Spring 2023 — Deep Learning
NeRF: Neural Radiance Fields — A deep dive into the variations and applications

Jash Rathod (Net ID: jsr10000, NYU ID: N13876910)
Aneek Roy (Net ID: ar8002, NYU ID: N18324528)
Github-Code Repository Link : https://github.com/aneekroy/nerf-deep-dive-testbench
New York University (NYU)

Abstract

The authors propose a technique that achieves excellent results in generating new views of intricate scenes by using a sparse collection of input views and optimizing an underlying continuous volumetric scene function. Their approach employs a fully-connected deep neural network that represents a scene by taking a single continuous 5-D coordinate as input (spatial location in x, y, z and viewing direction in θ, ϕ), and outputs the volume density and view-dependent emitted radiance at that location. The authors synthesize new views by querying 5-D coordinates along camera rays and then use standard volume rendering methods to project the colors and densities into an image. The differentiability of volume rendering enables the algorithm to optimize the network weights using only a set of images with known camera poses. The authors explain how they effectively optimize these neural radiance fields to render realistic views of complex scenes, surpassing previous work in neural rendering and view synthesis. We look into various related works on NeRF(Mildenhall et al. 2020), in a constrained resource environment, with sparse coding(Niemeyer et al. 2022), performance in real-time environment (Hedman et al. 2021), and representation from one or few images (Yu et al. 2021). Our report will focus on these applications, and in the process, we will visually compare them with prior works in neural rendering and view synthesis.

Introduction

In this work, the authors address the long-standing problem of view synthesis in a new way by directly optimizing the parameters of a continuous 5D scene representation to minimize the error of rendering a set of captured images. Earlier works in the field fail to reproduce realistic scenes with complex geometry, especially when compared to techniques that represent scenes using discrete representations such as triangle meshes or voxel grids.

Recent work in Neural 3D shape representations has investigated the implicit representation of continuous 3D shapes as level sets by optimizing deep networks that map xyz coordinates to signed distance functions (Jiang C. 2020) or occupancy fields (Genova K. 2020). However, one caveat in this field is its requirement to access the ground truth 3D geometry that is typically provided by synthetic 3D datasets like ShapeNet (Chang 2015). Subsequent work has relaxed this requirement of ground truth 3D shapes by formulating differentiable rendering functions that allow neural implicit shape representations to be optimized using only 2D images. But still these works are limited to simple shapes with low geometric complexity, resulting in over-smoothed renderings. The authors here show that an alternate strategy of optimizing networks to encode 5D radiance fields (3D volumes with 2D view-dependent appearance) can represent higher-resolution geometry and appearance to render photorealistic novel views of complex scenes.

The authors represent a static scene as a continuous 5D function that outputs the radiance emitted in each direction at each point (x; y; z) in space and a density at each point which acts like a differential opacity controlling how much radiance is accumulated by a ray passing through (x; y; z). Their method optimizes a deep fully-connected neural network without any convolutional layers (often referred to as a multilayer perceptron or MLP) to represent this function by regressing from a single 5D coordinate (x; y; z; θ, ϕ) to a single volume density and view-dependent RGB color. To render this neural radiance field (NeRF) from a particular viewpoint they: 1) march camera rays through the scene to generate a sampled set of 3D points, 2) use those points and their corresponding 2D viewing directions as input to the neural network to produce an output set of colors and densities, and 3) use classical volume rendering techniques to accumulate those colors and densities into a 2D image. Because this process is naturally differentiable, we can use gradient descent to optimize this model by minimizing the error between each observed image and the corresponding views rendered from our representation. Minimizing this error across multiple views encourages the network to predict a coherent model of the scene by assigning high volume densities and accurate colors to the locations that contain the true underlying scene content.

They represent a continuous scene as a 5D vector-valued function whose input is a 3D location x = (x; y; z) and 2D viewing direction (θ; ϕ), and whose output is an emitted color c = (r; g; b) and volume density σ. With direction as a 3D Cartesian Unit Vector d, We approximate this continuous 5D scene representation with an MLP network.

