Geometry-Free View Synthesis
Transformers and no 3D Priors

—

Supplementary Material

In this supplementary, we provide additional results obtained with our models in Sec. A. Sec. B summarizes models, architectures and hyperparameters that were used in the main paper. After describing details on the training and test data in Sec. C and on the uncertainty evaluations via the entropy in Sec. D, Sec. E concludes the supplementary material with a brief discussion of the compression artifacts introduced by the usage of the VQGAN as the compression model.

A. Additional Results

Interactive Scene Exploration Interface Fig. 9 shows a preview of the videos available at https://git.io/JOnwn, which demonstrate an interface for interactive 3D exploration of images. Starting from a single image, a user can use keyboard and mouse to move the camera freely in 3D. To provide orientation, we warp the starting image to the current view using a monocular depth estimate (corresponding to the MiDaS [52] baseline in Sec. 4.2). This enables a positioning of the camera with real-time preview of the novel view. Once a desired camera position has been reached, the spacebar can be pressed to autoregressively sample a novel view with our transformer model.

In the videos available at https://git.io/JOnwn, we use camera trajectories from the test sets of RealEstate10K and ACID, respectively. The samples are produced by our *impl.-depth* model, and for an additional visual comparison, we also include results obtained with the same methods that we compared to in Sec. 4.2.

Figure 9. Preview of the videos available at https://git.io/JOnwn, which demonstrate an interface for interactive 3D exploration of images. Starting from a single image, it allows users to freely move around in 3D. See also Sec. A.

Small Viewpoint Changes & Continuous Trajectories For very small viewpoint changes, distortions due to compression dominate the error of our approach (left of Fig. 10, where the x-axis uses PSIM between source and target view as a proxy for the difficulty/magnitude of viewpoint change). Still, our approach outperforms previous approaches when considering the average over small, medium and large viewpoint changes (solid lines at the left) and the gap quickly increases when considering more difficult examples (solid lines, right). See also Sec. E on the trade-off between distortion caused by compression and computational efficiency. Our approach can also be applied to small viewpoint changes and the generation of continuous, consistent trajectories; see Fig. 11. Note that the ability of previous approaches to synthesize small viewpoint changes well does not enable high-quality synthesis of even moderately long trajectories.

Transformer Variants Over the Course of Training Fig. 12 reports the negative log-likelihood (NLL) over the course of training on RealEstate and ACID, respectively. The models overfit to the training split of ACID early which makes training on ACID much quicker and thus allows us to perform multiple training runs of each variant with different initializations. This

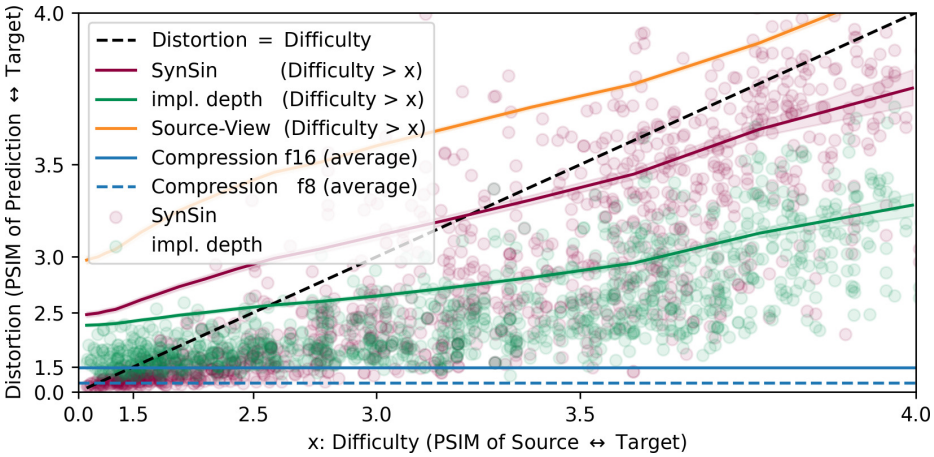

Figure 10. Comparison on small and large viewpoint changes.

Figure 11. Continuous trajectories: Ours (top) & SynSin (bottom).

enables an estimate of the significance of the results by computing the mean and standard deviation over three runs (solid line and shaded area in Fig. 12, respectively).

