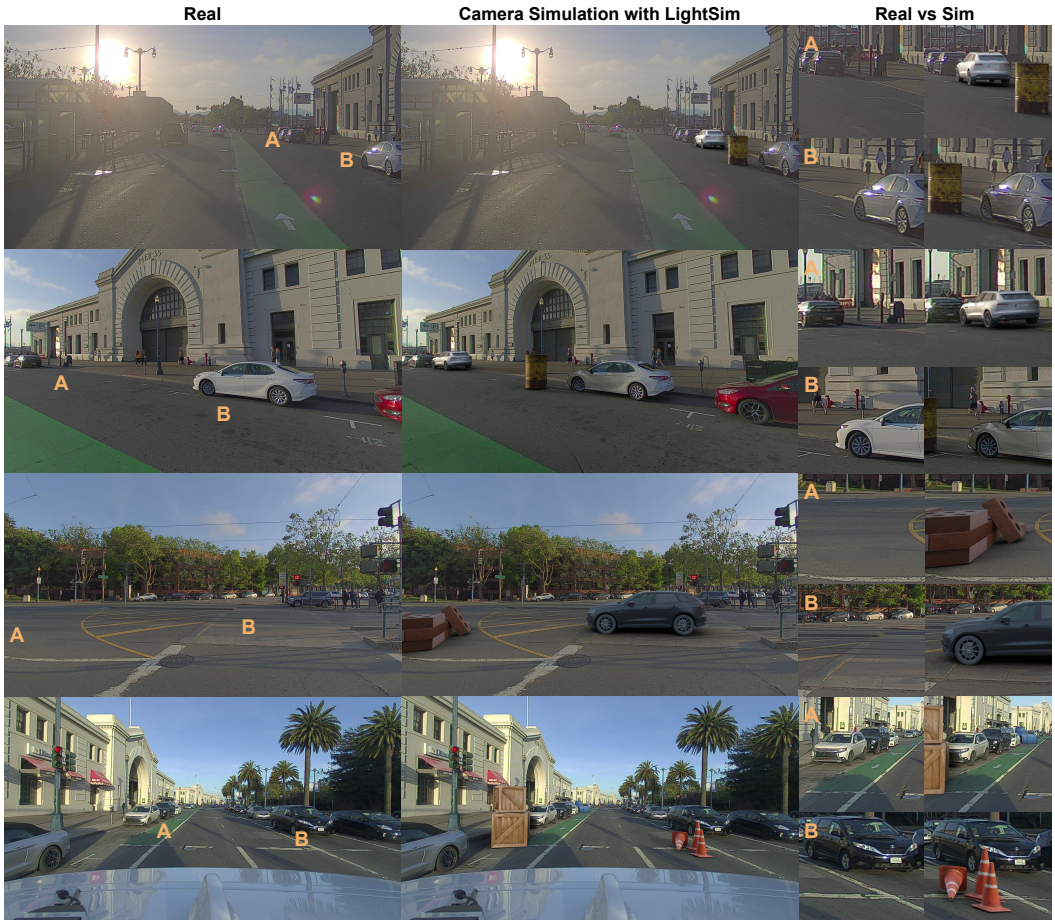

Figure A13: Additional lighting-aware actor insertion examples with LightSim.

E.5 Additional Camera Simulation Examples

Combining all these capabilities results in a controllable, diverse, and realistic camera simulation with LightSim. In Fig. A14, we show additional camera simulation examples similar to Fig. 1 and Fig. 7 in the main paper. We show the original scenario in the first block. In the second block, we show simulated scene variations with an actor cutting into the SDV's lane, along with inserted traffic barriers, resulting in a completely new scenario with generated video data under multiple lighting conditions. In the third block, we show another example where we add inserted barriers and replace all the scene actors with a completely new set of actors reconstructed from another scene. The actors are seamlessly inserted into the scenario with the new target lighting. Please refer to the project page for video examples.

