Supplementary Material for: Unconstrained Scene Generation with Locally Conditioned Radiance Fields

1. Model Architectures and Training Details

In this section we summarize the model architectures, hyperparameter settings, and other training details used for producing the GSN models presented in this paper.

1.1. Mapping Network

The mapping network maps the global latent code $\mathbf{z}$ to an intermediate non-linear latent space [8]. All models in our experiments, including those that do not use the global generator, use a mapping network which consists of a normalization step followed by three linear layers with LeakyReLU activations [10], as shown in Tab. 1.

	Activation	Output Shape
Input $\mathbf{z}$	–	128
Normalize	–	128
Linear	LeakyReLU (0.2)	128
Linear	LeakyReLU (0.2)	128
Linear	LeakyReLU (0.2)	128

Table 1: Mapping network architecture.

1.2. Global Generator

The purpose of the global generator (Tab. 2) is to map from a single global latent code $\mathbf{z}$ to a 2D grid of local latent codes $\mathbf{W}$ which represent the spatial layout of the scene. The global generator is composed of successive modulated convolutional layers [9] that are conditioned on the global latent code. Following StyleGAN [8], the model learns a constant input for the first layer. The first layers in every pair of modulated convolutional layers thereafter upsamples the feature map resolution by $2\times$, which is implemented as a transposed convolution with a stride of 2, followed by bilinear filtering [19].

The output resolution of the global generator (which is also the spatial resolution of $\mathbf{W}$) is a hyperparameter which effectively controls the size of the spatial region represented by each individual local latent code. We set the global generator output resolution to 32×32 for all experiments.

	Activation	Output Shape
Constant Input	–	$256 \times 4 \times 4$
ModulatedConv (3×3)	LeakyReLU (0.2)	$256 \times 8 \times 8$
ModulatedConv (3×3)	LeakyReLU (0.2)	$256 \times 8 \times 8$
ModulatedConv (3×3)	LeakyReLU (0.2)	$256 \times 16 \times 16$
ModulatedConv (3×3)	LeakyReLU (0.2)	$256 \times 16 \times 16$
ModulatedConv (3×3)	LeakyReLU (0.2)	$256 \times 32 \times 32$
ModulatedConv (3×3)	LeakyReLU (0.2)	$256 \times 32 \times 32$
ModulatedConv (3×3)	–	$32 \times 32 \times 32$

Table 2: Global generator architecture.

1.3. From Global to Local Coordinate System

In this section we describe how to express 3D points $\mathbf{p}$ in the local coordinate system respective to their corresponding latent code $\mathbf{w}_{ij}$ in the latent grid $\mathbf{W} \in \mathbb{R}^{c \times s \times s}$. We start by defining a global coordinate system so that it's origin lies at the center of $\mathbf{W}$. This global coordinate system is scaled to $[-1, +1]$, where the limits are set to encompass the max width, height and depth of trajectories on a dataset. In this global coordinate system, $u = \frac{2}{s}$ is size of a cell in $\mathbf{W}$ where s is the spatial dimension of $\mathbf{W}$. Given a point $\mathbf{p} = [p_x, p_y, p_z]$ in the global coordinate system we compute $\mathbf{p}' = [p'_x, p'_y, p'_z]$ as follows:

$$p'_x = p_x \mod u \tag{1}$$
$$p'_y = p_y \tag{2}$$
$$p'_z = p_z \mod u \tag{3}$$

Finally, after the modulo operation we scale $\mathbf{p}'$ to $[-1, +1]$.

1.4. Local Generator

The local generator is composed of a locally conditioned radiance field network which maps coordinates and view direction to appearance a and occupancy σ, a volumetric rendering step which accumulates along sampled rays to convert a and σ values to feature vectors, and a refinement network which upsamples feature maps to higher resolution RGB images.

The locally conditioned radiance field network (Fig. 1) mimics the architecture of the the original NeRF network [11]. To condition the network such that it can represent many different radiance fields we swap out the fixed linear layers for modulated linear layers similar to those used in CIPS [1], where each modulated linear layer is conditioned on $\mathbf{w_{ij}}$. Each modulated linear layer has 128 channels.

When performing volumetric rendering we threshold occupancy values σ with a softplus as in D-NeRF [14] as opposed to the standard ReLU, as we find it leads to more stable training. For all experiments we sample 64 samples per ray. When generating 64×64 images we sample the radiance field network to produce feature maps at 32×32 resolution, and when generating 128×128 resolution images we sample feature maps at 64×64 resolution.

Once volumetric rendering has been performed we upsample the resulting feature map with refinement blocks (Fig. 3) until the desired resolution is achieved, then apply a sigmoid to bound the final output, as in GIRAFFE [13]. In general, we found that sampling higher resolution feature maps directly from the radiance field produced higher quality results compared to sampling at low resolution and applying many refinement blocks, but the computational cost is significantly higher.