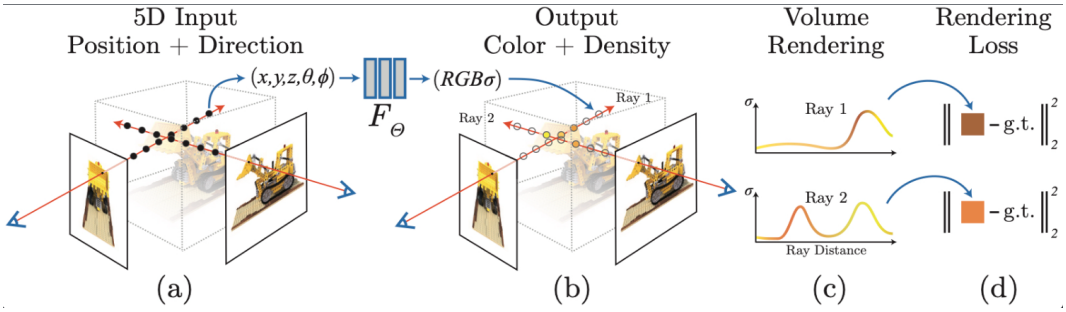

Figure 1: NeRF Architecture

Related Works

PixelNeRF

Alex et. al. (Yu et al. 2021) proposed a modified NeRF architecture to enable sharing of knowledge between scenes. They did so by adding a fully-convolutional image encoder which encodes the input image into a pixel-aligned feature grid, followed by a NeRF network that outputs color and density, given a spatial location and its corresponding encoded feature.

Given an input image I, they extract a feature volume by passing the image through the image encoder. Then, for a point on a camera ray x, they retrieve the corresponding image feature by projecting x onto the image plane to the image coordinates $\pi(x)$ using known intrinsics, then bilinearly interpolating between the pixel-wise features to extract the feature vector $W(\pi(x))$. The image features are then passed into the NeRF network, along with the position and view direction (both in the input view coordinate system) as:

$$f\left(\gamma\left(x\right), d; W\left(\pi\left(x\right)\right)\right) = (\sigma, c) \tag{1}$$

where γ is a positional encoding on x with six exponentially increasing frequencies, as in the original NeRF paper. The image feature is incorporated as a residual at each layer. They extend our model to allow for an arbitrary number of views at test time, which distinguishes our method from existing approaches that are designed to only use a single input view at test time.

With the assumption that only relative camera poses are known, they fix an arbitrary world world coordinate system. So, if we have an image $I^{(i)}$, then its associated camera transform from the world view is given by: $P^{(i)} = [R^{(i)}t^{(i)}]$. For a new target camera ray, the query point x is transformed, with view direction d, into the coordinate system of each input view I with the world to camera transform as:

$$x^{(i)} = P^{(i)}x, d^{(i)} = R^{(i)}d \tag{2}$$

To obtain the output density and color, they process the coordinates and corresponding features in each view coordinate frame independently and aggregate across the views within the NeRF network. If f1 denotes the initial layers which process inputs in each input view space separately, and f2 denotes the finals layers which process the aggregated views, then the intermediate vector V_i is given by:

$$V^{(i)} = f_1\left(\gamma\left(x^{(i)}\right), d^{(i)}; W^{(i)}\left(\pi\left(x^{(i)}\right)\right)\right) \tag{3}$$

The intermediate vector $V(i)$ are then aggregated with the average pooling operator ψ and passed into the final layers, denoted as f2, to obtain the predicted density and color:

$$(\sigma, c) = F_2\left(\psi\left(V^{(1)}, \ldots, V^{(n)}\right)\right) \tag{4}$$

