# LagerNVS: Latent Geometry for Fully Neural Real-time Novel View Synthesis

Stanislaw Szymanowicz [1,2] Minghao Chen [1,2] Jianyuan Wang [1,2]
Christian Rupprecht [1] Andrea Vedaldi [1,2]

[1]Visual Geometry Group, University of Oxford [2]Meta AI

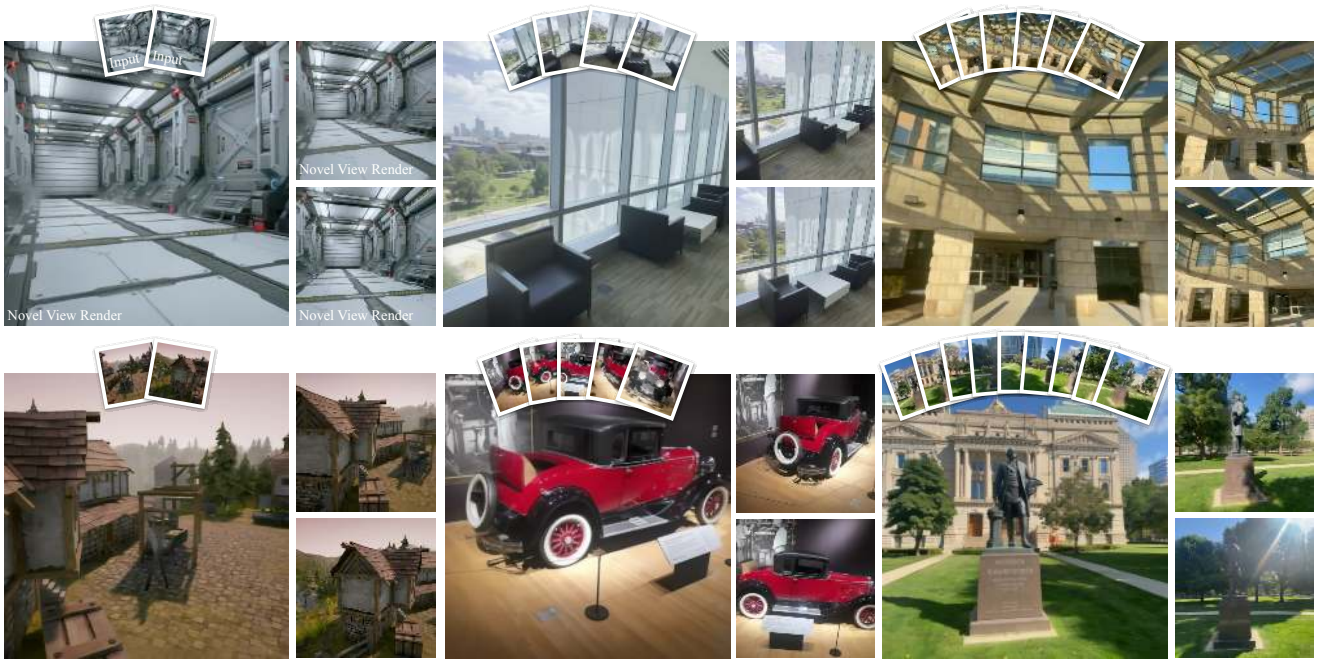


Figure 1. **LagerNVS** is a real-time, feed-forward, and generalizable Novel View Synthesis method. It achieves state-of-the-art quality by rendering views directly with a network, without explicit 3D reconstruction, and by incorporating strong 3D-aware features. These examples were rendered at more than 30 FPS at 512×512 on a single H100 GPU.

## Abstract

*Recent work has shown that neural networks can perform 3D tasks such as Novel View Synthesis (NVS) without explicit 3D reconstruction. Even so, we argue that strong 3D inductive biases are still helpful in the design of such networks. We show this point by introducing LagerNVS, an encoder-decoder neural network for NVS that builds on '3D-aware' latent features. The encoder is initialized from a 3D reconstruction network pre-trained using explicit 3D supervision. This is paired with a lightweight decoder, and trained end-to-end with photometric losses. LagerNVS achieves state-of-the-art deterministic feed-forward Novel View Synthesis (including 31.4 PSNR on Re10k), with and without known cameras, renders in real time, generalizes to in-the-wild data, and can be paired with a diffusion decoder for generative extrapolation. See szymanowiczs.github.io/lagernvs for code, models, and examples.*

## 1. Introduction

Novel View Synthesis (NVS) is the task of rendering new views of a scene from a set of other views. The most common approach to NVS is to fit a 3D model of the scene to the given views via optimization; then, the resulting *explicit 3D reconstruction* is rendered from the target viewpoints [36, 48]. This process can work well, but is slow and prone to overfitting unless the number of source views is large. A recent alternative is to learn a neural network that performs the 3D reconstruction in a feed-forward (optimization-free) manner. This is faster and can work well even with few or single source views because of the priors learned by the network [9, 11, 12, 30, 33, 62, 65, 66, 79, 82]. The logical next step is to bypass the 3D reconstruction altogether: methods like SRT [55], LVSM [34], and RayZer [32] have shown that the network can output the new views directly.

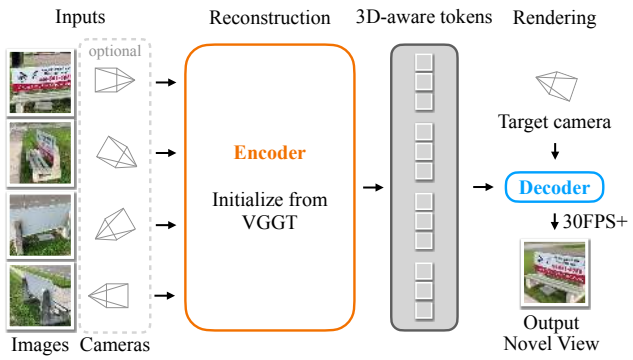


Figure 2. **Method.** The model takes any number of images and, optionally, their camera parameters as input. A large network initialized from a reconstruction model [70] outputs an intermediate feature representation with implicit 3D information. A lightweight network is queried on a target camera pose and renders a $512\times512$ image at more than 30 FPS on a single H100 GPU when provided with up to 9 input images.

However, foregoing 3D reconstruction does not mean that other 3D inductive biases are unimportant. We show this point by building a NVS network that, like SRT and LVSM, bypasses explicit 3D reconstruction and directly renders the new views. However, we incorporate 3D-aware features into it (Fig. 2) by initializing the model with the weights of the VGGT [70] backbone. This way, we extract features that, while not explicitly '3D', were pre-trained using explicit 3D supervision. We show that, compared to using strong but generic features like DinoV2 [49], using such 3D-aware features is highly beneficial for NVS.

In general, feed-forward NVS architectures are relatively unexplored. We thus compare several possible designs (Fig. 4). The simplest is a so-called *decoder-only* architecture that takes the source views and the target camera, and outputs the target views [34]. This works well, but re-evaluates the entire network for each new view generated. In contrast, *encoder-decoder* architectures [32, 34, 56] separate encoding from viewpoint-conditioned decoding. The encoder runs only *once per scene*, extracting a latent 3D representation of it, and only the decoder runs *for each new view*, which can be much more efficient. We further distinguish 'bottleneck' encoder-decoders, where the latent tokens are constrained to a fixed dimension before being decoded [34], and 'highway' encoder-decoders, where the decoder can access all image features directly. We show that the latter strikes an excellent quality and speed trade-off.

Based on these insights, we build *LagerNVS*, a **La**tent **Ge**ometry model for **R**eal-time NVS (Fig. 1). LagerNVS performs significantly better than LVSM, (+1.7dB PSNR margin on the standard RealEstate10k [91] benchmark), the previous state-of-the-art in reconstruction-free NVS. It also outperforms feed-forward 3D reconstruction networks, including those [33] that are based on VGGT.

We also show that training the network on a large mixture of datasets is important for NVS quality and generalization. Compared to prior models that are usually trained on single datasets, LagerNVS works well on ego-centric, 360°, and non-square images, as well as images collected in-the-wild, even when source camera poses are unknown, with a single set of model parameters (Fig. 3).

LagerNVS is also efficient: encoding requires mere seconds and decoding runs in real time (30FPS+) with up to nine source images at $512\times512$ resolution on a single H100 GPU. This is notable because the renderer/decoder is a standard neural network which does not use explicit 3D representations, custom kernels [37] or JIT compilation [2].

Like its peers, LagerNVS is trained using a regression loss, which tends to regress to the mean in the presence of ambiguity, such as when rendering parts of the scene that are not visible in the source views. This calls for *generative* models that can sample *plausible* views when information is missing. Motivated by this, we repurpose LagerNVS's decoder for denoising diffusion [25], with promising results.

To summarize, our contributions are as follows:

1. We show that reconstruction-free NVS still benefits from strong 3D biases implicitly captured by features pre-trained using explicit 3D supervision.
2. We explore three NVS architectures, decoder-only, bottleneck encoder-decoder, and highway encoder-decoder, and show that the latter has the best rendering quality for a given size of the renderer. Moreover, we propose a decoder design that enables real-time rendering on a single H100 with up to 9 source views at $512 \times 512$ resolution.
3. We set the new SoTA for deterministic NVS, outperforming the prior SoTA, LVSM, by a solid +1.7dB margin. We also outperform NVS models that *do* perform feed-forward 3D reconstruction, both with and without source camera poses.

We release several model checkpoints and code to reproduce and further extend our results (see the project website). A model performance card is given in Tab. A1.

## 2. Related work

**Encoder-decoder NVS with explicit 3D representations.** Many NVS methods are based on reconstructing the scene, extracting an *explicit* 3D representation of it. By explicit, we mean that the representation maps 3D locations to corresponding local properties like opacity and radiance. Neural Radiance Fields [48] and 3D Gaussians [36] fit (encode) explicit 3D scene representations to the source views via optimization, which is slow and overfits unless hundreds of views are given. Hence, authors have also proposed neural networks that can extract 3D representations in a feed-forward manner, quickly and from only a few views. Some output NeRFs [10, 42, 84] and others per-pixel 3D

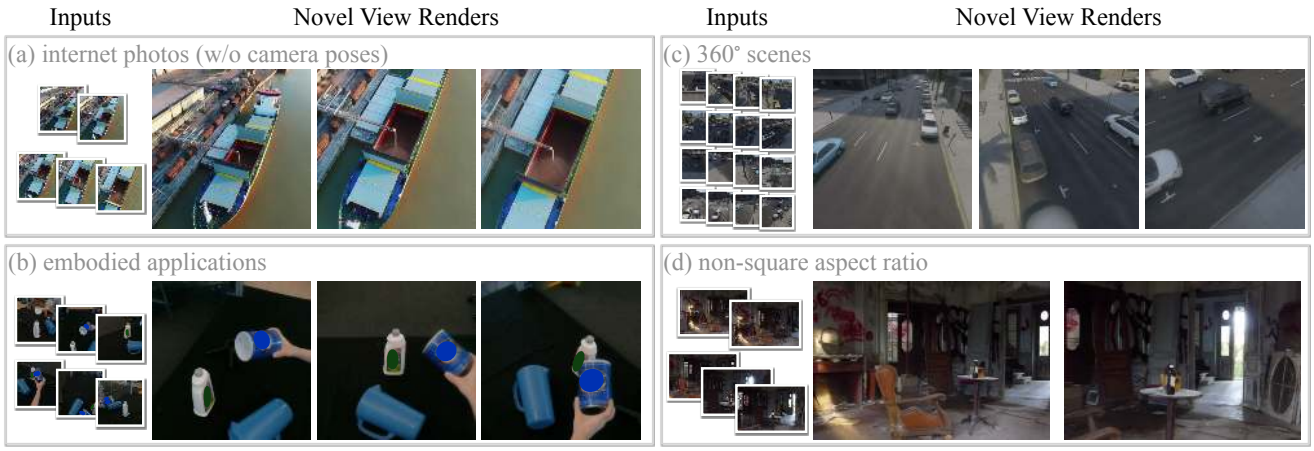


Figure 3. **Generalization.** Our model generalizes to a wide range of input types and applications, including (a) internet photos with unknown camera poses, (b) embodied applications (logos removed as postprocessing), (c) 360° scenes and (d) non-square aspect ratios.

Gaussians [9, 11, 65, 66, 80, 83, 88, 92]. Many methods assume known source cameras, but others relax this constraint [26, 33, 62, 82, 83, 90], often by using pre-trained multi-view reconstruction models [18, 70, 71, 73]. We too leverage such models, but use their latent feature representation instead of their explicit 3D outputs.