1.5. Discriminator

The discriminator is based on the architecture used in StyleGAN2[9] (Tab. 3), including residual blocks (Fig. 3) and minibatch standard deviation layer [7]. When including depth information as input to the discriminator we normalize it to $[0, 1]$. In the case that the radiance field network is sampled at a resolution lower than the final output resolution (such as when using the refinement network), then resulting depth maps will have lower resolution than real examples. To prevent the discriminator from using this difference in detail to differentiate real and fake samples we downsample all real depth maps to match the resolution of the generated depth maps, then upsample them both back to full resolution.

The decoder (Tab. 4) takes as input 4×4 resolution feature maps from the discriminator (before the minibatch standard deviation layer), and applies successive transposed convolutions with a stride of 2 and bilinear filtering [19] to upsample the input until the original resolution is recovered.

1.6. Sampling Camera Poses

We use poses from camera trajectories in the training set as candidate poses during sampling, as real camera poses better reflect the true distribution of occupable locations compared to uniformly sampling over the entire scene region. Sampled camera poses are normalized and expressed relative to the camera pose in the middle of the trajectory.

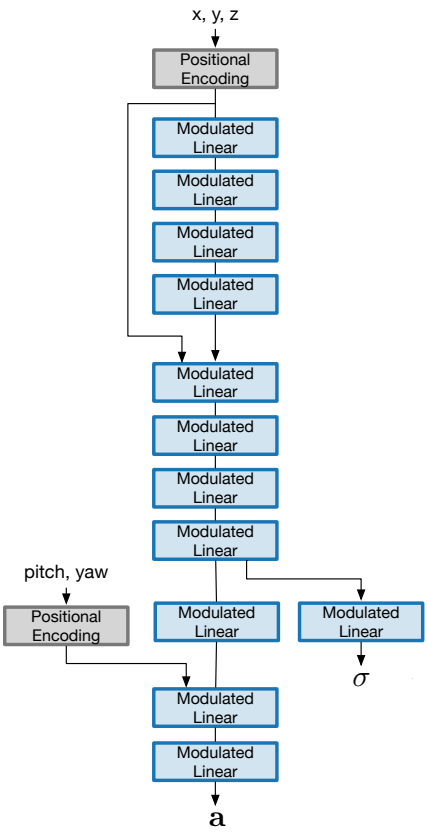

Figure 1: Locally conditioned radiance field network.

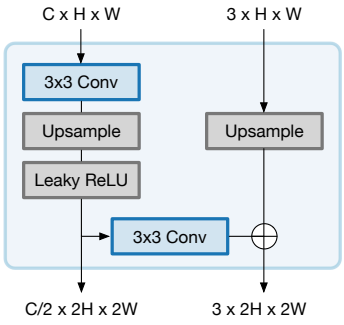

Figure 2: Refinement block. Feature maps pass through the left path, while RGB images pass through the right path. The input to the right path is not applied for the first refinement block after the volumetric rendering step.

This normalization enforces an egocentric coordinate system whose origin is placed at the center of $\mathbf{W}$. Note that despite working with trajectories of multiple camera poses, we still only sample a single camera pose per generated scene during training.

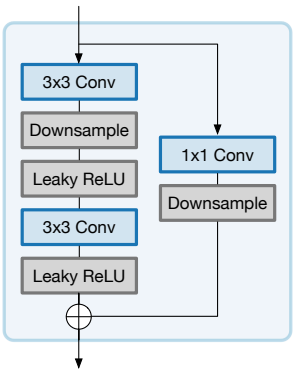

Figure 3: Residual block.

	Activation	Output Shape
Input RGB-D	–	$4 \times 64 \times 64$
Conv (3×3)	LeakyReLU (0.2)	$64 \times 64 \times 64$
Residual Block	LeakyReLU (0.2)	$128 \times 32 \times 32$
Residual Block	LeakyReLU (0.2)	$256 \times 16 \times 16$
Residual Block	LeakyReLU (0.2)	$512 \times 8 \times 8$
Residual Block	LeakyReLU (0.2)	$512 \times 4 \times 4$
Minibatch stdev	–	$513 \times 4 \times 4$
Conv (3×3)	LeakyReLU (0.2)	$512 \times 4 \times 4$
Flatten	–	8192
Linear	LeakyReLU (0.2)	512
Linear	–	1

Table 3: Discriminator architecture.

