Project 2b: Building Built in Minutes - Nerf

Ankit Mittal
Department of Robotics Engineering
Worcester Polytechnic Institute
Email: amittal@wpi.edu

Rutwik Kulkarni
Department of Robotics Engineering
Worcester Polytechnic Institute
Email: rkulkarni1@wpi.edu

***Abstract*—In this report, we explore the implementation of Nerf a method for synthesizing novel views of complex scenes, leveraging a sparse set of input views to optimize a continuous volumetric scene function represented by a deep fully-connected network. Processing 5D coordinates to predict volume density and view-dependent radiance, this approach employs classical volume rendering to achieve photorealistic renderings.**

I. Introduction

Leveraging a sparse array of input views, our approach optimizes a continuous volumetric scene function represented through a sophisticated fully-connected deep network. This network processes unique 5D coordinates—encompassing both spatial locations (x, y, z) and viewing directions (θ, ϕ)—to predict volume density and direction-dependent emitted radiance accurately.

By querying these 5D coordinates along camera rays, we employ classical volume rendering techniques to compose the resultant images, achieving an unprecedented level of photo-realism in the synthesis of novel views. The core innovation of our approach lies in its optimization process, which is facilitated by the natural differentiability of volume rendering, requiring only a series of images with known camera poses to refine our scene representation. In our comprehensive implementation review, we discuss the challenges encountered, optimizations applied, and the comparative results in different scenarios.[1]

II. Model Input

In our study, we utilize a dataset comprising images of a lego and ship structure, as shown in Figure 1 . Additionally, each image is accompanied by its corresponding camera pose, represented as camera-to-world transformation matrices. Our approach leverages Neural Radiance Fields (NeRF) and classic volume rendering techniques, treating every pixel in the image as a ray in the real world. To prepare our data for the neural network, we first convert all data to world coordinates. This conversion enables us to define the direction of each ray accurately and to compute 3D spatial coordinates by sampling points along these rays. This preprocessing step is crucial for feeding the correct input into our model to generate the desired output.

A. Ray generation

To synthesize views in our dataset, which includes images of a lego structure as illustrated in Figure 1, we generate

Fig. 1: Lego dataset - sample picture

rays originating from each pixel of the image. A ray is mathematically represented by the equation involving 'o' for the origin, 't' as the sampling parameter, and 'd' for the direction. For our purposes, the origin 'o' is determined by the position of the pixel within the image, and the direction 'd' is defined by a unit vector pointing from the camera center to the specific pixel position.

Initially, our values are confined to the 2D plane of the image and expressed in pixel coordinates. To transform these into rays suitable for our model, we follow these steps:

1) Transform the pixel coordinates into normalized coordinates relative to the camera center. This involves adjusting the coordinate frame to (X, -Y, -Z) as per the COLMAP convention, with an assumption that Z = -1.
2) Calculate the ray direction relative to the camera frame by first obtaining the vector in the camera's frame. Then, by multiplying this vector by the rotation part of the camera-to-world transformation matrix, we convert this vector into the world frame. The ray direction in unit vector form is achieved by normalizing this vector, i.e., dividing it by its magnitude.
3) The origin of the ray in world coordinates is identified simply by the translation component of the camera-to-world transformation matrix.

These steps ensure that we accurately generate rays from the image pixels to the world frame, allowing us to process and render the scene with our Neural Radiance Field model.

B. Sample Points

Having defined the direction and origin for each ray, the remaining element to specify is the sampling parameter, which determines how we sample points along the ray. To refine our model's ability to synthesize and render complex scenes, we incorporate uniform sampling along the ray's path, enhanced with the addition of some random noise. This technique of noise addition is aimed at introducing variability, ensuring that the model encounters a wider array of data during training. This variability is crucial for improving the model's performance, as it simulates a more diverse set of lighting and viewing conditions, leading to more robust and realistic outputs. The sampling strategy involves calculating midpoints along the ray, establishing intervals around these points, and then randomly selecting points within these intervals to simulate the effect of noise. In order to determine the midpoint for sampling along each ray, we start with two predefined points: the near ray point (hn=2) and the far ray point (hf=6). We then uniformly sample 192 points between these two bounds along the ray, introducing randomness into the sampling process to incorporate noise. This technique ensures that each ray is sampled with a distribution that simulates realistic lighting and material interactions within the scene. This approach helps in accurately capturing the nuances of light interaction within the scene, ultimately contributing to the generation of high-quality, photorealistic images.