RegNeRF

NeRF's performance drops significantly if the number of input views is sparse. This is because the optimization procedure in NeRF supervises these sparse viewpoints by reconstruction loss in the form:

$$\mathcal{L}_{MSE}(\theta, \mathcal{R}_i) = \sum_{r\epsilon\mathcal{R}_i} ||\hat{c}_\theta(r) - c_{GT}(r)||^2 \tag{5}$$

While the model might learn to reconstruct the input views perfectly, novel views may be degenerate because the model is not biased towards learning a 3D consistent solution with such a sparse input scenario. To overcome this, Michael Niemeyer et. al. (Niemeyer et al. 2022) regularize unseen viewpoints. To be more precise, they define a space of unseen but relevant viewpoints, and render small patches randomly sampled from these viewpoints. The key idea in RegNeRF is that these patches can be regularized to yield smooth geometry and high-likelihood colors.

A space is established that consists of potential camera positions as a bounding box of all provided target camera sites in order to obtain these unseen viewpoints. It is assumed that all cameras will focus on a central scene point with a common up axis, computing the normalized mean over the up axes of all target poses, and that there will be a mean focus point, determined by solving a least squares problem to find the 3D point with the shortest squared distance to the optical axes of all target poses. The set of all possible camera rotations is defined by

$$\mathcal{S}_R|t = R(\bar{p}_u, \bar{p}_f + \epsilon, t)|\epsilon \sim \mathcal{N}(0, 0.125) \tag{6}$$

where R(.,.,.) indicates the resulting "look-at" camera rotation matrix and ϵ is a small jitter added to the focus point.

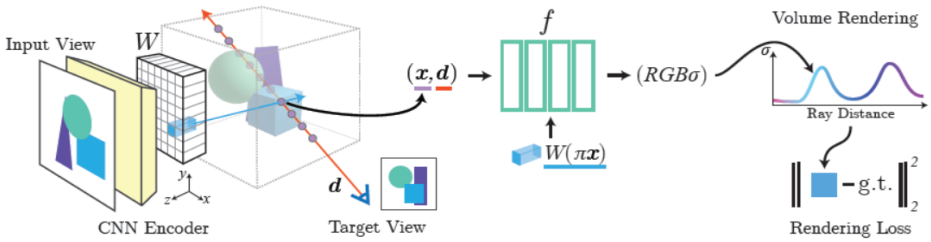

Figure 2: PixelNeRF architecture

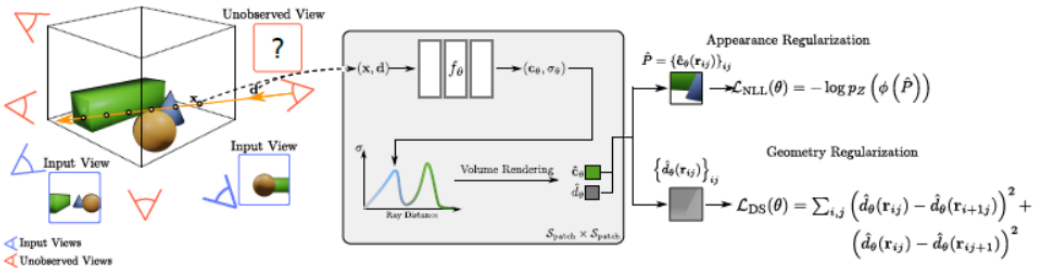

Figure 3: RegNeRF Architecture

In terms of Geometry regularization, the prior regarding real-world geometry tending to be piece-wise smooth is incorporated into the model by encouraging depth smoothness from unobserved viewpoints. In terms of Color regularization, sparse inputs lead to color shifts. This is countered by estimating the likelihood of rendered patches and maximizing it during optimization.

Another problem the authors engage in this paper is NeRF's highly divergent behavior at the start of training. Even though, it reconstructs the input views correctly over iterations, but this divergent behavior deteriorates novel views. To avoid this, the authors implement sample space annealing to restrict the scene sampling space to a smaller region, essentially introducing an inductive bias to explain input images with geometric structures in the center of the scene.

