P2 Phase2: NeRF (Neural Radiance Fields)

Sarthak Mehta
Department of Robotics Engineering
Worcester Polytechnic Institute
smehta1@wpi.edu

Prasham Soni
Department of Robotics Engineering
Worcester Polytechnic Institute
psoni@wpi.edu

***Abstract*—This report details the implementation of a NeRF (Neural Radiance Fields) pipeline for reconstructing a 3D scene from multiple 2D images. The process involves ray sampling, estimating the NeRF function through deep learning, and rendering the final scene using volume rendering.**

I. Phase2: NeRF

The fundamental goal of NeRF (Neural Radiance Fields) is to model how a given scene would appear from novel viewpoints by leveraging the spatial relationships between captured images. Given a set of posed images, where each image is associated with its camera's location and orientation, NeRF learns to synthesize novel views by estimating the scene's radiance and volumetric density at continuous spatial coordinates.

The NeRF model takes a 3D point along a ray and its corresponding viewing direction as input and predicts four outputs:

- r, g, b (color values): Representing the radiance emitted from that point in the given viewing direction.
- σ (density value): Indicating the probability of light being absorbed at that location.

Once these outputs are obtained, volume rendering is applied to aggregate the contributions of sampled points along each ray. By integrating radiance and density across the ray, NeRF produces realistic images from novel viewpoints, even if those perspectives were not explicitly present in the training data. This pipeline enables high-fidelity 3D scene reconstruction and view synthesis, making NeRF a powerful technique for scene representation.

II. Data Generation

A. Ray Generation for NeRF

Given an image along with its corresponding camera location, orientation, and transformation matrix, we generate rays that serve as the foundation for learning the scene representation in NeRF. The ray generation process follows these steps:

B. Camera Parameterization

Each image is associated with intrinsic parameters (focal length, principal point) and extrinsic parameters (camera position and orientation). The transformation matrix, which encodes the camera's pose in world coordinates, allows us to map image pixels to 3D space.

C. Ray Formation

For each pixel in the image, a primary ray is generated by computing its origin and direction. The camera center serves as the origin for all rays, while the direction is determined by unprojecting pixel coordinates into 3D space using the camera's intrinsic and extrinsic parameters.

D. Ray Parameterization

Each ray is parameterized as:

$$\mathbf{r}(t) = \mathbf{o} + t\mathbf{d} \tag{1}$$

where $\mathbf{o}$ is the camera origin, and $\mathbf{d}$ is the normalized ray direction computed from the camera transformation and intrinsic parameters.

This formulation defines the trajectory of rays in 3D space, forming the basis for subsequent scene reconstruction and rendering.

III. Ray Sampling

Once rays are generated, the next step is to sample points along each ray to estimate scene properties effectively. Sampling determines the spatial locations where the NeRF model evaluates radiance and density, which are crucial for volume rendering. Two sampling strategies were employed: uniform sampling and dynamic sampling.

A. Uniform Sampling

In uniform sampling, points are evenly distributed along the ray within a predefined range. This ensures a consistent representation of the scene along the entire ray. Specifically, the sampling was performed within the depth range $[0.1, 5]$ with a fixed number of samples per ray:

- Depth range: $[0.1, 5]$
- Number of samples per ray: 64

This approach provides a broad but coarse representation of the scene, which is useful for capturing global structure but may not be optimal for regions with high-frequency details.

1) Sampling Issues and Black Image Output: One of the primary reasons for incorrect output during the ray sampling process was the improper handling of radiance and density values. Specifically, the following factors contributed to the issue:

- The RGB values predicted by the NeRF model were typically small.

- The density values for each sampled point along a ray were nearly constant across all samples.

Since the final rendered color for each pixel is computed using the accumulated radiance and transmittance along the ray, the product of small RGB values and uniform density outputs resulted in extremely low intensity values. This effect was compounded further, leading to an output that appeared predominantly black.

B. Dynamic Sampling

Instead of using a fixed, uniform sampling strategy, an adaptive approach was introduced to focus sampling on regions of interest.

- Depth range: $[2, 6]$
- Number of samples per ray: 179

This adaptive approach enhances detail in targeted regions, improving rendering quality and reducing redundancy in less significant areas.

