Wonderland: Navigating 3D Scenes From a Single Image

Supplementary Material

Table of Contents

Table A1. **Mathematical notation used in the paper.**

Expression	**Specification**	**Explanation**
Commonly Used		
x	$x \in R^{T\times H\times W\times 3}$	source video clip
s	–	stride in sampling clip x from source video
z	$z \in R^{t\times h\times w\times c}$	video latent embedded from x
$\mathcal{E}$	–	encoder from 3D-VAE
r_s	$r_s = \frac{H}{h} = \frac{W}{w}$	spatial compression rate
r_t	$r_t = \frac{T}{t}$	temporal compression rate
p	$p \in R^{T\times H\times W\times 6}$	Plücker embedding of cameras of video clip x
Diffusion Related		
τ	–	diffusion time step
ϵ	$\epsilon \in R^{t\times h\times w\times c}$	random noise such that $\epsilon \sim \mathcal{N}(0, I)$
z_τ	$z_\tau \in R^{t\times h\times w\times c}$	noisy video latent
D_θ	–	diffusion model parameterized by θ
y	–	conditional signal, usually textual embedding
o_v	$o_v \in R^{N_v\times d_v}$	visual tokens as a sequence in the diffusion model
$o_{\text{ctrl}}, o_{\text{lora}}$	$o_{\text{ctrl}}, o_{\text{lora}} \in R^{N_v\times d_v}$	camera tokens as a sequence in the diffusion model
N	–	number of transformer blocks in the ControlNet branch
Reconstruction Related		
p_l	–	spatial patch size applied to z in the LaLRM
o_l	$o_l \in R^{N_l\times d_l}$	visual latent tokens as a sequence in the LaLRM
N_l	$N_l = t \cdot \frac{h}{p_l} \cdot \frac{w}{p_l}$	number of visual latent tokens in the LaLRM
o_{p}	$o_{\text{p}} \in R^{N_l\times d_l}$	camera tokens as a sequence in the LaLRM
V	–	number of supervision views in the LaLRM
G	$G \in R^{(T\times H\times W)\times 12}$	Gaussian feature map in the LaLRM

A. Implementation Details

Table A1 provides a list of the mathematical notation used in the paper.