Additional Qualitative Results For convenience, we also include additional qualitative results directly in this supplementary. Fig. 15 and 17 show additional qualitative comparisons on RealEstate10K and ACID, as in Fig. 5 and 6 of the main paper. Fig. 16 and 18 demonstrate the diversity and consistency of samples by showing them along with their pixel-wise standard deviation. Fig. 19 contains results from the depth-probing experiment of Sec. 4.3, and Fig. 13 from the entropy visualization of Sec. 4.1.

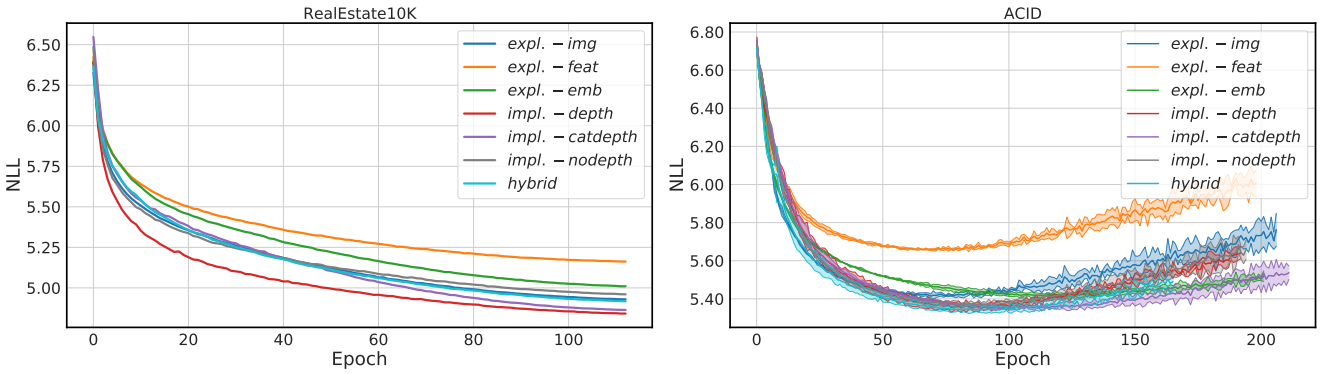

Figure 12. Negative log-likelihood over the course of training on RealEstate10K (left) and ACID (right). Implicit variants achieve the best results, see Sec. 4.1 for a discussion.

B. Architectures & Hyperparameters

Transformer The architecture of all transformer models discussed in this work follows the GPT-2 architecture [50]. More specifically, the transformer consists of L transformer blocks, where each block performs the following operation on an input sequence $z \in \mathbb{R}^{|z| \times d_e}$ (with $|z|$ the length of z):

$$z_1 = \text{LayerNorm}(z) \tag{15}$$
$$z_2 = \text{MultiHeadSelfAttention}(z_1) + z \tag{16}$$
$$z_3 = \text{LayerNorm}(z_2) \tag{17}$$
$$z_4 = \text{MLP}(z_3) + z_2 \tag{18}$$

In contrast to the global attention operation, the MLP is applied position-wise.

Given an input sequence s and an embedding produced by the conditioning function $f(x^{\text{src}}, T) \in \mathbb{R}^{n \times d_e}$ (see Sec. 3.2), the transformer maps s to a learnable embedding $e(s) + e^{\text{pos}} =: \hat{e}^0 \in \mathbb{R}^{|s| \times d_e}$, applies the L transformer blocks on the concatenated sequence $[f(x^{\text{src}}, T), \hat{e}^0]$ and finally projects to $|\mathcal{Z}|$ logits π^L via a linear transformation W_{head}, which correspond to a categorical distribution over sequence elements, *i.e.*

$$e^0 = [f(x^{\text{src}}, T), e(s) + e^{\text{pos}}] \tag{19}$$
$$e^l = \text{TransformerBlock}(e^{l-1}), \quad l = 1 \dots L \tag{20}$$
$$\pi^L = W_{\text{head}} \cdot \text{LayerNorm}(e^L). \tag{21}$$