	Activation	Output Shape
Input Feature Map	–	$512 \times 4 \times 4$
ConvTranspose (3×3)	LeakyReLU (0.2)	$256 \times 8 \times 8$
ConvTranspose (3×3)	LeakyReLU (0.2)	$128 \times 16 \times 16$
ConvTranspose (3×3)	LeakyReLU (0.2)	$64 \times 32 \times 32$
ConvTranspose (3×3)	LeakyReLU (0.2)	$32 \times 64 \times 64$
Conv (3×3)	–	$4 \times 64 \times 64$

Table 4: Decoder architecture

1.7. Training Details

We use the RMSprop [5] optimizer with $\alpha = 0.99$, $\epsilon = 10^{-8}$, and a learning rate of 0.002 for both the generator and discriminator. Following StyleGAN [8], we set the learning rate of the mapping network $100\times$ less than the rest of the network for improved training stability. Equalized learning rate [7] is used for all learnable parameters, and an exponential moving average of the generator weights [7] with a decay of 0.999 is used to stabilize test-time performance. Differentiable data augmentations [21] such as random translation, colour jitter, and Cutout [4] are applied to all inputs to the discriminator in order to combat overfitting. To save compute, the R1 gradient penalty is applied using a lazy regularization strategy [9] by applying it every 16 iterations. We set λ_{R1} to 0.01 and λ_{Recon} to 1000 for all experiments.

All 64×64 resolution models used for the generation performance evaluation (GSN and otherwise) were trained for 500k iterations with a batch size of 32. Training takes 4 days on two NVIDIA A100 GPUs with 40GB of memory each. Mixed precision training is applied to the generator for a small reduction in memory cost and training time. We do not apply mixed precision training to the discriminator as training stability decreases in this case.

2. Inverting GSN for View Synthesis

In order for GSN to deal with the view synthesis problem we follow common practices for GAN inversion [18], adopting a hybrid inversion approach where we first train an encoder $E_{\theta_E} : \mathbb{R}^{3\times w\times h} \times SE(3) \longrightarrow \mathbb{R}^{c\times s\times s\times s}$ on $\{(\hat{\mathbf{X}}, \mathbf{T}, \mathbf{W})_i\}_{i=1:n}$ tuples sampled from a trained GSN (trained on the training set of the same dataset). The goal of this encoder is to predict an initial grid of latent codes $\mathbf{W}_0$ given a set of posed views. Our encoder is conceptually similar to [6, 12] where views $\hat{\mathbf{X}}$ are first processed with a backbone (UNet [15] with a ResNet-50 encoder in our case) and the resulting feature maps are back-projected using camera poses $\mathbf{T}$ into a shared feature volume $\mathbf{V} \in \mathbb{R}^{c\times s\times s\times s}$. Finally, we perform average pooling over the height dimension of $\mathbf{V}$ to get $\mathbf{W}_0 \in \mathbb{R}^{c\times s\times s}$. We train the encoder by minimizing the following reconstruction loss:

$$\mathcal{L}(\hat{\mathbf{X}}, \mathbf{T}, \mathbf{W}; \theta_E) = \|\mathbf{W} - E_{\theta_E}(\hat{\mathbf{X}}, \mathbf{T})\|_2 + \quad (4)$$

$$+\|\hat{\mathbf{X}} - f(E_{\theta_E}(\hat{\mathbf{X}}, \mathbf{T}), \mathbf{T})\|_2, \quad (5)$$

where the first term encourages the reconstruction of the local latent grid, and the second term encourages samples from locally conditioned radiance field f to match the original input views.

At inference time, given a trained encoder E_{θ_E}, we feed the source views $\mathcal{S} = \{(\mathbf{X}, \mathbf{T})_i\}_{i=t-5:t}$ through our encoder to predict an initialization latent code grid $\mathbf{W}_0$ that we then optimize via SGD for 1000 iterations to get $\hat{\mathbf{W}}$. Given that scenes do not share a canonical orientation, we predict $\mathbf{W}_0$ at multiple rotation angles of $\{\mathbf{T}_i\}_{i=t-5:t}$ about the $y-$axis to find the generator's preferred orientation and use this orientation during optimization (note that relative transformations between camera poses do not change with this global transformation). We define the preferred orientation as the one that minimizes an auto-encoding LPIPS loss [20]. The optimization process is performed by freezing the weights of f_{θ_f} and computing a reconstruction loss w.r.t. $\mathcal{S}$. We then use $\hat{\mathbf{W}}$ in the locally conditioned radiance field and render observations using the camera poses of $\mathcal{S}$ to produce $\hat{\mathcal{S}}$ (*i.e.* to auto-encode source views), while also rendering from the

camera poses of the target views $\mathcal{T}$ to produce $\hat{\mathcal{T}}$ (*i.e.* to predict unseen parts of the scene). Future work will explore in depth how to improve the quality and efficiency of the inversion approach for GSN-based models where the generative model tends to prefer a certain orientation. In Fig. 4 we show qualitative results for view synthesis on Vizdoom on held out sequences not seen during training. We can see how GSN learns a robust prior that is able to fill in the scene with plausible completions (*e.g.* row 5 and 8), even if those completions do not strictly minimize the L1 reconstruction loss.

