PP-NeRF: Preprocessed Neural Radiance Fields

Christian Aagnes
caagnes@mit.edu

Kushagra Chitkara
kchitkar@mit.edu

Abstract

We propose an alternative framework to generate new perspectives of intricate scenes from sparse data. We achieve this by leveraging Neural Radiance Fields (NeRF) as our foundation, which utilizes a multilayer perceptron's weights to depict a scene's density and color based on its 3D coordinates. While NeRF excels with static subjects in controlled environments, its training and rendering times are extremely slow and the 3D renderings produced from low-resolution images are often quite poor. Although successful attempts have already been made to remedy these limitations of NeRF, they typically do not apply traditional image preprocessing techniques (if any) to further aid the NeRF in creating consistent and clear novel view renderings. This project implements a compressed version of the original NeRF architecture (from scratch) that still produces renderings of comparable quality and also utilizes traditional image preprocessing techniques prior to training the NeRF model to ultimately improve the final 3D renderings of the scene, especially for low-resolution images.

1. Introduction

In the rapidly evolving field of computer vision, the synthesis of photorealistic images from sparse data has emerged as a pivotal challenge, particularly in the context of generating new perspectives of complex scenes. An area which was relatively untouched since the past few decades has seen the emergence of Neural Radiance Fields (NeRF) [6], a groundbreaking approach that uses a deep neural network to model volumetric scenes with unprecedented detail and realism.

While a significant leap forward, NeRF has some drawbacks. In this paper, we wish to tackle two of those limitations: 1) Training a NeRF model is a computationally challenging task and 2) Training a NeRF model on low resolution images results in poor 3D renderings while training it on very high resolution images can result in computational explosions.

While there have been novel approaches to tackle the training and rendering times of the original NeRF model architecture [8] [7] [11] [2] [4], we show that a compressed version of the original NeRF architecture can perform well enough for many applications while reducing its training time significantly.

To improve the 3D renderings of scenes with low-resolution images we propose our framework PP-NeRF, which includes preprocessing the images by applying low- and high-pass filters as well as background masking prior to feeding them into our NeRF implementation. We compare our results of the original reconstruction with the preprocessed reconstructions on our own implementation of the NeRF architecture. Since our preprocessing steps are a function of the dataset rather than the model, our framework has the advantage that it can be extended to any future NeRF implementations as well.

2. Related Work

Neural Radiance Fields (NeRF) [6] introduced a novel method for synthesizing views of complex scenes by modeling the volumetric scene function using a fully connected multilayer perceptron. This approach encodes both color and density as functions of 3D coordinates, which are input along with a 2D viewing direction to the network. NeRF uses the color and density outputs from the neural network to reconstruct the entire scene pixel by pixel.

However, NeRF suffers from a few drawbacks. It requires a significant amount of computational resources for both training and inference. Rendering a single image can be very time-consuming because the model needs to sample thousands of points along each ray for hundreds of rays per image. The original NeRF model is also designed to work with completely static scenes. The performance heavily relies on accurate camera parameters and low-noise images.

Attempts have been made to speed up the NeRF model. For example, researchers at NVIDIA came up with Instant Neural Graphics Primitives (Instant NGP) that dramatically accelerates the training and inference processes compared to traditional NeRF [7]. Instead of using a simple multilayer perceptron (MLP) like in traditional NeRF, Instant NGP employs a hash table to map 3D coordinates to feature vectors efficiently. This approach reduces memory re-

quirements and accelerates lookup times. Even though we implemented our own implementation of NeRF which is faster than the original implementation, it still took very long to conduct extensive experiments and we often ran out of GPU. Hence, some of our experiments were ran using the Instant NGP software - this will be explicitly mentioned throughout the report where it is appropriate.

Researchers have also looked to tackle the limitations of well curated datasets [1], coming up with NeRF in the Wild, which uses a modified model to understand and simulate how different lighting conditions affect the appearance of the scene [5].

We wish to take a different approach, where we preprocess our images instead of having the model adapt to various conditions. This also reduces the computational challenges to the model and speeds up training time, as fewer iterations are required to obtain a reconstruction of the scene.

3. Models and Methodology

3.1. Model Pipeline

We divide the PP-NeRF framework into the following 5 steps:

1. **Dataset collection:** The first step involves collecting a dataset of a scene to conduct our experiments. For this we first used the Lego dataset as this was one of the datasets used in the original NeRF paper, which would allow us to compare our approach to previous state of the art results. To conduct experiments on a more complicated scene we used the Gerrard Hall dataset from COLMAP [10] [9]. It provides 100 images of the "Gerrard" hall at UNC Chapel Hill.