Note that non-conditioning elements, *i.e.* the last $|s|$ elements, are masked autoregressively [70]. For all experiments, we use an embedding dimensionality $d_e = 1024$, $L = 32$ transformer blocks, 16 attention heads, two-layer MLPs with hidden dimensionalities of $4 \cdot d_e$ and a codebook of size $|\mathcal{Z}| = 16384$. This setting results in a transformer with $437M$ parameters. We train the model using the AdamW [41] optimizer (with $\beta_1 = 0.9$, $\beta_2 = 0.95$) and apply weight decay of 0.01 on non-embedding parameters. We train for $500k$ steps, where we first linearly increase the learning rate from $2.5 \cdot 10^{-6}$ to $1.5 \cdot 10^{-4}$ during the first $5k$ steps, and then apply a cosine-decay learning rate schedule [40] towards zero.

VQGAN The architecture and training procedure of the VQGANs is adopted from [15], where we use a downsampling factor of $f = 2^4$. For the codebook $\mathcal{Z}$, we use $|\mathcal{Z}| = 16384$ entries and a dimensionality of $d_z = 256$. This means that any input $x \in \mathbb{R}^{H \times W \times C}$ will be mapped to a latent representation of size $E(x) \in \mathbb{R}^{H/2^4 \times W/2^4 \times 256}$. For our experiments on RealEstate and ACID, where $H = 208$ and $W = 368$, this corresponds to a latent code of size 13×23 (which is then unrolled to a sequence of length $|s| = 299$). We use the authors' official implementation and pretrained models[2] and perform finetuning on frames of RealEstate and ACID for 50'000 steps on either dataset, resulting in two dataset-specific VQGANs.

[2] see https://github.com/CompVis/taming-transformers

Other models For monocular depth estimates, we use MiDaS v2.1[3]. We use the official implementations and pretrained models for the comparison with 3DPhoto [60][4], SynSin [72][5] and InfNat [38][6].

C. Training and Testing Data

Training our conditional generative model requires examples consisting of $(x^{\text{dst}}, x^{\text{src}}, T)$. Such training pairs can be obtained via SfM [78] applied to image sequences, which provides poses (R_i, t_i) for each frame x^i with respect to an arbitrary world coordinate system. For two frames $x^{\text{src}}, x^{\text{dst}}$ from the sequence, the relative transformation is then given by $R = R_{\text{dst}} R_{\text{src}}^{-1}$ and $t = t_{\text{dst}} - R t_{\text{src}}$. However, the scale of the camera translations obtained by SfM is also arbitrary, and without access to the full sequence, underspecified.

To train the model and to meaningfully compute reconstruction errors for the evaluation, we must resolve this ambiguity. To do this, we also triangulate a sparse set of points for each sequence using COLMAP [57]. We then compute a monocular depth estimate for each image using MiDaS [52] and compute the optimal affine scaling to align this depth estimate with the scale of the camera pose. Finally, we normalize depth and camera translation by the minimum depth estimate.

All qualitative and quantitative results are obtained on a subset of the test splits of RealEstate10K [78] and ACID [38], consisting of 564 source-target pairs, which have been selected to contain medium-forward, large-forward, medium-backward and large-backward camera motions in equal parts. We will make this split publicly available along with our code.

A Note on Reported Metrics

Since our stated goal is to model *large* camera transformations, our evaluation focuses on this ability and thus differs from the evaluation of SynSin, which is biased to small changes; see Tab. 4: With the SynSin evaluation (small cam-Δ) we reproduce the officially reported numbers and our choice to evaluate at the original aspect ratio (at 208×368) hass minor effects. The last three rows show the deterioration of metrics if we remove biases to small changes Δ: We (i) evaluate all test pairs, not just the better of two views (w/o best of 2), (ii) remove 10.14% of test pairs that contain no camera change at all (w/o src $\equiv$ tgt) and (iii) add 50% of larger viewpoint changes (w/ 50% large). The main paper reports results at 100% medium and large viewpoint changes, but we include an additional analysis with small changes in Fig. 10.