NeRF in real-time

Prior implementations of NeRF (Mildenhall et al. 2020) are quite slow rendering a ray requires querying an MLP hundred of times, such that rendering a frame at 800 x 800 resolution takes roughly a minute on a modern GPU. This prevents NeRF from being used for interactive view synthesis applications. Peter Hedman et. al. (Hedman et al. 2021) designed a practical representation that enables the serving and real-time rendering of scenes reconstructed by NeRF.

The authors approach the problem in two ways: reformulating NeRF and designing a procedure to bake this modified NeRF into a discrete volumetric representation that is suited for real-time rendering. To reformulate NeRF, they modify the NeRF to output a diffused RGB color c_d and 4-dimensional vector v_s (similar to deferred rendering (Justus Thies and Niebner 2019)) in addition to the volume density σ at each input 3D location:

$$\sigma\left(t\right), C_d\left(t\right), v_s\left(t\right) = MLP_\theta\left(r\left(t\right)\right) \tag{7}$$

To render a pixel, they accumulate the diffuse colors and feature vectors along each ray and pass the accumulated feature vector and color, concatenated to the ray's direction, to a very small MLP with parameters ϕ (2 layers with 16 channels each) to produce a view-dependent residual that we add to the accumulated diffuse color.

$$\widehat{C}_d\left(r\right) = \sum_k T\left(t_k\right)\alpha\left(\sigma\left(t_k\right)S_k\right)c_d\left(t_k\right) \tag{8}$$

$$V_s\left(r\right) = \sum_k T\left(t_k\right)\alpha\left(\sigma\left(t_k\right)\delta_k\right)v_s\left(t_k\right) \tag{9}$$

$$\widehat{C}\left(r\right) = \widehat{C}_d\left(r\right) + MLP_\phi\left(v_s\left(r\right), d\right) \tag{10}$$

With this modification, they can precompute and store the diffuse colors and 4-dimensional feature vectors within our sparse voxel grid representation. So, the need to evaluate the MLP once per sample reduces to evaluating the MLP once per pixel. In addition, they regulate the sparsity of opacity within the scene with a regularizer that penalizes the predicted density using a Cauchy loss during training.

Now, to use the trained deferred NeRF in real-time, they replace the MLP evaluations in NeRF with fast lookups in a pre-computed data structure, i.e., baking, the diffuse colors c_d, volume densities σ, and 4-dimensional feature vectors v_s in a voxel grid data structure. Sparse Neural Radiance Grids or SNeRG represents an N^3 voxel grid in a block-sparse

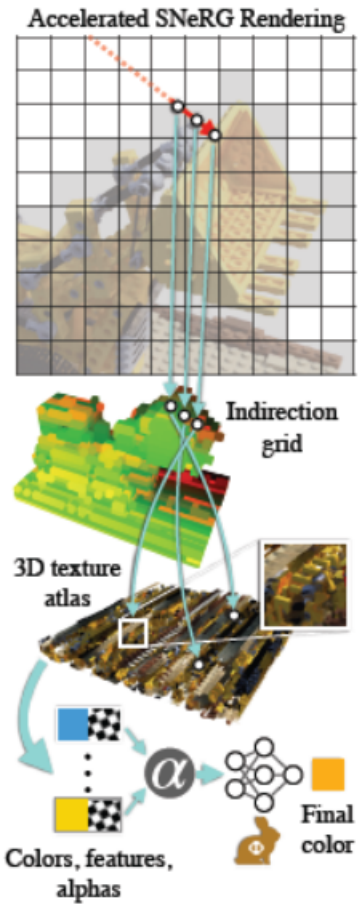

Figure 4: SNeRG architecutre

format using two smaller dense arrays: The first array is a 3D texture atlas containing densely packed "macroblocks" of size B^3 each, corresponding to the content (diffuse color, feature vectors, and opacity) that actually exists in the sparse volume, the second array is a low resolution $(N/B)^3$ indirection grid, which either stores a value indicating that the corresponding B^3 macroblock within the full voxel grid is empty, or stores an index that points to the high-resolution content of that macroblock within the 3D texture atlas. To minimize storage cost and rendering time, their procedure only allocates storage for voxels in the scene that are both non-empty and visible in at least one of the training views. Furthermore, they quantize all values in the baked SNeRG representation to 8 bits and separately compress the indirection grid and the 3D texture atlas.