By leveraging both uniform and dynamic sampling, the model efficiently balances global structure capture and fine-detail refinement.

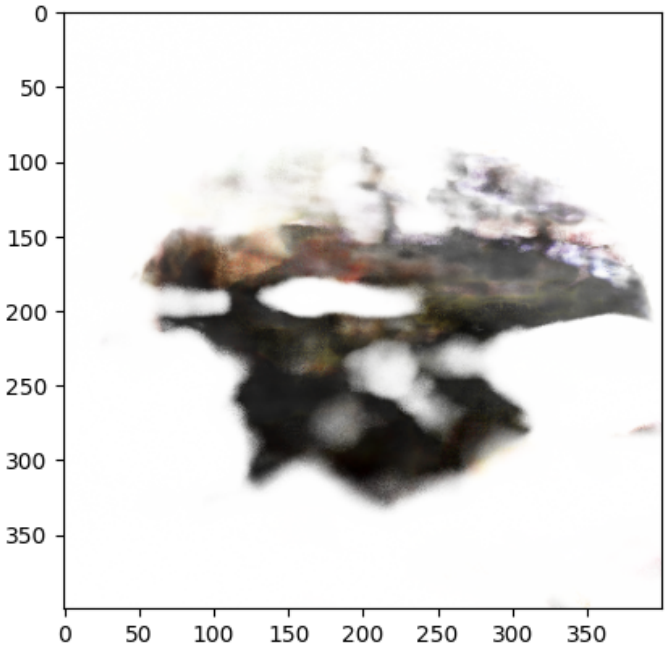

Fig. 1. Ship output with same parameters as Lego Set

While these improvements enhanced rendering quality for *Lego*, the reconstruction of the *ship* scene remained insufficient. Further refinements in sampling strategies, network architecture, and loss functions may be required to achieve high-fidelity results.

To address this issue and improve rendering quality, the dynamic sampling strategy was adjusted. The initial sampling configuration of $[2, 6]$ with 179 samples per ray failed to generate a proper reconstruction for the *ship* scene. To improve results, the dynamic sampling range was modified to:

- **Updated Depth Range:** $[0.8, 5]$
- **Number of Samples per Ray:** 360

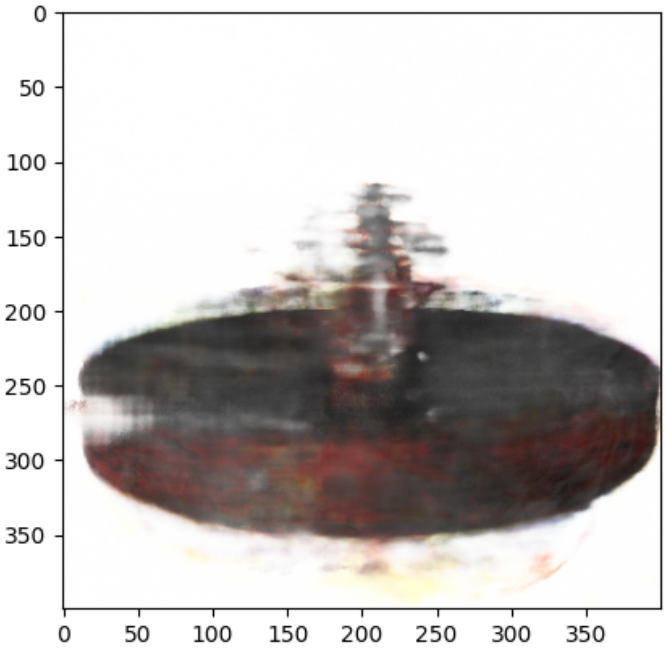

Fig. 2. Ship output with updated parameters

IV. Model Training

This section describes the methodology and strategies used in training our model. We provide details on the parameters as well as the hyper-parameters that influence the training process.

A. Parameters

In this section, we outline the key model parameters. These parameters include the weights, biases, and other trainable components that are initialized and updated during training. Further details on their initialization, regularization, and role in the learning process are discussed below.

B. Hyper-parameters

Hyper-parameters are crucial in controlling the training dynamics and final model performance. Here, we cover two important aspects: positional encoding and additional observations regarding the optimization strategies.