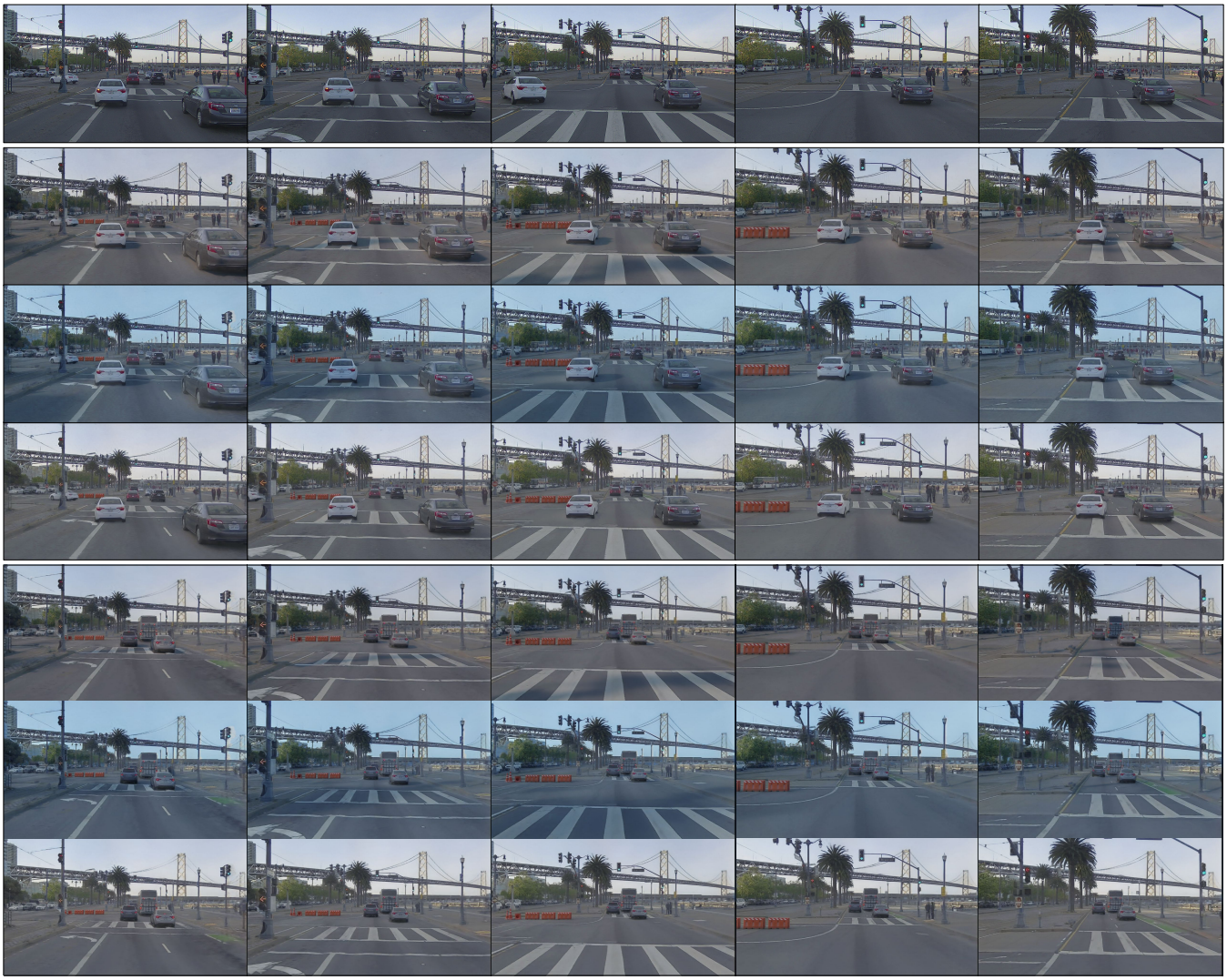

Figure A14: Additional controllable camera simulation examples.

E.6 Additional Qualitative Results on nuScenes

We further showcase LightSim's ability to generalize to driving scenes in nuScenes. We provide more qualitative scene relighting results in Fig. A15 and Fig. A16. Specifically, we select ten diverse scenarios that involve traffic participants such as vehicles, pedestrians and construction items. The sequence IDs are `011, 135, 154, 158, 159, 273, 274, 355, 544, 763`. As described in Sec. C.3, we conduct neural scene reconstruction and lighting estimation (pre-trained on PandaSet) to build the lighting-aware digital twins. Then, we apply the neural deferred rendering model pre-trained on PandaSet to obtain the relit images. Although the nuScenes sensor data are much more sparse compared to PandaSet (32-beam LiDAR, 2Hz sampling rate), LightSim still produces reasonable scene relighting results, indicating good generalization and robustness. Please refer to the project page for video examples.

