SinGRAF: Learning a 3D Generative Radiance Field for a Single Scene Supplementary Material

Minjung Son*[1,2] Jeong Joon Park*[2] Leonidas Guibas[2] Gordon Wetzstein[2]
[1]Samsung Advanced Institute of Technology (SAIT) [2]Stanford University
minjungs.son@samsung.com, {jjpark3d,gordon.wetzstein,guibas}@stanford.edu

A. Video Results

We highly encourage readers to view our supplementary video containing visualizations of our 3D scenes, latent interpolations, and comparisons against GSN. The video results are best suited for appreciating the 3D consistency, quality, and diversity of our generated scenes. The name of the attached video file is "singraf_video.mp4."

B. Implementation Details

B.1. Progressive Patch Scaling

In practice, we used the input resolution of 512×512 for the input images ($H' = 512$) and used the fixed patch size of 64×64 ($H = 64$). When the scale $s = 1$ the 64×64 patch covers the entire image.

To progressively scale down the patches, we reduce the patch scale s for the first 100 epochs. Because we are using 1,000 sample batches for each epoch, we gradually reduce s during the course of 100,000 iterations and then fix s for the rest of the training. Typically our model exhibits the best KID score around 300 to 400 epochs.

We randomly sample the scale factor s independently for each image instance during training. We use a time-varying uniform distribution for the patch scale: $s \sim \mathrm{U}(s_{\min}(t), s_{\max}(t))$, where t is the epoch index. In practice, we used $s_{\min}(0) = 0.6$ and $s_{\max}(0) = 0.8$ in the beginning of the training and $s_{\min}(100) = 0.25$ and $s_{\max}(100) = 0.55$ at epoch 100. $s_{\min}(t)$ and $s_{\max}(t)$ values are interpolated linearly as a function of t over the course of 100 epochs.

B.2. Data Augmentation

As described in the main text, we schedule the increase of angles used for perspective augmentation during training. Similar to the progressive patch scaling, we gradually linearly increase the maximum augmentation angle for 100 epochs. We start from the angle range of $[0°, 0°]$ to $[-15°, 15°]$ linearly over the course of 100 epochs. For each real image patch we randomly and independently sample an angle from the current range and apply the perspective augmentation on the height axis. An example visualization of this augmentation could be seen in Fig. 1.

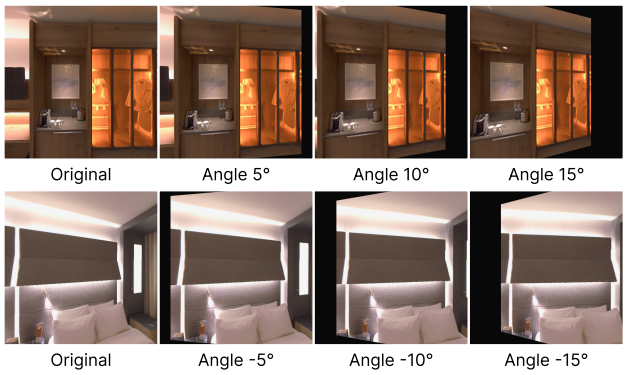

Figure 1. Visualizations of the perspective augmentation applied at different angles. During training, we apply up to $15°$ perturbation to promote diversity.

B.3. Camera Pose Optimization

As described in the main text, we non-parametrically optimize the camera pose distribution during training. Given a set of 1,000 camera poses: $\mathcal{T} = \{T_1, ..., T_{1000} | T_i \in SE(3)\}$. During training, we randomly sample camera poses from $\mathcal{T}$ and render the scenes from the cameras. We let these camera poses as optimizable variables and back-propagate the gradients from the adversarial loss to optimize the individual poses. Optimizing for the $SE(3)$ transformation is known to be a difficult task, and directly optimizing the values of the transformation matrix is difficult because the matrix can escape the manifold of $SE(3)$, e.g., $RR^\intercal \neq I$. Therefore, we decompose the matrix into multiple components for ease of optimization.