Experiments and Results

Our main objective was to compare different implementations of NeRF. For the codebase, we referred to the author's GitHub repositories. Most of the implementations were CLI-based that required inputs in the form of videos or images. We tried simulating our own environments or objects in front

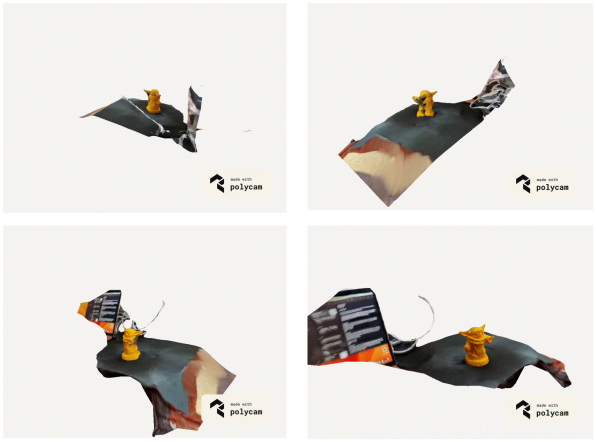

Figure 5: NeRF output

of us, notably, our library rooms and a 3D-printed Yoda toy, and the results are listed below.

To implement NeRF, NerfStudio (Tancik et al. 2023) was used. It provides an intuitive GUI and provides a selection of NeRF models to choose from, we went with the regular NeRF model. The output was in the form of a render, which was viewed on PolyCam. Figure 5 demonstrates screenshots of the 3D render generated by NeRF.

This result was achieved after fine-tuning two hyperparameters: Polygon Size and Smoothing. Polygon size determines the size of the smallest polygon. We explored the trade-off between polygon size and rendering time, larger polygon size means lower render time, but the rendered object won't be as detailed as a smaller polygon size render. But since we had limited computing power and time, we opted for a larger polygon size. The polygon size for all our experiments was set at 5 mm. Another parameter we experimented with was smoothing. Smoothing blurs the edges visually, makes the surface smoother, and hides the smaller edge errors. We kept the smoothing factor at 10%. Additonally, Adam optimizer was used and its hyperparameters are left at default values of $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 10^7$ for all the experiments for constancy.

PixelNeRF and RegNeRF try to resolve the same issue: sparsity in inputs. Feeding four images to each model yields the following 3D renders of the library rooms in NYU Dibner Library. (Fig 6 and Fig 7)

The output render has a lot of artifacts, but that is mainly because the images were taken on a cellphone without standardized parameters. Images fed into the PixelNeRF model encompassed the entire room, where we took an image from every end of the rectangular room. For the RegNeRF model, we took a smaller subsection of the same room and took photographs from all ends.

SNeRG was implemented on the images of the smaller subsection of the rectangular room. The inputs were screenshots from a video. The rendered views came out to be quite