Occasionally, we observe noticeable black holes (*e.g.,* on log `355` and `763`) in the relit images. This is because the reconstructed meshes are low-quality (non-watertight ground, broken geometry) due to sparse LiDAR supervision and mis-calibration. While the neural deferred rendering module is designed to mitigate this issue, it cannot handle large geometry errors perfectly. Stronger smoothness regularization during the neural scene reconstruction step can potentially improve the model's performance.

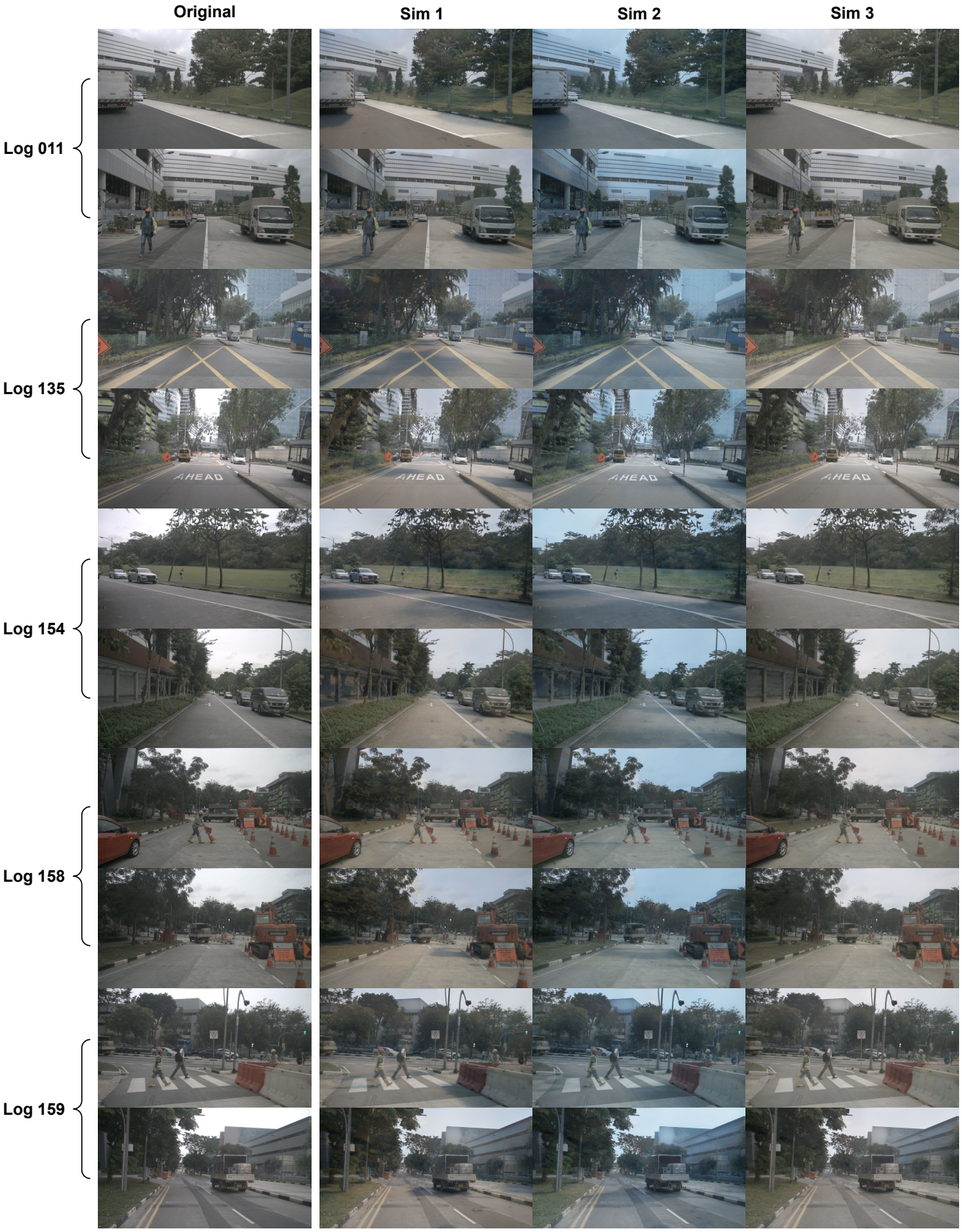

Figure A15: Generalization to nuScenes (Part 1).