Model	PSIM ↓	PSNR ↑	SSIM ↑
SynSin (small cam-Δ)	1.13 ± 0.54	23.03 ± 4.54	0.77 ± 0.12
SynSin (at 208×368)	1.32 ± 0.53	22.39 ± 4.24	0.76 ± 0.12
SynSin (w/o best of 2)	1.72 ± 0.72	19.75 ± 4.69	0.67 ± 0.16
SynSin (w/o src $\equiv$ tgt)	1.84 ± 0.66	18.86 ± 3.98	0.64 ± 0.15
SynSin (w/ 50% large)	2.48 ± 0.91	16.05 ± 3.99	0.57 ± 0.13

Table 4. Effects of removing evaluation biases to small changes.

[3] see https://github.com/intel-isl/MiDaS
[4] see https://github.com/vt-vl-lab/3d-photo-inpainting
[5] see https://github.com/facebookresearch/synsin/
[6] see https://github.com/google-research/google-research/tree/master/infinite_nature

D. Details on Entropy Evaluation

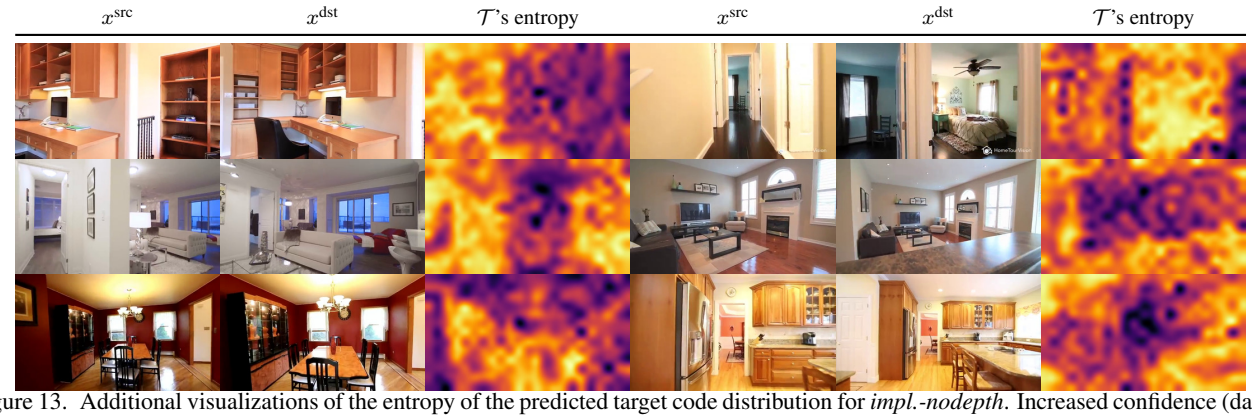

Figure 13. Additional visualizations of the entropy of the predicted target code distribution for *impl.-nodepth*. Increased confidence (darker colors) in regions which are visible in the source image indicate its ability to relate source and target geometrically, without 3D bias. See also Sec. 4.1 and Sec. D.

As discussed in Sec. 4.1, the relationship between a source view x^{src} and a target view x^{dst} can be quantified via the entropy of the probability distribution that the transformer assigns to a target view x^{dst}, given a source frame x^{src}, camera transformation T and conditioning function f. More specifically, we first encode target, camera and source via the encoder E and the conditioning function f (see Sec. 3.1), *i.e.* $s^{\text{dst}} = E(x^{\text{dst}})$ and $f(x^{\text{src}}, T)$. Next, for each element in the sequence s^{dst}, the (trained) transformer assigns a probability conditioned on the source and camera:

$$p\Big(s_i^{\text{dst}} | s_{<i}^{\text{dst}}, f(x^{\text{src}}, T)\Big), \quad 0 \leq i < |s^{\text{dst}}| \tag{22}$$

where for our experiments the length of the target sequence is always $|s^{\text{dst}}| = 13 \cdot 23 = 299$, see also Sec. B. The entropy $\mathbb{H}(s_i^{\text{dst}}, x^{\text{src}})$ for each position i is then computed as

$$\mathbb{H}(s_i^{\text{dst}}, x^{\text{src}})) = -\sum_{k}^{|\mathcal{Z}|} p_k(s_i^{\text{dst}} | s_{<i}^{\text{dst}}, f(x^{\text{src}}, T)) \log p_k(s_i^{\text{dst}} | s_{<i}^{\text{dst}}, f(x^{\text{src}}, T)) \tag{23}$$

Reshaping to the latent dimensionality $h \times w$ and bicubic upsampling to the input's size $H \times W$ then produces the visualizations of transformer entropy as in Fig. 4 and Fig. 13. Note that this approach quantifies the transformers uncertainty/surprise from a single example only and does not need to be evaluated on multiple examples.