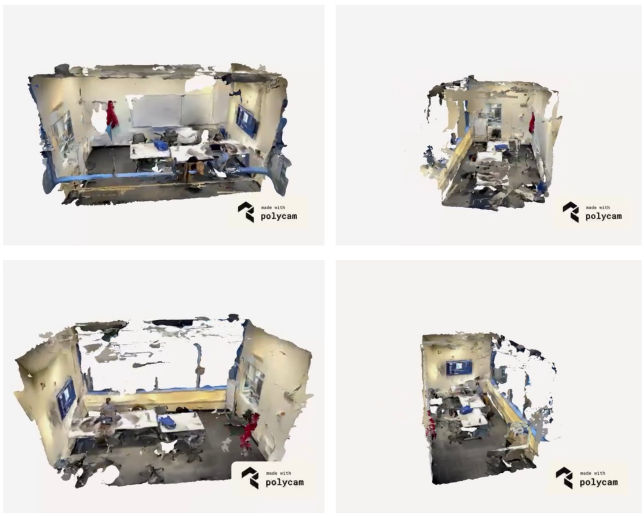

Figure 6: PixelNeRF Output

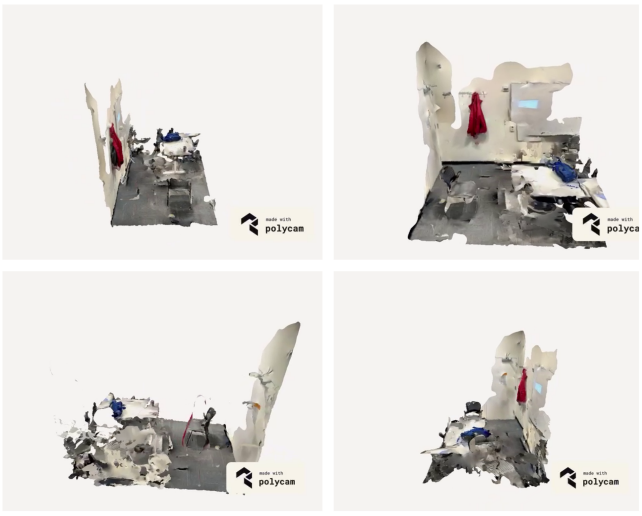

Figure 7: RegNeRF Output

detailed at the center of the scene but had a lot of artifacts at the outer fringes. Figure 8 shows the encompassing view of the scene, whereas figure 9 focuses on the center of the scene (the red jacket).

We see a high amount of detail on the red jacket. The model is able to retain the texture of the jacket as well as the surface reflection of the in-scene light. Peter et. al. (Hedman et al. 2021) also demonstrated the model's power to accurately reconstruct reflections on a shiny metal surface. So, going along the same line, this level of detail was expected.

For the trainings, the goal is to maximize the loss Peak Signal to Noise Ratio (PSNR). Figure 10 shows the plot for PSNR vs number of input views. Higher PSNR is better. Also, we have tabulated our best results for different models in table 1

Table 1: Best PSNR Values for Different Architectures

Architecture	Best PSNR
NeRF	24.324
PixelNeRF	23.173
RegNeRF	25.190
SNeRG	23.981

Conclusion

We started this project with an aim to achieve a comprehensive understanding of NeRF variations and applications, compare the performance of different NeRF implementations, and identify the strengths and weaknesses of each approach. Summarizing our work, we have implemented NeRF and different variations of it. The variations focused on two different aspects of it; PixelNeRF and RegNeRF improved NeRF's ability to work on sparse inputs, and SNeRG explored the real-time aspect of NeRF. Every variation modified the NeRF network to achieve better results with respect to the problem they were solving. NeRF has been a very active topic within view synthesis and there are a lot of research papers that have since built upon what we have presented here. Researchers might have already started working on the challenges we have stated above, but we will run more experiments to form a proper solution for the challenges mentioned above.

References

Chang, F. T. G. L. H. P. H. Q. L. Z. S. S. S. M. S. S. S. H. e. a., A.X. 2015. Shapenet: An information-rich 3d model repository. In *arXiv:1512.03012*.

Genova K., S. A. S. A. F. T., Cole F. 2020. Local deep implicit functions for 3d shape. In *ECCV*.

