A Study of Neural Radiance Fields

Computational Geometry Course Project

CS 5750 with Prof. Sudhanshu Semwal

Wes Robbins

11 May 2022

Abstract

Computational Geometry has played a fundamental role of the development of tasks such as image synthesis, image reconstruction, and depth maps. More recently, methods for these tasks have began to use neural networks. A recent—and already well-known—method for image synthesis and scene reconstruction that uses machine learning is Neural Radiance Fields (NeRFs). NeRFs have not only create eye-catching results, but achieve state-of-the-art performance on image synthesis and scene reconstruction benchmarks. In this report, I provide an overview of Neural Rendering Field Methods. Additionally, I discuss my own implementation and running time of training NeRF on my system. Last, I discuss some of the latest improvements in NeRF-based methods.

1 Neural Radiance Fields

Neural Radiance Fields (NeRFs) are a method for novel view synthesis. Here, I introduce the topic of NeRF in several subsections. First, in Section 1.1, I analyze the meaning of each word in the title of NeRF. Then, in Section 1.2, I introduce the neural network and discuss what the importance of the neural network is in NeRFs. Last, I discuss the method for training the NeRF model in Section 1.3.

After introducing Neural Radiance Fields, the rest of the report is organized as follows. Section 2 is an overview of my implementation details and computational cost. Section 3 covers four different recent improvement of the base NeRF approach. Then, I conclude the report in Section 4.

1.1 Why is the method called *Neural Radiance Fields*?

Understanding the title of NeRF can help give us context to better understanding the main ideas of NeRF. Below I define each word of the title in the context of this work.

Neural Here, neural implies the use of a neural network. In NeRF, a fully connected neural network with 7-layers is used. In NeRF, the neural network is the backbone for representing a scene.

Radiance In graphics processing, radiance defined by the quantity light that is being emitted from a point in space in each direction. NeRF adopts this term from traditional graphics processing, and uses it in the same context.

Field The NeRF method models a smooth (i.e., continuous) representation of a scene, which can also be described as a 'field'. NeRFs can generate views from arbitrary views, which are described by viewing angleds—and viewing angles are of course continuous. In other words, NeRF is not limited to a fixed number of discrete views.

1.2 Neural Classifier vs. Neural Scene Representation

To understand the Neural Radiance Fields, we must understand the purpose of the neural network within them. It can be challenging to understand the purpose of the neural network in NeRF, since it is much different than the standard use case of a neural network. Typically, a neural network is used as function approximator $F(x)$. The purpose of neural network $F(x)$ is to learn a mapping from an input space to a set of classes. One of the widely known use cases for neural networks is for image classification. Image

classification is an example where the model is learning a mapping from an arbitrary image input to an output class. However, the neural network in NeRF serves a much different purpose.

The neural network NeRF is used to learn a scene *representation*. As an input the model takes a 5d vector (x, y, z, Θ, ϕ) which is made of a 3 dimensional space coordinate and a 2 dimensional viewing angle on the surface of a viewing sphere. Instead of outputting the class the neural network outputs a 4 dimensional vector (R, G, B, α) which represents the color and radiance or transparency of that pixel. To synthesize a novel view, the neural network is queried for each pixel in the new image. Figure further shows the difference between a neural network used for standard classification versus a neural network used for scene representation.

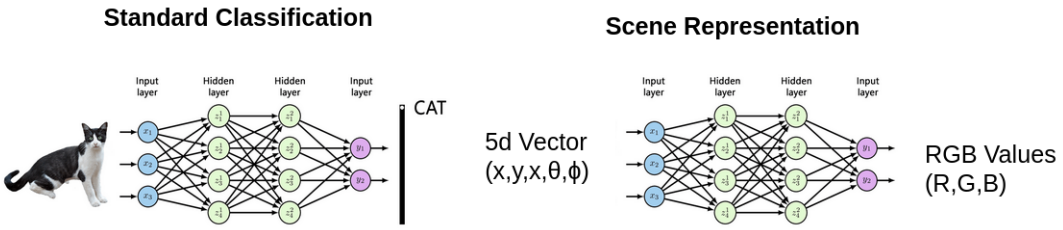

Figure 1: A neural network for standard classification vs. a neural network for scene representation.