2. **Image preprocessing:** We will be experimenting with several preprocessing techniques of the photographs including applying different filters to the main object and the background of the scene respectively as well as more naive approaches such as background removal techniques. These will be outlined in greater detail in section 3.2.2.

3. **Generating camera parameters:** We need to estimate the camera parameters for each preprocessed image in order to create sets of 3D coordinates and viewing directions, that will later be the input to our NeRF implementation. We use the pinhole camera model such that the parameters we have to estimate for each camera are the focal length f, two radial distortion parameters (k_1 and k_2), a rotation $\boldsymbol{R}$, and translation $\boldsymbol{t}$. Then the formula for projecting a 3D point $\boldsymbol{X}$ into a camera ($\boldsymbol{R}$, $\boldsymbol{t}$, f) is:

$$\boldsymbol{P} = \boldsymbol{R}\boldsymbol{X} + \boldsymbol{t} \qquad \text{(world to camera coordinates)}$$

$$\boldsymbol{p} = -\frac{1}{\boldsymbol{P}(z)} \cdot \boldsymbol{P} \qquad \text{(perspective division)}$$

$$\boldsymbol{p}^T = f \cdot r(\boldsymbol{p}) \cdot \boldsymbol{p} \qquad \text{(conversion to pixel coordinates)}$$

where $\boldsymbol{P}(z)$ is the z-coordinate of $\boldsymbol{P}$ and $r(\boldsymbol{p})$ is a function that computes a scaling factor to undo the radial distortion [3]:

$$r(\boldsymbol{p}) = 1.0 + k_1 \cdot ||\boldsymbol{p}||^2 + k_2 \cdot ||\boldsymbol{p}||^4$$

This gives a projection in pixels, where the origin of the image is the center of the image, the positive x-axis points right, and the positive y-axis points up (in addition, in the camera coordinate system, the positive z-axis points backwards, so the camera is looking down the negative z-axis). Finally, the equations above imply that the camera viewing direction is the third row of $\boldsymbol{R}$ and the 3D position of a camera is: $-\boldsymbol{R}^T\boldsymbol{t}$.

4. **Training NeRF model:** Now that we have both the camera viewing direction and position for each of the preprocessed images we are ready to train the NeRF on a specific scene. For this we used our own compressed version of the original NeRF model as well as NVIDIA's Instant-NGP model for more extensive experimentation [7].

5. **Visualize results:** The final step is to evaluate the final model's performance and use it to generate new views of the scene from arbitrary viewpoints. To do this, we use the same method to estimate the camera parameters for the desired viewpoint, and then use our NeRF model to render a new image from that viewpoint. We did a qualitative analysis of the rendered scenes, focusing on both the reconstruction of the scene as a whole as well as the finer details such as textures and shadows.

The full model pipeline is also shown in figure 1.

3.2. Methodology

3.2.1 Compressed NeRF Architecture

We created a faster, more compressed implementation of the NeRF model architecture from scratch using PyTorch in order to conduct our experiments. Our implementation can be viewed here. Just like the original NeRF implementation, it takes in a 5D input given by the 3D position of a sampled point along a ray from the camera position to the scene and a viewing direction given by two angles. It outputs a color (RGB-valued) as well as a scalar density value for that point. Unlike traditional machine learning methods, in this

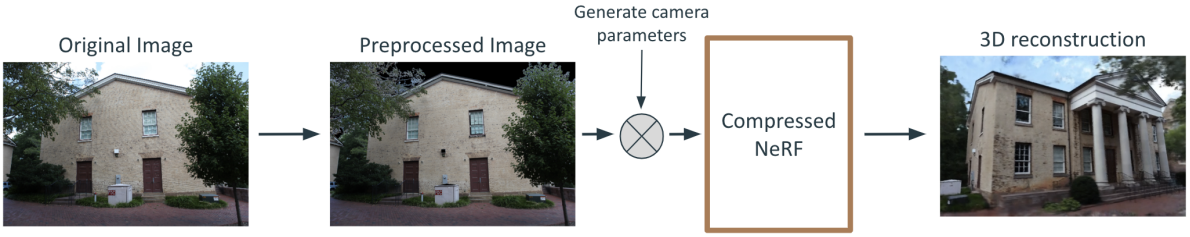
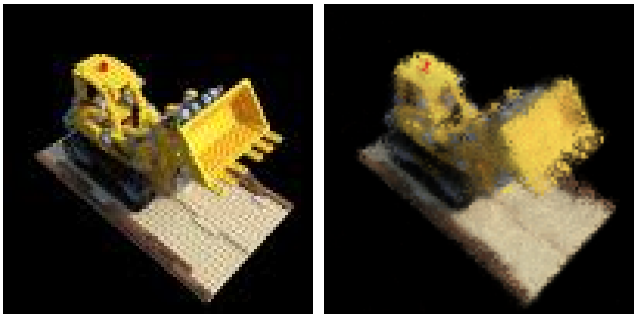