In addition, Fig. 5 shows qualitative view synthesis results on the Replica dataset [17], showing the applicability of GSN for view synthesis on realistic data. In this experiment we follow the settings described for Vizdoom in terms of $\mathcal{S}$ and $\mathcal{T}$. In Fig. 5 the top 3 rows show results on data from the training set (*e.g.* scenes that were observed during training) and the bottom 3 rows show test set results (*e.g.* results on unseen scenes). We can see how GSN successfully uses the prior learned from training data to find a plausible scene completion for unseen scenes that respects the global scene geometry.

3. Qualitative Results on Local vs. Global Coordinate Systems

In this section we demonstrate the robustness of GSN w.r.t. re-arrangement of the latent codes in $\mathbf{W}$. In order to do so we sample different scenes from our learned prior and apply a rigid transformation to their corresponding $\mathbf{W}$ (a 2D rotation). In principle, this rotation should amount to a rotation of the scene represented by $\mathbf{W}$ that does not change the radiance field prediction. To qualitatively evaluate this effect we sample different scenes and rotate their corresponding $\mathbf{W}$ by $\{0, 90, 180, 270\}$ degrees while (i) rotating the camera by the same amount about the $y-$axis so that the rendered image should remain constant and (ii) keeping the camera fixed so that the scene should rotate. In Fig. 6-7 we show the result of the setting in (i) for a local and global coordinate system respectively. In these results we see how a local coordinate system is drastically more robust to re-arrangements of the latent codes than a global coordinate system. In addition, we show results for the setting in (ii) in Fig. 8-9 for local and global coordinate systems respectively. In this case, we see how given a fixed camera, a rotation of $\mathbf{W}$ amounts to rotating the scene. In this case we can also see how a local coordinate system results in higher rendering quality compared to that of a global coordinate system, which suffers from degradation as the rotation angle increases.

4. Scene Editing

A nice property of the local latent grid produced by GSN is that it can be used to perform scene editing by directly altering $\mathbf{W}$. This property allows us a degree of manual control for scene synthesis beyond what we get from randomly sampling the generator. While the low resolution of $\mathbf{W}$ used in current models currently limits us to high level scene modifications, training with larger local latent grids could allow for more fine-grained control over scenes, such as rearrangement of furniture.

We find that, as with most image composition operations, the results of scene editing appear most convincing when the inputs are well aligned in terms of appearance and geometry. We demonstrate editing operations by manipulating the codes from single scenes, since we don't need to worry about matching appearance and geometry, but multiple scenes could be combined if they were similar enough. In Fig. 10-11 we manipulate the local latent codes by mirroring them along the horizontal axis to produce unique scenes.

5. Effect of Local Latent Grid Size

In Tab. 5 we investigate the effect of changing local latent grid resolution. Changes in latent resolution are achieved by adding or removing layers from the global generator. In this setting a grid size of 1×1 is equivalent to GSN + global latents. All models were trained for 125k iterations. We observe that performance improves as the size of the latent grid increases, albeit with diminishing returns.

Grid size	1×1	8×8	16×16	32×32
FID	100.7	85.3	80.1	78.3

Table 5: Generation results as a function of grid size. Grid size of 1×1 is equal to GSN+global latent.

6. Implicit Upsampling

A nice property of implicit scene representations (such as NeRF) is the ability to render images at arbitrary resolutions, even those larger than observed in training. We demonstrate this property in GSN by generating 256×256 resolution images from a model trained at 128×128 resolution (Fig. 12). The implicit representation excels at preserving sharp edges, however, it seems incapable of filling in the accompanying textural details as it has never seen images at the higher resolution before. As a result, the final higher resolution outputs may appear somewhat flat in their textures.