1) Positional Encoding: Positional encoding is implemented to incorporate sequential information into the model. This encoding allows the network to be aware of the relative or absolute positions of tokens in the input sequence. In our work, we adopt the standard sinusoidal positional encoding as described in [**?**], which is computed as:

$$\mathrm{PE}_{(pos,2i)} = \sin\left(\frac{pos}{10000^{\frac{2i}{d_{\text{model}}}}}\right), \quad \mathrm{PE}_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{\frac{2i}{d_{\text{model}}}}}\right)$$

where pos is the position and i is the dimension index.

2) Additional Observation: Additional observations were made during the optimization phase of training. Here, we compare two popular optimization strategies:

a) Using SGD: Stochastic Gradient Descent (SGD) was initially employed due to its simplicity and efficiency in large-scale problems. The learning rate and momentum were tuned to ensure stable convergence. Our experiments indicate that while SGD provides a solid baseline, its performance is sensitive to the choice of learning rate, particularly in deeper architectures.

b) Using ADAM: The ADAM optimizer was also utilized as an alternative to SGD. ADAM combines the advantages of two other extensions of SGD, namely Adaptive Gradient Algorithm and Root Mean Square Propagation. With adaptive learning rates for each parameter, ADAM has been observed to converge faster in practice. We adjusted the beta parameters and learning rate to optimize performance, resulting in smoother and more reliable convergence compared to vanilla SGD.

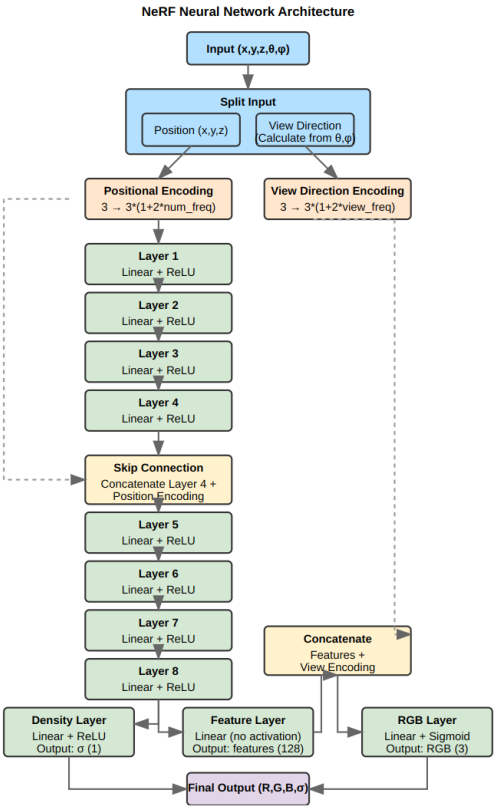

Fig. 3. Architecture

V. Volume Rendering

Volume rendering is a fundamental component of NeRF, enabling the synthesis of novel views from a continuous volumetric representation. In our implementation, the core idea is to compute the color along each ray by integrating the contribution of each sampled point in the scene.

A. Basic Mathematics

The color $C(\mathbf{r})$ along a ray $\mathbf{r}(t)$ is computed by accumulating contributions from each point along the ray. This is formalized by the following integral:

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

where:

- t_n and t_f are the near and far bounds along the ray.
- $\sigma(\mathbf{r}(t))$ represents the volume density at point $\mathbf{r}(t)$.
- $c(\mathbf{r}(t),\mathbf{d})$ denotes the color at $\mathbf{r}(t)$ given the ray direction $\mathbf{d}$.
- $T(t)$ is the accumulated transmittance up to t, defined as:

$$T(t) = \exp\left(-\int_{t_n}^{t} \sigma(\mathbf{r}(s))\,ds\right).$$

In practice, this continuous integral is approximated by a discrete sum over sampled points along the ray. This discretization, often enhanced by stratified or importance sampling, is at the core of our code implementation.