E. Faithful Reconstructions/Compression Artifacts

Efficient training of the transformer models is enabled by the strong compression achieved with the VQGAN, which to some degree introduces artifacts but allows to trade compute requirements for reconstruction quality. Larger discrete codes improve fidelity (see Fig. 14, Tab. 5) but a $4\times$ larger code leads to approximately $16\times$ larger costs when training the transformer ($\mathcal{O}(n^2)$-complexity of attention). Reducing such artifacts is thus a matter of scaling up hardware or training time. Additionally, it would also increase the time required to sample a novel view, which is currently 4.32 ± 0.04 seconds.

Model	PSIM ↓	R-FID ↓	PSNR ↑	SSIM ↑
VQGAN f16	1.48 ± 0.37	2.46	21.14 ± 2.45	0.67 ± 0.14
VQGAN f8	0.80 ± 0.28	1.52	25.23 ± 2.96	0.82 ± 0.11

Table 5. Reconstruction metrics for different compressions.

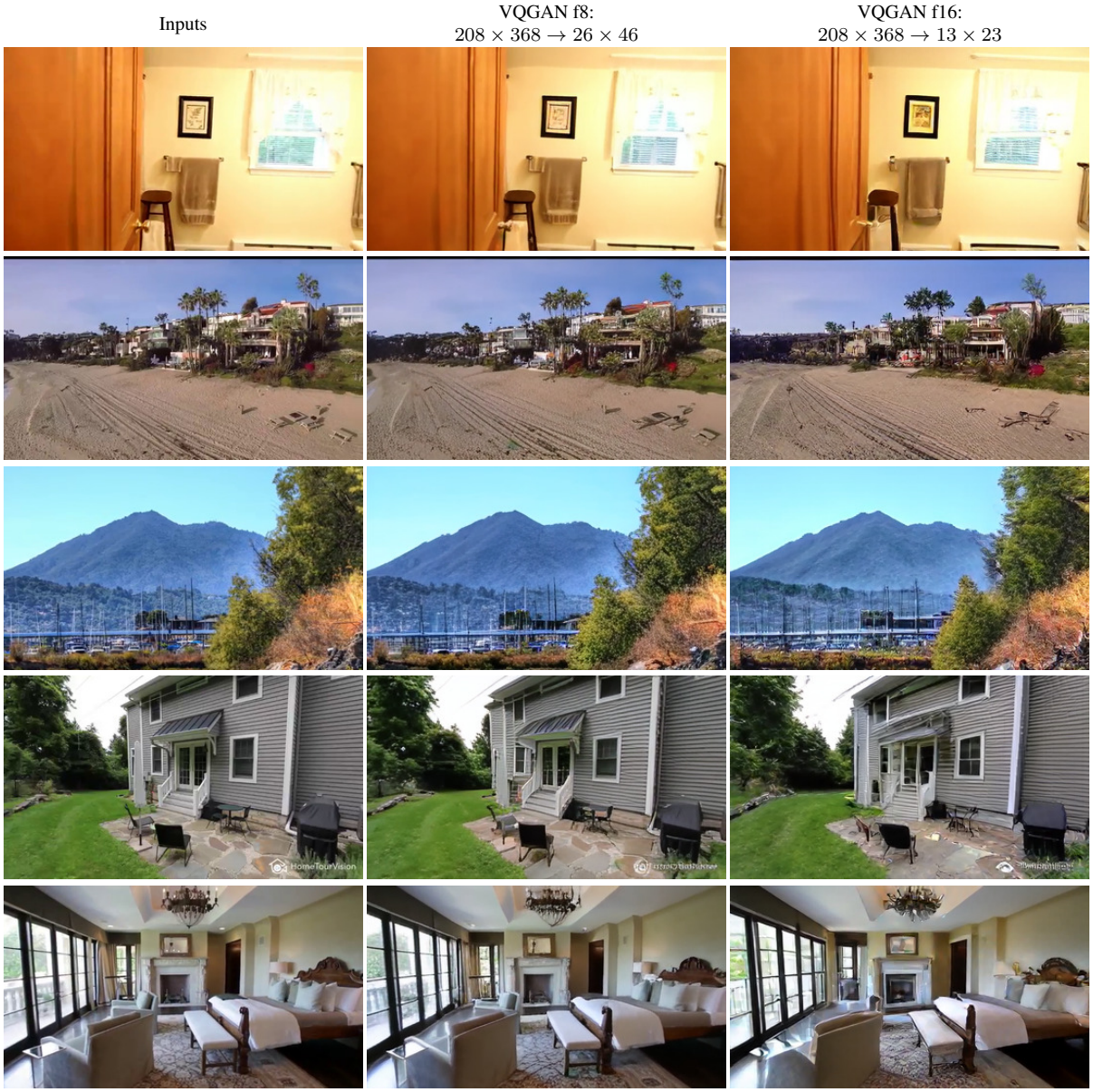

Figure 14. Qualitative visualization of different compression rates.

Figure 15. Additional qualitative comparisons on RealEstate10K.

Figure 16. Additional samples on RealEstate10K. The second column depicts the pixel-wise standard deviation σ obtained from $n = 32$ samples.

