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

Supplementary Material

Dataset	Views	Posed	Split	PSNR	SSIM	LPIPS
Re10k	2	✓	[4]	29.05	0.901	0.147
Re10k	2	✗	[4]	28.28	0.885	0.155
Re10k	2	✓	[33]	26.40	0.867	0.188
Re10k	2	✗	[33]	25.64	0.848	0.201
DL3DV	2	✓	[29]	21.77	0.692	0.287
DL3DV	2	✗	[29]	21.33	0.670	0.301
DL3DV	4	✓	[29]	24.94	0.780	0.188
DL3DV	4	✗	[29]	23.99	0.744	0.206
DL3DV	6	✓	[29]	26.14	0.808	0.159
DL3DV	6	✗	[29]	24.97	0.769	0.178
DL3DV	16	✓	[10]	25.42	0.782	0.171
DL3DV	16	✗	[10]	23.49	0.719	0.211
CO3D	3	✓	[27]	21.31	0.691	0.386
CO3D	3	✗	[27]	20.22	0.667	0.431
CO3D	6	✓	[27]	23.65	0.733	0.317
CO3D	6	✗	[27]	21.65	0.684	0.377
CO3D	9	✓	[27]	24.74	0.747	0.292
CO3D	9	✗	[27]	22.37	0.697	0.352
Mip360	3	✓	[27]	18.08	0.434	0.497
Mip360	3	✗	[27]	17.45	0.413	0.531
Mip360	6	✓	[27]	19.39	0.469	0.436
Mip360	6	✗	[27]	18.97	0.447	0.466
Mip360	9	✓	[27]	20.39	0.493	0.402
Mip360	9	✗	[27]	19.68	0.462	0.438

Table A1. **Generalizable model performance at** 512×512**.** 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.

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).

A. Additional experimental results

Our project page szymanowiczs.github.io/lagernvs includes video results, which we encourage the reader to view.

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 512×512, 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 from the main paper. 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.

A.3. SVSM

SVSM [13], 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 cameras, further illustrating that our method improves perfor-