Specifically, given a transformation T, we can decompose it into:

$$T = \begin{pmatrix} R & p \\ 0 & 1 \end{pmatrix} : \; R \in SO(3), \, p \in \mathbb{R}^3. \tag{1}$$

The rotation matrix can be further decomposed into a multiplication of 3 matrices:

$$R = R_z R_y R_x : R_{\{\}} \in SO(3), \tag{2}$$

where R_z, R_y, and R_x are respectively rotation matrix about z, y, and x axis. For example, we have that:

$$R_z = \begin{pmatrix} \cos\theta_z & -\sin\theta_z & 0 \\ \sin\theta_z & \cos\theta_z & 0 \\ 0 & 0 & 1 \end{pmatrix} : \theta_z \in \mathbb{R}, \tag{3}$$

where θ_z is the rotation around the z axis. While it is possible to parameterize our rotations with θ_z, θ_y, and θ_x, directly optimizing for the Euler angles is known to be difficult [6], as there is a discontinuity at $\theta = 0$. Therefore, we parameterize each rotation with the cosine and sine of the Euler angle for each axis. Let us denote $\mathcal{R}_z$ to be the data structure we carry for rotation about the z axis:

$$\mathcal{R}_z := [\cos\theta_z, \sin\theta_z]. \tag{4}$$

We similarly define $\mathcal{R}_y$ and $\mathcal{R}_x$ using cosine and sine of the angles. Then, we know that we can uniquely construct the rotation matrix R from the $\mathcal{R}$'s. For the whole transformation, we carry the four data structures to fully describe and construct the matrix T, which are: $[\mathcal{R}_z, \mathcal{R}_y, \mathcal{R}_x, p]$. We can optimize for these variables during training by backpropagating the adversarial losses. Note that for each update of the rotation parameter $\mathcal{R}_{\{\}}$, we need to make sure that the cosines and sines are proper, by normalizing it so that $||\mathcal{R}_{\{\}}|| = 1$. In practice, we assume that cameras are located at the same height and rotate along vertical axis. Thus, we only optimize p_x, p_z and $\mathcal{R}_y$ during the early stages of training where the expected patch scale is larger than 0.5.

B.4. Training Details

The balancing parameters for the regularization terms in Eq.(5) of the main paper are $\lambda_1 = 0.5$ and $\lambda_2 = 50$. We use the spatial resolution of 256×256 and feature channel of $C = 32$ for tri-plane representation [1], which is generated by a StyleGAN2 [3] generator which is modulated by a noise vector $\mathbf{z} \sim \mathbb{R}^{128}$. For rendering, we compute the feature of each sample along a ray via bilinear interpolation followed by concatenation (for the three planes), and process the feature using a decoder MLP to finally obtain color and density value. The decoder MLP is composed of two shared linear layers, one layer for the density branch, and two additional layers for the RGB branch. Each hidden layer uses 64 hidden units with leakyReLU activation except the final ones used for outputting density and RGB values. For volume-rendering we used 96 samples per ray without importance sampling, to generate patches of 64×64 resolution. When applied the patch scale of $s = 0.25$, the effective resolution of each patch is 256×256, i.e., the amount of details that exists in each patch would be obtained when rendering at 256×256. Therefore, we can render our models at 256×256 resolution with details even though we trained our models with 64×64 patches, without any 2D upsampling networks. As described in the main text, we adopt the StyleGAN2 discriminator architecture to process the patches, but we additionally concatenate the scale of each patch. We will release the source code upon acceptance.

C. Additional Analysis

C.1. Dataset Selection

Training 3D GANs, including GSN and SinGRAF, takes a long time. With the finite computational resources at our disposal, it was simply not possible to run SinGRAF on all scenes of a large 3D dataset such as Matterport3D. Therefore, we chose a representative subset of the Replica dataset and further stress-test our method on wildly different scene examples of a ballroom of Matterport3D and a custom-captured outdoor scene. We provide results on one additional Matterport3D scene (Fig. 2; SinGRAF: KID **0.050** and Div. **0.447**; GSN: KID 0.087 and Div. 0.001). Similar to other scenes, GSN fails to produce diversity, while SinGRAF generates diverse and realistic scenes.