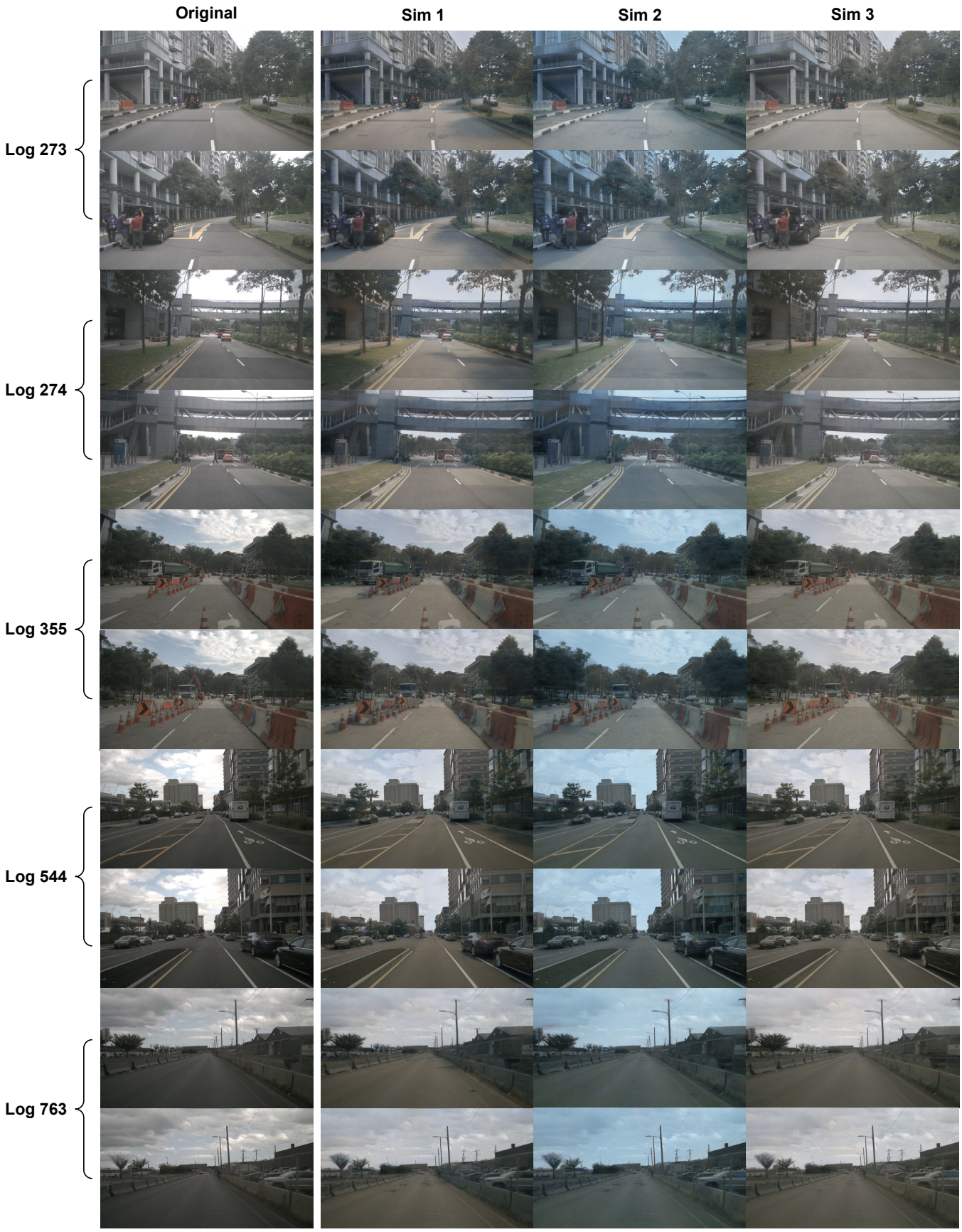

Figure A16: Generalization to nuScenes (Part 2).