1.3 Training NeRFs

In order for the neural network in a NeRF to accurately represent a scene, it must be trained on a dataset with images from many viewing angles of a scene. During training, each image from the dataset is cycled through. To create ground truth for the model, traditional graphics rendering methods are used to create a ray going through the image. The values from this ray are the ground truth for the neural network. The neural network is queried to return the value of the pixel for each position along the ray. The loss penalty is based on the for far the neural networks prediction is from the pixel value in the ground truth ray. Gradient descent is used to update the weights of the model.

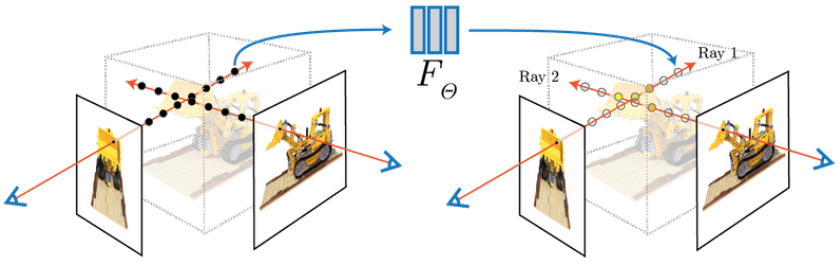

Figure 2: A diagram of training a NeRF with two example training images. The neural network is shown as F_Θ, the ground truth ray is shown on the left, and the predicted ray is shown on the right. Diagram is adopted from [2].

2 Implementation

For my implementation of NeRFs I use the PyTorch deep learning framework [3]. I implemented the neural network as described in the paper. Since the neural network in NeRF is relatively simple, this part was not too challenging. The more difficult part was setting up the pipeline with the data and the training loop. Since I own a Nvidia GPU I was able to train the model. However, without a GPU it would not be feasible to train a NeRF model. Below in Figure 4, I show system information with respect to the computational cost of NeRF. Then, in Figure 5, I show the loss plot over 200,000 training iterations.

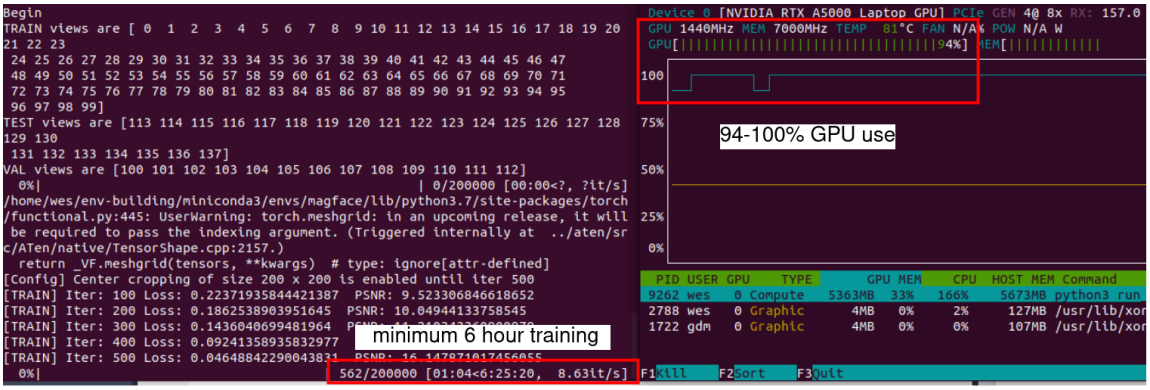

Figure 3: Screenshots of the terminal information from my system while running my NeRF implementation. On the right side of the Figure, it shows logging output. It can be seen that this screenshot is from the start of a run of 200,000 iteration, which is the recommended number of training iterations for the lego bulldozer dataset. The output shows that the training is on pace to take 6 hours 23 miuntes by running at 8.6 iterations per second. In reality, my system would thermal throttle and training took up to 12 hours.
The right side of the Figure shows live GPU information in my system. On my system I have a Nvidia RTX A6000 laptop GPU with 16GB device ram and 6,144 CUDA cores. The maximum theoretical performance of this GPU is 21 teraflops (floating point operations per second), which makes it a high-end consumer GPU. The screenshot shows NeRF uses between 94-100% of the GPU while running, which demonstrates that NeRF is a very computationally expensive method to train.