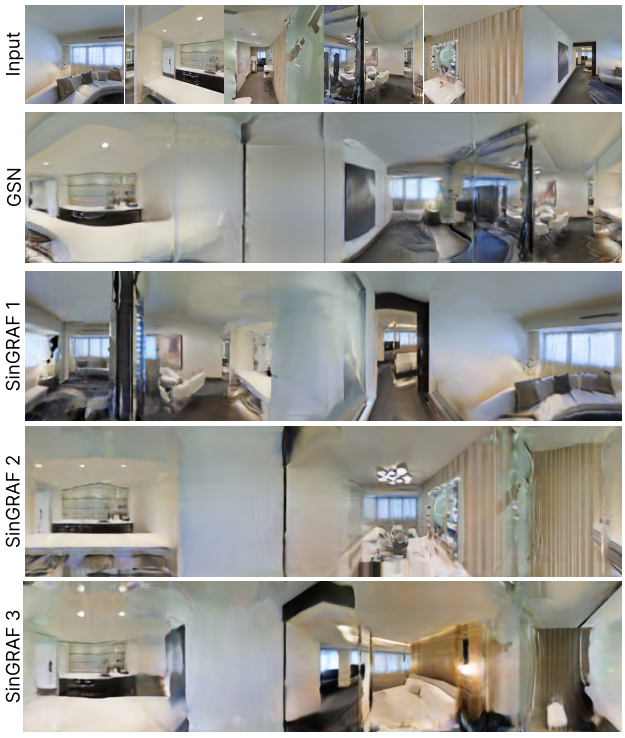

Figure 2. Additional Matterport3D scene.

C.2. Additional Empirical Analysis

Number of input images We conduct an empirical study to gauge the effects of varying the number of input images. As shown in Tab. 1, we observe that reducing the input image number degrades the quality and SinGRAF can generate diverse scenes for as few as 50 images. On the other hand, GSN fails to generate diversity for all ablated experiments due to mode collapse.

	GSN (128^2)		SinGRAF (128^2)		SinGRAF (256^2)		NeRF (256^2)	
#	KID↓	Div.↑	KID↓	Div.↑	KID↓	Div.↑	KID↓	Div.↑
100	.052	.001	**.037**	**.335**	**.055**	**.408**	.277	0.0
50	.113	.001	**.071**	**.362**	**.104**	**.426**	.269	0.0
10	.238	.001	**.130**	**.002**	**.159**	**.005**	.263	0.0

Table 1. Ablation over the number of input images using the "frl_apartment_4" scene.

Comparison with NeRF variants While NeRF and their variants are used for obtaining 3D structure of a specific scene rather than generating diversity, we believe that NeRF variants are interesting baselines to put the realism of our generated scenes in context. To this end, we train Instant-NGP [4] and measure KID of generated views at held-out poses on a Replica scene. To make the comparison fair, we do not use the ground truth camera poses but instead estimate the camera poses using an off-the-shelf structure-from-motion library, i.e., COLMAP [5]. As shown in the numerical results on Tab. 1, due to the small number of training views, the test view images of the NeRF variant are of very low quality.

Comparison with EG3D We trained a EG3D [1] baseline model for "frl_apartment_4" to show an additional comparison against a 3D-GAN algorithm. As expected, it fails to learn diversity (KID 0.078, Div. 0.008) without our continuous-scale patch discrimination.

Discriminator scale conditioning As described in the main text, we condition our discriminator with the patch scale. We conduct an experiment to test of effect of the scale conditioning, which shows that without discriminator scale conditioning, the KID of the ablated SinGRAF on "frl_apartment_4" is 0.070 (worse quality) with a similar diversity score of 0.341 compared to our model with the scale conditioning (KID 0.037, Div. 0.335).

Depth map visualization To test the validty of the 3D structure generated by SinGRAF, we show example depth maps in Fig. 3.

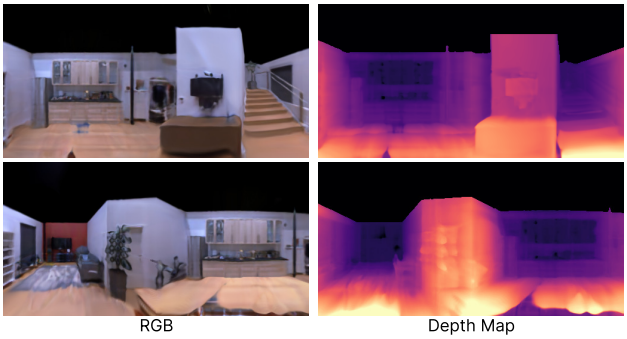

Figure 3. Depth map visualization ("frl_apartment_dynamic").

C.3. Note on the GSN comparison

We trained GSN [2] using the exact same settings used in their paper and published code except that we did not use the depth maps. The random poses are sampled from a 10K pose basket and jittered.