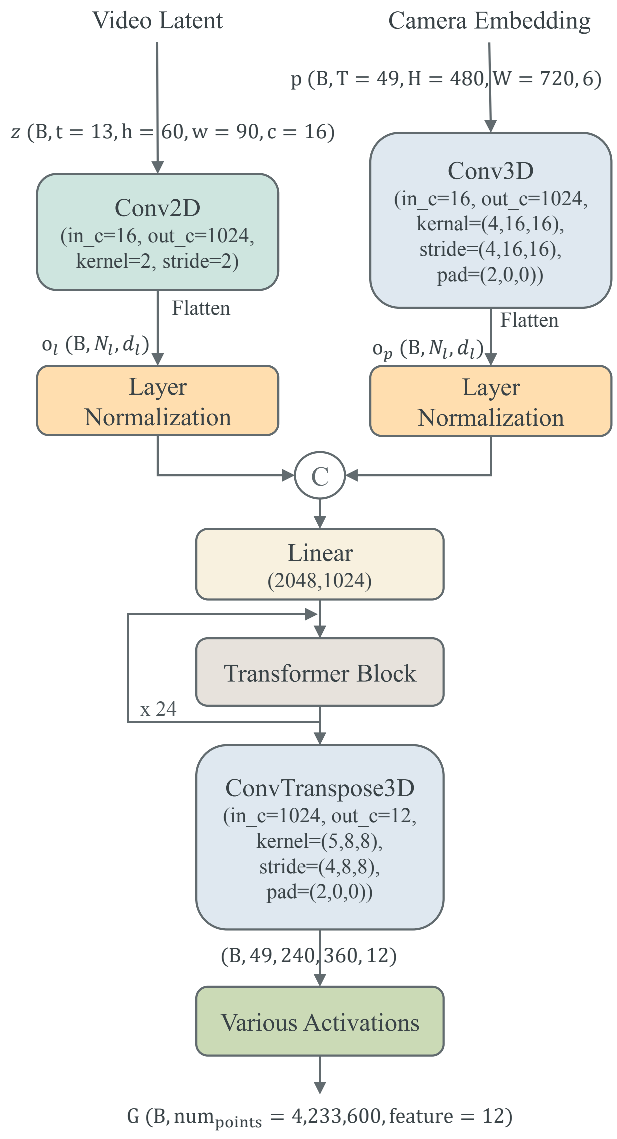

Video Latent
Camera Embedding
p (B, T = 49, H = 480, W = 720, 6)
z (B, t = 13, h = 60, w = 90, c = 16)
Conv2D (in_c=16, out_c=1024, kernel=2, stride=2)
Conv3D (in_c=16, out_c=1024, kernal=(4,16,16), stride=(4,16,16), pad=(2,0,0))
Flatten
Flatten
o_l (B, N_l, d_l)
o_p (B, N_l, d_l)
Layer Normalization
Layer Normalization
C
Linear (2048,1024)
Transformer Block
x 24
ConvTranspose3D (in_c=1024, out_c=12, kernel=(5,8,8), stride=(4,8,8), pad=(2,0,0))
(B, 49, 240, 360, 12)
Various Activations
G (B, num_points = 4,233,600, feature = 12)

Figure A1. **Architecture of the Latent Large Reconstruction Model (LaLRM).** Given a video latent z and the the Plücker embedding of cameras, the LaLRM directly regresses the 3DGS of the scenes in a feed-forward manner.

A.1. Additional Architectural Details

Our framework comprises a Camera-Guided Video Diffusion Model and a Latent Large Reconstruction Model (LaLRM) capable of generating 3D scenes conditioned on single images. Our architectural designs effectively align the generation and reconstruction tasks, bridging the image space and 3D space through the video latent space.