Hedman, P.; Srinivasan, P. P.; Mildenhall, B.; Barron, J. T.; and Debevec, P. 2021. Baking Neural Radiance Fields for Real-Time View Synthesis. *ICCV*.

Jiang C., A. M. A. H. J. N. M. F. T., Sud. 2020. Implicit grid representations for 3d scenes. In *CVPR*.

Justus Thies, M. Z.; and Niebner, M. 2019. Deferred neural rendering: Image synthesis using neural textures. In *ACM Transactions on Graphics*.

Mildenhall, B.; Srinivasan, P. P.; Tancik, M.; Barron, J. T.; Ramamoorthi, R.; and Ng, R. 2020. NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis. In *ECCV*.

Niemeyer, M.; Barron, J. T.; Mildenhall, B.; Sajjadi, M. S. M.; Geiger, A.; and Radwan, N. 2022. RegNeRF: Regularizing Neural Radiance Fields for View Synthesis from Sparse Inputs. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.

Tancik, M.; Weber, E.; Ng, E.; Li, R.; Yi, B.; Kerr, J.; Wang, T.; Kristoffersen, A.; Austin, J.; Salahi, K.; Ahuja, A.; McAllister, D.; and Kanazawa, A. 2023. Nerfstudio: A Modular Framework for Neural Radiance Field Development. *arXiv preprint arXiv:2302.04264*.

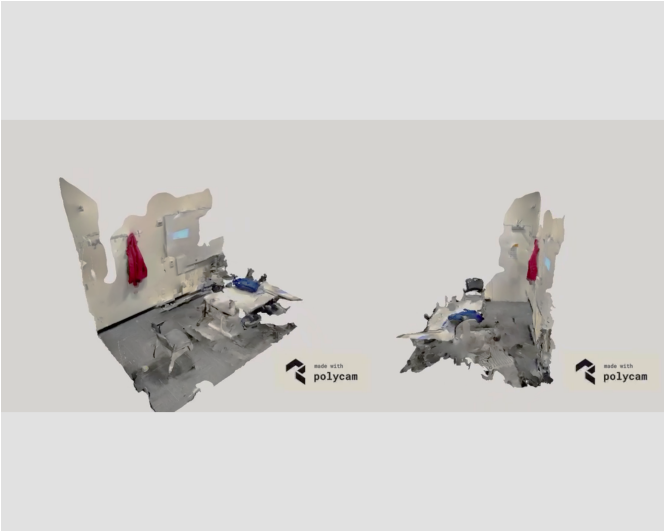

Figure 8: SNeRG Entire View

Yu, A.; Ye, V.; Tancik, M.; and Kanazawa, A. 2021. pixelNeRF: Neural Radiance Fields from One or Few Images. In *CVPR*.

Figure 9: SNeRG with focused center

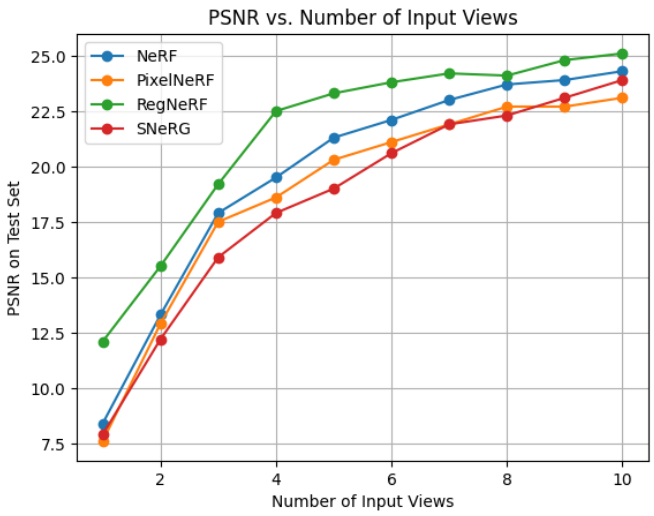

Figure 10: Results plot. Higher PSNR is better.