F Limitations and Future Works

While LightSim can simulate diverse outdoor lighting conditions, there are several areas where it could benefit from further improvements. First, LightSim cannot seamlessly remove shadows, as shown in Fig. A17, particularly in bright, sunny conditions where the original images exhibit distinct cast shadows. This is because the shadows are mostly baked during neural scene reconstruction (see view-independent reconstruction in Fig. A18), producing flawed synthetic data that confuses the neural deferred rendering module. Moreover, we specify fixed materials [2] and predict sky domes that are not ideal for different urban scenes and may cause real-sim discrepancies as shown in Fig. 9. Those issues can potentially be addressed by better intrinsic decomposition with priors and joint material/lighting learning [24, 19]. More discussions for inverse rendering on urban scenes are provided in Sec D.

Second, LightSim uses an HDR sky dome to model the major light sources for outdoor daytime scenes. Therefore, LightSim cannot handle nighttime local lighting sources such as street lights, traffic lights, and vehicle lights. A potential solution is to leverage semantic knowledge and create local lighting sources (*e.g.,* point/area lights [14] or volumetric local lighting [23]). Moreover, our experiments also focus on camera simulation for perception models, and we may investigate the performance of downstream planning tasks in future works. Lastly, our current system implementation relies on the Blender Cycles rendering engine [1], which is slow to render complex lighting effects. Faster rendering techniques can be incorporated to further enhance the efficiency of LightSim [20, 14].

Apart from the further method improvements mentioned above, it is important to collect extensive data from real-world urban scenes under diverse lighting conditions (*e.g.*, repeating the same driving route under varying lighting conditions). Such data collection aids in minimizing the ambiguity inherent in intrinsic decomposition. Moreover, it paves the way for multi-log training with authentic data by providing a larger set of real-real pairs for lighting training.

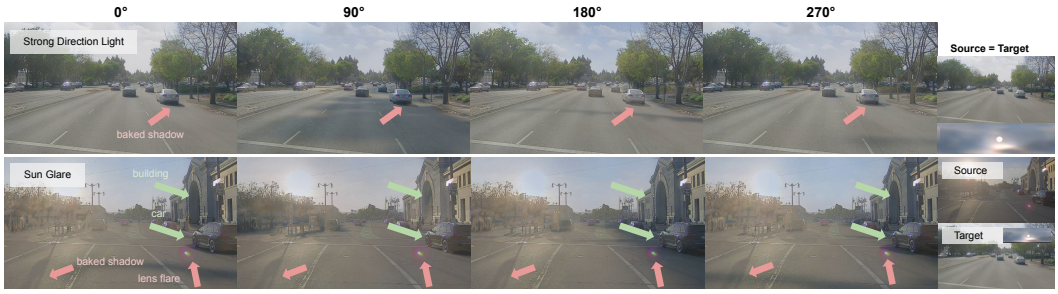

Figure A17: **Failure cases with strong directional lighting.** The neural deferred rendering network cannot fully remove the baked shadows (top row) and other sensor effects (e.g., lens flare – bottom row). We use green and red arrows to highlight areas where LightSim performs well and not well.

Figure A18: **View-independent reconstruction results for LightSim.** Shadows are baked at this stage, which are then mitigated by neural deferred rendering. The relighting failure cases for the last example are shown in Fig. A17.

In Fig. A17, we highlight two examples of LightSim applied to scenes with strong directional lighting and high sun intensity. Each row shows the shadow editing/relighting results under four different sun angles of the target environment map. In the top row, LightSim cannot fully remove source shadows

in bright and sunny conditions due to the baked shadows in the view-independent reconstruction. Moreover, due to inaccurate HDR peak intensity estimation, the brightness of cast shadows cannot match the original images well. In the bottom row, we depict a source image with high sun intensity and glare relit to a new target lighting. It is challenging to remove the sun glare and alter the over-exposed regions in this setting, but we can still apply some relighting effects to the cars and buildings in the scene (see arrows).