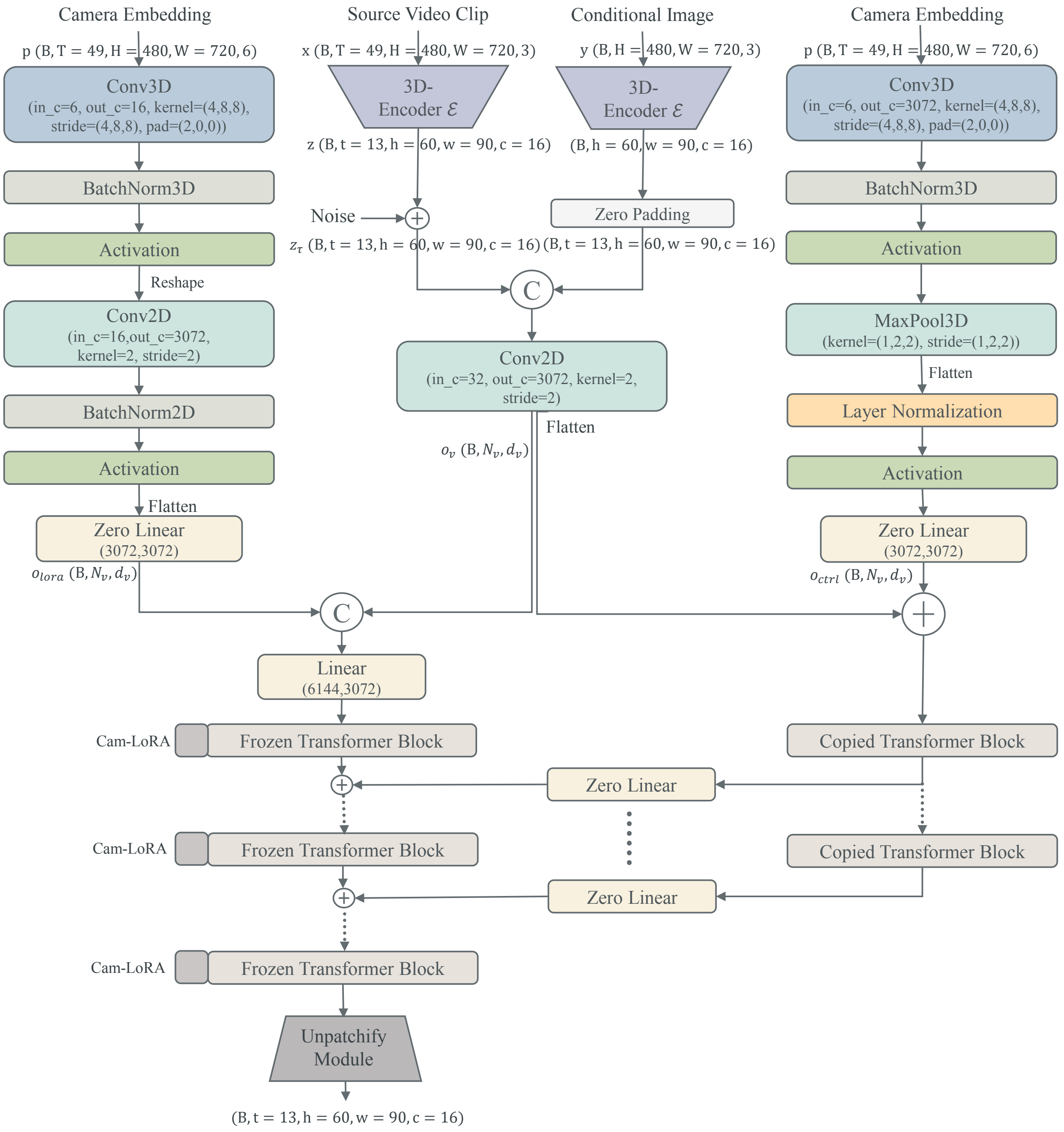
Figure A1 illustrates the architectural details of the LaLRM. Given video latents that can be generated by the video diffusion model (during inference time) or can be embedded from source video clips (during training time), a lightweight tokenization module projects these latents to visual tokens. Patchification is performed on the video latent along the spatial dimensions with a mild patch size p_l to obtain $o_l \in R^{N_l \times d_l}$, where $N_l = t \cdot \frac{h}{p_l} \cdot \frac{w}{p_l}$. For the Plücker embedding, we 3D-patchify along spatiotemporal dimensions with 3DConv. We purposefully set the spatial patch size to $p_l \cdot r_s$ and the temporal patch size to r_t (defined in Section 3.1 of the paper), producing pose tokens o_p that match the length with o_l. The two sets of tokens are merged with channel-wise concatenation and fed into a sequence of transformer blocks to regress the 3D Gaussian features. We fulfill pixel-level correspondence between the Gaussians and source video x in the RGB space via the latent decoding module, which involves a 3D-DeConv layer with upsampling strides $(r_t, p_l{\cdot}r_s, p_l{\cdot}r_s)$ and the 12-channel output is the Gaussian feature map $G \in R^{(T\cdot H\cdot W)\times 12}$.

Figure A2 illustrates the architectural details of integrating camera embeddings into the pre-trained video diffusion transformer during the training stage, with a ControlNet-branch and a LoRA-branch. Each branch involves a lightweight camera encoder composed of convolutional layers and zero-linear layers. The camera encoders project the camera embedding into camera tokens of the same dimension as the visual tokens. The visual tokens and camera tokens are concatenated or element-wise added before they are fed into the main branch and the ControlNet branch. The visual-camera tokens are further processed by a sequence of transformer blocks and mapped to the same dimension of the added noise by the unpatchify module. Note that for simplicity in Figure A2, we have omitted the text tokens, diffusion time embeddings, and positional embeddings.

A.2. Training and Evaluation Details

The camera-guided video diffusion model uses a transformer-based video diffusion model; namely, CogVideoX-5B-I2V [106] that generates 49 frames of 480 × 720 pixel resolution.[2] An encoder from 3DVAE compresses video clips with ratios of $r_t = 4$ and $r_s = 8$, yielding latents of dimensionality $13 \times 60 \times 90$. To build the ControlNet branch, we use the first $N = 21$ base transformer blocks from the video backbone to initialize the weights. The camera-LoRA has a low rank of dimension 256. The model is trained with a batch size of 24 for 40K steps, using the Adam optimizer [22] with a learning rate of 2×10^{-5}, $\beta_1 = 0.9$, $\beta_2 = 0.95$, and weight decay 1×10^{-4}. During the evaluation, we randomly sampled 300, 300, and 100 video clips from the RE10K test set, DL3DV-140, and Tanks-and-Temples, respectively. For each video clip, we sampled a

[2]The references cited in this document are listed in the main paper.