**Encoder-decoder NVS with latent 3D representations.** Other approaches extract a *latent* 3D representation of the scene that can be decoded directly into new views, but not necessarily into explicit 3D properties. These representations can be viewed as an encoding of the scene's *light field* [1, 22], a concept associated with NVS [7]. Early neural approaches like LFN [61] used auto-decoding to fit compact light field representations. SRT [56] later proposed an encoder-decoder network to extract such representations in a feed-forward manner from source views. LVSM [34] further improved this approach with increased decoder capacity, while RayZer [32] enabled training without camera labels for ordered image collections. Like LVSM, we too use a transformer-based architecture, but we (1) propose a different encoder information flow, and (2) leverage a pre-trained 3D reconstruction network for our encoder. Both (1) and (2) substantially improve performance. Unlike RayZer, we can operate on unordered image collections. Concurrently to us, SVSM [38] analyzed the scaling laws of encoder-decoder NVS transformers to maximize training efficiency. We use a similar architecture, but instead of optimizing compute usage, we focus on the role of 3D pre-training and on how it enables inference both with and without known cameras, with strong generalization.

**Decoder-only NVS.** Decoder-only methods directly map source images and a target camera to the target image without extracting a camera-independent intermediate representation, thus requiring the entire model to run for every new view rendered and limiting rendering speed. LVSM [34] also considers a variant that follows this paradigm.

**Generative NVS.** NVS is ambiguous when the target camera points at a part of the scene that is not represented in the source images. In such a case, a *generative* model that can hallucinate *plausible* completions is required. Generative diffusion decoders were used both in decoder-only NVS methods [4, 5, 21, 28, 31, 41, 46, 57, 67, 75, 77], and in encoder-decoder methods [8, 12, 20, 23, 45, 52, 76, 85]. While we focus on deterministic NVS, we include a preliminary experiment to demonstrate how our model can support diffusion-based generation too.

## 3. Method

In *Novel View Synthesis* (NVS), we are given $V$ source images $I_1, \dots, I_V \in \mathbb{R}^{3\times H\times W}$ and the parameters $\boldsymbol{g}$ of a target camera, expressed with respect to the viewpoint of image $I_1$ taken as *reference*. The goal is to output a new image $I \in \mathbb{R}^{3\times H\times W}$ captured by the target camera $\boldsymbol{g}$, which we write as a function $I = f(\boldsymbol{g}; I_1, \dots, I_V)$. If the camera parameters $\boldsymbol{g}_1, \dots, \boldsymbol{g}_V$ of the source images are also known, then the function becomes $I = f(\boldsymbol{g}; I_1, \boldsymbol{g}_1, \dots, I_V, \boldsymbol{g}_V)$.

In practice, we further assume that the focal length is the same across all source and target cameras, and that the vertical and horizontal focal lengths are equal. When the input cameras are unknown, we pass to the model a target camera with the horizontal field of view set to the canonical value $k_0$, but still train it to render an image at the (unknown) source focal length.[1]

Following [70], we parameterize the cameras with 11-dimensional vectors as $\boldsymbol{g} = (\boldsymbol{q}, \boldsymbol{t}, \boldsymbol{k}, \boldsymbol{w})$, where $\boldsymbol{q} \in \mathbb{S}^3$,

[1] In the first version of the paper, we did not make this distinction and effectively "leaked" the source focal length to the model that is not supposed to receive source cameras as input. Using this canonical value for the target focal length fixes this corner case with marginal impact on the results. See the appendix and limitations for details.

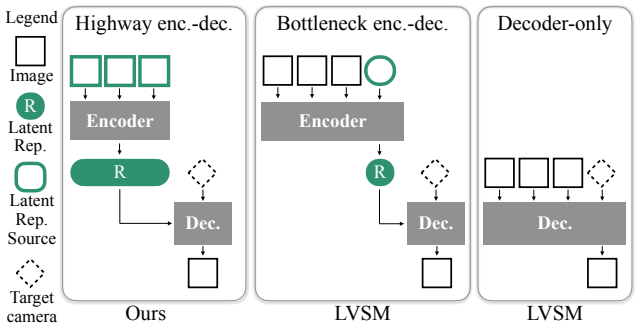


Figure 4. **Architectures.** Our highway encoder-decoder (left) maximizes information flow from source images to the latent representation (R), making it more expressive than the bottleneck encoder-decoder (middle). Unlike decoder-only models (right), our design allows scaling the encoder without slowing decoding.

is the camera rotation (expressed as a unit quaternion), $\boldsymbol{t} \in \mathbb{R}^3$, is the camera translation, $\boldsymbol{k} \in \mathbb{R}^2_+$ are the horizontal and vertical fields-of-view (the optical center is assumed to be at the image center). All $\boldsymbol{g}_i$ are relative to the first camera $\boldsymbol{g}_1$. We also introduce an auxiliary input parameter $\boldsymbol{w} \in \mathbb{R}^2_+$ to represent the scene scale, discussed in the supplement.

When $f$ is implemented as a neural network, we call the model *decoder only* if the whole network is evaluated for each new $\boldsymbol{g}$. We call it an *encoder-decoder* if it first encodes the source images into an intermediate representation $\boldsymbol{z}$ independently of the target camera, so that the NVS function $f$ is the composition of an encoder $e$ and a decoder $h$:

$$\boldsymbol{z} = e(I_1, \boldsymbol{g}_1, \ldots, I_V, \boldsymbol{g}_V), \quad I = h(\boldsymbol{z}; \boldsymbol{g}). \tag{1}$$

This allows us to amortize the cost of computing $\boldsymbol{z}$ across the generation of multiple target views.

We further distinguish "highway" and "bottleneck" encoder-decoders (Fig. 4). In *highway* encoder-decoders (named so to reflect non-attenuated information flow [64]), the representation $\boldsymbol{z} = (\boldsymbol{z}_1, \ldots, \boldsymbol{z}_V)$ contains separate feature vectors for each source image, whereas in *bottleneck* ones the number of tokens in $\boldsymbol{z}$ is independent of the number $V$ of source views, thus constraining information flow.

While we are particularly interested in models where both the encoder $e$ and the decoder $h$ are neural networks, encoder-decoders can also be based on explicit 3D reconstruction. In this case, $\boldsymbol{z}$ is a 3D representation, such as 3D Gaussians, and $h$ is the corresponding renderer.

### 3.1. An encoder with an implicit 3D bias

One obvious way to incorporate 3D inductive biases in the model $f$ is to opt for a reconstruction approach where $\boldsymbol{z}$ is an explicit 3D representation of the scene. A recent example is AnySplat [33], which builds on VGGT [70] for its encoder and uses a Gaussian splat renderer for the decoder. Alternatively, both the encoder and decoder can be neural networks, and the representation $\boldsymbol{z}$ can be a set of features [32, 34, 56], avoiding explicit 3D reconstruction and representations altogether. In this case, there are few 3D inductive biases, mostly limited to choosing an encoder for the camera parameters (e.g., ray maps).

Here we explore a third approach (Fig. 2), where encoder and decoder are neural networks, but the encoder is initialized from a network pre-trained for 3D reconstruction via explicit 3D supervision. In this way, there is no explicit 3D reconstruction, but the features $\boldsymbol{z}$ output from the encoder are "3D-aware".

We call our model LagerNVS, and we implement its encoder by building on top of the VGGT model [70]. Recall that VGGT is a feed-forward 3D reconstruction network that maps one or more images of a scene to a number of geometric quantities, including the camera $\boldsymbol{g}_i$ and the depth map $D_i$ for each source image $I_i$. However, we do not make use of these outputs directly. Instead, for each source image $I_i$, we extract an array of tokens $\boldsymbol{z}_i \in \mathbb{R}^{P \times C}$ from the last layers of VGGT's transformer backbone (before the decoding heads)—we extract the tokens output by the last local attention layer and the last global attention layer, and remove the so-called camera tokens. We concatenate these two sets of tokens channel-wise, pass them through a linear layer to project them to the desired dimension $C$ that our decoder expects, and normalize with LayerNorm [3] to improve learning stability.

**Augmenting the encoder with camera inputs.** In applications where the source cameras $\boldsymbol{g}_1, \ldots, \boldsymbol{g}_V$ are available, we wish to pass them to the encoder too. Since VGGT does not take cameras as input, we modify it by adding a 2-layer Multi-Layer Perceptron (MLP) that projects the camera parameters $\boldsymbol{g}$ to a 1024-dimensional token. When a camera is not provided as input, we set $\boldsymbol{g}$ to the null vector (keeping only the scene scale parameter $\boldsymbol{w}$). We add the projected tokens to the VGGT default initial values for the camera tokens, and then feed these to the VGGT feature backbone as usual. With these modifications, the encoder $e$ is now a function of images and optional cameras.

### 3.2. An efficient decoder

The decoder $h(\boldsymbol{g}; \boldsymbol{z}_1, \ldots, \boldsymbol{z}_V)$ takes as input the tokens $\boldsymbol{z}$ output by the encoder (Sec. 3.1) and the target camera $\boldsymbol{g}$.

**Encoding the target camera.** The goal of the decoder is to *render* the image from the viewpoint of the target camera $\boldsymbol{g}$, and we represent it densely in the form of a *Plucker ray map* [87]. Hence, $\boldsymbol{g}$ is input to the decoder as a $6 \times H \times W$ image where each pixel contains its corresponding camera ray in Plucker coordinates, namely $(r_d, r_m)$ where $r_d \in \mathbb{R}^3$ is the ray direction and $r_m \in \mathbb{R}^3$ is the ray moment. When the source camera intrinsics are unknown, we form target camera rays with a nominal horizontal field of view $\boldsymbol{k}_x = \pi/2$ to prevent information leakage (see above). A

convolutional layer with kernel size and stride $r' = 8$ is then applied to this image to extract $HW/r'^2$ tokens. Four additional register tokens [15] are concatenated to these. We use the symbol $\boldsymbol{s}$ to denote the resulting set of $4 + HW/r'^2$ tokens encoding the target camera $\boldsymbol{g}$.

**Decoder architecture.** We design the decoder as a transformer network, where the dense camera tokens $\boldsymbol{s}$ attend the encoded source images $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_V$ to form the target image. We experiment with two variants (see Alg. 1 in the supplement) trading off quality and speed. The first variant has complexity $\mathcal{O}(V^2)$ and uses *full attention* on the concatenation of target camera and scene tokens $(\boldsymbol{s}, \boldsymbol{z}_1, \ldots, \boldsymbol{z}_V)$, using them as queries, keys and values in the transformer:

$$q = k = v = (\boldsymbol{s}, \boldsymbol{z}_1, \ldots, \boldsymbol{z}_V). \tag{2}$$

The second variant has complexity $\mathcal{O}(V)$ and uses full attention on the target camera tokens $\boldsymbol{s}$ only, setting $q = k = v = \boldsymbol{s}$, followed by *cross attention* between these and the scene tokens, with two layers that use respectively

$$\begin{aligned} q_1 &= \boldsymbol{s}; \quad k_1 = v_1 = (\boldsymbol{z}_1, \ldots, \boldsymbol{z}_V), \text{ and} \\ q_2 &= (\boldsymbol{z}_1, \ldots, \boldsymbol{z}_V); \quad k_2 = v_2 = \boldsymbol{s} \end{aligned} \tag{3}$$

We otherwise use a standard transformer architecture (see Sec. 3.4 and the supplement). At the output, the register tokens are discarded, and the dense target camera tokens are projected with a linear layer to $8 \times 8$ patches and reshaped to the original image size to obtain the target image $I$.

### 3.3. Training

We train our model by minimizing the NVS loss, i.e., the distance between ground truth novel views and the ones estimated by the model. In particular, we use a combination of mean-squared (L2) error and perceptual [35] losses, i.e., $\mathcal{L} = \lambda_2 \mathcal{L}_2 + \lambda_p \mathcal{L}_p$.

Given that we start from a pre-trained VGGT model for our encoder, we have a choice on whether to fine-tune the entire model end-to-end, or restrict learning to the new parameters, most of which reside in the decoder. Empirically, we found that fine-tuning the entire model is essential to obtain good results. This is perhaps not surprising as the VGGT features were not trained with the goal of retaining the *appearance* of the source images or with the capability to understand camera pose conditioning.

**Data.** To train our model, we thus require a collection of tuples $(I, \boldsymbol{g}, I_1, \boldsymbol{g}_1, \ldots, I_V, \boldsymbol{g}_V)$, consisting of posed images of approximately static 3D scenes. Inspired by VGGT, we train on a rich mix of 13 multi-view datasets (see supplement), including typical NVS datasets such as RealEstate10k [91], DL3DV [44], WildRGBD [78]. Our full dataset roughly matches the size and diversity of the data used for VGGT.

### 3.4. Implementation details

**Architecture.** We use the pre-trained VGGT model for our encoder and start from its pre-trained weights. We use a ViT-B [17] transformer for our decoder with FlashAttention [13, 14, 60] attention kernels.

**Augmentations.** We train our model to be robust to the inputs to the model: we randomly sample the number of source views, varying between 1 and 10, selectively drop out camera tokens and vary the aspect ratio.

**Optimization.** We use AdamW optimizer and cosine learning rate decay with linear warmup. We use QK-norm [24], and gradient clipping for improved training stability, as well as gradient checkpointing for reduced memory usage. Our main model is trained at 512 resolution (longer side) on the full dataset mix for 250k iterations with batch size 512, but we adjust hyperparameters (e.g., batch size, learning rate and training iterations) to the dataset and baselines used in each experiment.

