DIFFUSIONRENDERER: Neural Inverse and Forward Rendering with Video Diffusion Models

Supplementary Material

In the supplementary material, we provide additional implementation details (Sec. A) and further results and analysis (Sec. B). Please refer to the ACCOMPANYING VIDEO for more qualitative results and comparisons.

A. Experimental Settings

Implementation details. We fine-tune our models based on Stable Video Diffusion[1] [8].

For the *inverse renderer*, we modify the diffusion UNet by expanding four additional channels in the first convolutional layer to include image conditions. We optimize both the diffusion UNet parameters and the domain embedding parameters using a learning rate of 3×10^{-5}. The training is conducted with a batch size of 256, with a mix of multiple scene attributes. When generating the single-channel depth, metallic, and roughness maps, we average the outputs across the three channels to obtain the final result for each map.

In the *forward renderer*, we expand the first convolutional layer of the diffusion UNet by 20 additional channels to concatenate the additional pixel-aligned G-buffer conditions. Since the depth, metallic, and roughness maps are single-channel properties, we replicate each to create three-channel inputs before passing them into the VAE encoder $\mathcal{E}$. The weights of the cross-attention layers are repurposed for lighting conditions, and are reset prior to training. We use a learning rate of 1×10^{-4} for optimization.

Both models are trained using the AdamW optimizer for 20,000 iterations, with mixed-precision (fp16) training at a resolution of 512×512 pixels. The training takes around 2 days on 32 A100 GPUs. We have empirically observed that the video model performs best when trained on video lengths that it will encounter during inference. To ensure robust generalization across different frame lengths, we randomly select training video lengths of 1, 4, 8, 16, and 24 frames. This strategy allows the model to adapt effectively to varying video lengths during inference without compromising output quality. As a result, the models can also effectively process a single image by treating it as a video with one frame. During the training of both models, a 0.1 dropout is applied independently to each condition channel to reduce reliance on individual conditions and potentially enhance robustness. During inference, we empirically observe that a small classifier-free guidance (CFG) such as 1.2 enhances the visual quality of the forward rendering model. CFG does not provide noticeable benefit for the inverse rendering model and we do not use it for the inverse rendering model.

Data preparation. For synthetic data curation, we begin with the Objaverse [16] LVIS split, containing 46,207 3D models. The 3D assets are filtered based on the following criteria: (i) assets include valid PBR attributes such as roughness and metallic, (ii) assets can be rendered without geometry/texture artifacts. This process yields a final set of 36,500 3D assets. We collect 766 HDR panoramas from three sources: PolyHaven[2], DoschDesign[3], and HDRMaps[4]. For PBR textures, we collect 6,300 CC0 textures from multiple sources: 3D Textures[5], ambientCG[6], cgbookcase[7], PolyHaven[8], sharetextures[9], and TextureCan[10]. We remove textures that include only diffuse channels or lack diffuse textures, and manually exclude non-tileable textures, resulting in 4,260 high-quality PBR textures.

In each scene, we place a plane with a randomly selected PBR material, and sample up to three 3D objects, and place them on the plane after randomly rotating, translating, and scaling. We perform collision detection to avoid intersecting objects. We also place up to three primitives (cube, sphere, and cylinder) with randomized materials to cover complex lighting effects such as inter-reflections. The materials of primitives can be from the aforementioned texture maps or a monolithic material with varying albedo, roughness, and metallic. A randomly selected HDR environment map illuminates the scene. We also add random horizontal rotation, flipping, and intensity scaling to the environment map. The rendered videos contain 5 types of motions, 1) 360-degree camera orbits; 2) small-scale regional camera oscillation; 3) 360-degree rotating light with a fixed camera; 4) rotating objects with a fixed camera; and 5) translating objects around the plane.

We render videos of all scenes with corresponding intrinsic images in a custom path tracer based on OptiX [58], with 256 spp, OptiX denoising and AgX tonemapper[11]. In total, there are 150,000 videos with paired ground-truth G-buffers and environment maps, at 24 frames per video in 512x512

[1] https://huggingface.co/stabilityai/stable-video-diffusion-img2vid

[2] polyhaven.com/hdris (License: CC0)

[3] doschdesign.com(License: link)

[4] hdrmaps.com (License: Royalty-Free)

[5] https://3dtextures.me/tag/cc0/

[6] https://ambientcg.com

[7] https://www.cgbookcase.com/textures

[8] https://polyhaven.com/

[9] https://www.sharetextures.com

[10] https://www.texturecan.com

[11] https://github.com/sobotka/AgX

CVVDP ↑	*SyntheticObjects*	*SyntheticScenes*
DiLightNet [82]	5.44	2.99
Neural Gaffer [30]	6.49	3.47
Ours	**6.77**	**6.40**

Table S1. Quantitative evaluation of relighting in terms of ColorVideoVDP. ColorVideoVDP reports video quality in the JOD (Just-Objectionable-Difference) units. The highest quality (no difference) is reported as 10 and lower values are reported for distorted content. We compute a JOD value per clip for three novel lighting conditions in each series and report the average over all clips.

resolution.