Camera Embedding
p (B, T = 49, H = 480, W = 720, 6)
Conv3D (in_c=6, out_c=16, kernel=(4,8,8), stride=(4,8,8), pad=(2,0,0))
BatchNorm3D
Activation
Reshape
Conv2D (in_c=16,out_c=3072, kernel=2, stride=2)
BatchNorm2D
Activation
Flatten
Zero Linear (3072,3072)
o_{lora} (B, N_v, d_v)
Source Video Clip
x (B, T = 49, H = 480, W = 720, 3)
3D-Encoder $\mathcal{E}$
z (B, t = 13, h = 60, w = 90, c = 16)
Noise
z_τ (B, t = 13, h = 60, w = 90, c = 16)
Conditional Image
y (B, H = 480, W = 720, 3)
3D-Encoder $\mathcal{E}$
(B, h = 60, w = 90, c = 16)
Zero Padding
(B, t = 13, h = 60, w = 90, c = 16)
C
Conv2D (in_c=32, out_c=3072, kernel=2, stride=2)
Flatten
o_v (B, N_v, d_v)
Camera Embedding
p (B, T = 49, H = 480, W = 720, 6)
Conv3D (in_c=6, out_c=3072, kernel=(4,8,8), stride=(4,8,8), pad=(2,0,0))
BatchNorm3D
Activation
MaxPool3D (kernel=(1,2,2), stride=(1,2,2))
Flatten
Layer Normalization
Activation
Zero Linear (3072,3072)
o_{ctrl} (B, N_v, d_v)
C
Linear (6144,3072)
Cam-LoRA
Frozen Transformer Block
Zero Linear
Copied Transformer Block
Cam-LoRA
Frozen Transformer Block
Zero Linear
Copied Transformer Block
Cam-LoRA
Frozen Transformer Block
Unpatchify Module
(B, t = 13, h = 60, w = 90, c = 16)

Figure A2. **Architecture of Dual-branch Camera-guided Video Diffusion Model.** We show the skeletons of the training pipeline, where random noise is added to the video latents. The conditional image is merged to the noisy latents via feature concatenation. The camera guidance is integrated with LoRA-branch (left) and ControlNet-branch (right). We ignore the text tokens, the diffusion time embeddings, the positional embeddings, and some reshaping operations for simplicity in the figure. In the foundation diffusion transformer, the text tokens are concatenated along *number-of-token* dimension with visual tokens. Thus we apply zero-padding to camera tokens to guarantee the same length before concatenation or element-wise sum. By default, we use SiLu as our activation function.

starting frame as the image condition and the subsequent n camera poses as pose conditions, where n is determined by the length of the generated video. Since different datasets have diverse FPS, we sampled n poses at different strides s ($s = 3$ for RE10K, $s = 1$ for DL3DV-140, and $s = 4$ for Tanks-and-Temples) to ensure smooth and noticeable view transitions. Compared with the baselines, our method generates the longest videos, so the baselines perform inference using the front subset of our sampled poses. The generated videos are COLMAPed to obtain camera poses, followed by conversion of the camera system to be relative to the first frame and normalization of all cameras to a common scale [4, 31]. We use translation and rotation error to measure camera-guidance precision [2, 4, 31, 101]. For a fair comparison, we measure the mean errors across the first 16 frames for each method; *i.e.*, VD3D [4] (16-frame), ViewCrafter [111] (25-frame), and Ours (49-frame). SVD-based [6] MotionCtrl [95] generates 14-frame videos, and all frames are used. Visual similarity is assessed by calculating PSNR, SSIM [94], and LPIPS [115] between the generated images and ground-truth views. Similarly, for a fair comparison, the first 14 frames in the generated videos are evaluated. An important issue is that generated videos tend to deviate from the conditional view and present diverse appearances as the scene progresses. It is therefore less reliable to evaluate quality using similarity metrics that measure the differences between generated frames and ground-truth views.