## 4. Experiments

We begin by comparing LagerNVS to LVSM, the prior SoTA for NVS (Sec. 4.1), and by demonstrating the benefits of training the model on a large number of different datasets (Sec. 4.2). Then, we study the importance of 3D-aware pre-training and the choice of encoder-decoder architecture (Sec. 4.3). We also show that our method, which uses an implicit 3D representation, outperforms methods that reconstruct the 3D scene explicitly (Sec. 4.4). Finally, we demonstrate how our model can be used in combination with diffusion models (Sec. 4.5).

We assess LagerNVS in terms of its ability to deliver high-quality NVS for a given complexity of the decoder, as the latter ultimately determines the rendering speed, which we aim to maintain in the real-time range. In all comparisons we thus treat the number of decoder transformer blocks as a control variable. We describe the most important experimental parameters and results here, and refer the reader to the supplement for details.

**Evaluation datasets.** We test our method on 3 datasets commonly used for Novel View Synthesis: RealEstate10k [91], DL3DV [44], and CO3D [51]. We adapt the training and testing setup to match that of our baselines, and include more details in each sub-section.

**Metrics.** On all quantitative tasks we use standard metrics: Peak Signal-to-Noise Ratio (PSNR), Structural Image Similarity (SSIM [74]), and Perceptual Similarity (LPIPS [89]).

### 4.1. Comparison to the state-of-the-art in NVS

We first evaluate the ability of LagerNVS to deliver SoTA NVS (Fig. 5 and Tab. 1). The SoTA is LVSM, which, like

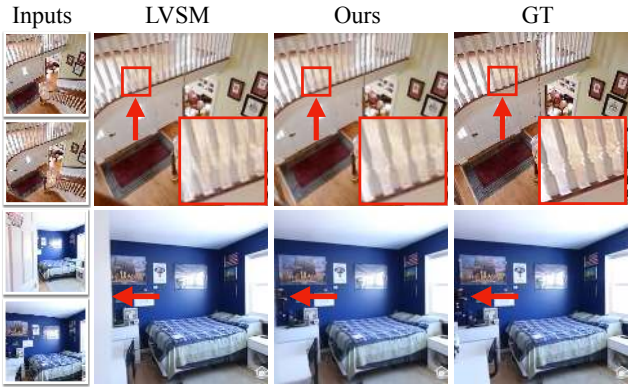


Figure 5. **Comparison to LVSM.** Our model estimates geometry better than LVSM in regions where 2D matching is challenging (top). It also uses monocular cues better for depth (bottom).

| | Method | Batch | PSNR ↑ | SSIM ↑ | LPIPS ↓ |
|---|---|---|---|---|---|
| (a) | LVSM [34] Enc-dec. bottleneck | 64 | 28.32 | 0.888 | 0.117 |
| (b) | LVSM [34] Decoder-only | 64 | 28.89 | 0.894 | 0.108 |
| (c) | Ours Enc-dec. highway— x-attn. | 64 | 30.11 | 0.912 | 0.089 |
| (d) | Ours Enc-dec. highway— full | 64 | **30.48** | **0.918** | **0.086** |
| (e) | LVSM [34] Enc-dec. bottleneck | 512 | 28.58 | 0.893 | 0.114 |
| (f) | LVSM [34] Decoder-only | 512 | 29.67 | 0.906 | 0.098 |
| (g) | Ours Enc-dec. highway— x-attn. | 512 | 31.06 | 0.924 | 0.080 |
| (h) | Ours Enc-dec. highway— full | 512 | **31.39** | **0.928** | **0.078** |

Table 1. **Comparison to LVSM.** Our model outperforms LVSM (SoTA) both at small (batch size 64) and large (batch size 512) training scales.

us, uses a latent 3D representation. We follow LVSM's evaluation protocol and train and test on RealEstate10k [91] with $V = 2$ source views. The data contains 65k video clips of indoor scenes. We use the same $256 \times 256$ resolution, training steps (100,000), and batch size as LVSM, as well as the source and target views in pixelSplat [9].

**Results.** We test the LVSM bottleneck encoder-decoder (Tab. 1 (a), (e)) and decoder-only (Tab. 1 (b), (f)) variants. The LVSM authors trained their model using batch size 512 (Tab. 1 (e-h)) and 64 (Tab. 1 (a-d)) to simulate resource-constrained training. We report scores for both settings, and with the full and cross-attention variants of our model.

The LVSM authors noted that their decoder-only models are better than their bottleneck encoder-decoder ones, which they attribute to the model architecture. However, encoder-decoder architectures can amortize the encoding cost, thus benefiting from more lightweight decoders and consequently faster rendering. It is thus notable that our *highway* encoder-decoder is significantly better than both LVSM variants in both settings (up to +1.7dB PSNR; Tab. 1 (c, d), (f, g)). Figure 5 indicates that our model benefits from improved multi-view matching and monocular depth estimation. As we further analyze in Sec. 4.3, this is due to a combination of factors, including using pre-trained 3D-aware features and not having an encoding bottleneck.

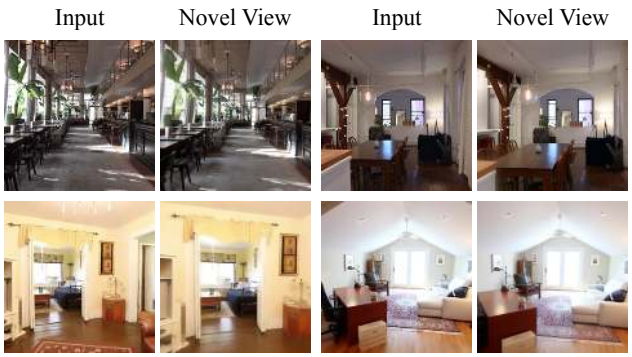


Figure 6. **Single-view NVS** is supported for small camera motion with LagerNVS.

### 4.2. Generalizable NVS

While training on a single dataset like Re10k is common practice in many NVS works [9, 11, 34, 88], multi-view geometry models [40, 70, 71] have shown the benefits of training on large collections of data, particularly for zero-shot generalization. Here, we look into this setup as well.

Our full model, trained as described in Sec. 3.4 on all the datasets of Sec. 3.3, achieves excellent generalization to in-the-wild data and different scene types, and can work without knowledge of the source camera poses, as shown in Fig. 3. It also supports single-view NVS, as shown in Fig. 6. Table A1 in the supplement provides a detailed quantitative evaluation of this model on a number of standard benchmarks.

### 4.3. Analysis and ablations

We assess the importance of 3D pre-training of the encoder features and architectural design choices via ablations.

**Experimental protocol.** As in Sec. 4.2, we train models on all datasets of Sec. 3.3. We report results in the 2-view setting with known cameras on the DL3DV split set by DepthSplat with $256 \times 256$ center-crops, taking care to remove source views from the target view set. We use the VGGT architecture as the encoder backbone. We modify the training hyperparameters compared to the main model, including batch size and training iterations, to reduce computation (see supp.). In the bottleneck variant, the global and local aggregator blocks process additional 3072 bottleneck tokens (following [34]).

**Importance of using pre-trained 3D-aware features.** Table 2 shows that initializing the model from VGGT (row (a)) is much better (+2.9dB PSNR) than starting from scratch (row (e)). Using generic 2D-pretraining (DinoV2 [49] in row (f)) is also marginally better, but not nearly as much as using 3D pre-training. Figure 7 shows that 3D pre-training improves especially the foreground objects, which benefit from better depth estimation.

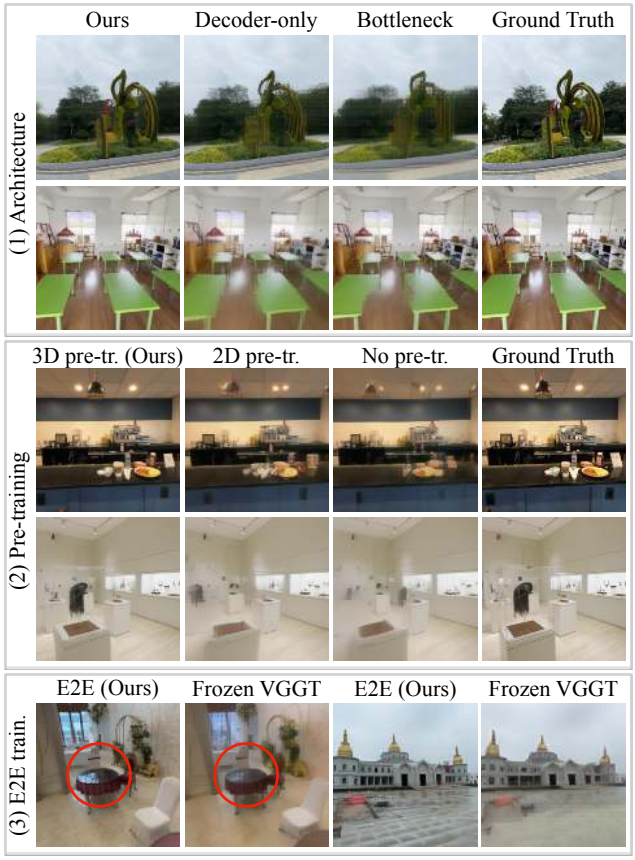


Figure 7. **Ablations.** Our architecture (1), pre-training (2) and end-to-end training (3) result in the best NVS quality.

Another important finding is that it is necessary to fine-tune the entire model end-to-end (E2E) to obtain good performance, unfreezing the VGGT backbone (rows (a) vs (g)). This is because VGGT is pre-trained to reconstruct only the geometry and thus to treat appearance attributes like colors, reflectance and transparency, which are important for NVS, as nuances. The results in Fig. 7 corroborate this hypothesis by showing that using a frozen backbone results in a lack of reflections and poor textures. Freezing VGGT also makes it harder for the model to learn to understand the source cameras (Sec. 3.1) when these are given.

Future versions of VGGT and similar reconstruction models should consider including a rendering head and loss during training to preserve appearance information and thus be more useful for tasks like NVS.

| | Encoder | E2E | X-attn | Pre-Tr. | PSNR ↑ | SSIM ↑ | LPIPS ↓ |
|---|---|---|---|---|---|---|---|
| (a) | Highway | ✓ | ✓ | 3D | 21.02 | 0.652 | 0.257 |
| (b) | Highway | ✓ | ✗ | 3D | **21.30** | **0.667** | **0.248** |
| (c) | Decoder only | ✓ | ✗ | — | 18.39 | 0.533 | 0.407 |
| (d) | Bottleneck | ✓ | ✗ | — | 17.53 | 0.480 | 0.461 |
| (e) | Highway | ✓ | ✓ | — | 18.03 | 0.551 | 0.405 |
| (f) | Highway | ✓ | ✓ | 2D | 18.17 | 0.515 | 0.388 |
| (g) | Highway | ✗ | ✓ | 3D | 19.01 | 0.553 | 0.334 |

Table 2. **Ablations.** Even when handicapped by using faster cross-attention, our highway encoder-decoder model outperforms the decoder-only and bottleneck encoder-decoder variants, which are analogous to the models introduced by LVSM [34]. Using 3D-aware pre-training (Pre-tr.) is crucial, but only effective if the features are fine-tuned end-to-end ("E2E").

| | Method | Test data | PSNR ↑ | SSIM ↑ | LPIPS ↓ |
|---|---|---|---|---|---|
| w cameras single dataset | DepthSplat [80] | DL3DV 4-view | 22.30 | 0.765 | 0.189 |
| | Ours | | **27.56** | **0.869** | **0.095** |
| | DepthSplat [80] | DL3DV 6-view | 23.47 | 0.812 | 0.154 |
| | Ours | | **29.45** | **0.904** | **0.068** |
| w/o cameras generalizable | AnySplat [33] | Re10k 2-view | 17.05 | 0.626 | 0.349 |
| | Flare [90] | | 23.77 | 0.801 | 0.191 |
| | NopoSplat [82] | | 24.06 | 0.820 | 0.178 |
| | Ours (v2) | | **25.54** | **0.828** | **0.158** |
| | AnySplat [33] | CO3D 9-view | 15.87 | 0.526 | 0.423 |
| | Ours (v2) | | **22.05** | **0.689** | **0.365** |

Table 3. **Feed-forward 3DGS comparison.** Our model outperforms state-of-the-art feed-forward 3DGS models across the board, both with and without input camera poses available, illustrating the strength of *latent* 3D representations. NoPoSplat only trained on RealEstate10k (a potentially unfair comparison), but we include it here for completeness. See Sec. C.3 for details of 'Ours (v2)'.

**Decoder-only vs bottleneck/highway encoder-decoder.** As in Sec. 4.1, the decoder-only model (Tab. 2 row (c)) outperforms the bottleneck encoder-decoder (row (d)), but our highway encoder-decoder is better than both (row (a)). The latter has no bottleneck, can leverage the pre-trained 3D-aware features (VGGT), and, by offloading much of the computation that is independent of the target view to the encoder, has more capacity to learn a high-quality decoder (for a fixed decoding budget, which is our control variable).