Baseline configurations. DiLightNet [82] requires a text prompt per example, so we used `meta/llama-3.2-11b-vision-instruct`[12] to generate a short prompt for each example in *SyntheticObjects* and *SyntheticScenes* based on the first image in each clip and the instruction *"What is in this image? Describe the materials. Be concise and produce an answer with a few sentences, no more than 50 words."*

Environment map encoder pre-training. As detailed in the main paper, the environment lighting condition in our forward rendering model is encoded through cross-attention between the UNet's spatial latent features and the environment map representation. To provide effective lighting encodings, similar to VAE and CLIP embeddings in diffusion models, we propose pre-training an environment map auto-encoder specifically designed to capture HDR light intensity and orientation.

With both LDR space and log space environment maps ($\mathbf{E}_{\text{ldr}}$ and $\mathbf{E}_{\text{log}}$) as the model input and auto-encoder's reconstruction target, our encoder can retain detailed ambient lighting information while emphasizing high-intensity HDR light spots. To ensure precise control over light orientation in scene rendering, we introduce a directional encoding map, $\mathbf{E}_{\text{dir}}$, where each pixel represents a unit vector corresponding to a light direction in the camera coordinate system. By modifying $\mathbf{E}_{\text{dir}}$, the light orientation in the scene can be adjusted accordingly.

The pre-training process aims to produce an environment map encoder $\mathcal{E}_{\text{env}}$, capable of encoding complex directional HDR lighting. For this, we pair $\mathcal{E}_{\text{env}}$ with two auxiliary modules: an environment map decoder $\mathcal{D}_{\text{env}}$ and a direction query encoder $\mathcal{E}_{\text{dir}}$. This forms an auto-encoder training pipeline, as illustrated in Fig. S1. The encoder $\mathcal{E}_{\text{env}}$ processes concatenated VAE-encoded inputs $\mathbf{h}_{\mathbf{E}} = (\mathcal{E}(\mathbf{E}_{\text{ldr}}), \mathcal{E}(\mathbf{E}_{\text{log}}), \mathcal{E}(\mathbf{E}_{\text{dir}}))$, generating $K = 4$ levels of multi-resolution features $(\mathbf{h}^i_{\text{env}})_{i=1}^K$. Similarly, $\mathcal{E}_{\text{dir}}$ takes a VAE-encoded directional map $\mathbf{h}_{\mathbf{D}} = \mathcal{E}(\mathbf{E}'_{\text{dir}})$, producing features $(\mathbf{h}^i_{\text{dir}})_{i=1}^K$ of the same shape. The decoder $\mathcal{D}_{\text{env}}$ reconstructs the inputs $\mathbf{E}'_{\text{ldr}}$ and $\mathbf{E}'_{\text{log}}$ using the features $(\mathbf{h}^i_{\text{env}})_{i=1}^K$ and $(\mathbf{h}^i_{\text{dir}})_{i=1}^K$ through cross-attention layers. To

[12] https://www.llama.com/

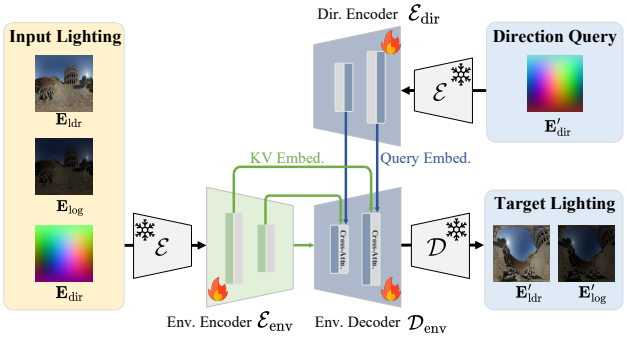

Figure S1. The overview of our environment map auto-encoder training pipeline.