For the latent large reconstruction model, we use a patch size $p_l = 2$ for the visual latent, and use a temporal patch size of 4 and a spatial patch size of 16 for the camera Plücker embedding. We follow reference [113] and use the same architecture for the transformer blocks. We use 24 base transformer blocks with a hidden dimension of 1,024. The latent decoding module has a 3D-DeConv layer with upsampling strides (4, 16, 16). The backbone transformer network is efficiently implemented with FlashAttentionV2 [21] and optimized with mixed precision training [67] using the BF16 datatype.

We first train the model with low-resolution video clips of dimensionality $49 \times 240 \times 360$ and their corresponding latents of dimensionality $13 \times 30 \times 45$. Then, we fine-tune the model with high-resolution $49 \times 480 \times 720$ dimensional video clips and corresponding $13 \times 60 \times 90$ dimensional latents. At this stage, due to memory constraints, we modify the 3D-DeConv layer in the latent decoding module with upsampling strides (4, 8, 8). Even with a smaller upsampling rate, for each 3D scene, our Gaussian prediction yields an enormous quantity of Gaussians to construct each scene; *i.e.*, $T \times \frac{H}{2} \times \frac{W}{2}$ (4,233,600). A total of $V = 48$ supervision views are used, for which we randomly select $V' = 24$ frames from each sampled video clip as seen views, and an additional 24 frames disjoint from the video clip as unseen views. The model is trained on low-resolution and high-resolution datasets for 200K and 100K iterations, respectively, using a cosine annealing schedule at a peak learning rate of 4×10^{-4} and 1×10^{-5}. We use a batch size of 24 with the Adam optimizer with $\beta_1 = 0.9$, $\beta_2 = 0.95$, and weight decay 1×10^{-4}. In the evaluation stage, following the video generation stage, we sample starting frames and poses with different strides in multiple benchmark datasets for video generation. The generated video latents are leveraged to reconstruct 3D scenes from which images are rendered for evaluation. At this stage, we assess performance exclusively using similarity-based metrics. Consistent with the video generation evaluation, we measure these metrics on the frames rendered from the first 14 camera poses.

B. Additional Analysis of Controllable Video Generation

This section presents further ablation analysis of our camera-conditioned video generation pipeline.

B.1. Effect of LoRA on Static Scene Generation and Camera Controllability

We employ LoRA fine-tuning in the main branch of our camera-guided video generation model. LoRA is advantageous for its compatibility with pre-trained models, as it introduces extensions without altering the original model weights. The lightweight module offers a cost-effective approach to fine-tuning heavy models, requiring minimal computational resources and reducing the risk of overfitting to customized datasets. Our framework takes advantage of LoRA to enhance static scene generation and camera controllability.

To evaluate the impact on *static scene generation*, we fine-tuned the I2V source model using LoRA on customized datasets dominated by static scenes, including RealEstate10K (RE10K), ACID, and DL3DV. In this scenario, no pose control was applied. We compared the performance of the fine-tuned model against the source model on 20 in-the-wild image prompts (along with text descriptions). The results show that the fine-tuned model generates significantly more varied static scenes compared to the source model, especially for cases with humans and animals. The visualizations in Figure A3 illustrate that LoRA enables the generation of more varied static scenes without compromising visual quality.

To assess the role of LoRA in *enhancing camera controllability*, we trained our full model without incorporating LoRA modules. For these scenarios, in the main branch, the camera embeddings are fed into the network in a channel-concatenation manner without LoRA tuning in the main backbone. Only the LoRA-camera encoder and linear processing layers at the top are learned. Following the ex-