The bottleneck model's unique advantage is that the decoding speed is independent of the number of source views. Even so, the cross-attention variant of the highway architecture still allows real-time rendering (56–30 FPS) with 1–9 source views.

**Full vs cross-attention.** Using full attention (Tab. 2 row (a)) performs slightly better than cross-attention (row (b)), but is slower: the maximum number of source images for real-time (30FPS+) rendering is 9 for the cross-attention variant, and 6 for the full attention variant (see Tab. A3 in the supplement), due to the fact that the complexity increases from $O(V)$ to $O(V^2)$ in the number $V$ of source views.

### 4.4. NVS with Explicit 3D Representations

Most NVS methods reconstruct the scene, extracting an explicit 3D representation of it, which also provides a strong 3D inductive bias. In this section, we show that our *latent* 3D representation performs better than these solutions.

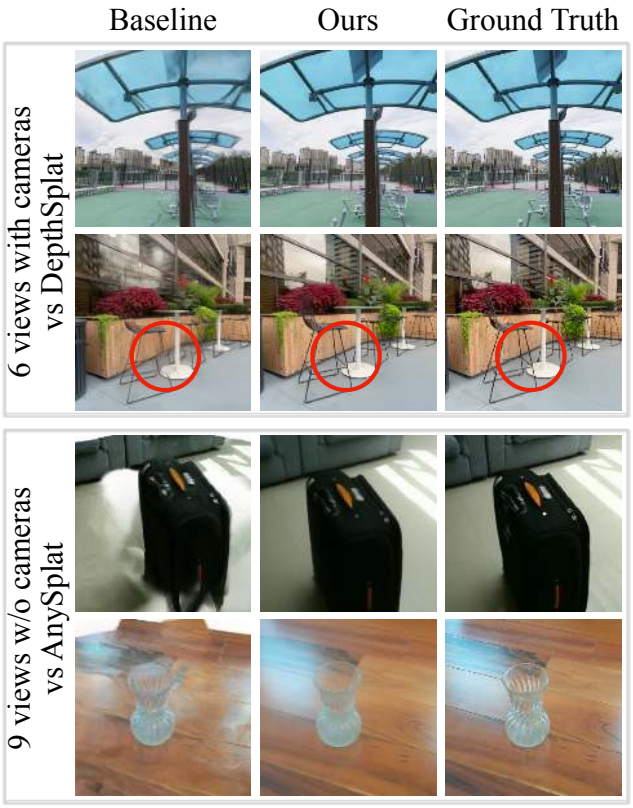


Figure 8. **Comparison to feed-forward 3DGS.** Our model better handles thin parts, reflections, and occlusions, and supports inference with and without source camera poses.

**Baselines and evaluation protocols.** First, we consider the case where the source views come with known camera poses. We evaluate our method against DepthSplat, which uses explicit 3D priors in the form of pre-trained monocular depth features, cost volume matching and an explicit 3D representation. We use their train and test data (DL3DV) and their 4-view and 6-view splits (while taking care to remove source images from evaluation targets) with $256 \times 256$ center-crops. Second, we consider the case where the source cameras are unknown and compare against Flare [90] and AnySplat [33], which were also trained on a large, diverse dataset. Following standard practice, we evaluate on 2-view RealEstate10k with the split from [82] at $256 \times 256$. Like Flare, we use the subset of that split with low- and medium-overlap scenes. AnySplat uses pre-trained VGGT features, like us. We further test the latter on the more challenging CO3D [51] 360° captures, using the 9-view split from [76] at $512 \times 512$ resolution.

**Results.** As shown in Tab. 3, we outperform all methods across the board, often by a significant margin. Figure 8 shows that our implicit representation better captures reflective surfaces and thin structures. Note that 3DGS should have no problem representing such elements, but it is evidently difficult to learn a feed-forward model that predicts them accurately.


Figure 9. **Diffusion.** Our decoder can be fine-tuned with a diffusion objective, enabling hallucination of unobserved regions.

An even larger difference can be observed in comparison on CO3D in Tab. 3. Figure 8 shows this can be attributed to the fact that pixel-aligned Gaussian methods completely fail in areas that are not visible in any source view—even with 9 views there are significant occlusions, which are clearly visible (third row of Fig. 8). Occlusion coverage is possible with feed-forward 3DGS [65, 66], but AnySplat and Flare force Gaussian-ray alignment—this makes the observed content sharper (notice their competitive LPIPS) but prevents infilling occluded areas. In contrast, even though our decoder is limited by its deterministic nature, it can, to some extent, 'imagine' simple regions, such as the floor (rows 3 and 4 in Fig. 8, more examples in supp.).

### 4.5. Generative NVS

While our model, trained as a deterministic regressor, is capable of simple infilling, high-quality hallucination of unobserved regions requires a generative model. LagerNVS can be adapted to become one, by fine-tuning it to operate as a diffusion model. In order to do so, we keep the encoder unmodified and frozen, and fine-tune the decoder with a denoising diffusion objective. The decoder is minimally modified and fine-tuned: we allow the input layer to accept additional channels corresponding to the noisy image, and add adaLN-zero [50] conditioning on the denoising timestep.

In Fig. 9 we demonstrate two scenarios where this formulation is better than the deterministic one: (1) occlusion and (2) extrapolation. In these cases, deterministic models like ours and LVSM regress to the mean and produce blurry outputs, whereas the diffusion variant can hallucinate plausible completions. Remarkably, our pre-trained model learns diffusion very quickly, after fine-tuning for just 60k iterations, operating in pixel space (without latent diffusion [54]) and using only 12 transformer blocks.

## 5. Conclusion

We showed the benefits of leveraging strong 3D inductive biases even in NVS models that do not use explicit 3D

representations. We have done so by building our model, LagerNVS, from a strong pre-trained 3D reconstruction network. The resulting model achieves state-of-the-art NVS results when compared to implicit feed-forward NVS models, as well as models that do extract a 3DGS representation of the scene. Furthermore, our lightweight decoder allows us to render new views at $512 \times 512$ in real time on a single GPU. Our model is robust, efficient, generalizes well, and requires no camera poses, which makes it practical. While we have focused on deterministic NVS, experiments show that the model can be applied to generative NVS as well, with further benefits. We share the code and model checkpoints for the benefit of the open-source community.

**Acknowledgments** We thank Thu Nguyen-Phuoc, Roman Shapovalov, Antoine Toisoul and Zihang Lai for fruitful discussions, David Charatan for assistance with data and Shanghzhan Zhang for confirming evaluation details. This work was done while S. Szymanowicz and M. Chen were interns at Meta. S. Szymanowicz is supported by the EPSRC Doctoral Training Partnerships Scholarship (DTP) EP/R513295/1 and the Oxford-Ashton Scholarship.

## References

[1] E. Adelson and R. Bergen. *The Plenoptic Function and the Elements of Early Vision*. MIT Press, 1991. 3

[2] Jason Ansel, Edward Yang, Horace He, et al. PyTorch 2: Faster machine learning through dynamic Python bytecode transformation and graph compilation. In *Proc. ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2024. 2

[3] Lei Jimmy Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. *arXiv.cs*, abs/1607.06450, 2016. 4, 15

[4] Sherwin Bahmani, Tianchang Shen, Jiawei Ren, Jiahui Huang, Yifeng Jiang, Haithem Turki, Andrea Tagliasacchi, David B. Lindell, Zan Gojcic, Sanja Fidler, Huan Ling, Jun Gao, and Xuanchi Ren. Lyra: Generative 3D scene reconstruction via video diffusion model self-distillation. *arXiv*, 2509.19296, 2025. 3

[5] Jianhong Bai, Menghan Xia, Xiao Fu, Xintao Wang, Lianrui Mu, Jinwen Cao, Zuozhu Liu, Haoji Hu, Xiang Bai, Pengfei Wan, and Di Zhang. ReCamMaster: camera-controlled generative rendering from a single video. In *Proc. ICCV*, 2025. 3, 16

[6] Gilad Baruch, Zhuoyuan Chen, Afshin Dehghan, Tal Dimry, Yuri Feigin, Peter Fu, Thomas Gebauer, Brandon Joffe, Daniel Kurz, Arik Schwartz, and Elad Shulman. ARKitscenes - a diverse real-world dataset for 3d indoor scene understanding using mobile RGB-d data. In *Proc. NeurIPS*, 2021. 17

[7] Chris Buehler, Michael Bosse, Leonard McMillan, Steven J. Gortler, and Michael F. Cohen. Unstructured lumigraph rendering. In *Proc. SIGGRAPH*, 2001. 3

[8] Eric R. Chan, Koki Nagano, Matthew A. Chan, Alexander W. Bergman, Jeong Joon Park, Axel Levy, Miika Aittala, Shalini De Mello, Tero Karras, and Gordon Wetzstein. Generative novel view synthesis with 3D-aware diffusion models. In *Proc. ICCV*, 2023. 3, 15

[9] David Charatan, Sizhe Li, Andrea Tagliasacchi, and Vincent Sitzmann. pixelSplat: 3D Gaussian splats from image pairs for scalable generalizable 3D reconstruction. In *Proc. CVPR*, 2024. 1, 3, 6, 13

[10] Anpei Chen, Zexiang Xu, Fuqiang Zhao, Xiaoshuai Zhang, Fanbo Xiang, Jingyi Yu, and Hao Su. MVSNeRF: Fast generalizable radiance field reconstruction from multi-view stereo. In *Proc. ICCV*, 2021. 2

[11] Yuedong Chen, Haofei Xu, Chuanxia Zheng, Bohan Zhuang, Marc Pollefeys, Andreas Geiger, Tat-Jen Cham, and Jianfei Cai. MVSplat: efficient 3D gaussian splatting from sparse multi-view images. In *Proc. ECCV*, 2024. 1, 3, 6

[12] Yuedong Chen, Chuanxia Zheng, Haofei Xu, Bohan Zhuang, Andrea Vedaldi, Tat-Jen Cham, and Jianfei Cai. MVSplat360: Feed-forward 360 Scene Synthesis from Sparse Views. In *Proc. NeurIPS*, 2024. 1, 3

[13] Tri Dao. FlashAttention-2: Faster attention with better parallelism and work partitioning. In *Proc. ICLR*, 2024. 5

[14] Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. In *Proc. NeurIPS*, 2022. 5

[15] Timothée Darcet, Maxime Oquab, Julien Mairal, and Piotr Bojanowski. Vision transformers need registers. In *Proc. ICLR*, 2024. 5

[16] Jimmy Ba Diederik P. Kingma. Adam: A method for stochastic optimization. In *Proc. ICLR*, 2015. 18

[17] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth $16\times16$ words: Transformers for image recognition at scale. In *Proc. ICLR*, 2021. 5, 15, 16

[18] Bardienus Duisterhof, Lojze Zust, Philippe Weinzaepfel, Vincent Leroy, Yohann Cabon, and Jerome Revaud. MASt3R-SfM: a fully-integrated solution for unconstrained structure-from-motion. In *Proc. 3DV*, 2025. 3

[19] Stefan Elfwing, Eiji Uchibe, and Kenji Doya. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural Networks*, 107, 2018. 15

[20] Tobias Fischer, Samuel Rota Bulò, Yung-Hsu Yang, Nikhil Varma Keetha, Lorenzo Porzi, Norman Müller, Katja Schwarz, Jonathon Luiten, Marc Pollefeys, and Peter Kontschieder. FlowR: flowing from sparse to dense 3d reconstructions. In *Proc. ICCV*, 2025. 3

[21] Ruiqi Gao, Aleksander Holynski, Philipp Henzler, Arthur Brussee, Ricardo Martin-Brualla, Pratul Srinivasan, Jonathan T. Barron, and Ben Poole. CAT3D: create anything in 3d with multi-view diffusion models. In *Proc. NeurIPS*, 2024. 3

[22] Steven J. Gortler, Radek Grzeszczuk, Richard Szeliski, and Michael F. Cohen. The lumigraph. In *Proc. SIGGRAPH*, 1996. 3

[23] Jiatao Gu, Alex Trevithick, Kai-En Lin, Josh M. Susskind, Christian Theobalt, Lingjie Liu, and Ravi Ramamoorthi. NerfDiff: Single-image view synthesis with nerf-guided distillation from 3D-aware diffusion. *arXiv.cs*, abs/2302.10109, 2023. 3

[24] Alex Henry, Prudhvi Raj Dachapally, Shubham Pawar, and Yuxuan Chen. Query-key normalization for transformers. In *Findings of EMNLP*, 2020. 5

[25] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In *Proc. NeurIPS*, 2020. 2

[26] Sunghwan Hong, Jaewoo Jung, Heeseong Shin, Jiaolong Yang, Seungryong Kim, and Chong Luo. Unifying correspondence, pose and nerf for generalized pose-free novel view synthesis. *Proc. CVPR*, 2024. 3

[27] Emiel Hoogeboom, Jonathan Heek, and Tim Salimans. simple diffusion: End-to-end diffusion for high resolution images. In *Proc. ICML*, 2023. 14