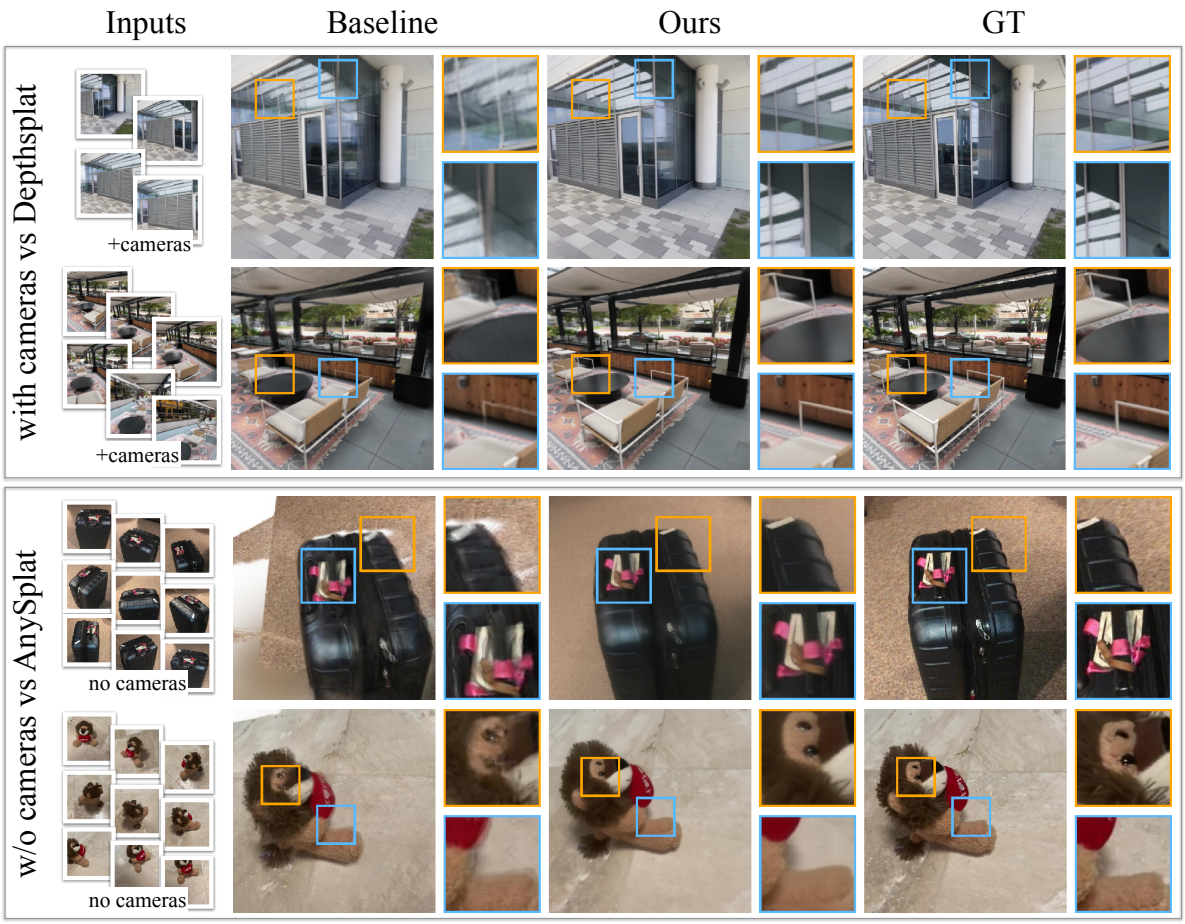

Figure A2. **Qualitative comparison vs Feed-forward 3DGS.** Our model more accurately represents reflective areas (mirror, top row and table, second row) and thin structures (metal rod, top row and chair arm, second row). The global, latent representation leads to better surface alignment (suitcase handle,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

	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.

mance 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 [10]), 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 [17] 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 [22] 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

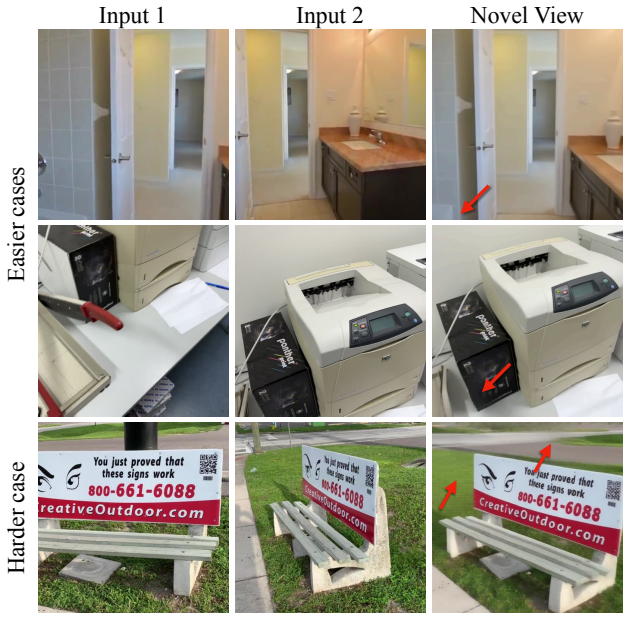

Figure A3. **Occlusions.** Our model can handle simple occlusions, such as the corner of the bathtub (top) or the bottom part of the black box (middle). In more difficult settings (bottom), completions are blurry (as expected), but still reasonable.

the clean sample directly (i.e., $\mathbf{x}_0$ prediction) in pixel space. We use a zero-SNR noise scheduler [15] shifted with a ratio of 4 to account for a relatively high (256×256) resolution [8]. Training uses Mean-Squared Error and Perceptual Losses [12] with weights 1.0 and 4.0, respectively, and a learning rate of 1×10^{-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 [3]. 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 [25], 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 9×1024 and 1024×1024, respectively, with SiLU [7] 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 [25] 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 2048×768 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 [1] 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 [6] ((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 Appendix 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

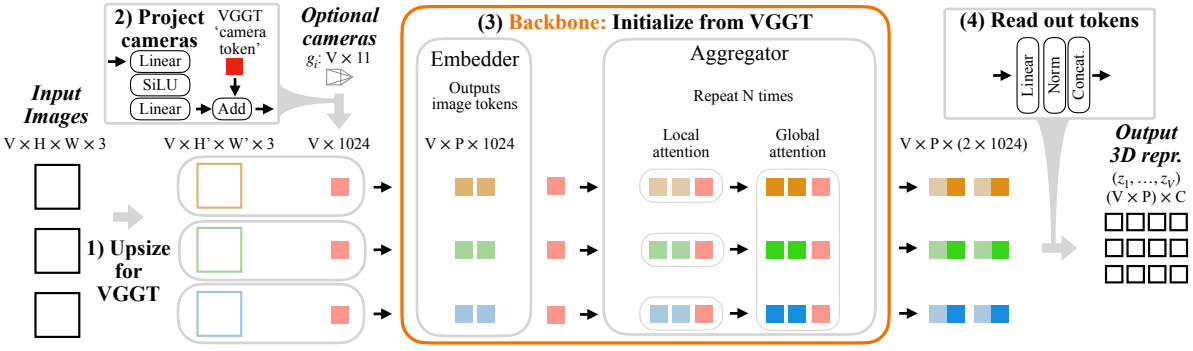

Figure A4. **Encoder.** The encoder takes as source V *images* and, optionally, V *cameras*. The images are upsized (1) to the dimension expected by VGGT, and cameras $\boldsymbol{g}$ are projected (2) to conditioning tokens. Image and camera tokens are fed together to the encoder backbone, which is 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* $(\boldsymbol{z}_1, \ldots, \boldsymbol{z}_V)$.

LVSM [11], 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 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 [19], 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.