enhance directional encoding, the training objective involves re-projecting the environment map with random rotations applied to the lighting sphere. This rotation information can be precisely represented by $\mathbf{E}'_{\text{dir}}$. To reconstruct the re-projected environment map, we use the features $(\mathbf{h}^i_{\text{dir}})_{i=1}^K$ encoded from $\mathbf{E}'_{\text{dir}}$ as embedding to query the directional HDR lighting encoded in $(\mathbf{h}^i_{\text{env}})_{i=1}^K$ (serving as key-value embedding) through the cross-attention layers in environment map decoder $\mathcal{D}_{\text{env}}$. The training objective therefore is:

$$\mathcal{L}_{\text{env}} = \|\mathbf{h}_{\mathbf{E}'} - \mathcal{D}_{\text{env}}(\mathcal{E}_{\text{env}}(\mathbf{h}_{\mathbf{E}}), \mathcal{E}_{\text{dir}}(\mathbf{h}_{\mathbf{D}}))\|^2 \tag{7}$$

where $\mathbf{h}_{\mathbf{E}'} = (\mathcal{E}(\mathbf{E}'_{\text{ldr}}), \mathcal{E}(\mathbf{E}'_{\text{log}})) \in \mathbb{R}^{\times h_{\text{env}} \times w_{\text{env}} \times 8}$.

Object Insertion. We provide additional details of object insertion application shown in main paper Fig. 8. The objective is to seamlessly insert an object (either 2D or 3D) into a given background image $\mathbf{I}_{\text{bg}}$, ensuring consistent appearance with the background (e.g., aligned lighting effects). Our method achieve this task with a combination of the inverse and forward rendering processes, as illustrated in Fig. S2.

First, our inverse rendering model estimates the G-buffer of the background image $\mathbf{I}_{\text{bg}}$. The G-buffer of the object to be inserted is obtained either through our inverse renderer or directly from a rendering engine. Based on the known foreground object mask $\mathbf{M}$, these G-buffers are then blended to create a composite G-buffer. Additionally, we estimate the lighting using an off-the-shelf model [61].

Using the composite G-buffer and estimated lighting, our forward rendering model generates two images: $\mathbf{I}^*_{\text{ins}}$ representing the scene with the inserted object, and $\mathbf{I}^*_{\text{bg}}$, the re-rendering of the original background. To minimize unintended changes to the original background image, we follow [41, 44, 74] and compute a shading ratio $\boldsymbol{\rho} = \mathbf{I}^*_{\text{ins}}/\mathbf{I}^*_{\text{bg}}$ that accounts for the relative shading effects introduced by the inserted object.

The final edited image $\mathbf{I}_{\text{ins}}$ is computed by multiplying the shading ratio with the original background image $\mathbf{I}_{\text{bg}}$ and compositing the masked foreground object $\mathbf{M} \cdot \mathbf{I}^*_{\text{ins}}$ onto the

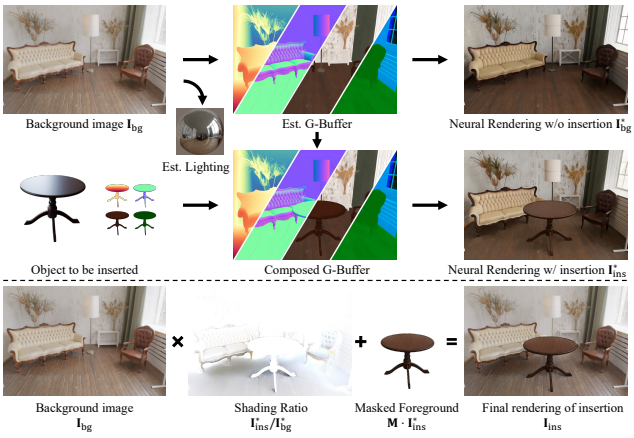

Figure S2. Overview of the object insertion workflow.

shaded background:

$$\mathbf{I}_{\text{ins}} = (1-\mathbf{M})\cdot\mathbf{I}_{\text{bg}}\cdot\frac{\mathbf{I}^*_{\text{ins}}}{\mathbf{I}^*_{\text{bg}}} + \mathbf{M}\cdot\mathbf{I}^*_{\text{ins}} \tag{8}$$

This process is visualized in Fig. S2 (bottom).

B. Additional Results

Runtime cost. Since our models are built on top of Stable Video Diffusion, the inference runtime cost of our models is roughly on the same level as Stable Video Diffusion. For a 24-frame video with a resolution of 512x512, the peak GPU memory cost for both models at inference time is around 21 GB. the inverse rendering model takes 9.7 seconds to perform 20 denoising steps including VAE encoding and decoding, clocked on one A100 GPU. The forward rendering model takes 20.3 seconds to run 20 denoising steps including VAE encoding and decoding. The increased runtime of the forward renderer is due to additional condition signals, which require extra time for encoding.