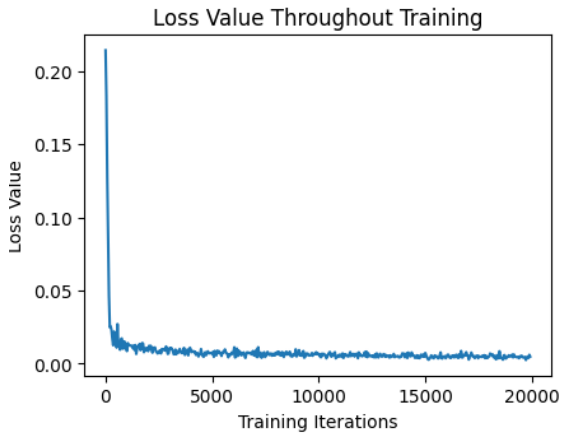

Figure 4: This Figure is a plot of training loss throughout a NeRF training run. A low loss means the model is performing well on the training data. In this Figure, it can be see than in early training iterations (approximately 0-10,000), there is a large drop in training loss. This is due to the fact that originally the model is generating random views. The model is quickly able to learn a *reasonable* representation of the scene. However, after a 10,000 iteration, there are still many unnatural artifacts that seem to 'float' around in the output images. As we approach the 200k iterations, these artifacts slowly disappear.

3 Recent Improvements

Even though NeRF was only published in 2020, it has been rapidly adopted due to the high-quality results. The original paper now has over 800 citations, and improving Neural Rendering Field methods is an active area of research. In this section, I describe a few recent improvements to NeRF. The improvements include broadening the scope of the scene; accelerating NeRF training and inference; and synthesizing 3D scenes from fewer images.

3.1 NeRF in the Wild

While the original NeRF paper created very smooth results, the the results were limited to constricted scenes. For example, in the Lego Bulldozer dataset that I used for training, the background is edited out and replaced with a white background. This allows much simpler training for the neural network because it cannot learn spurious correlations with random objects or lighting changes in the background.

In a paper titled *Nerf in the wild: Neural radiance fields for unconstrained photo collections*, several changes were proposed to handle the problem described in the last paragraph [1]. The main idea of this paper was to 'transient' items in an image from the static items in the image. Transient items include people or birds in the sky, which may appear in one image, but not another. Additionally, the paper uses a separate embedding for appearance which handles different lighting conditions. A diagram of these additions can be seen in the below Figure 5. Example images of novel views with different lighting conditions of an in-the-wild scene can be seen in Figure 6.

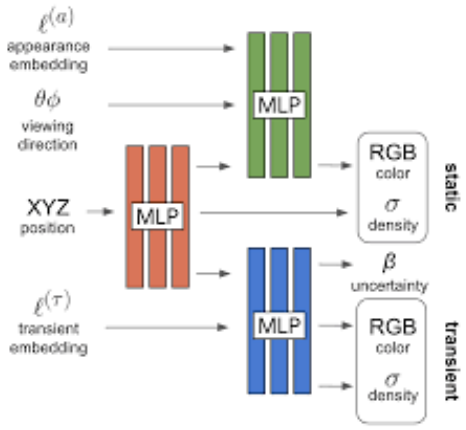

Figure 5: A diagram of the architecture from NeRF in the Wild. Instead only one fully connected neural network, also know as multi-layer-perceptron (MLP), three different neural networks are used. This allow decoupling of lighting condition, transient items, and static items. Figure adopted from [1].