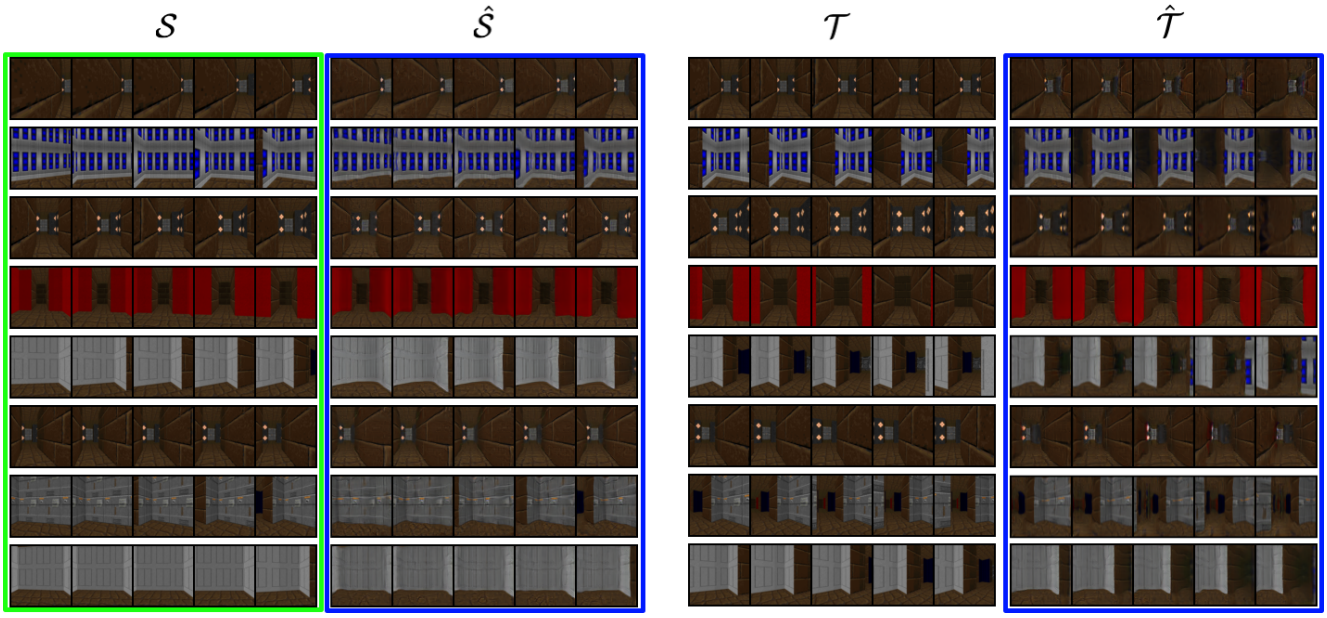

Figure 4: Qualitative view synthesis results on Vizdoom sequences not seen during training. Given source views $\mathcal{S}$ we invert GSN to obtain a local latent code grid $\hat{\mathbf{W}}$, which is then use both to reconstruct $\mathcal{S}$, denoted as $\hat{\mathcal{S}}$, and also to predict target views $\mathcal{T}$ (given their camera poses) which are denoted as $\hat{\mathcal{T}}$. Each row corresponds to a different set of source views $\mathcal{S}$. Frames highlighted in green are input to GSN, frames highlighted in blue are predictions.

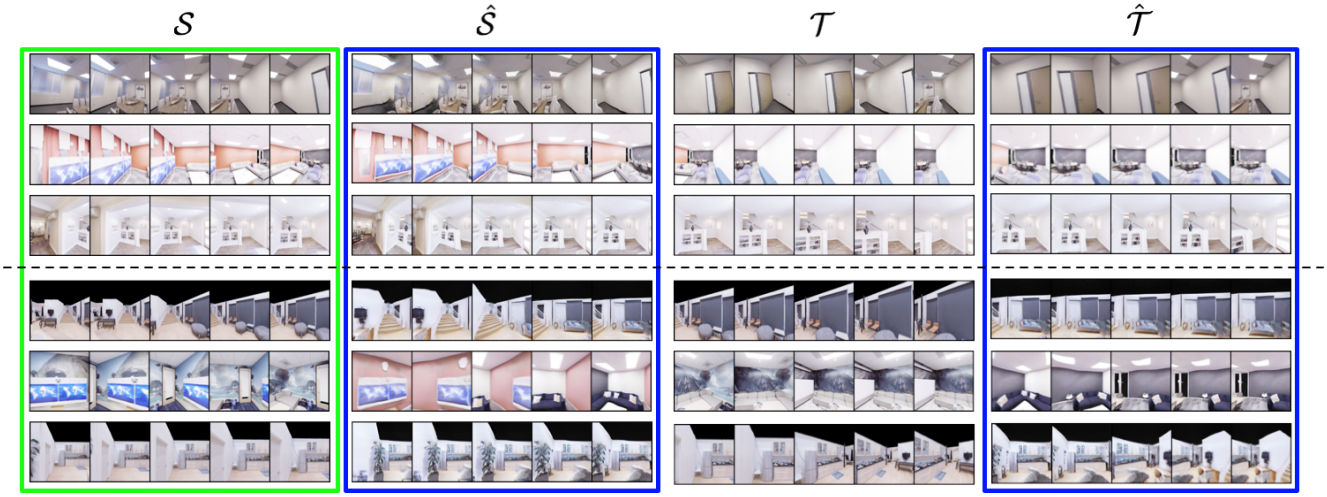

Figure 5: Qualitative view synthesis results on Replica. Given source views $\mathcal{S}$ we invert GSN to obtain a local latent code grid $\hat{\mathbf{W}}$, which is then use both to reconstruct $\mathcal{S}$, denoted as $\hat{\mathcal{S}}$, and also to predict target views $\mathcal{T}$ (given their camera poses) which are denoted as $\hat{\mathcal{T}}$. Each row corresponds to a different set of source views $\mathcal{S}$ (top 3 rows are scenes from the training set, bottom 3 rows are scenes in a heldout test set). Frames highlighted in green are input to GSN, frames highlighted in blue are predictions.