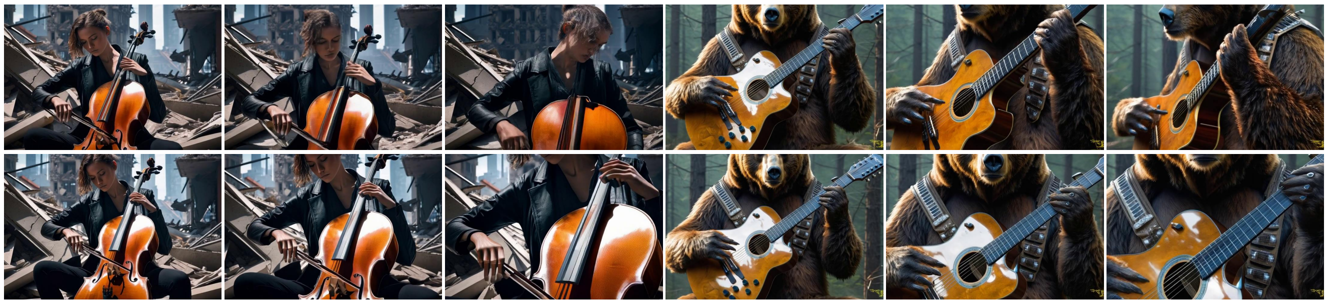


Figure A3. **Comparison of video generations** between the source model (*top row*) and the model fine-tuned on static-scene datasets with LoRA modules (*bottom row*). The results demonstrate that fine-tuning the model on static-scene datasets equipped with LoRA produces significantly more static scenes.

Table A2. **Analysis of architectural designs in camera-guided video generation model.** We report the performance for visual quality (FID and FVD) and pose control precision (R_{err} and T_{err}) from models trained on the RealEstate10K dataset. The first part of the table is adopted from Table 1 in the paper.

Architecture	Metrics			
	FID ↓	FVD ↓	R_{err} ↓	T_{err} ↓
Lora-branch	19.02	212.74	0.102	0.157
Ctrl-branch	18.75	205.45	0.058	0.104
Dual-branch	**17.22**	**183.54**	**0.052**	**0.095**
Dual w/o LoraModule	17.84	195.07	0.062	0.101
Ctrl-branch only				
w/o weight copy	18.92	206.75	0.065	0.108
block-1	19.90	214.66	0.114	0.162
blocks-10	19.15	210.74	0.075	0.126
blocks-30	20.15	221.61	0.056	0.105

perimental setup outlined in Section 4.3 of the paper, we evaluated the models on 100 video clips sampled from RealEstate10K. As shown in the middle part of Table A2, comparing the "*Dual w/o LoraModule*" and "*Dual-branch*" configurations reveals that LoRA plays a critical role in fine-tuning the main branch. Excluding LoRA results in a noticeable performance drop in both visual quality and camera-guidance precision.

B.2. Analysis of the ControlNet-Branch Design

In our full model, the ControlNet branch is designed by utilizing a trainable copy of the first 21 base transformer blocks of the foundational video model, which consists of 42 blocks in total. We extensively evaluated the design by using the ControlNet branch only. Specifically, we trained the model with ControlNet conditioning under various configurations, including using the first 21 blocks without weight copying, denoted *w/o weight copy*, and using the first 1, 10, or 30 blocks with weight copying, denoted block-1, blocks-10, and blocks-30, respectively. As shown in the third part of Table A2, comparisons among the different architectural variants and the"*Ctrl-branch*" reveal that weight copying substantially improves all metrics, particularly visual quality. Using only one block results in weak camera controllability, whereas increasing the number of blocks strengthens the ability of the model to guide camera poses. Notably, using 21 blocks ("*Ctrl-branch*") achieves similar levels of pose controllability as using 30 blocks, while maintaining high visual quality. Based on these observations, we selected the trainable copy of the first 21 base transformer blocks, as it provides an optimal balance between pose controllability and computational efficiency.

C. Additional Analysis of 3D Reconstruction

This section provides further analysis underlying our design choices related to the large-scale 3D reconstruction model.

C.1. Fine-Tuning With the In-the-Wild Dataset

In deploying the LaLRM, we adopt a progressive training strategy. During the second stage, we fine-tune the model by involving a self-generated in-the-wild dataset. To assess the impact of this dataset, we further fine-tuned a separate reconstruction model, LaLRM–, which excludes the in-the-wild dataset. We quantitatively compared the performances of LaLRM– and LaLRM on benchmark datasets and qualitatively evaluated them on 20 disjoint in-the-wild image prompts. The results reported in Table A3 indicate that incorporating the in-the-wild dataset during fine-tuning

Table A3. **Analysis on involving in-the-wild dataset to fine-tune LaLRM.** We report the performance on various benchmark datasets for novel view synthesis of 3D scenes, which are built from *single view* condition.