[28] Basile Van Hoorick, Rundi Wu, Ege Ozguroglu, Kyle Sargent, Ruoshi Liu, Pavel Tokmakov, Achal Dave, Changxi Zheng, and Carl Vondrick. Generative camera dolly: Extreme monocular dynamic novel view synthesis. In *Proc. ECCV*, 2024. 3

[29] Po-Han Huang, Kevin Matzen, Johannes Kopf, Narendra Ahuja, and Jia-Bin Huang. DeepMVS: Learning multi-view stereopsis. In *Proc. CVPR*, 2018. 17

[30] Tooba Imtiaz, Lucy Chai, Kathryn Heal, Xuan Luo, Jungyeon Park, Jennifer Dy, and John Flynn. LVT: large-scale scene reconstruction via local view transformers. In *Proc. SIGGRAPH Asia*, 2025. 1

[31] Jensen, Zhou, Hang Gao, Vikram Voleti, Aaryaman Vasishta, Chun-Han Yao, Mark Boss, Philip Torr, Christian Rupprecht, and Varun Jampani. Stable virtual camera: Generative view synthesis with diffusion models. In *Proc. ICCV*, 2025. 3

[32] Hanwen Jiang, Hao Tan, Peng Wang, Haian Jin, Yue Zhao, Sai Bi, Kai Zhang, Fujun Luan, Kalyan Sunkavalli, Qixing Huang, and Georgios Pavlakos. RayZer: a self-supervised large view synthesis model. In *Proc. ICCV*, 2025. 1, 2, 3, 4, 13, 14

[33] Lihan Jiang, Yucheng Mao, Linning Xu, Tao Lu, Kerui Ren, Yichen Jin, Xudong Xu, Mulin Yu, Jiangmiao Pang, Feng Zhao, Dahua Lin, and Bo Dai. AnySplat: feed-forward 3D Gaussian Splatting from unconstrained views. *ACM Trans. on Graphics (TOG)*, 44(6):1–16, 2025. 1, 2, 3, 4, 7, 8

[34] Haian Jin, Hanwen Jiang, Hao Tan, Kai Zhang, Sai Bi, Tianyuan Zhang, Fujun Luan, Noah Snavely, and Zexiang Xu. LVSM: a large view synthesis model with minimal 3D inductive bias. In *Proc. ICLR*, 2025. 1, 2, 3, 4, 6, 7, 15

[35] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *Proc. ECCV*, 2016. 5, 14

[36] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3D Gaussian Splatting for real-time radiance field rendering. *Proc. SIGGRAPH*, 42(4), 2023. 1, 2

[37] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3D Gaussian Splatting for Real-Time Radiance Field Rendering. *ACM Trans. on Graphics (TOG)*, 2023. 2

[38] Evan Kim, Hyunwoo Ryu, Thomas W. Mitchel, and Vincent Sitzmann. Scaling view synthesis transformers. In *Proc. CVPR*, 2026. 3, 13

[39] Hoang-An Le, Partha Das, Thomas Mensink, Sezer Karaoglu, and Theo Gevers. EDEN: Multimodal Synthetic Dataset of Enclosed garDEN Scenes. In *Proc. WACV*, 2021. 17

[40] Vincent Leroy, Yohann Cabon, and Jérôme Revaud. Grounding Image Matching in 3D with MAST3R. In *Proc. ECCV*, 2024. 6

[41] Hanwen Liang, Junli Cao, Vidit Goel, Guocheng Qian, Sergei Korolev, Demetri Terzopoulos, Konstantinos N. Plataniotis, Sergey Tulyakov, and Jian Ren. Wonderland: Navigating 3D scenes from a single image. In *Proc. CVPR*, 2025. 3

[42] Kai-En Lin, Yen-Chen Lin, Wei-Sheng Lai, Tsung-Yi Lin, Yi-Chang Shih, and Ravi Ramamoorthi. Vision transformer for nerf-based view synthesis from a single input image. In *Proc. WACV*, 2023. 2

[43] Shanchuan Lin, Bingchen Liu, Jiashi Li, and Xiao Yang. Common Diffusion Noise Schedules and Sample Steps are Flawed. In *Proc. WACV*, 2024. 14

[44] Lu Ling, Yichen Sheng, Zhi Tu, Wentian Zhao, Cheng Xin, Kun Wan, Lantao Yu, Qianyu Guo, Zixun Yu, Yawen Lu, Xuanmao Li, Xingpeng Sun, Rohan Ashok, Aniruddha Mukherjee, Hao Kang, Xiangrui Kong, Gang Hua, Tianyi Zhang, Bedrich Benes, and Aniket Bera. DL3DV-10K: a large-scale scene dataset for deep learning-based 3d vision. In *Proc. CVPR*, 2024. 5

[45] Fangfu Liu, Wenqiang Sun, Hanyang Wang, Yikai Wang, Haowen Sun, Junliang Ye, Jun Zhang, and Yueqi Duan. ReconX: reconstruct any scene from sparse views with video diffusion model. *arXiv*, 2408.16767, 2024. 3

[46] Ruoshi Liu, Rundi Wu, Basile Van Hoorick, Pavel Tokmakov, Sergey Zakharov, and Carl Vondrick. Zero-1-to-3: Zero-shot one image to 3D object. In *Proc. ICCV*, 2023. 3

[47] Xingchen Liu, Piyush Tayal, Jianyuan Wang, Jesus Zarzar, Tom Monnier, Konstantinos Tertikas, Jiali Duan, Antoine Toisoul, Jason Y. Zhang, Natalia Neverova, Andrea Vedaldi, Roman Shapovalov, and David Novotny. uCO3D uncommon objects in 3D. In *Proc. CVPR*, 2025. 17

[48] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. NeRF: Representing scenes as neural radiance fields for view synthesis. In *Proc. ECCV*, 2020. 1, 2

[49] Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy V. Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel HAZIZA, Francisco Massa, Alaaeldin El-Nouby, Mido Assran, Nicolas Ballas, Wojciech Galuba, Russell Howes, Po-Yao Huang, Shang-Wen Li, Ishan Misra, Michael Rabbat, Vasu Sharma, Gabriel Synnaeve, Hu Xu, Herve Jegou, Julien Mairal, Patrick Labatut, Armand Joulin, and Piotr Bojanowski. DINOv2: Learning robust visual features without supervision. *Transactions on Machine Learning Research*, 2024. 2, 6

[50] William Peebles and Saining Xie. Scalable diffusion models with transformers. In *Proc. ICCV*, 2023. 8, 14

[51] Jeremy Reizenstein, Roman Shapovalov, Philipp Henzler, Luca Sbordone, Patrick Labatut, and David Novotny. Common objects in 3d: Large-scale learning and evaluation of real-life 3d category reconstruction. In *Proc. ICCV*, 2021. 5, 8

[52] Xuanchi Ren, Tianchang Shen, Jiahui Huang, Huan Ling, Yifan Lu, Merlin Nimier-David, Thomas Müller, Alexander Keller, Sanja Fidler, and Jun Gao. GEN3C: 3D-informed world-consistent video generation with precise camera control. In *Proc. CVPR*, 2025. 3

[53] Mike Roberts, Jason Ramapuram, Anurag Ranjan, Atulit Kumar, Miguel Angel Bautista, Nathan Paczan, Russ Webb, and Joshua M. Susskind. Hypersim: A photorealistic synthetic dataset for holistic indoor scene understanding. In *Proc. ICCV*, 2021. 17

[54] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proc. CVPR*, 2022. 8, 16

[55] Mehdi S. M. Sajjadi, Henning Meyer, Etienne Pot, Urs Bergmann, Klaus Greff, Noha Radwan, Suhani Vora, Mario Lucic, Daniel Duckworth, Alexey Dosovitskiy, Jakob Uszkoreit, Thomas A. Funkhouser, and Andrea Tagliasacchi. Scene representation transformer: Geometry-free novel view synthesis through set-latent scene representations. *CoRR*, abs/2111.13152, 2021. 1

[56] Mehdi S. M. Sajjadi, Henning Meyer, Etienne Pot, Urs Bergmann, Klaus Greff, Noha Radwan, Suhani Vora, Mario Lucic, Daniel Duckworth, Alexey Dosovitskiy, Jakob Uszkoreit, Thomas Funkhouser, and Andrea Tagliasacchi. Scene representation transformer: Geometry-free novel view synthesis through set-latent scene representations. In *Proc. CVPR*, 2022. 2, 3, 4

[57] Kyle Sargent, Zizhang Li, Tanmay Shah, Charles Herrmann, Hong-Xing Yu, Yunzhi Zhang, Eric Ryan Chan, Dmitry Lagun, Li Fei-Fei, Deqing Sun, and Jiajun Wu. ZeroNVS: Zero-shot 360-degree view synthesis from a single real image. *arXiv.cs*, abs/2310.17994, 2023. 3

[58] Johannes Lutz Schönberger and Jan-Michael Frahm. Structure-from-motion revisited. In *Proc. CVPR*, 2016. 17

[59] Philipp Schröppel, Jan Bechtold, Artemij Amiranashvili, and Thomas Brox. A benchmark and a baseline for robust multi-view depth estimation. In *Proc. 3DV*, 2022. 17

[60] Jay Shah, Ganesh Bikshandi, Ying Zhang, Vijay Thakkar, Pradeep Ramani, and Tri Dao. FlashAttention-3: Fast and accurate attention with asynchrony and low-precision. In *Proc. NeurIPS*, 2024. 5

[61] Vincent Sitzmann, Semon Rezchikov, Bill Freeman, Josh Tenenbaum, and Frédo Durand. Light field networks: Neural scene representations with single-evaluation rendering. In *Proc. NeurIPS*, 2021. 3

[62] Brandon Smart, Chuanxia Zheng, Iro Laina, and Victor Adrian Prisacariu. Splatt3R: Zero-shot gaussian splatting from uncalibrated image pairs. *arXiv*, 2408.13912, 2024. 1, 3

[63] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. In *Proc. ICLR*, 2021. 14

[64] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. In *Proc. ICML Workshops*, 2015. 4

[65] Stanislaw Szymanowicz, Christian Rupprecht, and Andrea Vedaldi. Splatter Image: Ultra-fast single-view 3D reconstruction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024. 1, 3, 8

[66] Stanislaw Szymanowicz, Eldar Insafutdinov, Chuanxia Zheng, Dylan Campbell, João F. Henriques, Christian Rupprecht, and Andrea Vedaldi. Flash3D: Feed-forward generalisable 3D scene reconstruction from a single image. In *Proceedings of the International Conference on 3D Vision (3DV)*, 2025. 1, 3, 8

[67] Stanislaw Szymanowicz, Jason Y. Zhang, Pratul Srinivasan, Ruiqi Gao, Arthur Brussee, Aleksander Holynski, Ricardo Martin-Brualla, Jonathan T. Barron, and Philipp Henzler. Bolt3D: Generating 3D scenes in seconds. In *Proc. ICCV*, 2025. 3

[68] Bill Triggs. Camera pose and calibration from 4 or 5 known 3D points. In *Proc. ICCV*, 1999. 18

[69] Suhani Vora, Noha Radwan, Klaus Greff, Henning Meyer, Kyle Genova, Mehdi S. M. Sajjadi, Etienne Pot, Andrea Tagliasacchi, and Daniel Duckworth. NeSF: Neural semantic fields for generalizable semantic segmentation of 3D scenes. *Trans. on Machine Learning Research*, 2022. 17

[70] Jianyuan Wang, Minghao Chen, Nikita Karaev, Andrea Vedaldi, Christian Rupprecht, and David Novotny. Vggt: Visual geometry grounded transformer. In *Proc. CVPR*, 2025. 2, 3, 4, 6, 15, 17

[71] Shuzhe Wang, Vincent Leroy, Yohann Cabon, Boris Chidlovskii, and Jerome Revaud. DUSt3R: Geometric 3D vision made easy. In *Proc. CVPR*, 2024. 3, 6

[72] Wenshan Wang, Delong Zhu, Xiangwei Wang, Yaoyu Hu, Yuheng Qiu, Chen Wang, Yafei Hu, Ashish Kapoor, and Sebastian Scherer. TartanAir: a dataset to push the limits of visual SLAM. In *Proc. IROS*, 2020. 17

[73] Yifan Wang, Jianjun Zhou, Haoyi Zhu, Wenzheng Chang, Yang Zhou, Zizun Li, Junyi Chen, Jiangmiao Pang, Chunhua Shen, and Tong He. $\pi^3$: Permutation-equivariant visual geometry learning. In *Proc. ICLR*, 2026. 3

[74] Zhou Wang, Alan C. Bovik, Hamid R. Sheikh, and Eero P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE Trans. on Image Processing*, 13(4), 2004. 5

[75] Daniel Watson, William Chan, Ricardo Martin-Brualla, Jonathan Ho, Andrea Tagliasacchi, and Mohammad Norouzi. Novel view synthesis with diffusion models. In *Proc. ICLR*, 2023. 3