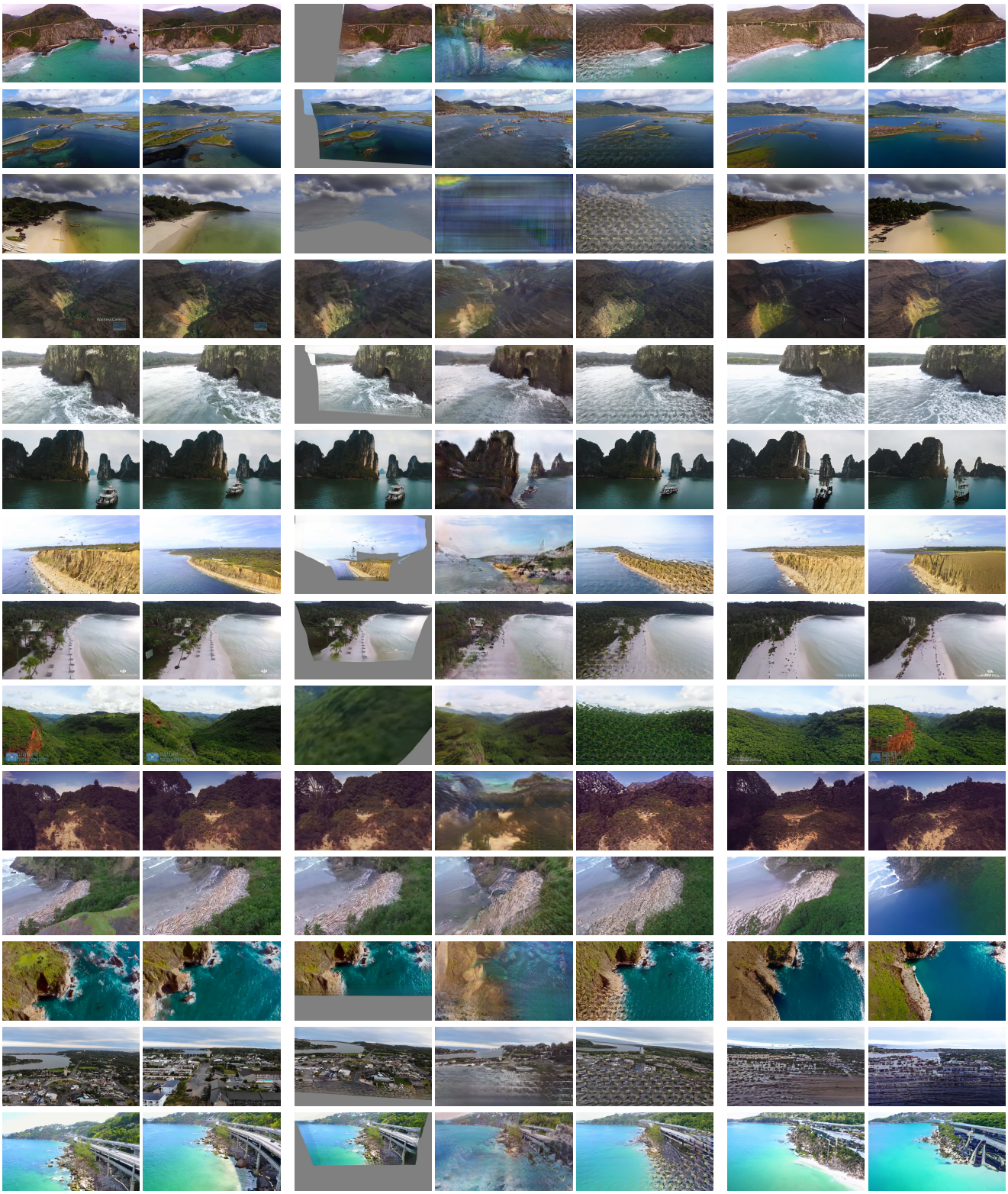

Figure 17. Additional qualitative comparisons on ACID.