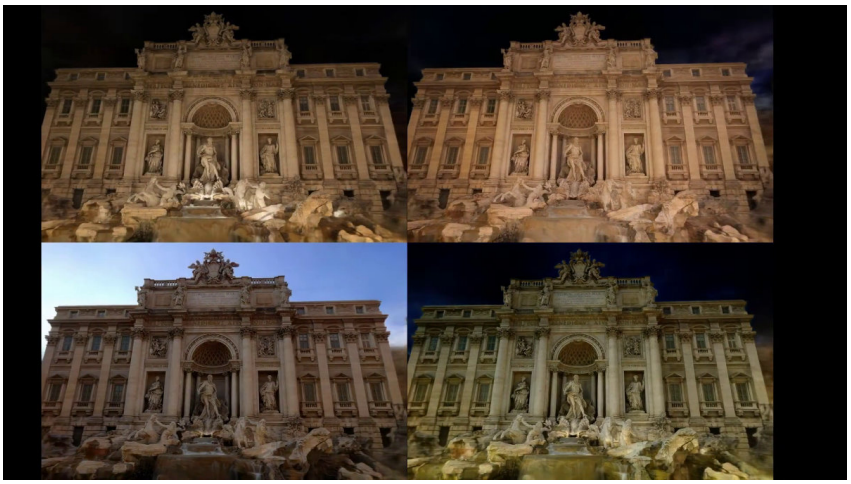

Figure 6: Novel view synthesis from NeRF in the Wild.

3.2 Accelerating NeRF by using Many Neural Networks

Another path of research in neural rendering fields is reducing the computational cost of training and inference. As discussed in Section 2, NeRF models have a high computational cost during both training and inference. For example, on a consumer GPU it took me up to 12 hours to train NeRF model on the Lego Bulldozer dataset, which is a relatively simple scene. At inference, after the model was trained, it

took my system four minutes to generate a 360° view. It is ideal to accelerate training time to reduce carbon footprint and to allow faster iteration. Additionally, if an application needs real time rendering it must run at least 30 frames per second, which is much faster than the original NeRF model can be run.

A method to reduce the computational cost of was proposed in *Kilonerf: Speeding up neural radiance fields with thousands of tiny mlps* [4]. The idea of the kilo NeRF method is to have many small neural networks represent small parts of the scene rather than a single large neural network to represent the entire scene. In Kilonerf, since there are many small neural networks, they can all be parallelized and many pixels and can be generated at once. Additionally, the space complexity is reduced significantly because the space complexity of a neural network is $O(n^2)$ where n is the width of the layers. Since Kilonerf uses much narrower (and shallower) neural networks, much less space is needed on the GPU. The authors claim they are able get over 2,000x speed up from the baseline with their method. The general idea of Kilonerf can be seen in Figure 7.

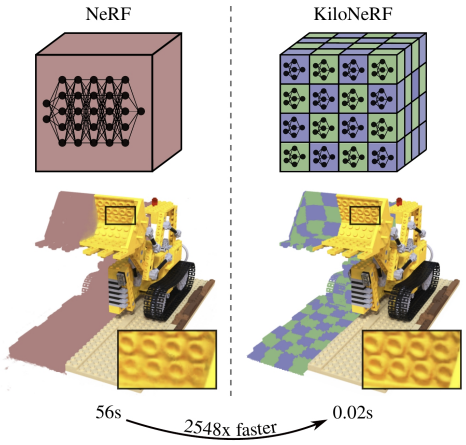

Figure 7: This Figure shows the difference between standard NeRF and Kilonerf.

3.3 Scaling NeRF Scene to Representation to City Blocks

Another limitation of vanilla NeRF is that it is only limited to small scenes. While NeRF in the Wild enabled NeRFs to represent large outdoor scenes, it was still limited to only one scene. An interesting solution to this problem was proposed in *Block-NeRF: Scalable Large Scene Neural View Synthesis* [6]. The Block-NeRF method is proposed by a group of Waymo researchers who have the goal of synthesizing novel views across many city blocks. The visual results from the paper are quite impressive. The model is able to generate a 'fly over' of different blocks with high fidelity. The NeRF model was trained on images takes from different areas around the block. One drawback that still remains of this approach is that the camera locations and viewing angles need to be known exactly. The results of this paper can be found at https://waymo.com/research/block-nerf/.