[76] Rundi Wu, Ben Mildenhall, Philipp Henzler, Keunhong Park, Ruiqi Gao, Daniel Watson, Pratul P. Srinivasan, Dor Verbin, Jonathan T. Barron, Ben Poole, and Aleksander Holynski. ReconFusion: 3D Reconstruction with Diffusion Priors. In *Proc. CVPR*, 2024. 3, 8, 13

[77] Rundi Wu, Ruiqi Gao, Ben Poole, Alex Trevithick, Changxi Zheng, Jonathan T. Barron, and Aleksander Holynski. CAT4D: create anything in 4D with multi-view video diffusion models. In *Proc. CVPR*, 2025. 3

[78] Hongchi Xia, Yang Fu, Sifei Liu, and Xiaolong Wang. RGBD objects in the wild: Scaling real-world 3D object learning from RGB-D videos. In *Proc. CVPR*, 2024. 5, 17

[79] Haodong Xiang, Xinghui Li, Kai Cheng, Xiansong Lai, Wanting Zhang, Zhichao Liao, Long Zeng, and Xueping Liu. GaussianRoom: improving 3D Gaussian splatting with SDF guidance and monocular cues for indoor scene reconstruction. In *Proc. ICRA*, 2025. 1

[80] Haofei Xu, Songyou Peng, Fangjinhua Wang, Hermann Blum, Daniel Barath, Andreas Geiger, and Marc Pollefeys. DepthSplat: Connecting Gaussian Splatting and Depth. In *Proc. CVPR*, 2025. 3, 7, 13

[81] Yao Yao, Zixin Luo, Shiwei Li, Jingyang Zhang, Yufan Ren, Lei Zhou, Tian Fang, and Long Quan. Blendedmvs: A large-scale dataset for generalized multi-view stereo networks. In *Proc. CVPR*, 2020. 17

[82] Botao Ye, Sifei Liu, Haofei Xu, Li Xueting, Marc Pollefeys, Ming-Hsuan Yang, and Peng Songyou. No Pose, No Problem: Surprisingly Simple 3D Gaussian Splats from Sparse Unposed Images. In *Proc. ICLR*, 2025. 1, 3, 7, 8

[83] Botao Ye, Boqi Chen, Haofei Xu, Daniel Barath, and Marc Pollefeys. Yonosplat: You only need one model for feedforward 3d gaussian splatting. *Proc. ICLR*, 2026. 3

[84] Alex Yu, Vickie Ye, Matthew Tancik, and Angjoo Kanazawa. PixelNeRF: Neural radiance fields from one or few images. In *Proc. CVPR*, 2021. 2

[85] Wangbo Yu, Jinbo Xing, Li Yuan, Wenbo Hu, Xiaoyu Li, Zhipeng Huang, Xiangjun Gao, Tien-Tsin Wong, Ying Shan, and Yonghong Tian. ViewCrafter: taming video diffusion models for high-fidelity novel view synthesis. pages 1–18, 2025. 3

[86] Amir Roshan Zamir, Alexander Sax, William B. Shen, Leonidas J. Guibas, Jitendra Malik, and Silvio Savarese. Taskonomy: Disentangling task transfer learning. In *Proc. CVPR*, 2018. 17

[87] Jason Y. Zhang, Amy Lin, Moneish Kumar, Tzu-Hsuan Yang, Deva Ramanan, and Shubham Tulsiani. Cameras as rays: Pose estimation via ray diffusion. In *Proc. ICLR*, 2024. 4, 16

[88] Kai Zhang, Sai Bi, Hao Tan, Yuanbo Xiangli, Nanxuan Zhao, Kalyan Sunkavalli, and Zexiang Xu. GS-LRM: Large Reconstruction Model for 3D Gaussian Splatting. In *Proc. ECCV*, 2024. 3, 6

[89] Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *Proc. CVPR*, 2018. 5

[90] Shangzhan Zhang, Jianyuan Wang, Yinghao Xu, Nan Xue, Christian Rupprecht, Xiaowei Zhou, Yujun Shen, and Gordon Wetzstein. Flare: Feed-forward geometry, appearance and camera estimation from uncalibrated sparse views. In *Proc. CVPR*, 2025. 3, 7, 8, 13

[91] Tinghui Zhou, Richard Tucker, John Flynn, Graham Fyffe, and Noah Snavely. Stereo magnification: Learning view synthesis using multiplane images. *ACM Trans. on Graphics (TOG)*, 37(4):1–12, 2018. 2, 5, 6, 17

[92] Chen Ziwen, Hao Tan, Kai Zhang, Sai Bi, Fujun Luan, Yicong Hong, Li Fuxin, and Zexiang Xu. Long-LRM: Long-sequence Large Reconstruction Model for Wide-coverage Gaussian Splats. *Proc. ICCV*, 2025. 3

# LagerNVS: Latent Geometry for Fully Neural Real-time Novel View Synthesis

## Supplementary Material

| Dataset | Views | Posed | Split | PSNR | SSIM | LPIPS |
|---|---|---|---|---|---|---|
| Re10k | 2 | ✓ | [9] | 28.99 (29.05) | 0.900 (0.901) | 0.149 (0.147) |
| Re10k | 2 | ✗ | [9] | 27.88 (28.28) | 0.875 (0.885) | 0.161 (0.155) |
| Re10k | 2 | ✓ | [90] | 26.36 (26.40) | 0.866 (0.867) | 0.190 (0.188) |
| Re10k | 2 | ✗ | [90] | 25.11 (25.64) | 0.833 (0.848) | 0.210 (0.201) |
| DL3DV | 2 | ✓ | [80] | 21.66 (21.77) | 0.688 (0.692) | 0.290 (0.287) |
| DL3DV | 2 | ✗ | [80] | 21.27 (21.33) | 0.666 (0.670) | 0.303 (0.301) |
| DL3DV | 4 | ✓ | [80] | 24.90 (24.94) | 0.778 (0.780) | 0.189 (0.188) |
| DL3DV | 4 | ✗ | [80] | 23.89 (23.99) | 0.738 (0.744) | 0.208 (0.206) |
| DL3DV | 6 | ✓ | [80] | 26.09 (26.14) | 0.806 (0.808) | 0.161 (0.159) |
| DL3DV | 6 | ✗ | [80] | 24.86 (24.97) | 0.761 (0.769) | 0.181 (0.178) |
| DL3DV | 16 | ✓ | [32] | 25.20 (25.42) | 0.776 (0.782) | 0.174 (0.171) |
| DL3DV | 16 | ✗ | [32] | 23.31 (23.49) | 0.713 (0.719) | 0.214 (0.211) |
| CO3D | 3 | ✓ | [76] | 21.18 (21.31) | 0.688 (0.691) | 0.393 (0.386) |
| CO3D | 3 | ✗ | [76] | 19.88 (20.22) | 0.660 (0.667) | 0.445 (0.431) |
| CO3D | 6 | ✓ | [76] | 23.41 (23.65) | 0.728 (0.733) | 0.326 (0.317) |
| CO3D | 6 | ✗ | [76] | 21.37 (21.65) | 0.680 (0.684) | 0.388 (0.377) |
| CO3D | 9 | ✓ | [76] | 24.40 (24.74) | 0.740 (0.747) | 0.302 (0.292) |
| CO3D | 9 | ✗ | [76] | 22.05 (22.37) | 0.689 (0.697) | 0.365 (0.352) |
| Mip360 | 3 | ✓ | [76] | 17.74 (18.08) | 0.428 (0.434) | 0.505 (0.497) |
| Mip360 | 3 | ✗ | [76] | 17.29 (17.45) | 0.409 (0.413) | 0.535 (0.531) |
| Mip360 | 6 | ✓ | [76] | 19.17 (19.39) | 0.466 (0.469) | 0.444 (0.436) |
| Mip360 | 6 | ✗ | [76] | 18.81 (18.97) | 0.446 (0.447) | 0.472 (0.466) |
| Mip360 | 9 | ✓ | [76] | 20.19 (20.39) | 0.487 (0.493) | 0.412 (0.402) |
| Mip360 | 9 | ✗ | [76] | 19.69 (19.68) | 0.461 (0.462) | 0.440 (0.438) |

Table A1. **Generalizable model performance at** $512 \times 512$**.** To facilitate future comparisons, we report the performance of our generalizable model on a number of standard high-resolution NVS benchmarks, with and without source camera poses. Scores are reported for version 2 of the model in black, described in Sec. 3 and Sec. C.3. Gray numbers represent scores for the previous version (v1).

## A. Additional experimental results

Video results are available on our project page.

### A.1. Generalizable, high-resolution NVS

Previous works on deterministic Novel View Synthesis typically evaluate quantitative performance in a single-dataset setting at low resolution (256). To facilitate future comparisons, we include quantitative evaluation on a number of standard benchmarks at $512 \times 512$, with and without known camera poses in Tab. A1.

### A.2. Additional comparisons to LVSM

Additional qualitative comparisons to LVSM in Fig. A1 complement Tab. 1 and Fig. 5. Figure A1 shows that our model performs better on examples where matching in 2D is difficult, due to low overlap (first row), repeated patterns (second row), and geometry close to the camera (third row). This indicates that the 3D bias from pre-training helps to disambiguate such challenging cases.


Figure A1. **Additional comparison vs LVSM.** Our model better maintains consistent shape in regions where 2D matching across inputs is challenging (top 2 rows) and shows better monocular depth estimation (bottom row).

| | PSNR | SSIM | LPIPS |
|---|---|---|---|
| SVSM | 30.01 | 0.910 | 0.096 |
| Ours | **31.39** | **0.928** | **0.078** |

Table A2. LagerNVS outperforms SVSM due to greater model capacity and 3D pre-training.

### A.3. SVSM

SVSM [38], a concurrent work, is a variant of LVSM that, similarly to us, adopts an encoder-decoder network with a cross-attention decoder. Unlike us, however, they focus on efficient compute usage, and use unidirectional cross-attention (Alg. 1). We instead study the role of 3D pre-training, use a larger encoder and bidirectional cross-attention. As shown in Tab. A2, we achieve better quality, although our network is larger. The training efficiency findings from SVSM are orthogonal to our 3D pre-training and design and can likely be combined for further gains.

### A.4. Additional comparisons to 3DGS

We include additional visual comparisons to feed-forward 3DGS methods in Fig. A2, with and without known cam-

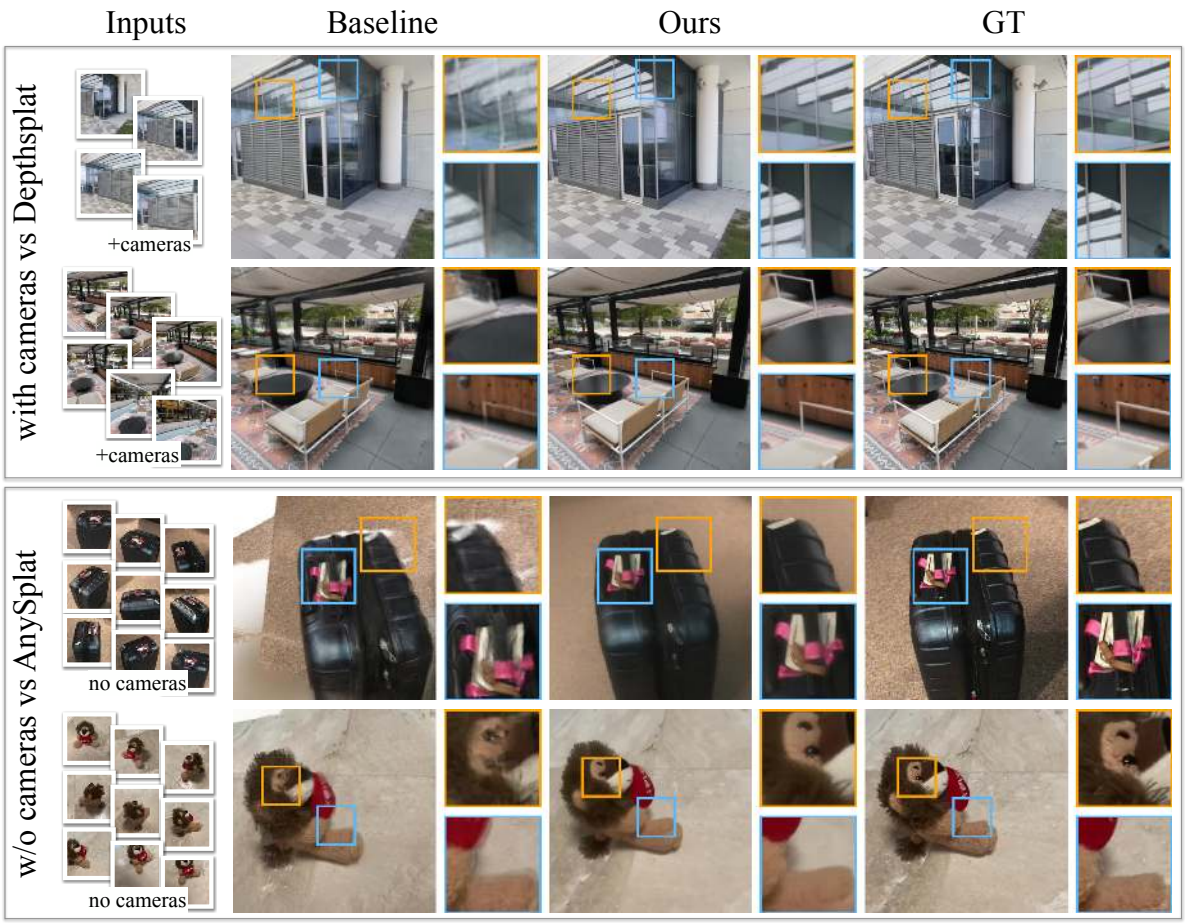


Figure A2. **Qualitative comparison vs Feed-forward 3DGS.** Our model more accurately represents reflective areas (mirror, top row and table, second row) and thin structures (metal rod, top row and chair arm, second row). The global, latent representation leads to better surface alignment (suitcase handle, third row and teddy bear, bottom row) and is more robust to occlusions (floor, third row).

eras, further illustrating that our method improves performance on reflective surfaces, thin structures, and occluded regions. Other methods can suffer from surface misalignment, especially in 360° captures.

