Supplementary material

A. Implementation details

A.1. Model architecture

As described in Sec. 3.2, R3D2 builds upon the work of [24], with key differences in the training procedure, application domain, and dataset. One of the most notable architectural changes to the base pretrained model lies in the autoencoder design. In particular, we modify the skip connections to capture the latent representations at three hierarchical levels of the encoder (or four in the case of R3D2-BIG), specifically just before each downsampling operation. These latent features are then processed using zero convolutions and subsequently added to the corresponding latent representations in the decoder after each upsampling stage. The zero convolutions are initialized as zeros during training.

A.2. Training setup

The pretrained model struggles with high resolution generation as it wasn't trained for it. When fine-tuned for our application at low resolutions the results are noticeably worse. Therefore, we train our model at a resolution of 1080×1920, the same resolution used for performing our inference tasks and benchmarks.

Given the high demands of training at high resolution we are limited to batch size of 1 per GPU. In total $8\times$ A100 80GB GPUs are used for training in order to speed up development process. The training took about 10 hours, amounting to around 81 GPU hours.

We use the AdamW [19] optimizer with a learning rate of $lr = 0.0001$ and employ a linear warm-up schedule for the first 500 steps. Our loss is a combination of the perceptual LPIPS loss [51], and the Gram-matrix loss [28]

$$\mathcal{L} = \lambda_{\text{LPIPS}}\mathcal{L}_{\text{LPIPS}} + \lambda_{\text{gram}}\mathcal{L}_{\text{gram}},$$

with $\lambda_{\text{LPIPS}} = 1.0$ and $\lambda_{\text{gram}} = 0.5$.

A.3. Inference setup

To evaluate the inference speed of the diffusion model with all our modifications we compile the models with the public available package `stable-fast`. Some of the optimizations performed by stable-fast are: fused CUDNN convolution kernels, fused GEMM kernels, fused multi-head attention and the use of CUDA graph format. All tests were performed at a resolution of 1080×1920 and at a batch size of one to make them relevant to real time data stream applications.

Tests were performed on the top consumer grade and enterprise grade GPUs as these should provide a more significant benchmark for years to come. As explained in Sec. 4.4, 1000 warm-up steps were performed followed by 1000 forward passes through the model. This number was chosen by continuously running forwards passes and evaluating how both GPUs perform over time and how many steps they take to stabilize to a given speed. In our tests the NVIDIA H200 took the longest to stabilize and the value 1000 was considered as a safe value to ensure proper warm-up for both. We performed the same test multiple times to ensure it resulted in negligible value changes. From the 1000 forward passes we can compute the inference speed and its standard deviation. The results of the benchmark are shows in Tab. 4.

Table 4. Results for inference speed and deviation in frames per second (FPS) on both consumer hardware and enterprise server hardware.

Method	Rendering speed (FPS) RTX 5090	H200
Naive insertion	-	-
+ R3D2	13.359 ± 0.024	18.129 ± 0.078
+ R3D2-BIG	4.067 ± 0.005	5.964 ± 0.037

B. Shortcoming of current methods

B.1. Image-to-image style transfer baselines

To further highlight the shortcomings of current image-to-image methods we provide extra samples with various prompts and different resolutions (Fig. 8). With GPT4o it is not possible to control the resolution of generation, but one can ask it to generate a square image instead which is the closes solution, however it drastically changes geometry and objects. SDEdit struggles to generate consistent images, particularly at higher resolution. Both instruct-Pix2Pix and CosXL-Edit fail to follow the directions and have a tendency to degrade the image further.

Given the similarities in methodology and application, we tested Difix3D+ [43] and showcase how foreign object insertion with realistic light effects cannot be generalized from training a model to fix gaussians (see Tab. 5). Some samples are provided in Fig. 9, demonstrating how this method is incapable of adding the shadows and light effects needed to blend the foreign asset to the environment.