Method	RealEstate10K			DL3DV			Tanks-and-Temples		
Metrics	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑
LaLRM–	0.295	17.06	0.538	0.343	16.62	0.570	0.359	15.85	0.502
LaLRM	**0.292**	**17.15**	**0.550**	**0.325**	**16.64**	**0.574**	**0.344**	**15.90**	**0.510**

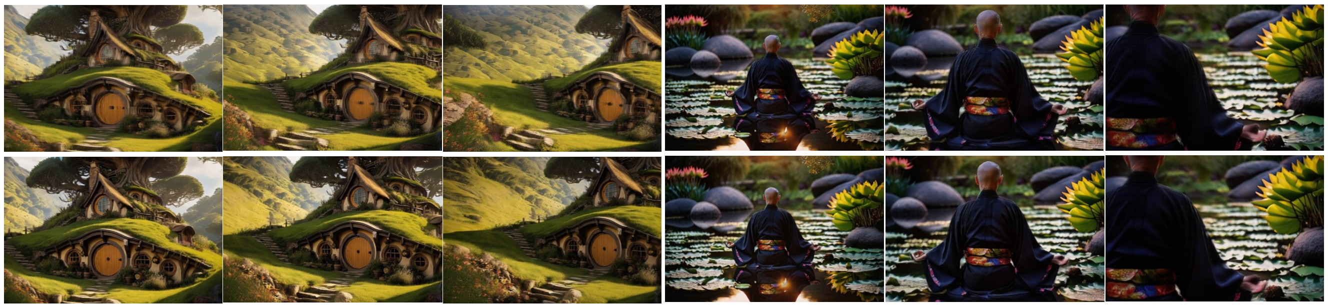


Figure A4. **Comparison of 3D rendering performance** between latent reconstruction models fine-tuned *without* in-the-wild dataset (*upper row*) and *with* in-the-wild dataset (*lower row*). Involving in-the-wild datasets during fine-tuning improves the generalization capability.

enhances the generalization capabilities of our model. Furthermore, as shown in Figure A4, LaLRM demonstrates noticeably better rendering quality compared to LaLRM–. This further validates the benefits of using in-the-wild data in the fine-tuning process.

D. Additional Discussion and Comparison of Related Work

This section provides a detailed discussion of our work relative to prior research in the use of generative priors for 3D rendering.

Significant advancements have been achieved in static and kinetic 3D object generation from text or single image prompts with notable improvements in quality and efficiency [1, 3, 10, 18, 27, 30, 34, 44, 52, 55, 63, 73, 79, 90, 93, 99, 100, 102, 105]. However, progress in 3D scene generation has lagged behind [16, 25, 64, 80, 86, 91, 108, 109, 111, 116]. Most approaches to 3D scene generation follow a two-stage process: First, novel views are generated from a single image and, second, these views are used to train a 3D representation with a per-scene optimization strategy.

Early methods, such as LucidDreamer [20] and RealmDreamer [80], explored scene-level 3D generation conditioned on text descriptions or single images. They relied on the 3D priors from incomplete point clouds constructed via depth prediction from single images. Then, they combined depth-based warping with diffusion-based image inpainting to complete the scenes in an autoregressive manner. These methods often struggle with inconsistencies in occluded regions, as the per-view inpainting process can introduce severe artifacts and discontinuities, particularly in unseen areas. WonderJourney [109], which targets wide-scene generation, also employs image inpainting diffusion models to fill unseen regions rendered from limited point clouds. However, as shown in our main comparisons, this method similarly suffers from 3D incoherence in occluded areas. Also, all these works do not have automatic and explicit control over camera poses during the generation process.

Other efforts, such as Cat3D [25] and ReconFusion [96], address multi-view consistency by incorporating camera conditioning into image diffusion models. Nonetheless, a noticeable issue is the tendency to produce blurry or distorted background regions, particularly when conditioned on a single image. This arises from the use of image diffusion models to obtain dense views auto-regressively, which are then used for 3D reconstruction via per-scene optimization [48]. Image diffusion models lack built-in mechanisms to guarantee cross-view consistency and such a multiple-shot generation strategy often introduces inconsistencies, especially for wide-view scenarios.