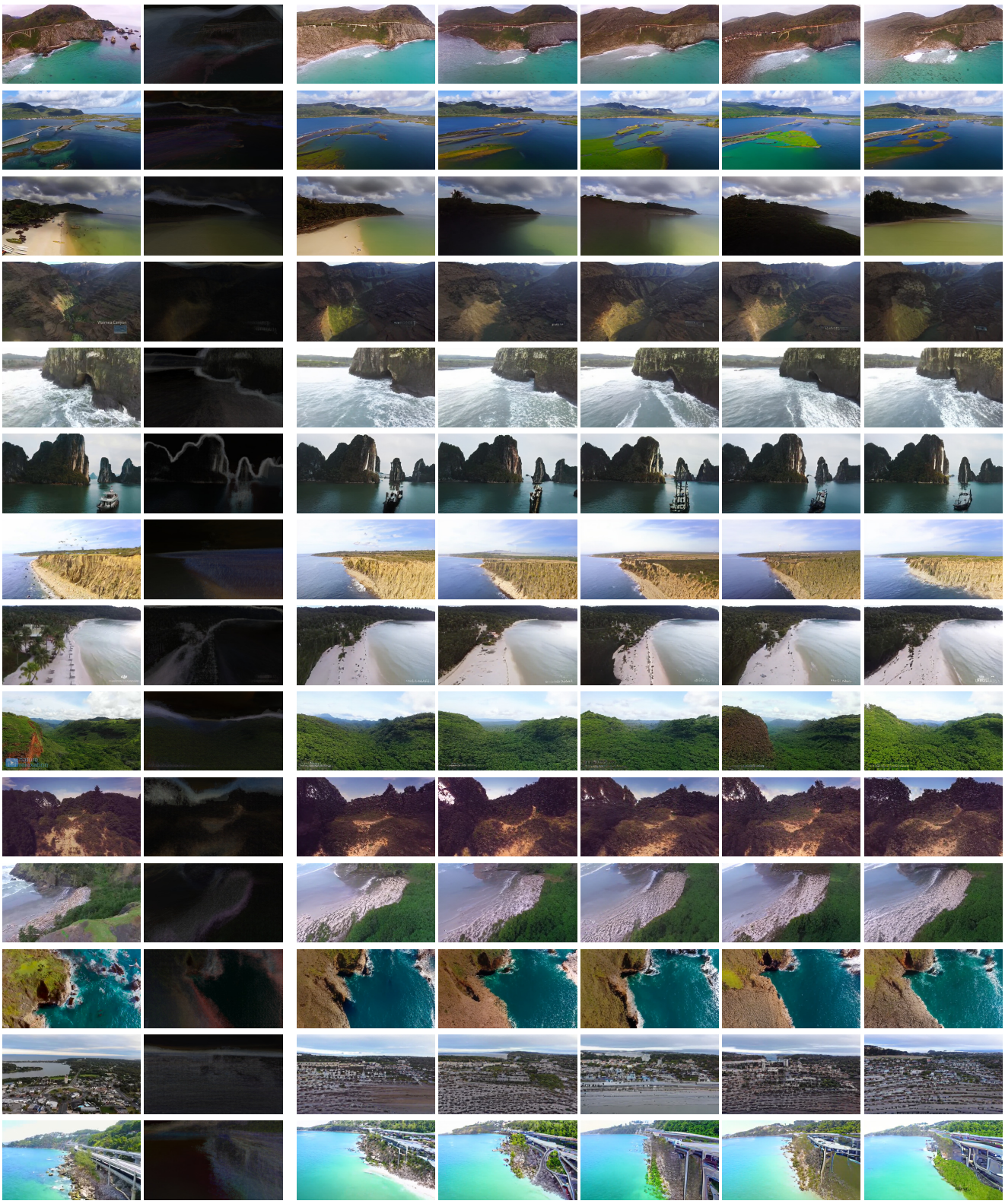

Figure 18. Additional samples on ACID. The second column depicts the pixel-wise standard deviation σ obtained from $n = 32$ samples.