Table 5. Comparison between R3D2 and Difix3D+ on inserted assets

Method	LPIPS ↓	SSIM ↑	PSNR ↑
Original Reconstruction	0.177	0.896	28.94
Naive Insertion	0.211	**0.866**	24.48
Naive Insertion + R3D2	**0.142**	0.862	**25.29**
Naive Insertion + Difix3D	0.254	0.806	23.58

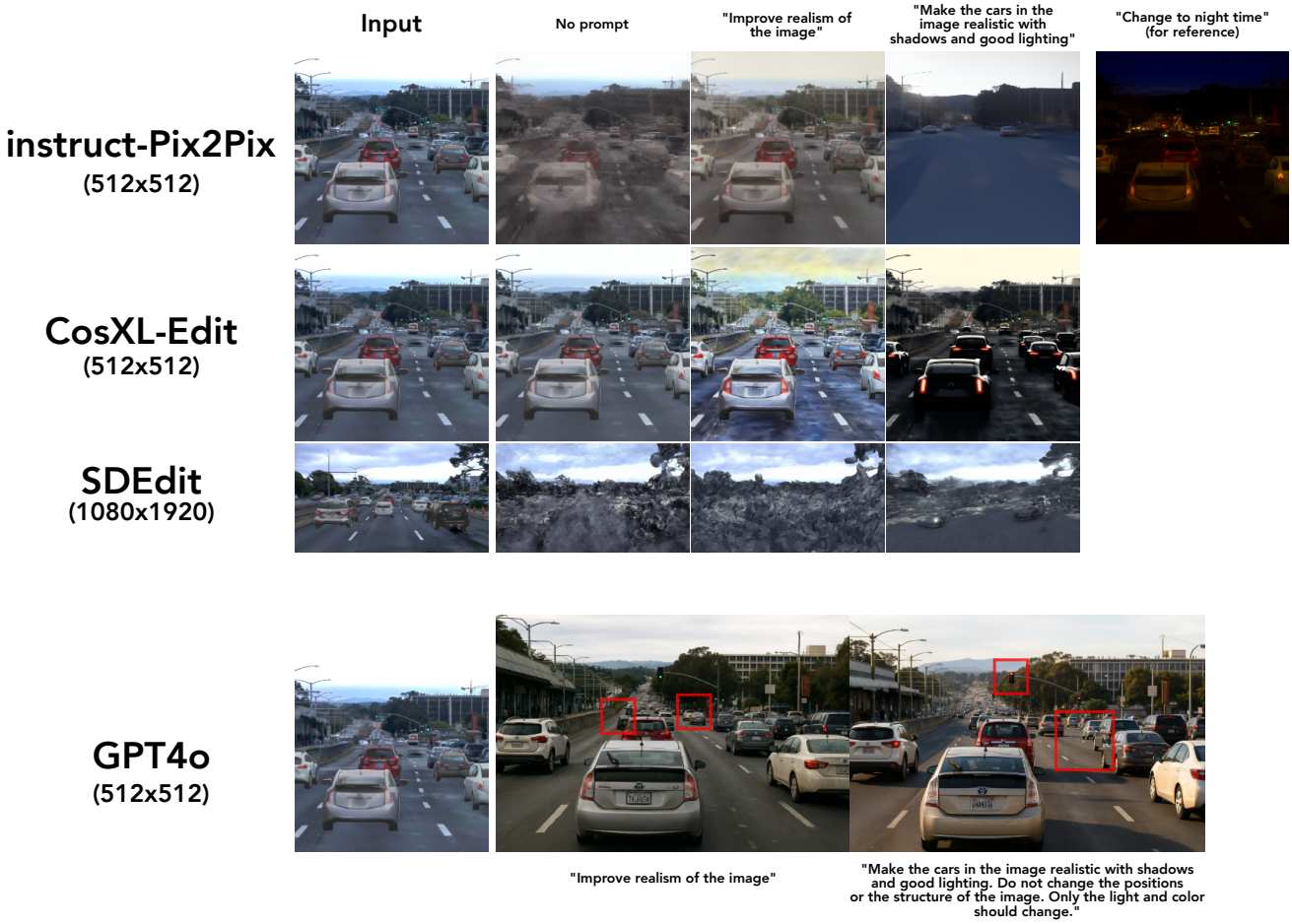

Figure 8. Additional samples of off-the-shelf image editing models applied to our use case.

Figure 9. Samples of Difix3D+ [43] applied on the R3D3 dataset. The model sharpens the image, saturates the colors, adds contrast and improves the look of the vehicle but it does not attempt to add any shadows or realistic light effects.