Figure 1. PP-NeRF framework: The preprocessing steps aims to improve the final 3D renderings reconstructed from the original, low-resolution images while the "Compressed NeRF" implementation aims to improve the slow training and inference times of the original NeRF architecture.

case we actually want our neural network to overfit to our scene, as we want the weights to be an accurate representation of the 3D scene. The original NeRF implementation has 10 hidden layers including 9 fully-connected ReLU layers with a dimension of 256, as well as one fully-connected ReLU layer with a dimension of 128. Due to GPU time and memory constraints, conducting experiments using this model architecture was not computationally feasible (Note: we asked the teaching staff for more compute but we were not given access to this). Our compressed NeRF model uses only 5 hidden layers: 4 fully-connected ReLU layers with filter sizes of 128 and one fully-connected ReLU layer with a filter size of 68. Just like the original implementation we also included positional encodings in order to properly capture the high-frequency variations in the images. To optimize our network we use the Adam optimizer with a learning rate set to $5e-3$ and we sampled 32 depths for each ray. The final 3D renderings of our compressed NeRF implementation on the Lego dataset can be seen in figure 2.



(a) Original Image (b) Reconstructed Image

Figure 2. The reconstructed image was rendered using our own "compressed NeRF" implementation, by training it for 10,000 iterations on the Lego dataset (which was also used by the original NeRF authors).

What we obtain is a blurry yet effective reconstruction of the ground truth, as it captures the overall scene but fails to reconstruct the finer details of the textures of the Lego pieces. The original NeRF model requires training for several hours, whereas 10,000 iterations on our compressed NeRF takes only 30 minutes. Note that this is for the Lego dataset which includes relatively simple images with no background or transient objects. The training and inference times on more complex images like the Gerrard Hall dataset required much more time to train as it required more images to accurately reconstruct the scene.

3.2.2 Image Preprocessing

In addition to implementing a more computationally efficient version of the original NeRF architecture, we also experiment with different preprocessing techniques of the images in our dataset, prior to feeding them into our compressed NeRF implementation. This aims to improve the final renderings of the scene, especially for applications when the images are inherently blurry or of low resolution.

Motivated by the Lego dataset we first tried remove, mask, and blur the background of the images. To blur the images, we first identified the background and then applied a Gaussian filter to these sections of the images specifically. The intuition behind this step was to aid the NeRF model in focusing on the main object of the scene (which in this case was the hall) rather than objects in the background such as the sky, trees, and other objects.

As these images were of quite low resolution, we also experimented with sharpening the edges and the finer details of the hall. To do this we used high-pass filters such as the Laplacian filter as well as simple sharpening kernels. We also experimented with combining the original dataset with the processed images rather than solely using the preprocessed images. An example of different preprocessed images are shown in figure 3.

4. Results and Discussion

Since we wanted to test our framework on a more complex dataset, we focused on running experiments on the Gerrard Hall dataset. We first trained our baseline model on the unedited images of the hall. As aforementioned, we experimented with background masking, sharpening, and a

(a) Masked background.

(b) Sharpened image with a high-pass filter.

(c) Masked background and sharpened hall.

Figure 3. Three examples of the preprocessed images from the Gerrard Hall dataset.

combination of the two to create 3D renderings of the hall. Figure 4, 5, and 6 show images of different models from similar viewing angles. Please view the presentation for the full videos of the 3D renderings of the hall. Note that those videos were created using the Instant-NGP model, and the images below are screenshots from those videos.

Figure 4. Images from baseline model reconstruction

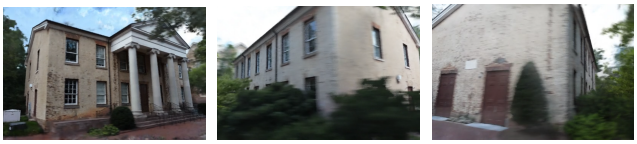


Figure 5. Images from sharpened model reconstruction

Figure 6. Images from masked and sharpened model reconstruction

From the figures above we can see that the sharpened model reconstruction (figure 5) is significantly better than the baseline model (figure 4). This is probably due to the fact that the images were of relatively low resolution, and thus the high-pass filters were able to sharpen the edges of the hall. This is especially clear when we focus on the pillars of the main entrance of the hall as we can see that the sharpened model reconstruction accurately depicts the texture of the pillars. With regards to the masked and sharpened images (figure 6), at first glance it seems as though the masking negatively affected the final rendering. This might be due to the fact that the masking of the background was not perfect on all of the images in our dataset, resulting in several artifacts in the 3D rendering as the model saw both the sky in some images and a masked background in others. However, due to the sharpening on the hall itself, again we see that some finer details are better represented in this rendering compared to the baseline model. Interestingly, we also found that applying a Gaussian filter on the background of the images instead of masking it also did not improve the model's performance and resulted in indistinguishable renderings from the baseline model. Understanding why exactly this is the case would be an interesting avenue for future work.