G Computation Resources

In this project, we ran the experiments primarily on NVIDIA Tesla T4s provided by Amazon Web Services (AWS). For prototype development and small-scale experiments, we used local workstations with RTX A5000s. Overall, this work used approximately 8,000 GPU hours (a rough estimation based on internal GPU usage reports), of which 3,000 were used for the final experiments and the rest for exploration and concept verification during the early stages of the research project. We provide a rough estimation of GPU hours used for the final experiments in Table A5, where we convert one A5000 hour to two T4 hours approximately.

Experiment	T4 Hours	Comments
Table 1 (perceptual quality validation)	1850	LightSim (100), NeRF-OSR (1500), EPE (150)
Table 2 (downstream training)	150	6× models, each takes 25 GPU hours
Table A4 (lighting estimation)	40	15h NLFE, 25h LightSim
Fig. 5 & Table A2 (ablations)	400	50h each model
Fig. 8 (nuScenes)	40	20h for digital twins, 20h for relighting
Others (data generation & demos)	510	500h lighting data generation + 10h demos

Table A5: Summary of GPU hours used for the final experiments.

H Licenses of Assets

We summarize the licenses and terms of use for all assets (datasets, software, code, pre-trained models) in Table A6.

Assets	License	URL
Blender 3.5.0 [1]	GNU General Public License (GPL)	`https://www.blender.org/`
HDRMaps [7]	Royalty-Free[9]	`https://hdrmaps.com/`
HoliCity [30]	Non-commercial purpose[10]	`https://holicity.io/`
TurboSquid 3D Models	Royalty-Free[11]	`https://www.turbosquid.com`[12]
PandaSet [25]	CC BY 4.0[13]	`https://scale.com/open-av-datasets/pandaset`
nuScenes [3]	Non-commercial (CC BY-NC-SA 4.0)[14]	`https://www.nuscenes.org/`
SOLDNet [21]	Apache License 2.0	`https://github.com/ChemJeff/SOLD-Net/`
EPE [16]	MIT License	`https://github.com/isl-org/PhotorealismEnhancement`
Self-OSR [16]	Apache License 2.0	`https://github.com/YeeU/relightingNet`
NeRF-OSR [19]	Non-commercial purpose[15]	`https://github.com/r00tman/NeRF-OSR`
Color Transfer [15]	MIT License	`https://github.com/jrosebr1/color_transfer`

Table A6: Summary of the licenses of assets.

I Broader Impact

LightSim offers enhancements in camera-based robotic perception, applicable to various domains such as self-driving vehicles. Its ability to generate controllable camera simulation videos (*e.g.*, actor insertion, removal, modification, and rendering from new viewpoints) and adapt to varying outdoor lighting conditions can potentially improve the reliability and safety of intelligent robots for a broad range of environmental conditions. Additionally, LightSim's capacity to create lighting-aware digital twins can improve realism in digital entertainment applications such as augmented reality or virtual reality. However, as with any technology, the responsible use of LightSim is important. Privacy concerns may arise when creating digital twins of real-world locations. We also caution that our system might produce unstable performance or unintended consequences under different datasets, especially when the sensory data are very sparse and noisy.

References

[1] Blender Foundation. Blender, 2021.

[2] Brent Burley and Walt Disney Animation Studios. Physically-based shading at disney. In *ACM SIGGRAPH*, 2012.

[3] Holger Caesar, Varun Bankiti, Alex H Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. nuscenes: A multimodal dataset for autonomous driving. *CVPR*, 2020.

[4] Yun Chen, Frieda Rong, Shivam Duggal, Shenlong Wang, Xinchen Yan, Sivabalan Manivasagam, Shangjie Xue, Ersin Yumer, and Raquel Urtasun. Geosim: Realistic video simulation via geometry-aware composition for self-driving. *CVPR*, 2021.

[5] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. *CVPR*, 2016.

[6] Amos Gropp, Lior Yariv, Niv Haim, Matan Atzmon, and Yaron Lipman. Implicit geometric regularization for learning shapes. *ICML*, 2020.