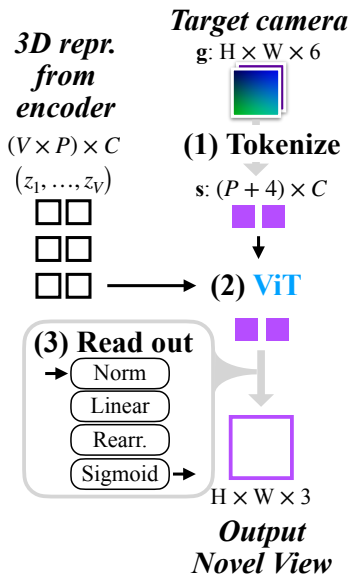

Figure A5. **Decoder.** The decoder tokenizes the *target camera* (1), in the form of a Plucker ray map [32]. The *scene tokens* (left) and target camera tokens $\boldsymbol{s}$ are fed to a Vision Transformer (2) [6], with two variants of the attention mechanism (see text). The register tokens and scene tokens are discarded after the transformer, and *output novel views* are read out from the remaining tokens.

C.3. Training details

Data One advantage of building on multi-view stereo models like VGGT [25] 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 [26], Eden [14], ARKit [2], BlendMVS [30], Hypersim [18], UCo3D [16], Taskonomy [31], RealEstate10k [34], StaticThings [21], NeSF [24], MVSSync [9], WildRGBD [28], and approximately 45k Internet-style videos, whose cameras were annotated using a structure-from-motion pipeline [20] with

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)	$\mathcal{O}(V)$	<u>9</u>	<u>21.02</u>	<u>23.65</u>	<u>24.54</u>	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.

post-filtering. We sample the datasets with variable proportions, balancing their relative size and diversity.

Shared hyperparameters Throughout all experiments, we use AdamW [5] 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

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.

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 [23]). 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 λ so that that $\lambda w_1 := 1.35$.
2. If both camera poses and points are available, the we proceed as follows.
   - We choose a λ. To do so, with 50% probability, we choose λ so that $\lambda w_1 = 1.35$ (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 $\lambda w_1 := 1.35$ 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 λ in the dataloader so $\lambda w_1 := 1.35$ as required. If the evaluation is on unposed images, as discussed in the main paper, we set $\boldsymbol{g}_i$ to the null token, keeping only scale parameter λw_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 λ 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 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.

References

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

[2] 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. 4

[3] 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

[4] 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

[5] Jimmy Ba Diederik P. Kingma. Adam: A method for stochastic optimization. In *Proc. ICLR*, 2015. 5

[6] 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 16×16 words: Transformers for image recognition at scale. In *Proc. ICLR*, 2021. 3, 4

[7] Stefan Elfwing, Eiji Uchibe, and Kenji Doya. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural Networks*, 107, 2018. 3

[8] Emiel Hoogeboom, Jonathan Heek, and Tim Salimans. simple diffusion: End-to-end diffusion for high resolution images. In *Proc. ICML*, 2023. 3

[9] Po-Han Huang, Kevin Matzen, Johannes Kopf, Narendra Ahuja, and Jia-Bin Huang. DeepMVS: Learning multi-view stereopsis. In *Proc. CVPR*, 2018. 4

[10] 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

[11] 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. 4

[12] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *Proc. ECCV*, 2016. 3

[13] Evan Kim, Hyunwoo Ryu, Thomas W. Mitchel, and Vincent Sitzmann. Scaling view synthesis transformers. In *Proc. CVPR*, 2026. 1

[14] Hoang-An Le, Partha Das, Thomas Mensink, Sezer Karaoglu, and Theo Gevers. EDEN: Multimodal Synthetic Dataset of Enclosed garDEN Scenes. In *Proc. WACV*, 2021. 4

[15] Shanchuan Lin, Bingchen Liu, Jiashi Li, and Xiao Yang. Common Diffusion Noise Schedules and Sample Steps are Flawed. In *Proc. WACV*, 2024. 3

[16] 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. 4

[17] William Peebles and Saining Xie. Scalable diffusion models with transformers. In *Proc. ICCV*, 2023. 2

[18] 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. 4

[19] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proc. CVPR*, 2022. 4

[20] Johannes Lutz Schönberger and Jan-Michael Frahm. Structure-from-motion revisited. In *Proc. CVPR*, 2016. 4

[21] Philipp Schröppel, Jan Bechtold, Artemij Amiranashvili, and Thomas Brox. A benchmark and a baseline for robust multi-view depth estimation. In *Proc. 3DV*, 2022. 4

[22] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. In *Proc. ICLR*, 2021. 2

[23] Bill Triggs. Camera pose and calibration from 4 or 5 known 3D points. In *Proc. ICCV*, 1999. 6

[24] 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. 4

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

[26] 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. 4

[27] 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. 1

[28] 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. 4

[29] 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. 1

[30] 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. 4

[31] 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. 4

[32] 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

[33] 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. 1

[34] 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. 4