7. Generating Novel Scenes

GSN is able to model the distribution of scenes in smaller datasets well, but due to the limited variety in these datasets the synthesized scenes do not differ much from those present in the training set. Ideally, we would like to be able to generate completely novel scenes. In order to demonstrate that GSN extends to larger datasets, we train our model on 30 scenes sampled from the Matterport dataset [3]. In Fig. 13 we present some novel scenes generated by GSN as well as the pixel-wise nearest neighbours

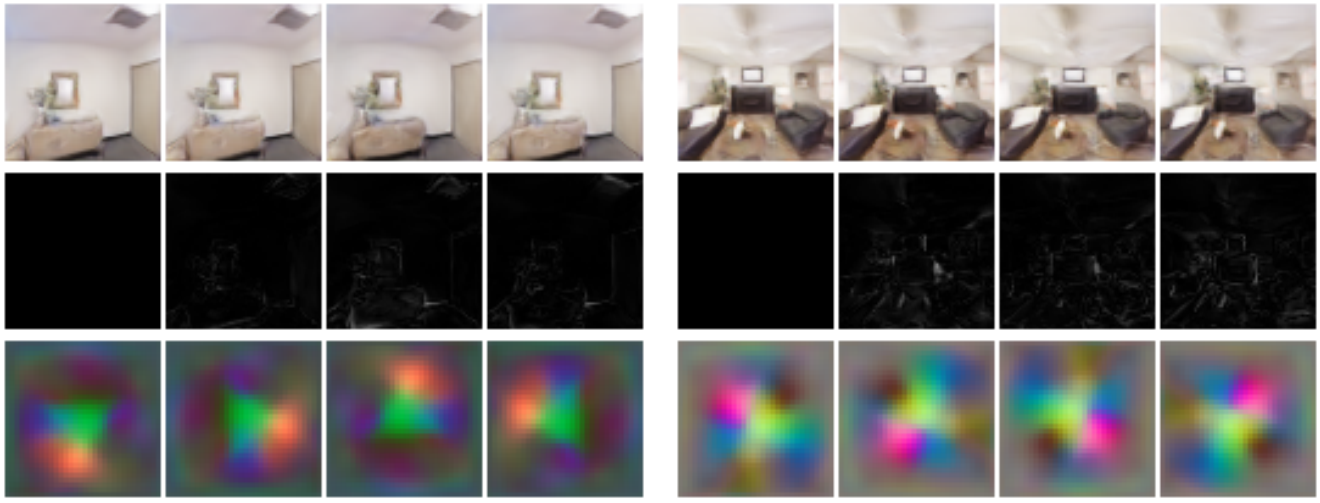

Figure 6: Change in generation output as local latent codes are rotated with a *local* coordinate system for two different scenes. (Top) Rendered image. (Middle) Residual w.r.t. 0 degree rotation. (Bottom) Visualization of **W**. Each column corresponds to a rotation of the camera and **W** in $\{0, 90, 180, 270\}$.