To make the comparison fair against our setup using the 1K pose basket, we tried reducing the size of the random pose distribution from 10K to 1K for GSN on "frl_apartment_4" but did not see any difference in quality or diversity (KID 0.055, Div. 0.001). Moreover, we tried GSN training with our pose sampling method, but it fails to converge (KID 0.599, Div. 0.0). The total number of parameters of GSN (25.4M) is larger than ours (20.7M). We tried varying GSN parameters, discriminator resolution, and pose sampling, but couldn't achieve diversity.

C.4. Visualization of the Diversity Metric

As discussed in the main text, we measure the diversity of the 3D generative models by fixing a camera and rendering with randomly sampled latent codes. In Fig. 4, we show examples of such renderings. Note how we can tell that GSN's model has collapsed to a single mode, thus no variations from the fixed viewpoints. On the other hand, notice how SinGRAF generates highly diverse renderings of the same scenes from the fixed camera with varying latent.

C.5. Latent Code Interpolation

To showcase the rich and smooth latent space we learn via training SinGRAF on single scenes, we visualize the latent space interpolation by fixing a camera and rendering the scene using latent vectors obtained via linear interpolation between two latent vectors. The results are shown in Fig. 6, demonstrating high-quality, and diverse latent embedding of the single scenes. We highly encourage readers to view our video results for animated interpolation in the latent space.

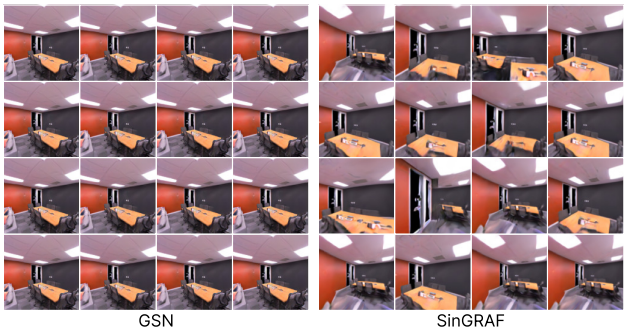

Figure 4. Visualizing randomly sampled 3D scenes from a fixed camera view. Notice how GSN's scenes do not change with the varying latent, indicating mode collapse, while SinGRAF presents highly diverse renderings.

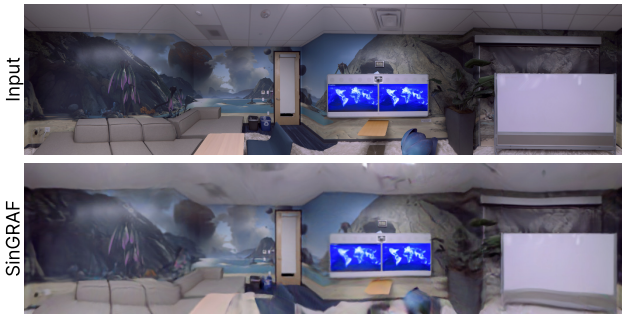

Figure 5. Failure case of the "office_0" scene. The lack of diversity in this scene is likely due to the paintings on the walls that uniquely determine the relative locations of most patches. Still, SinGRAF generates high-quality scene which resemble to input.

C.6. Perspective Rendering Results

In the main text, we have only visualized our scenes in the panorama form. In Fig. 7, we show renderings of the scenes from randomly chosen latent codes and camera poses using a perspective camera model. Note that we sampled the camera poses using the distribution $\mathcal{T}$, which is the result of the optimization process described in Sec. B.3 during training.

C.7. Failure Case

We observe that when the scene contains too many local details to be identified from small field-of-view patches, SinGRAF often learns a mode-collapsed latent space. Such an example can be found in Fig. 5, where the scene contains detailed paintings on the wall that uniquely determine the locations of the patches. We note that, while this mode-collapse behavior is unpredictable and thus is a limitation of our approach, the reconstructed scene closely resembles the ground truth scene of the input images. This is surprising, given that our model is given only *unposed* images, suggesting a promising future direction toward reconstructing challenging scenes via adversarial training.