Figure 8: This Figure shows the general idea of Block-NeRF. Images from many adjacent blocks can be used to train a single NeRF model, which is able to synthesize novel views in any of the city blocks.

3.4 360° Image Synthesis with Few Images

The final path NeRF research I will discuss is decreasing the number of images. In some cases, the number images needed to train a NeRF may be limiting factor. In the Lego Bulldozer, there is 100 training spread evenly across a viewing sphere. Furthermore, the location and viewing angle must be known exactly for each training image. This may not be realistic in some applications. Several works have proposed methods to reduce the number training images needed for NeRF model. One method I found particularly interesting is from a paper titled *Sharf: Shape-conditioned radiance fields from a single view* [5]. In this work they propose a projecting the 2d image into a voxelized representation from a single image. With the voxelized image, they train the NeRF method. The authors show they are able to generate views from multiple that are geometrically faithful to the original image with only a single image as training input. The drawback of this method is that the novel views which have viewing angles significantly different from the original image are not high fidelity. Further, they may not miss details of the object not shown in the single original training image.

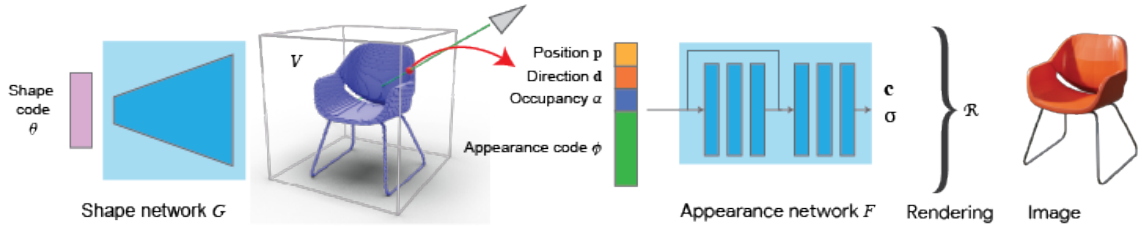

Figure 9: An architecture diagram of the Shape Conditional NeRF method. First, a shape network is used to create a voxelized version of the image. Next, the voxelized image is used to train the neural network and synthesize new views.

4 Conclusion

In this project, a did an overview of Neural Rendering Field methods, also known as NeRFs. NeRFs are a method for novel image view synthesis, which requires a model to generate combine several previous views into a single, novel view. NeRFs have significantly boosted state-of-the-art performance on image synthesis and scene reconstruction benchmarks since they were originally published in 2020. In this report, I over viewed the fundamental concepts Neural Rendering Field Methods. Additionally, I discuss some of the latest improvement in NeRF-based methods and discuss my own implementation of NeRF and the execution time of training on my system. Overall, I enjoyed doing a project that combined deep learning computational geometry.

References

[1] Ricardo Martin-Brualla, Noha Radwan, Mehdi SM Sajjadi, Jonathan T Barron, Alexey Dosovitskiy, and Daniel Duckworth. Nerf in the wild: Neural radiance fields for unconstrained photo collections. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7210–7219, 2021.

[2] 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. In *European conference on computer vision*, pages 405–421. Springer, 2020.

[3] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[4] Christian Reiser, Songyou Peng, Yiyi Liao, and Andreas Geiger. Kilonerf: Speeding up neural radiance fields with thousands of tiny mlps. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 14335–14345, 2021.

[5] Konstantinos Rematas, Ricardo Martin-Brualla, and Vittorio Ferrari. Sharf: Shape-conditioned radiance fields from a single view. *arXiv preprint arXiv:2102.08860*, 2021.

[6] Matthew Tancik, Vincent Casser, Xinchen Yan, Sabeek Pradhan, Ben Mildenhall, Pratul P Srinivasan, Jonathan T Barron, and Henrik Kretzschmar. Block-nerf: Scalable large scene neural view synthesis. *arXiv preprint arXiv:2202.05263*, 2022.