C. Position Encoding

In the seminal work, it was observed that employing positional encoding to represent high-frequency variations in both color and geometry yields improved results compared to operating on the raw spatial coordinates (x, y, z) and viewing directions (θ, ϕ). Positional encoding is utilized to enhance the network's capability in modeling fine details by transforming low-dimensional input coordinates into a higher-dimensional space. This transformation is realized through the function $\gamma(p)$, which maps a scalar input p to a vector of sinusoidal functions, as follows:

$$\gamma(p) = \big[\sin(2^0\pi p), \cos(2^0\pi p), \sin(2^1\pi p), \cos(2^1\pi p), \ldots, \sin(2^{L-1}\pi p), \cos(2^{L-1}\pi p)\big]$$

where L denotes the number of frequency octaves. The use of sinusoidal functions of exponentially increasing frequencies – where 2^l indicates the frequency scaling factor for the l-th octave – allows the encoding to capture complex patterns across different scales. The inclusion of both sine and cosine terms for each frequency octave ensures a complete and rich representation of periodic functions within the encoded space. This encoding is applied separately to each of the spatial coordinates (L = 10) and direction vector components(L = 4), resulting in a nuanced, high-dimensional representation that significantly enhances the neural network's performance in rendering photorealistic images with high fidelity.

III. Network -MLP

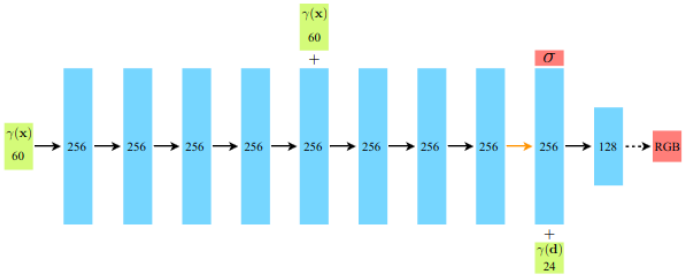

Fig. 2: Network Architecture

A visual representation of our fully-connected neural network is provided. The architecture comprises input vectors in green, intermediate hidden layers in blue, and output vectors in red, with the dimension of each vector indicated inside its corresponding block. All layers are fully-connected with black arrows denoting ReLU activations, orange arrows for layers without activation, dashed black arrows for sigmoid activations, and a "+" symbol representing vector concatenation.

The positional encoding of the input location, $\gamma(\mathbf{x})$ and raw spatial coordinates (x, y, z) (input vector size = 60 +3), is processed through eight fully-connected layers with ReLU activations, each consisting of 256 channels. Adhering to the DeepSDF architecture, a skip connection is implemented, appending the encoded input to the activation of the fifth layer. The network includes an additional layer that yields the volume density σ, which is subjected to a ReLU function to guarantee nonnegative density values, along with a 256-dimensional feature vector.

This feature vector is combined with the positional encoding of the input viewing direction, $\gamma(\mathbf{d})$ & raw view direction vector (input vector size = 24 + 3), and then forwarded through another fully-connected ReLU layer with 128 channels. The concluding layer, applying a sigmoid activation, computes the emitted RGB radiance at location $\mathbf{x}$ from a ray with direction $\mathbf{d}$.

A. Network Parameter

Sr. No.	Parameter	Value
1	Epochs	3
2	MiniBatchSize	1024
3	Learning rate	0.0005
4	Image Size	400 X 400
5	Optimizer	Adam

TABLE I: Network Parameter

IV. Volume Rendering

The network's output comprises the RGB color value and the volume density at a specific location. These predicted values are utilized to render the 3D scene. The predictions

are integrated into the classical volume rendering equation to compute the color of a specific point in space. The volume rendering equation is typically expressed as follows:

$$C(\mathbf{r}) = \int_{t_n}^{t_f} T(t)\sigma(\mathbf{r}(t))\mathbf{c}(\mathbf{r}(t), \mathbf{d})\, dt$$

where $C(\mathbf{r})$ represents the resulting color along the ray $\mathbf{r}$, t_n and t_f define the near and far bounds of integration along the ray, $T(t)$ is the accumulated transmittance from t_n to t, σ is the volume density at position $\mathbf{r}(t)$, and $\mathbf{c}$ is the emitted radiance (color) at position $\mathbf{r}(t)$ for a ray traveling in direction $\mathbf{d}$. The integration accumulates the contribution of light at each point along the ray, modulated by the transmittance and density to determine the final pixel color.