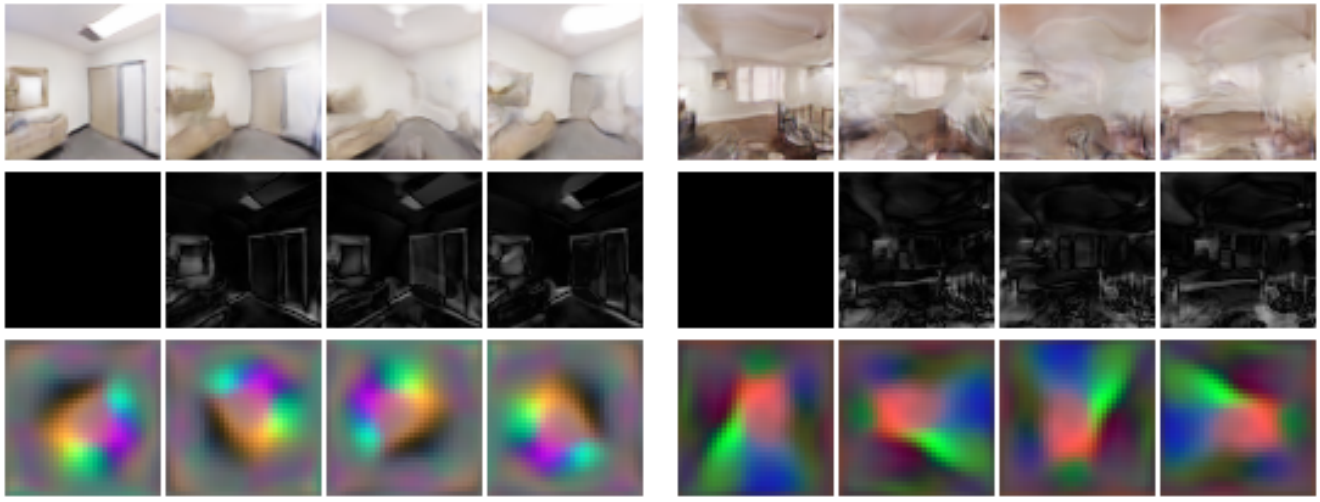

Figure 7: Change in generation output as local latent codes are rotated with a *global* coordinate system. (Top) Rendered image. (Middle) Residual w.r.t. 0 degree rotation. (Bottom) Visualization of **W**. Each column corresponds to a rotation of the camera and **W** in $\{0, 90, 180, 270\}$.

from the training set. We observe minimal overlap between the generated images and the nearest neighbours, suggesting that the model is capable of producing unique scenes when exposed to larger amounts of diverse data.

8. Qualitative Comparison of Models

In Fig. 14 we visually inspect the effects of different architectural design decisions. Pi-GAN [2] uses a global latent code and modulated linear layers with sinusoidal activations in its radiance field network. This model failed to fully converge in our setting, resulting in scenes with monotone colours and incorrect scene geometry. GRAF [16] also uses a global latent code, but is conditioned with concatenation instead of modulation. GRAF generates scene with reasonable appearance and geometry, although some artifacts are still present. GSN + global latent code is similar in construction to GRAF, but with modulated linear layers as conditioning instead of concatenation. Similar to GRAF, it generates a diverse set of scenes, but they still contain some artifacts. Replacing the global latent code in GSN for the local latent code resolves the issue with sporadic artifacts, resulting in clean generated scenes.

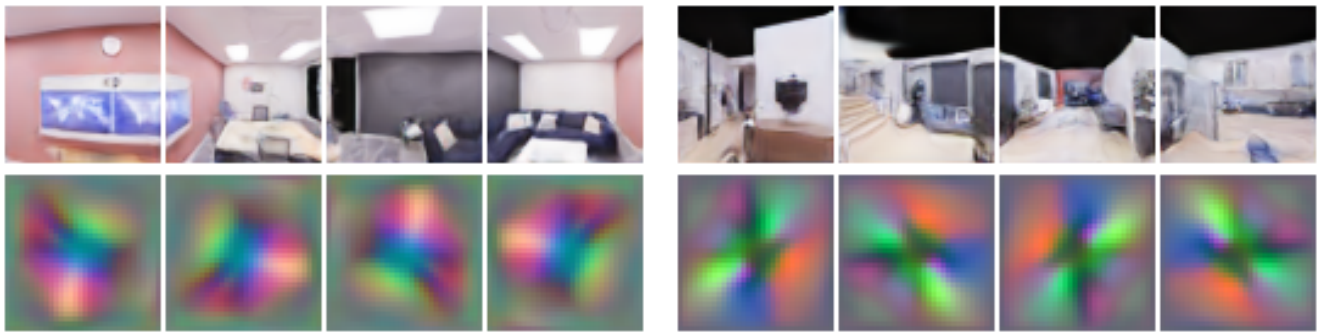

Figure 8: Change in generation output for a *fixed camera* as local latent codes are rotated with a *local* coordinate system for two different scenes. (Top) Rendered image. (Bottom) Visualization of $\mathbf{W}$. Each column corresponds to a rotation of $\mathbf{W}$ in $\{0, 90, 180, 270\}$.

Figure 9: Change in generation output for a *fixed camera* as local latent codes are rotated with a *global* coordinate system for two different scenes. (Top) Rendered image. (Bottom) Visualization of $\mathbf{W}$. Each column corresponds to a rotation of $\mathbf{W}$ in $\{0, 90, 180, 270\}$.