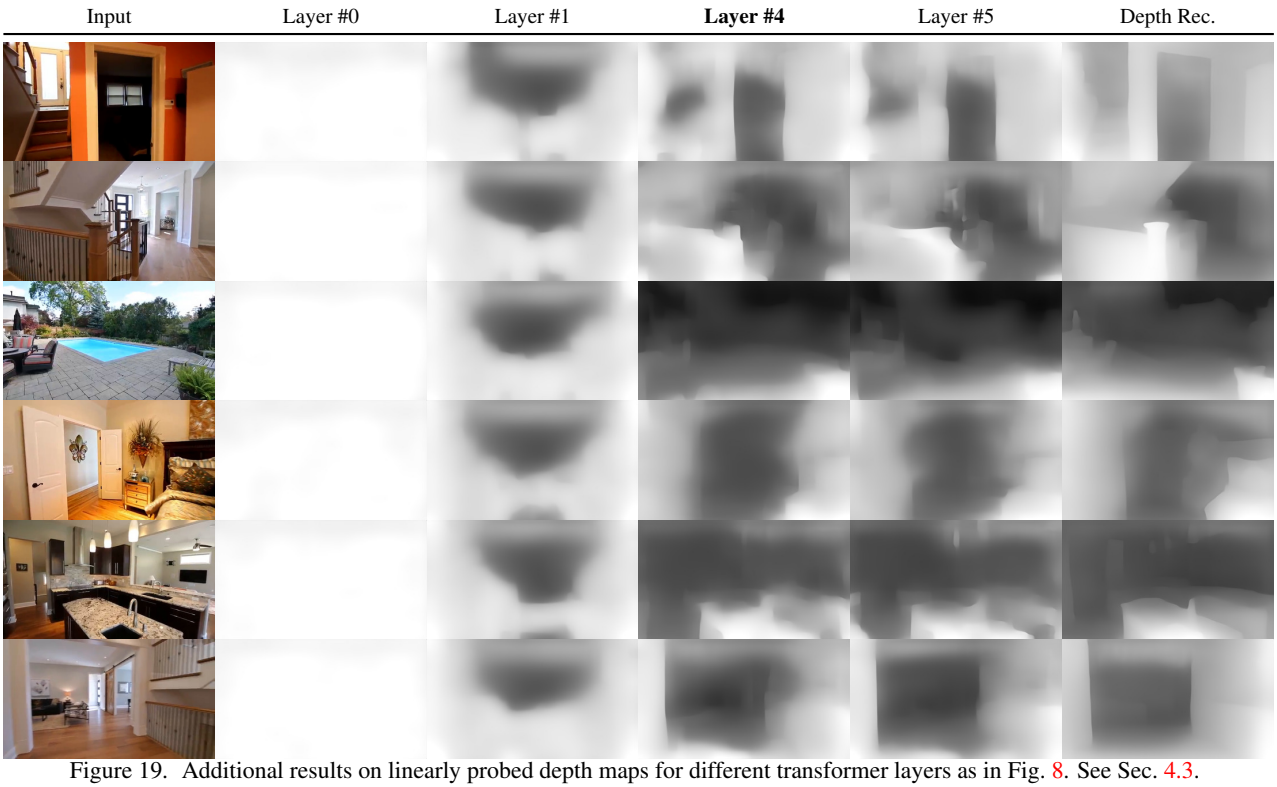

Figure 19. Additional results on linearly probed depth maps for different transformer layers as in Fig. 8. See Sec. 4.3.