B. Crucial Role in NeRF

The volume rendering formulation is critical for NeRF for several reasons:

1) **Differentiable Rendering:** The continuous formulation of volume rendering allows the use of gradient-based optimization to update the model parameters. By back-propagating through the rendering integral, the network learns an implicit representation of the scene.
2) **Continuous Scene Representation:** Instead of relying on discrete voxels or meshes, NeRF represents scenes as continuous functions. This continuous nature is efficiently captured through the integration over the scene's volume, enabling high-quality novel view synthesis.
3) **Handling Complex Effects:** The formulation naturally incorporates view-dependent effects and occlusions. The exponential transmittance term $T(t)$ models the accumulation of absorption along the ray, ensuring that contributions from occluded or distant points are appropriately diminished.

Overall, the mathematical foundation of volume rendering not only ensures that our approach remains theoretically sound but also directly informs the design of the code, resulting in robust and high-fidelity view synthesis.

VI. Results

A. Loss Function

1) Mean Squared Error (MSE) Loss: MSE loss measures the pixel-wise difference between the predicted and ground truth images by computing the squared difference between corresponding pixel values. It is primarily used during training to update the model's weights and biases, ensuring that the reconstructed images closely match the ground truth.

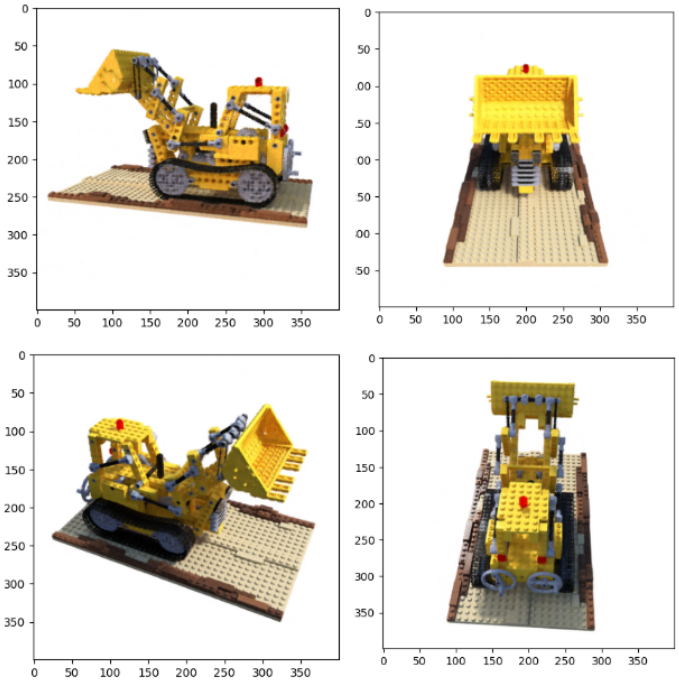

Fig. 4. NeRF: Lego Output

2) Peak Signal-to-Noise Ratio (PSNR): PSNR is computed for the test output images to quantify the quality of reconstruction. A higher PSNR value indicates lower reconstruction error and better image fidelity, making it a useful metric for assessing how well the model generalizes to unseen viewpoints.

3) Structural Similarity Index Measure (SSIM): SSIM is calculated for the test output images to evaluate the structural differences between the predicted and ground truth images. Unlike MSE and PSNR, SSIM considers luminance, contrast, and structural information, making it a more perceptually relevant measure of image quality.

Loss	Lego	Ship
PSNR: Avg Loss over 200 images (in dB)	32.188	9.9012
SSIM: Avg Loss over 200 images	0.1966	0.4872

TABLE I
LOSS TABLE FOR PSNR SSIM

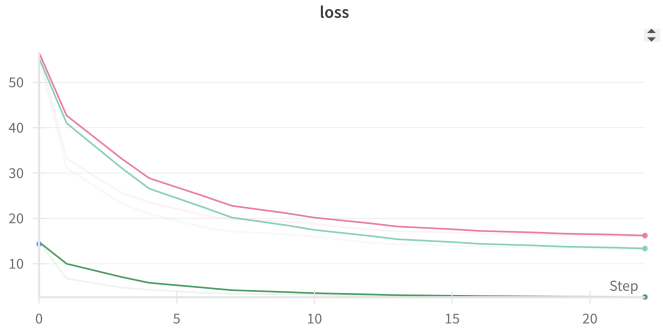

Fig. 5. NeRF: Lego: Epoch v/s Training Loss (This is for 3 different runs, green is the optimal one)