V. Loss Function

After completing 3D volume rendering to obtain all the RGB color values, we proceed to calculate the photometric loss between these predicted color values and the actual image values, employing the Sum of Squared Differences (SSD) loss for this purpose.

$$\mathcal{L} = \sum_i \left\| I_i - \hat{I}_i \right\|_2^2 \tag{1}$$

VI. Training

We received two distinct datasets for our study, one consisting of lego images and the other of ship images, with each dataset containing 100 images. We trained our model using all the images from both datasets with the specified parameters. Additionally, we conducted an experiment on the LEGO dataset without incorporating positional encoding to understand its impact. The outcomes of these training sessions are discussed and compared in the results section.

VII. Results

In the initial section, we will present a comparative evaluation of the test-set views from synthetic scenes produced through a physically-based rendering method.

A. Lego

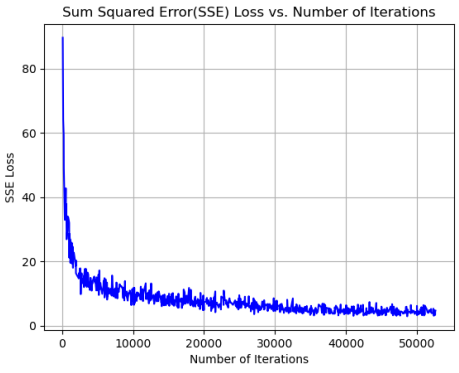

Fig. 3: SSD vs Iterations

(a) Ground Truth (b) Rendered view

Fig. 4: Lego Dataset.

	PNSR	SSIM
Lego (Test Set)	24.67	0.74

TABLE II: Avg PNSR and SSIM

B. Lego(without position encoding)

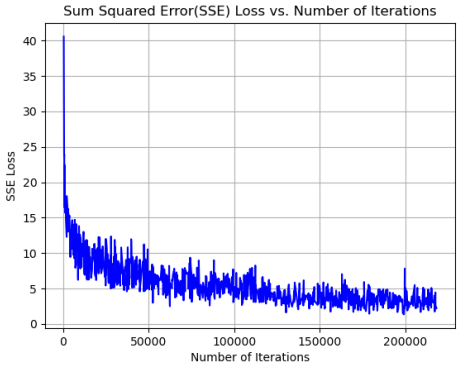

Fig. 5: SSD vs Iterations

(a) Ground Truth (b) Rendered view

Fig. 6: Lego Dataset.

	PNSR	SSIM
Lego (Test Set)	21.77	0.63

TABLE III: Avg PNSR and SSIM

C. *ship*

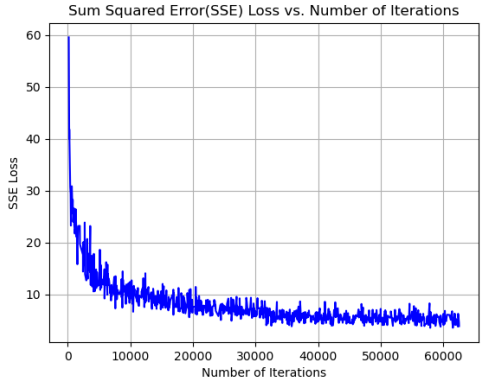

Fig. 7: SSD vs Iterations

(a) Ground Truth (b) Rendered view

Fig. 8: Lego Dataset.

	PNSR	SSIM
Ship (Test Set)	23.11	0.70

TABLE IV: Avg PNSR and SSIM

VIII. Acknowledgment

The author would like to thank Prof. Nitin Sanket, Teaching Assistant, and Grader of this course RBE549- Computer Vision.

References

[1] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, "Nerf: Representing scenes as neural radiance fields for view synthesis," *CoRR*, vol. abs/2003.08934, 2020. [Online]. Available: https://arxiv.org/abs/2003.08934