### A.5. Occlusions

The main model we present is deterministic, so in case of ambiguity (e.g., from occlusions), it tends to regress to the mean. Interestingly, however, we find that in simple cases, the "mean" is a good enough approximation to the ambiguous regions. In Fig. A3, we show two examples where the completion is successful: the corner of the bathtub, and the bottom of the box and shelf. In more difficult examples, as in other deterministic methods (see the limitations section of RayZer [32]), completions are necessarily blurry, but they are still surprisingly reasonable (occluded grass and road in the bottom row of Fig. A3). A more principled solution to handling occlusions correctly is to use a *diffusion decoder*, which can sample from the conditional probability distribution. We illustrated an example of how our model can be fine-tuned as a diffusion model in Fig. 9 of the main paper. The next section includes more details.

## B. Diffusion

**Implementation details.** We make two modifications to the architecture of the decoder to allow it to operate as a diffusion model. First, we add adaLN-zero [50] conditioning layers to the decoder to condition it on the denoising timestep. Second, the input patch embedding layer is changed to accept 3 additional channels, corresponding to the noisy input, leading to a total of $3 + 6 = 9$ input channels in the decoder. We employ a DDIM scheduler [63] with 1000 training timesteps, a linear beta schedule from $\beta_{\text{start}} = 0.001$ to $\beta_{\text{end}} = 0.02$, and train the model to predict the clean sample directly (i.e., $\mathbf{x}_0$ prediction) in pixel space. We use a zero-SNR noise scheduler [43] shifted with a ratio of 4 to account for a relatively high (256×256) resolution [27]. Training uses Mean-Squared Error and Perceptual Losses [35] with weights 1.0 and 4.0, respectively, and

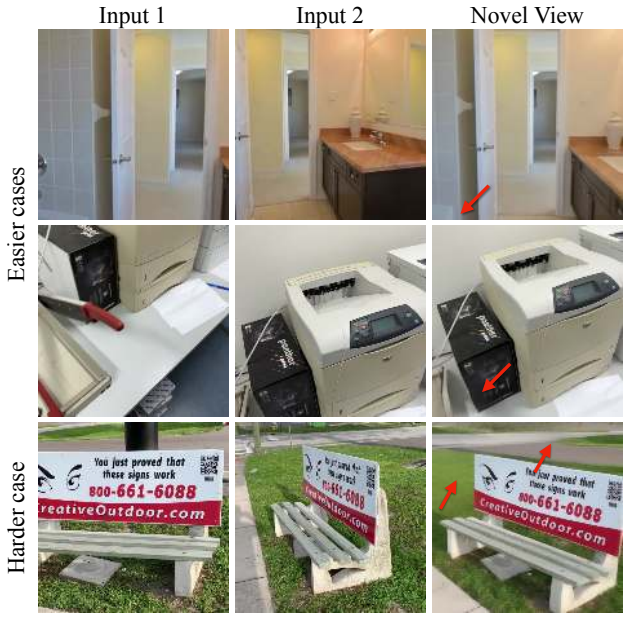


Figure A3. **Occlusions.** Our model can handle simple occlusions, such as the corner of the bathtub (top) or the bottom part of the black box (middle). In more difficult settings (bottom), completions are blurry (as expected), but still reasonable.

a learning rate of $1 \times 10^{-5}$ for 60k iterations.

**Discussion.** Our diffusion decoder can only generate individual Novel Views, thus generating a video would necessarily result in flicker. Generating a consistent video would require a video diffusion model as a decoder, or an autoregressive formulation like in [8]. Both are possible, are orthogonal to our contributions, and are an exciting direction for future work.

# C. Implementation details

## C.1. Architecture: encoder

Here we include additional details of the encoder described in the main paper. Figure A4 serves as a visual reference.

Recall that VGGT operates with images with longer side $H' = 518$ or $W' = 518$ and token dimension 1024. We wish to support images with longer side up to $H = 512$ or $W = 512$ (to follow standard resolutions on NVS benchmarks). Source images are thus first resized to the dimension expected by VGGT [70], i.e., with longer side 518, using bilinear interpolation ((1) in Fig. A4).

Camera tokens $\boldsymbol{g}_i$ are projected to the token dimension 1024 ((2) in Fig. A4) using two linear layers with weight size $9 \times 1024$ and $1024 \times 1024$, respectively, with SiLU [19] activation in between. The VGGT "camera token," which distinguishes the first camera from the remaining ones, is added to the projected token. When cameras $\boldsymbol{g}_i$ are not available, we set the first 9 elements of the camera vector to 0: $\boldsymbol{g}_i[:9] = \mathbf{0}$, but keep the scale normalization factor. Similarly to VGGT, we use the camera pose of the first camera $\boldsymbol{g}_1$ as the reference frame.

Image and camera tokens are fed to the encoder backbone ((3) in Fig. A4), which has weights initialized from VGGT. The backbone consists of an image embedder (a 24-layer transformer), which operates per-image, and the aggregator (a 48-layer transformer), which interleaves local (per-frame) and global (per-sequence) attention layers. We keep the architecture of the backbone unmodified; see [70] for details. We keep the tokens output by the last local attention layer and the last global attention layer, and concatenate them channel-wise. The backbone output thus has shape $V \times P \times (2 \times 1024)$, where $P$ is the number of tokens processed by VGGT, after discarding the camera and register tokens.

Next (in (4) in Fig. A4), a linear layer of weight matrix with size $2048 \times 768$ is used to project two features (one from a global attention layer and one from local attention layer) of VGGT channel dimension, 1024, to the one expected by the decoder, 768; this is followed by a Layer Normalization [3] layer. Such operations output per-image tokens $\boldsymbol{z}_i$, which are concatenated for the whole sequence to form the latent 3D representation $(\boldsymbol{z}_1, \ldots, \boldsymbol{z}_V)$.

## C.2. Architecture: decoder

Here we include additional details of the decoder described in the main paper, with a visual illustration shown in Fig. A5.

The decoder receives the *latent 3D representation* of the 3D scene $(\boldsymbol{z}_1, \ldots, \boldsymbol{z}_V)$, as well as the target camera pose $\boldsymbol{g}$. The camera is first tokenized ((1) in Fig. A5) with a strided convolution layer, reshaped, and followed by concatenating 4 register tokens, as described in Sec 3.2 of the main paper.

Next, the scene representation tokens and the camera tokens $\boldsymbol{s}$ are input to the backbone of the decoder, a Vision Transformer [17] ((2) in Fig. A5). We use ViT-B, which is a transformer with channel dimension $C = 768$, 12 heads, and 12 transformer blocks (implemented as in Sec. C.2.1 and Alg. 1). The feed-forward layers have a hidden dimension expansion factor of 4. Finally, we discard the scene reconstruction tokens and the register tokens, and read out ((3) in Fig. A5) the novel view from the updated target camera tokens. The readout process includes normalization, projecting each of $P$ tokens from channel dimension $C$ to $3 \times r'^2$ (recall $r'$ is the patch size of the decoder), rearranging to the final image dimension $H \times W \times 3$, and a per-element sigmoid activation function. Following LVSM [34], we do not use biases in the decoder layers.

### C.2.1. Attention mechanism

As introduced in Sec. 3.2 of the main paper, we consider two variants of the attention mechanism in the transformer

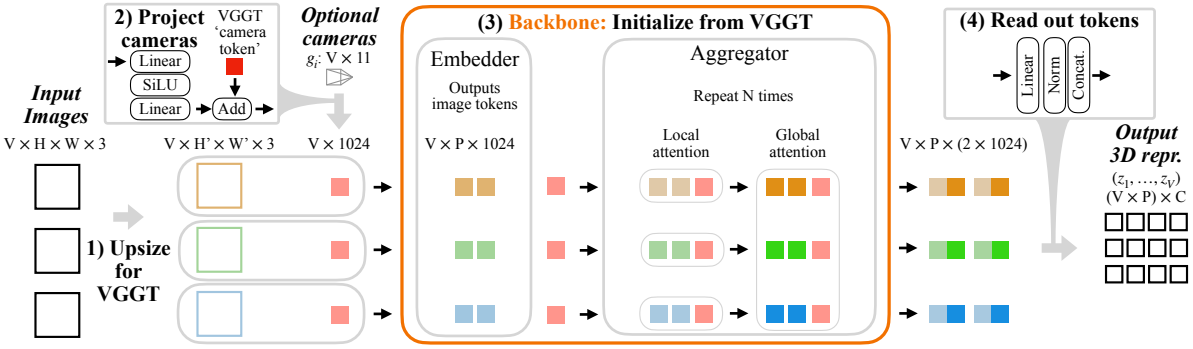


Figure A4. **Encoder.** The encoder takes as source $V$ *images* and, optionally, $V$ *cameras*. The images are upsized (1) to the dimension expected by VGGT, and cameras $\boldsymbol{g}$ are projected (2) to conditioning tokens. Image and camera tokens are fed together to the encoder backbone, which is initialized with VGGT weights (3). The tokens from the last local attention layer and global attention layer of VGGT are concatenated, after discarding camera tokens. These tokens are then (4) projected to decoder channel dimension $C$ to form the *output latent 3D representation* $(\boldsymbol{z}_1, \ldots, \boldsymbol{z}_V)$.

blocks (see Alg. 1). First, we consider the "full attention" variant, where both scene reconstruction tokens and the target camera tokens serve as queries, keys, and values in the attention mechanism. In the "cross-attention" variant, we implement bidirectional cross-attention. In this variant, both the camera and scene tokens are updated via cross-attention to the other. The target camera tokens are allowed to self-attend, but there is no self-attention between the scene tokens. Both sets of tokens are updated via feed-forward layers. The algorithm for both is described in Alg. 1. Such an implementation allows the target camera tokens and latent geometry tokens to communicate with each other and update, without quadratic growth of complexity of the attention operation.

A variant which we do not consider in the main paper, but is commonly used in multimodal transformers such as text-to-image diffusion models [54], is unidirectional cross-attention. We detail this variant in Alg. 1 as well, but do not use it in practice, because we find that it performs poorly as the number of source views increases. Concretely, in Tab. A3, we find that while the gap in performance to full attention is small ($-0.8$ PSNR) with two source views, it grows substantially (to $-1.9$ PSNR) as the number of views increases to 6. The unidirectional cross-attention variant is faster because it does not require additional cross-attention and feed-forward update of scene reconstruction tokens.

We select the "bidirectional cross-attention" variant for our generalizable model (described in Sec 4.2 of the main paper) as a middle ground between speed and quality.

### C.3. Differences compared to the first version of the model (v1)

As noted in Sec. 3, the paper has been updated compared to the CVPR version to prevent leaking the camera focal length to the model when no source cameras are passed as input. The new version, which we call v2, sets the target camera horizontal field of view set to the canonical value $k_0$ when source cameras are unknown. The value of the canonical horizontal field of view is chosen to be $k_0 = 53.13° = \arctan(0.5)$ such that horizontal focal length $f_x$ is equal to the image width, $f_x = W$. We do so without change to the target image in supervision. This effectively trains the network to output an image with the same intrinsics as the source images [5]. In the case when source cameras are known, no change is made compared to v1: we set the target camera intrinsics to the same value as the source camera intrinsics.

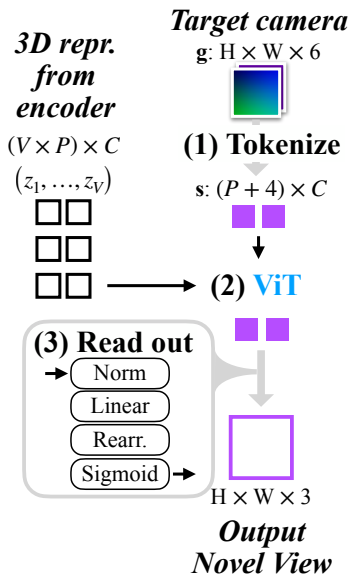


Figure A5. **Decoder.** The decoder tokenizes the *target camera* (1), in the form of a Plucker ray map [87]. The *scene tokens* (left) and target camera tokens $\boldsymbol{s}$ are fed to a Vision Transformer (2) [17], with two variants of the attention mechanism (see text). The register tokens and scene tokens are discarded after the transformer, and *output novel views* are read out from the remaining tokens.