More recent efforts, such as ReconX [59] and ViewCrafter [111], leverage video diffusion models and global point clouds to enhance multi-view consistency. However, as demonstrated in our main comparisons, these methods are sensitive to the initialization of point clouds and are restricted to generating narrow-scope scene representations. Additionally, they lack explicit pose control during the generation process.

Importantly, all previous methods depend on time-consuming per-scene optimization such as NeRF [68] or 3DGS [47]. By contrast, our approach integrates explicit camera control into a video diffusion model to enable precise and expansive scene generation. Our large-scale 3D reconstruction model is capable of efficiently constructing 3D scenes from video latents. Its design effectively aligns the generation and reconstruction tasks and bridges the image space and 3D space through the video latent space, eliminating the need for time-consuming per-scene optimization.

Comparison of Mip-NeRF: We next compare our method with Cat3D [25] on more complex scenes from Mip-NeRF [5]. Due to the lack of open-source code for Cat3D, we retrieved the demonstration results directly from the source webpage of reference [25]. The images in Figure A5 were rendered from 3DGS representations generated with orbital camera trajectories. For each scene, we

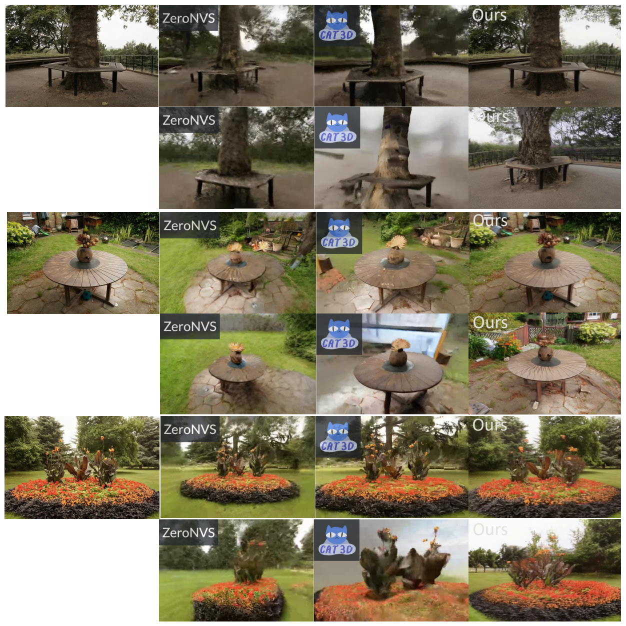


Figure A5. **Comparison of ZeroNVS and Cat3D on the Mip-Nerf dataset** in 3D scene generation from *single* input images. For each scene, the conditional image is shown in the left column along with renderings from two viewpoints, one at the conditional image (starting) view (upper) and another at around a 120°rotation from the starting view (lower).

show renderings from two viewpoints: one at the conditional image (starting) view and another rotated at around 120°from the starting view. We observe that for views close to the conditional image, our method achieves rendering quality similar to Cat3D and noticeably better than ZeroNVS. However, as the viewpoint deviates from the conditional image, Cat3D suffers from severe blurring, particularly in the background. By contrast, our method generates scenes with clearer textures, sharper details, and greater consistency with the conditional images.

E. Limitations and Future Work

While our method achieves superior generation performance and higher efficiency relative to prior work, there remain some limitations.

First, the development of the model is time and compute intensive. Although the inference process is efficient, the inference speed of the video generation model remains a bottleneck. Most of the compute time in our pipeline is consumed during the video generation phase. This drawback could be mitigated via parallel computation; *e.g.*, using xDiT (`https://github.com/xdit-project/xDiT`) for parallel inference, or utilizing a more efficient denoising strategy.

Second, our approach focuses on static scenes. We occasionally observe motions in the generated videos, which hampers the reconstruction effect. In future work, we will aim to extend our pipeline to dynamic scenes, exploring its potential for generating 4D content incorporating temporal dynamics.

By addressing the aforementioned limitations, our framework can be ameliorated for broader application with enhanced performance.