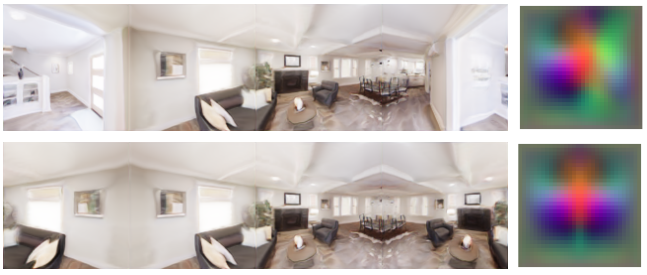

Figure 10: Panoramas and corresponding local latent codes for scenes produced by GSN. Mirroring the local latent code from a single room (top row) produces a new room (bottom row).

Figure 11: Panoramas and corresponding local latent codes for scenes produced by GSN. Mirroring the local latent code from a single room (top row) produces a new room (bottom row).

Figure 12: High resolution samples (256 × 256) implicitly upsampled from model trained at 128 × 128 resolution. The model preserves sharp edges during upsampling, but cannot resolve higher fidelity textures. Zoom for details.

Figure 13: GSN samples (64 × 64 resolution) from model trained on a subset of Matterport dataset, with top-3 pixel-wise nearest neighbours from training set.

Figure 14: Qualitative comparison of different model architectures at 64 × 64 resolution. GSN+global is equivalent to a GRAF model with modulated linear layers. GSN+local indicates the full model, with modulated linear layers, local latent grid, and local coordinate system. GT indicates ground truth images from the training set.

References

[1] Ivan Anokhin, Kirill Demochkin, Taras Khakhulin, Gleb Sterkin, Victor Lempitsky, and Denis Korzhenkov. Image generators with conditionally-independent pixel synthesis. *arXiv preprint arXiv:2011.13775*, 2020. 2

[2] Eric R Chan, Marco Monteiro, Petr Kellnhofer, Jiajun Wu, and Gordon Wetzstein. pi-GAN: Periodic implicit generative adversarial networks for 3d-aware image synthesis. *arXiv preprint arXiv:2012.00926*, 2020. 6

[3] Angel Chang, Angela Dai, Thomas Funkhouser, Maciej Halber, Matthias Niessner, Manolis Savva, Shuran Song, Andy Zeng, and Yinda Zhang. Matterport3d: Learning from rgb-d data in indoor environments. *arXiv preprint arXiv:1709.06158*, 2017. 5

[4] Terrance DeVries and Graham W Taylor. Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552*, 2017. 3

[5] Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. 2012. 3

[6] Abhishek Kar, Christian Häne, and Jitendra Malik. Learning a multi-view stereo machine. *arXiv preprint arXiv:1708.05375*, 2017. 3

[7] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. *arXiv preprint arXiv:1710.10196*, 2017. 2, 3

[8] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4401–4410, 2019. 1, 3

[9] 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. 1, 2, 3

[10] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 3. Citeseer, 2013. 1

[11] 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. *arXiv preprint arXiv:2003.08934*, 2020. 2

[12] Zak Murez, Tarrence van As, James Bartolozzi, Ayan Sinha, Vijay Badrinarayanan, and Andrew Rabinovich. Atlas: End-to-end 3d scene reconstruction from posed images. *arXiv preprint arXiv:2003.10432*, 2020. 3

[13] Michael Niemeyer and Andreas Geiger. Giraffe: Representing scenes as compositional generative neural feature fields. *arXiv preprint arXiv:2011.12100*, 2020. 2

[14] Keunhong Park, Utkarsh Sinha, Jonathan T Barron, Sofien Bouaziz, Dan B Goldman, Steven M Seitz, and Ricardo-Martin Brualla. Deformable neural radiance fields. *arXiv preprint arXiv:2011.12948*, 2020. 2

[15] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015. 3

[16] Katja Schwarz, Yiyi Liao, Michael Niemeyer, and Andreas Geiger. GRAF: Generative radiance fields for 3d-aware image synthesis. *arXiv preprint arXiv:2007.02442*, 2020. 6

[17] Julian Straub, Thomas Whelan, Lingni Ma, Yufan Chen, Erik Wijmans, Simon Green, Jakob J Engel, Raul Mur-Artal, Carl Ren, Shobhit Verma, et al. The replica dataset: A digital replica of indoor spaces. *arXiv preprint arXiv:1906.05797*, 2019. 4

[18] Weihao Xia, Yulun Zhang, Yujiu Yang, Jing-Hao Xue, Bolei Zhou, and Ming-Hsuan Yang. GAN inversion: A survey. *arXiv preprint arXiv:2101.05278*, 2021. 3

[19] Richard Zhang. Making convolutional networks shift-invariant again. In *International Conference on Machine Learning*, pages 7324–7334. PMLR, 2019. 1, 2

[20] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 586–595, 2018. 3

[21] Shengyu Zhao, Zhijian Liu, Ji Lin, Jun-Yan Zhu, and Song Han. Differentiable augmentation for data-efficient gan training. *arXiv preprint arXiv:2006.10738*, 2020. 3