**Algorithm 1** Transformer block

**Require:** Camera tokens $s$, Reconstructed tokens $\mathbf{z}$, Attention Mode $M$

**procedure** ATTENTIONBLOCK($s, \mathbf{z}, M$)
  **if** $M =$ "Full attention" **then**
    ▷ Tokens are concatenated prior to the block: $c = [\mathbf{z} \parallel s]$
    $c \leftarrow c + \text{SelfAttn}\ (Q = c\ ,\quad K = c\ ,\quad V = c\ )$
    $c \leftarrow c + \text{FFN}(c)$
    **return** $c$ ▷ Sequence split occurs after all blocks
  **else if** $M =$ "Unidirectional cross-attn." **then**
    ▷ Standard cross-attention block
    $s \leftarrow s + \text{SelfAttn}\ (Q = s\ ,\quad K = s\ ,\quad V = s\ )$
    $s \leftarrow s + \text{CrossAttn}(Q = s\ ,\quad K = \mathbf{z}\ ,\quad V = \mathbf{z}\ )$
    $s \leftarrow s + \text{FFN}(s)$
    **return** $s$
  **else if** $M =$ "Bidirectional cross-attention" **then**
    ▷ Bidirectional conditioning
    $s \leftarrow s + \text{SelfAttn}\ (Q = s\ ,\quad K = s\ ,\quad V = s\ )$
    $s \leftarrow s + \text{CrossAttn}(Q = s\ ,\quad K = \mathbf{z}\ ,\quad V = \mathbf{z}\ )$ ▷ Standard cross-attention
    $\mathbf{z} \leftarrow \mathbf{z} + \text{CrossAttn}(Q = \mathbf{z}\ ,\quad K = s\ ,\quad V = s\ )$ ▷ Cross-attention the 'opposite' direction
    $\mathbf{z} \leftarrow \mathbf{z} + \text{FFN}(\mathbf{z})$ ▷ FFN applied to scene tokens
    $s \leftarrow s + \text{FFN}(s)$ ▷ FFN applied to camera tokens
    **return** $s, \mathbf{z}$
  **end if**
**end procedure**

| Attention | Complexity | Max # imgs @ 512 real-time | Quality (PSNR) 2-view | 4-view | 6-view | Params |
|---|---|---|---|---|---|---|
| Full | $\mathcal{O}(V^2)$ | 6 | **21.30** | **24.17** | **24.84** | 85M |
| Unidirectional cross-attn. (not used) | $\mathcal{O}(V)$ | **26** | 20.51 | 22.56 | 22.91 | 113M |
| Bidirectional cross-attn. (ours) | $\mathcal{O}(V)$ | 9 | 21.02 | 23.65 | 24.54 | 170M |

Table A3. **Decoder attention variants.** The bidirectional cross-attention mechanism used for the main model offers a good trade-off between real-time rendering speed and NVS quality.

This change was necessary because during training the source and target images have the same focal length. As a consequence, specifying the focal length for the target images could potentially "leak" the focal length information of the source cameras. Moreover, specifying target camera intrinsics different from (potentially unknown) source images resulted in deteriorated performance in model v1 due to a train-test mismatch. Providing a nominal value $k_0$ as horizontal field of view solves both issues. First, it provides no information about source camera intrinsics. Second, such nominal value can be applied both at training and testing time, thus removing the train-test mismatch.

After the change, the new model has slightly worse performance on the benchmarks (but is still state-of-the-art by a good margin). Table Tab. A1 shows the difference. We have updated the results in the paper to refer to 'v2' when relevant, and marked this explicitly in Tab. 3. Note that only the 'generalizable' variant of the model is affected by this change (because that is the only model we train with unposed source images), so some of the rows are unaffected.

Note also that model v1 is completely valid for the case where source cameras are passed as input, but we prefer for the users to use model v2 as default for all cases as the performance difference is minor.

## C.4. Training details

**Data** One advantage of building on multi-view stereo models like VGGT [70] is that they generalize well due to training on a large collection of datasets. In line with this work, we consider a rich mix of datasets to supervise our model too, including TartanAir [72], Eden [39], ARKit [6], BlendMVS [81], Hypersim [53], UCo3D [47], Taskonomy [86], RealEstate10k [91], StaticThings [59], NeSF [69], MVSSync [29], WildRGBD [78], and approximately 45k Internet-style videos, whose cameras were annotated using a structure-from-motion pipeline [58] with post-filtering. We sample the datasets with variable proportions, balancing their relative size and diversity.

**Shared hyperparameters** Throughout all experiments, we use AdamW [16] optimizer with beta coefficients $(\beta_1, \beta_2) = (0.9, 0.95)$ and weight decay of $0.05$. We skip iterations when the gradient norm is larger than $5.0$. The learning rate is warmed up linearly for $3k$ iterations, followed by cosine annealing to zero with warm restarts. We detail the remaining hyperparameters in Tab. A4.

**Multiple targets per one scene encoding.** The forward pass through the encoder, which has to process $V \geq 1$ images, is much slower than a pass through the decoder, which processes a single target view. We thus rebalance the costs by predicting $V_{\text{target}} \geq 1$ target views for each set of source images in a single batch (these images are rendered in parallel but still independently).

**View, image, and camera augmentation.** We randomly sample the number of source views $V$ during training for each batch to allow our model to accept a variable number of source images at test time. We randomly use $1 - 10$ source views in each batch, inclusive. We sample (uniformly) an even number of source views with probability $4 : 1$ to sampling an odd number of views. Each example has a fixed number of target views $V_{target} = 8$. We randomly drop the camera pose token for $40\%$ of training examples, jointly training for posed and unposed NVS. Aspect ratio is sampled uniformly in the log domain between $[0.5, 2.0]$.

**Scene scale.** To generate the target image $I$, an NVS algorithm is given the target camera $\boldsymbol{g}$, which specifies how much the camera moves with respect to the imaged scene. Due to scale ambiguity, cameras and 3D scene are generally only defined up to a common scaling factor. The NVS algorithm must determine the relationship between the scene scale and the target camera scale to interpret the target viewpoint correctly.

When there are several source views with know source cameras, the NVS algorithm can infer the scale of the scene relative to those cameras from triangulation, and use this information to interpret the scale of the target camera too. However, when the source cameras are *not* specified as input, or when there is a single source view and triangulation is not possible, the NVS algorithm requires additional information and/or assumptions to find the relation between the scene scale and the target camera scale.

A possible approach is to assume that the target camera motion is expressed meters, and that the NVS model can implicitly perform metric reconstruction of the scene. We discount this approach because it is challenging to collect metric data at large enough volumes for training accurate metric NVS systems.

Another approach is to define the scale to be the maximum distance of any camera pose from the reference (first) camera, i.e., $w_1 = \max_i(\|t_i\|_2)$. This is convenient as camera poses can usually be estimated reliably and quickly (e.g., from the 5-point algorithm [68]). Camera poses are also available for the majority of evaluation benchmarks, thus making this normalization method well-suited for quantitative evaluations.

On the other hand, when there is a single source image, or when all the source cameras overlap, $w_1 = 0$ and using this scale is not possible. We thus consider a second method of normalization, based on the scale of the points observed in the scene, which we define as the average distance from the reference camera to all points visible in all source images

$$w_2 = \frac{1}{\sum_i \sum_u \sum_v \mathbb{1}_{iuv}} \sum_i \sum_u \sum_v \mathbb{1}_{iuv} \|x_{iuv}\|_2,$$

where $\mathbb{1}_{iuv}$ is the indicator function that evaluates to 1 if pixel in view $i$ at coordinate $u, v$ holds a valid depth, and $x_{iuv}$ is the world-coordinate location of the point.

We train our model to use either normalization method, depending on what is available at inference time. We do this by inputting both normalization factors $w_1$ and $w_2$ to the network as part of the camera parameters; in other words, highlighting the target camera parameters, the NVS function is of the type:

$$I = f(\boldsymbol{q}, \boldsymbol{t}, \boldsymbol{k}, w_1, w_2, \dots)$$

For each given training scene and camera, this gives us a certain value for the tuple $(\boldsymbol{q}, \boldsymbol{t}, \boldsymbol{k}, w_1, w_2)$. Furthermore, we can rescale the scene and cameras by a common factor to obtain a different tuple $(\boldsymbol{q}, \lambda\boldsymbol{t}, \boldsymbol{k}, \lambda w_1, \lambda w_2)$, obtaining another valid training example. Because one scaling factor may be unavailable at inference time, at training time we also randomly drop either of them out, setting it to zero, in which case the model learns to rely only on the other.

In practice, this is done as follows.

1. If only camera poses are available for a particular training scene, then $w_2 = 0$ and we choose $\lambda$ so that that $\lambda w_1 := 1/1.35$.
2. If both camera poses and points are available, the we proceed as follows.
   - We choose a $\lambda$. To do so, with 50% probability, we choose $\lambda$ so that $\lambda w_1 = 1/1.35$ (following LVSM) and with 50% probability so that $\lambda w_2 = 1$.
   - We choose which, if any, scaling factor to drop out. With 1/3 probability, we drop out $w_1$, $w_2$ or neither, thus training with $(0, \lambda w_2)$, $(\lambda w_1, 0)$, or $(\lambda w_1, \lambda w_2)$.

At test time, whenever possible, we use the scaling factor $\lambda w_1 := 1/1.35$ and pass $w_2 = 0$ to the network. As discussed above, this is a viable option whenever there are at least two cameras with non-overlapping origins. During quantitative evaluation, we compute $\lambda$ in the dataloader so that $\lambda w_1 := 1/1.35$ as required. If the evaluation is on unposed images, as discussed in the main paper, we set $\boldsymbol{g}_i$ to

| Experiment | Iter. | LR | Batch | Data | Source Views | Res | Grad clip |
|---|---|---|---|---|---|---|---|
| LVSM comparison Tab 1. (a-c) | 100k | $4e-4$ | 64 | Re10k | 2 | 256 | 1.0 |
| LVSM comparison Tab 1. (d-f) | 100k | $4e-4$ | 512 | Re10k | 2 | 256 | 1.0 |
| Ablations Tab 2. | 100k | $1e-4$ | 64 | All | 2–4 | 256 | 3.0 |
| 3DGS Tab 3. DL3DV posed | 100k | $4e-4$ | 512 | DL3DV | 2–6 | 256 | 3.0 |
| Final model, 3DGS Tab 3. Unposed and Tab 4. | 250k | $1e-4$ | 512 | All | 1–10 | 512 | 3.0 |

Table A4. **Hyperparameters.** We detail the training hyperparameters for each experiment. The model used for comparison in "3DGS Tab 3. Unposed" can also accept camera poses as conditioning, and we use it as our final model.

the null token, keeping only scale parameter $\lambda w_1$, so that the source cameras are not available to the model, but scale is unambiguous. The network is able to implicitly learn the meaning of this scaling protocol during training, and thus uniquely estimate the image $I$ for the target camera $\boldsymbol{g}$.

When such scaling is not possible due to single source image or overlapping camera origins, we use the other scaling factor. In the dataloader, we can compute $\lambda$ so that $\lambda w_2 = 1.0$, and pass $(\lambda w_1, \lambda w_2) = (0, 1.0)$. In some cases, we may not be able to compute $w_2$ either (due to a lack of depth information). Here, we simply pass $(w_1, w_2) = (0, 1)$, which means that the model implicitly understand the target camera translation to be expressed as a fraction of the scale of the points it observes in the scene.

Such training protocol was necessary for supporting single-view NVS with our main generalizable model. When training only with $w_1$ (scaling based on cameras) and evaluation with one source image, the model was capable of rendering images under rotation around center of projection of the source camera, but unable to render images under camera translation. Adding scaling based on points during training, using $w_2$, resulted in successful renders under camera translation (as shown in Fig. 6 in the main paper).

## D. Limitations

Our model is trained with identical camera intrinsics for all source images, equal to target camera intrinsics. As a consequence, providing source images with different focal lengths or different target camera intrinsics can lead to deteriorated performance. A more general version of the model should be trained to respond correctly to an arbitrary target focal length and relax the constraint that focal lengths are the same within the source cameras and with the target camera. This could be achieved by focal length randomization during training.

Next, the model is not capable of high-quality hallucination of unseen regions, and exhibits blurry renderings with block artifacts when rendering unobserved regions. Occasionally, when rendering a video, such block artifacts result in an impression of flicker, which often stems from uncertainty in geometry estimation, unobserved regions, or difficulty in estimating source camera poses. Additionally, regions with high-frequency patterns, such as grass or trees, are systematically poorly represented by our model—investigating this failure mode is an interesting avenue for future work. Moreover, our model is suited only to static data, did not include humans in the training data, and did not include images with distortion (e.g., fish-eye). As a consequence, we do not expect the model to work well in such scenarios.