Without a separate environment map encoder, Ours (w/o Env. Encoder) completes 20 denoising steps in 19.9 seconds. The runtime overhead introduced by the environment map encoder is negligible.

Temporal consistency. In Table S1 we report ColorVideoVDP [50] (CVVDP) scores for the relighting comparison (c.f., Table 2 and Fig. 6 in the main paper). CVVDP predicts the perceptual difference between pairs of videos and accounts for spatial and temporal aspects of vision. We note that our method has the highest CVVDP score for both test sets, which is consistent with visual inspections. Please refer to the supplemental video to assess temporal consistency. In contrast to Neural Gaffer and DiLightNet, which leverage image diffusion models, our approach builds upon video diffusion models, which provide considerably improved temporal consistency. For reproducibility, CVVDP was configured according to:

```
ColorVideoVDP v0.4.2, 75.4 [pix/deg], Lpeak=200,
Lblack=0.2, Lrefl=0.3979 [cd/m^2] (standard_4k).
```

User study. We conducted a user study to evaluate the image perceptual quality of our method. In this study, participants were shown a reference path-traced rendering alongside a pair of renderings: one from our method and one from a baseline (randomly shuffled). They were asked to select which rendering perceptually more closely resembles the reference, considering aspects like lighting, shadows, and reflections. This user study was conducted for both neural rendering and relighting tasks. The evaluation data were sampled from SyntheticScenes and SyntheticObjects (the same datasets used for Table 1 and Table 2) (70 scenes). For each comparison, we collected 9 user selections to determine the preferred rendering by majority voting. The preference percentages for our method compared to baseline approaches are reported across all examples. Inspired by GPTEval3D [77], we repeat this experiment using GPT-4V as perceptual evaluators. Reported in Table S2, the user study results align with our findings in the main paper, and indicate a reasonable level of agreement between human and GPT-4V assessments.

		Neural Rendering				Relighting	
		SSRT	SplitSum	RGB↔X	DiLightNet	DiLightNet	N.Gaffer
Scenes	Human	72%	75%	85%	85%	90%	65%
	GPT4V	40%	50%	80%	85%	60%	68%
Objs	Human	37%	43%	76%	83%	57%	57%
	GPT4V	57%	45%	87%	54%	55%	52%

Table S2. **User study.** We report the percentage of images where users preferred Ours over baselines. A preference $> 50\%$ indicates Ours outperforming baselines. Evaluation follows main paper Table 1, 2 on *SyntheticScenes* and *SyntheticObjects*.

Comparison with FEGR [75] and UrbanIR [46]. We additionally compare to 3D inverse rendering and relighting approaches FEGR [75] and UrbanIR [46] in Fig. S5. These methods optimize neural 3D representation, then use volume rendering and PBR to produce the final relighting result. As the input data is limited to a single illumination condition, they often cannot cleanly remove shadows from the albedo, resulting in shadow artifacts in re-lit results. Additionally, existing scene reconstruction methods struggle to handle highly detailed structures such as trees, and dynamic scenes, which limits their fidelity for PBR path tracing. In contrast, our method consistently generates more photorealistic results without relying on explicit 3D geometry constraints. We refer to the accompanying video for animated results.

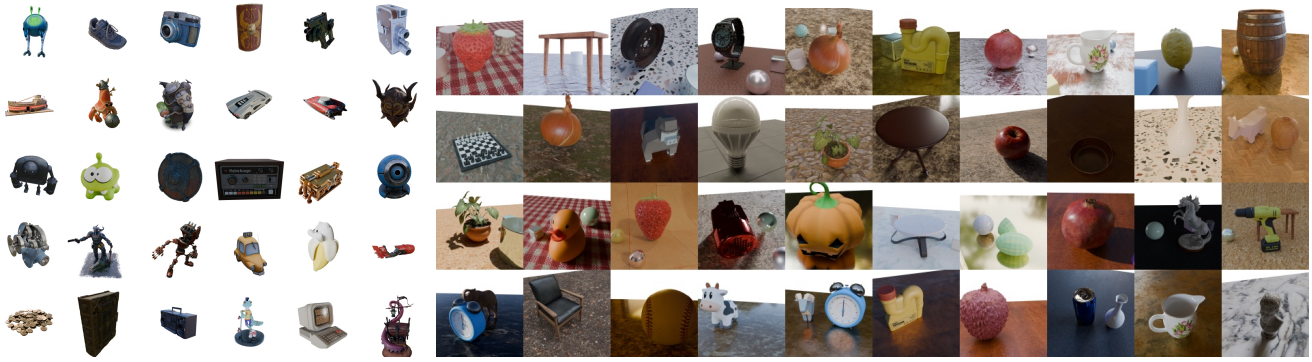

Figure S3. Visualization of the synthetic datasets for quantitative evaluation.