5. Future Work

With our current PP-NeRF implementation there are two main directions for further work and improvement of our framework: 1) experimenting with more preprocessing techniques and 2) change the current implementation to handle in-the-wild photographs.

So far, we have only focused on simple preprocessing techniques, and from the results we could see that different techniques worked best depending on the context of the images. This is undesirable since it means that our framework is not very generalizable to other scenes. Due to limited computational resources, it was very challenging to conduct many different experiments on the preprocessing techniques and we were only able to work extensively on two datasets. It is clear that there are still more techniques that should be explored as well as to properly understand why some techniques worked better than others. For example, applying the Gaussian filter to the backgrounds might have created indistinguishable renderings only for this specific dataset due to the static nature of the scene, or it could be that it is important to maintain a consistent level of sharpness throughout the image. This would be important questions to consider if we want to create a framework that is applicable to all scenes.

Furthermore, all of our experiments were conducted on datasets of static scenes with no varying lighting conditions. Thus, a direction for future work could be to extend our framework to account for these conditions. One idea to achieve this could be to train a network to initialize the latent associated with each image in order to improve their consistency for easier 3D reconstruction. This could enable amortization similar to variational autoencoders, although we do note that such a network would need a strong inductive bias and a 3D prior.

6. Contributions

Overall, we think we worked very well as a team and we thoroughly enjoyed working together on this project! We tried to delegate the tasks as equally as possible, and you can find more specific details about our individual contri-

butions below. Note that all the conceptual decisions such as the direction of the project, the preprocessing techniques, and the overall model framework were always discussed together and decided upon as a group.

Christian: My main task consisted of creating the pipeline to preprocess the images prior to feeding them into our compressed NeRF implementation such that we could conduct all of our experiments of the different preprocessing techniques. I also helped to implement our compressed NeRF implementation.

Kushagra: My main task consisted of following the NeRF implementation as described in the original paper and compressing it such that we could conduct our experiments with the limited computational resources we had access to. I also ran the final experiments using NVIDIA's Instant-NGP model.

References

[1] Jonathan T. Barron, Ben Mildenhall, Matthew Tancik, Peter Hedman, Ricardo Martin-Brualla, and Pratul P. Srinivasan. Mip-nerf: A multiscale representation for anti-aliasing neural radiance fields. *CoRR*, abs/2103.13415, 2021. 2

[2] Zhiqin Chen, Thomas Funkhouser, Peter Hedman, and Andrea Tagliasacchi. Mobilenerf: Exploiting the polygon rasterization pipeline for efficient neural field rendering on mobile architectures. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16569–16578, 2023. 1

[3] Pierre Drap and Julien Lefèvre. An exact formula for calculating inverse radial lens distortions. *Sensors (Basel)*, 2016. 2

[4] Stephan J. Garbin, Marek Kowalski, Matthew Johnson, Jamie Shotton, and Julien P. C. Valentin. Fastnerf: High-fidelity neural rendering at 200fps. *CoRR*, abs/2103.10380, 2021. 1

[5] Ricardo Martin-Brualla, Noha Radwan, Mehdi S. M. Sajjadi, Jonathan T. Barron, Alexey Dosovitskiy, and Daniel Duckworth. Nerf in the wild: Neural radiance fields for unconstrained photo collections. *CoRR*, abs/2008.02268, 2020. 2

[6] 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. *CoRR*, abs/2003.08934, 2020. 1

[7] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Trans. Graph.*, 41(4):102:1–102:15, July 2022. 1, 2

[8] Christian Reiser, Songyou Peng, Yiyi Liao, and Andreas Geiger. Kilonerf: Speeding up neural radiance fields with thousands of tiny mlps. *CoRR*, abs/2103.13744, 2021. 1

[9] Johannes Lutz Schönberger and Jan-Michael Frahm. Structure-from-motion revisited. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 2

[10] Johannes Lutz Schönberger, Enliang Zheng, Marc Pollefeys, and Jan-Michael Frahm. Pixelwise view selection for unstructured multi-view stereo. In *European Conference on Computer Vision (ECCV)*, 2016. 2

[11] Alex Yu, Ruilong Li, Matthew Tancik, Hao Li, Ren Ng, and Angjoo Kanazawa. Plenoctrees for real-time rendering of neural radiance fields. *CoRR*, abs/2103.14024, 2021. 1