[7] HDRMaps. Hdrmaps. *https://hdrmaps.com/*, Access date: 2023-05-17.

[8] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *NeurIPS*, 2017.

[9] Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. Poisson surface reconstruction. In *Proceedings of the fourth Eurographics symposium on Geometry processing*, volume 7, page 0, 2006.

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

[11] Zhi-Hao Lin, Bohan Liu, Yi-Ting Chen, David A. Forsyth, Jia-Bin Huang, Anand Bhattad, and Shenlong Wang. Urbanir: Large-scale urban scene inverse rendering from a single video. *CoRR*, abs/2306.09349, 2023.

[12] William E Lorensen and Harvey E Cline. Marching cubes: A high resolution 3d surface construction algorithm. *ACM SIGGRAPH computer graphics*, 1987.

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

[14] Merlin Nimier-David, Delio Vicini, Tizian Zeltner, and Wenzel Jakob. Mitsuba 2: A retargetable forward and inverse renderer. *ACM Transactions on Graphics (TOG)*, 38(6):1–17, 2019.

[15] Erik Reinhard, Michael Adhikhmin, Bruce Gooch, and Peter Shirley. Color transfer between images. *IEEE Computer graphics and applications*, 2001.

[16] Stephan R Richter, Hassan Abu AlHaija, and Vladlen Koltun. Enhancing photorealism enhancement. *PAMI*, 2021.

[17] Stephan R Richter, Vibhav Vineet, Stefan Roth, and Vladlen Koltun. Playing for data: Ground truth from computer games. *ECCV*, 2016.

[18] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*, 2015.

[19] Viktor Rudnev, Mohamed Elgharib, William Smith, Lingjie Liu, Vladislav Golyanik, and Christian Theobalt. Nerf for outdoor scene relighting. In *ECCV*, 2022.

[20] Dave Shreiner, Bill The Khronos OpenGL ARB Working Group, et al. *OpenGL programming guide: the official guide to learning OpenGL, versions 3.0 and 3.1*. 2009.

[21] Jiajun Tang, Yongjie Zhu, Haoyu Wang, Jun Hoong Chan, Si Li, and Boxin Shi. Estimating spatially-varying lighting in urban scenes with disentangled representation. In *ECCV*, 2022.

[22] TurboSquid. *https://www.turbosquid.com*, Access date: 2023-05-17.

[23] Zian Wang, Wenzheng Chen, David Acuna, Jan Kautz, and Sanja Fidler. Neural light field estimation for street scenes with differentiable virtual object insertion. In *ECCV*, 2022.

[24] Zian Wang, Tianchang Shen, Jun Gao, Shengyu Huang, Jacob Munkberg, Jon Hasselgren, Zan Gojcic, Wenzheng Chen, and Sanja Fidler. Neural fields meet explicit geometric representations for inverse rendering of urban scenes. In *CVPR*, 2023.

[25] Pengchuan Xiao, Zhenlei Shao, Steven Hao, Zishuo Zhang, Xiaolin Chai, Judy Jiao, Zesong Li, Jian Wu, Kai Sun, Kun Jiang, et al. Pandaset: Advanced sensor suite dataset for autonomous driving. In *ITSC*, 2021.

[26] Ze Yang, Yun Chen, Jingkang Wang, Sivabalan Manivasagam, Wei-Chiu Ma, Anqi Joyce Yang, and Raquel Urtasun. Unisim: A neural closed-loop sensor simulator. In *CVPR*, 2023.

[27] Ye Yu, Abhimitra Meka, Mohamed Elgharib, Hans-Peter Seidel, Christian Theobalt, and William AP Smith. Self-supervised outdoor scene relighting. In *ECCV*, 2020.

[28] Ye Yu and William AP Smith. Inverserendernet: Learning single image inverse rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3155–3164, 2019.

[29] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. Pyramid scene parsing network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2881–2890, 2017.

[30] Yichao Zhou, Jingwei Huang, Xili Dai, Shichen Liu, Linjie Luo, Zhili Chen, and Yi Ma. Holicity: A city-scale data platform for learning holistic 3d structures. *arXiv*, 2020.