Figure S4. Additional qualitative comparison of relighting. Our method produces more accurate specular reflections compared to the baselines.

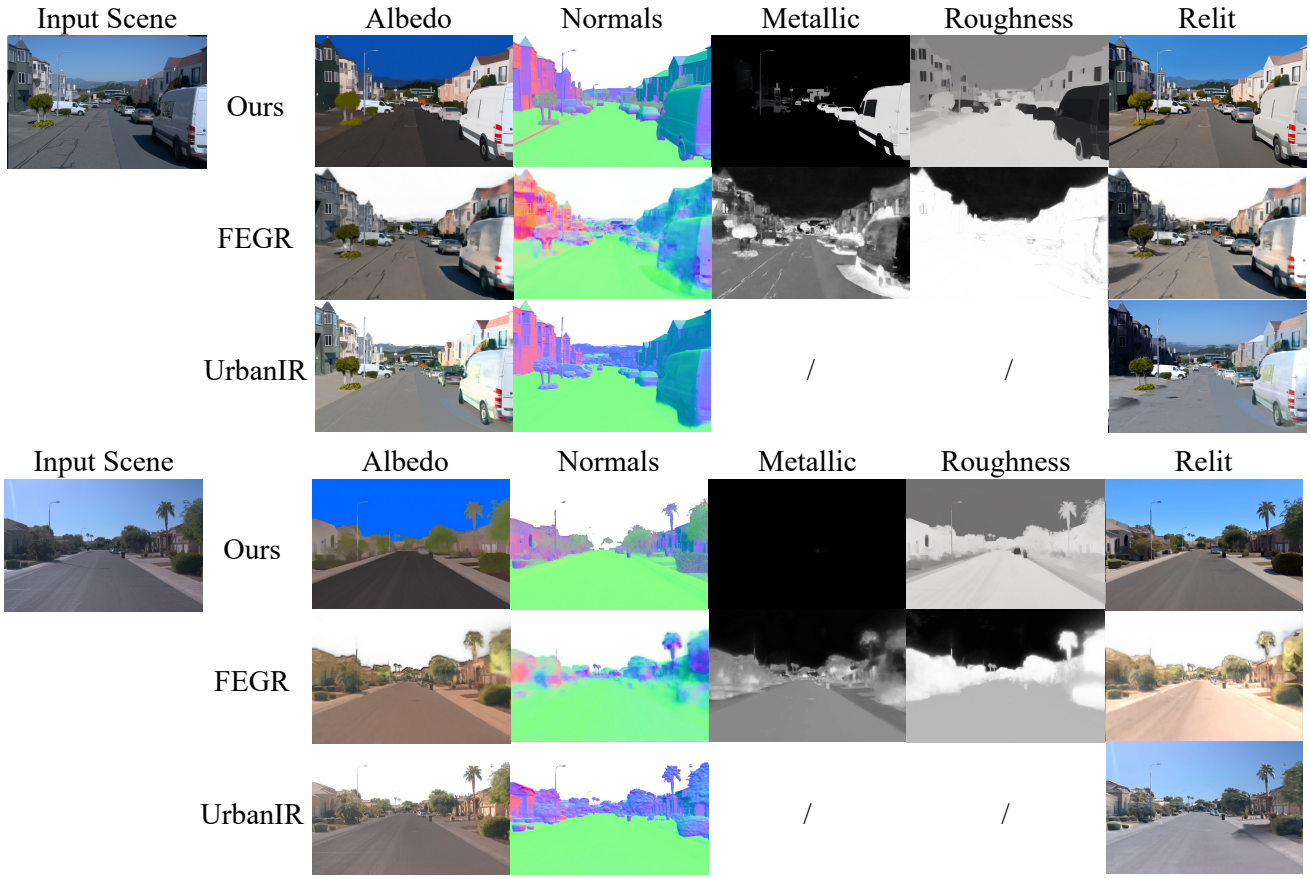

Figure S5. Qualitative comparison of inverse rendering and relighting on Waymo dataset with FEGR [75] and UrbanIR [46].