References

[1] Eric R. Chan, Connor Z. Lin, Matthew A. Chan, Koki Nagano, Boxiao Pan, Shalini De Mello, Orazio Gallo, Leonidas Guibas, Jonathan Tremblay, Sameh Khamis, Tero Karras, and Gordon Wetzstein. Efficient geometry-aware 3D generative adversarial networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2022. 2, 3

[2] Terrance DeVries, Miguel Angel Bautista, Nitish Srivastava, Graham W. Taylor, and Joshua M. Susskind. Unconstrained scene generation with locally conditioned radiance fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 14304–14313, October 2021. 3

[3] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of StyleGAN. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 8110–8119, 2020. 2

[4] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Transactions on Graphics (ToG)*, 41(4):1–15, 2022. 3

[5] Johannes L Schonberger and Jan-Michael Frahm. Structure-from-motion revisited. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4104–4113, 2016. 3

[6] Yi Zhou, Connelly Barnes, Jingwan Lu, Jimei Yang, and Hao Li. On the continuity of rotation representations in neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5745–5753, 2019. 2

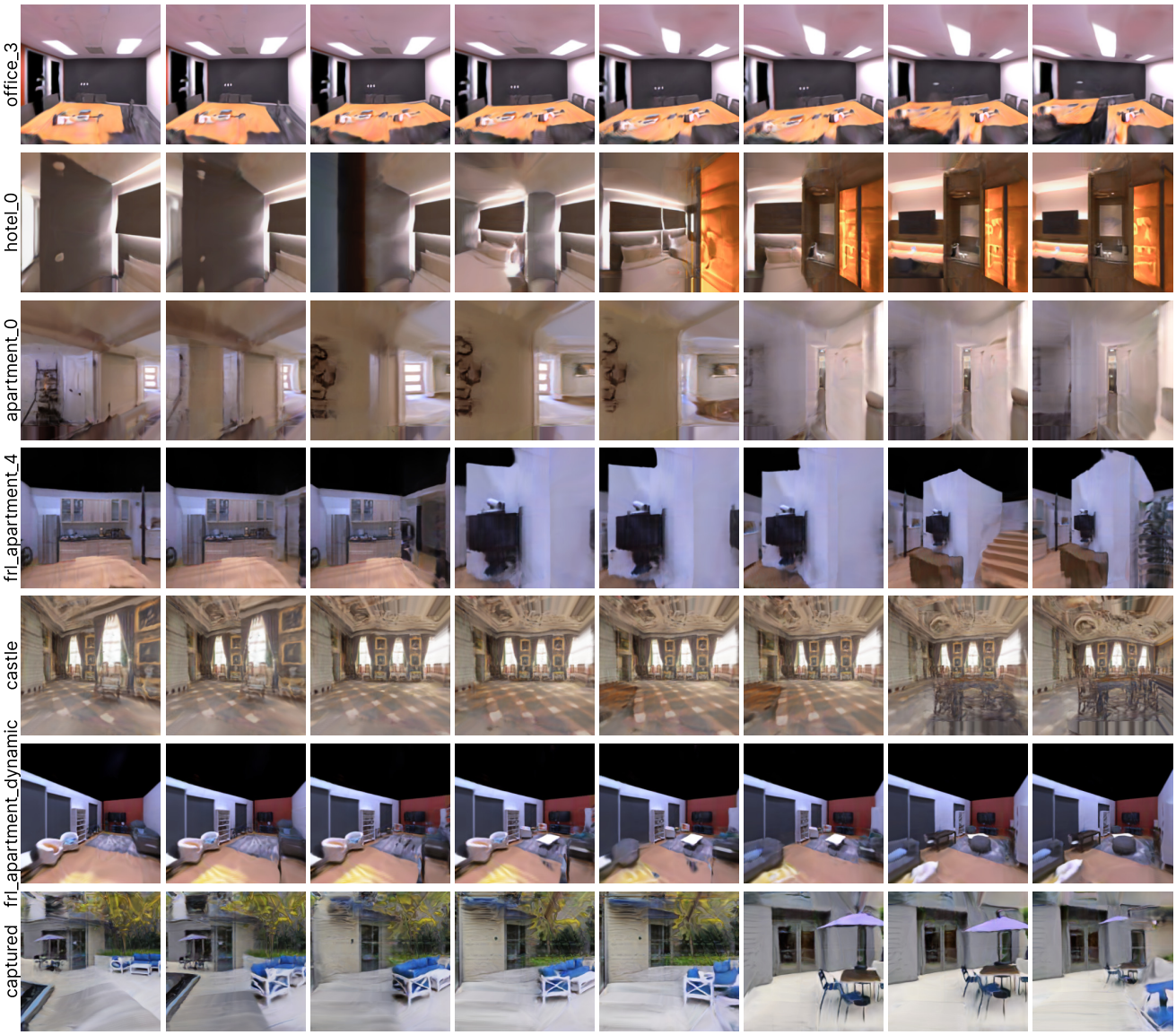

Figure 6. Latent code interpolations. We fix the camera viewpoint per scene and render the scene using latent vectors interpolating between two latent vectors, whose scenes are shown on the two extreme sides.