B.2. Shortcomings of R3D2

As discussed in Sec. 5, currently R3D2 does not address the issue of temporal consistency. Because R3D2 processes each frame independently, applying it to sequential frames in a video or across views from a multi-camera rig can result in subtle flickering or visual inconsistencies. While we believe that extending our approach to video models will resolve this, current video-based architectures are computationally intensive.

C. Neural reconstruction details

C.1. Reconstruction sequence selection

Current neural reconstruction methods struggle in extreme weather conditions, such as heavy rain/fog/snow or when the cameras experience harsh sun glares. We base our selection on filtering such scenes as well as removing scenes without any dynamic object presents.

C.2. Deviations from original reconstruction method

SplatAD [9] employs *feature splatting*, meaning that instead of rendering an image with RGB-color ($H \times W \times 3$) as vanilla gaussian splatting does, they render a feature-map $H \times W \times f$, where $f \in \mathbf{R}^d$. They then use a CNN-based decoder to obtain the final image $\mathcal{D} : \mathbf{R}^d \rightarrow \mathbf{R}^3$ and show that this yields better reconstruction results [9]. However, as the decoder is also scene-specific, it hinders seamless transfer of objects across scenes. We therefore disable the feature splatting and use the vanilla approach of directly rendering the RGB color of the scene.

C.3. Partial reconstruction of 3D assets in traditional neural rendering frameworks

As discussed in the main manuscript, although state-of-the-art autonomous driving (AD) scene reconstruction methods achieve impressive results, they often produce partially reconstructed objects. This limitation arises because objects in typical AD data are frequently observed from limited, often similar, viewpoints. To illustrate this issue, we present multi-view images of partially reconstructed objects in Fig. 10. As shown, objects are generally well reconstructed from certain viewpoints but lack detail and completeness when viewed from novel directions.

D. Dataset details

D.1. Asset generation with Amodal3R

TRELLIS [45] enables text- and/or image-conditioned 3D asset generation. A key limitation is its assumption that conditioning images are object-centric and non-occluded. This poses a problem for in-the-wild vehicle data, where objects are often partially occluded (e.g., by other cars or barriers). To overcome this, we employ Amodal3R [44], which facilitates image conditioning with occluded objects. Amodal3R requires three inputs: 1) the object-centric (potentially occluded) original image crop, 2) a segmentation mask of the object of interest, and 3) a foreground occlusion mask highlighting the object's occluded regions.

While the original image crop and the object's segmentation mask are readily available from WOD [37] and its object-assets extension (WOD-OA) [35], the foreground occlusion mask requires further processing. We generate this mask using pix2gestalt [23], a method that recovers an object's complete shape from a partially occluded view. Specifically, pix2gestalt produces an amodal (whole object) mask. Because we observed that pix2gestalt tends to slightly overestimate the object's size, we first erode its generated amodal mask using a kernel of size $k = 10$. The foreground occlusion mask is then obtained by subtracting the original object segmentation mask (from WOD-OA) from this eroded amodal mask.

Amodal3R employs a dual-diffusion process: the first generates the object's structure, and the second, conditioned on this structure, generates its features (which are used to generate properties like color and opacity when decoding Gaussian primitives). We use $n_{\text{steps}} = 250$ for each process, keeping other parameters at their default values. To condition on multiple images of the target object, we utilize Amodal3R's *stochastic multi-sampling*, which randomly samples a new conditioning image at each diffusion timestep. Further details are available in the original publication [44].

D.2. Text-to-3D generation

To generate our text-to-3D assets we use TRELLIS [45]. As text-conditioning we use a combination of randomly sampled types, styles, and colors according to the sets of prompts below. Example assets are shown in Fig. 11.

For vehicle types we use: `car`, `truck`, `van`, `SUV`, `sedan`, `coupe`, `hatchback`, `convertible`.

For styles we use: `new`, `old`, `rusty`, `shiny`, `dusty`, `muddy`, `sporty`, `modern`, `luxury`, `vintage`.

For colors we use: `red`, `blue`, `green`, `yellow`, `black`, `white`, `silver`, `gray`, `brown`, `orange`.

E. Additional qualitative examples

We show additional qualitative examples in Fig. 12 highlighting the capabilities of R3D2.

Figure 10. Multi-view images of partially reconstructed dynamic objects.

Figure 11. Example of text-to-3D assets generated with TRELLIS [45].

Figure 12. Samples of R3D2 applied on reinserted actors.