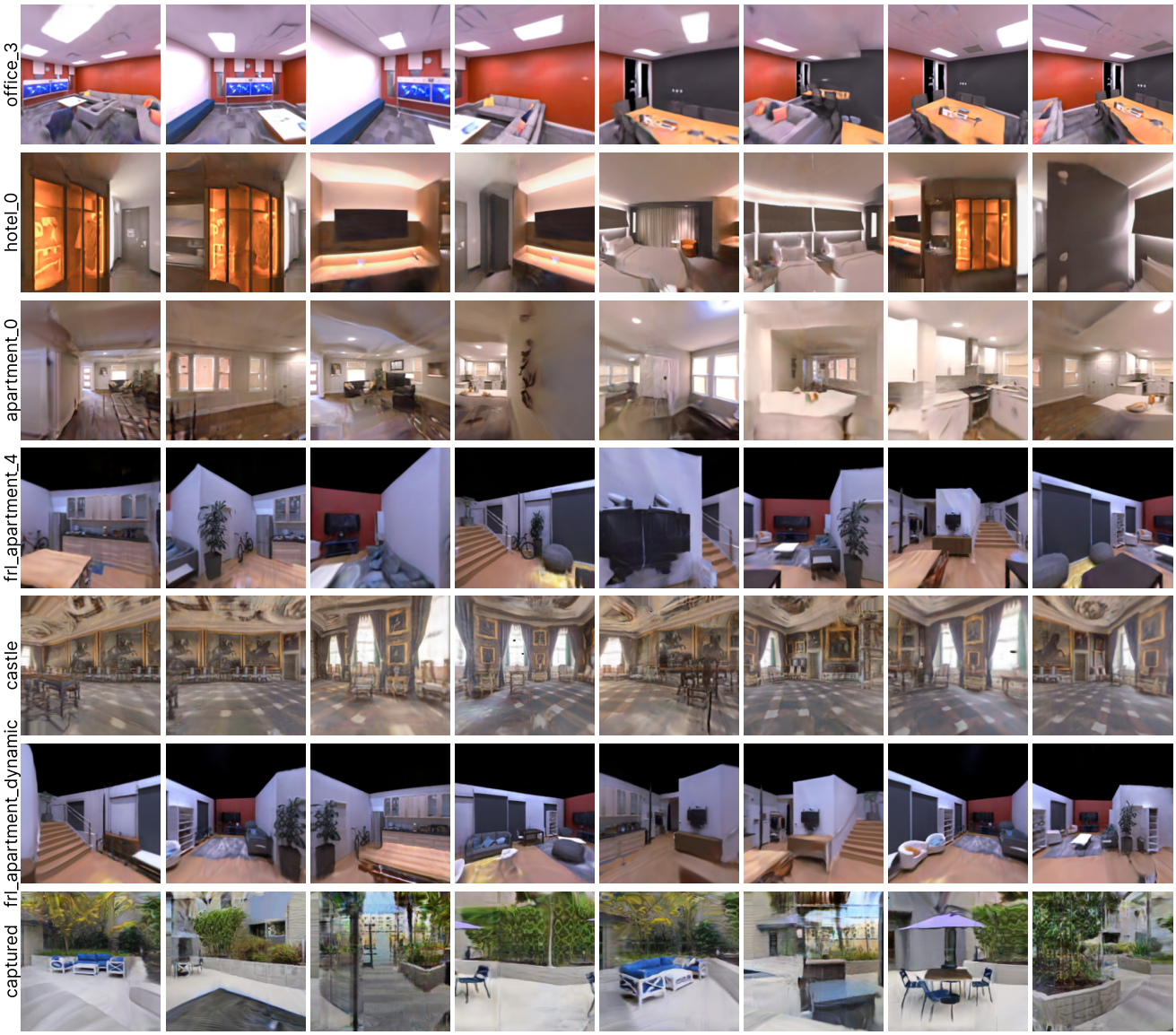

Figure 7. Perspective rendering results. We show perspective renderings of our trained scenes from randomly chosen latent codes and camera poses.