NeRF - Neural Radiance Fields for View Synthesis RBE549 Project 2

Sumukh Porwal, Piyush Thapar
MS Robotics Engineering
Worcester Polytechnic Institute
Email: sporwal@wpi.edu, pthapar@wpi.edu

***Abstract*—This report presents a comprehensive investigation into a deep learning technique for reconstructing 3D scenes using the Neural Radiance Field (NeRF) method. NeRF employs a fully connected (non-convolutional) deep network that takes as input a continuous 5D coordinate—comprising the spatial location (x, y, z) and the viewing direction (θ, ϕ)—and outputs both the volume density and the view-dependent radiance at that location.**

I. Introduction

NeRF introduces a novel method for view synthesis by optimizing the parameters of a continuous 5D scene representation to reduce rendering error across a set of captured images. It generates new views by querying 5D coordinates along camera rays and then applies traditional volume rendering techniques to project the resulting colors and densities onto an image. Since volume rendering is inherently differentiable, only images with known camera poses are needed to optimize the representation. This work explains how to effectively adjust neural radiance fields to render photorealistic novel views of scenes with complex geometry and appearance, achieving results that surpass earlier approaches in neural rendering and view synthesis. A physical depiction of NeRF is provided in Figure 1.

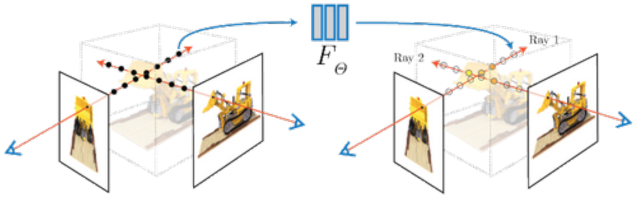


Fig. 1. NeRF

II. Methodology

We model the continuous 5D scene using a multilayer perceptron (MLP) network F_Θ, which maps an input coordinate (x, y, z, θ, ϕ) to an output tuple (r, g, b, σ). Here, σ represents the volume density and (r, g, b) the RGB color, both as functions of the spatial position (x, y, z) and the viewing direction (θ, ϕ). The weights Θ are optimized so that each 5D input is accurately associated with its corresponding density and view-dependent color.

A. Volume Rendering with Radiance Fields

The 5D neural radiance field encapsulates a scene by specifying the volume density and the directionally emitted radiance at every point. To render the color along any ray passing through the scene, we rely on classical volume rendering principles. In this context, the volume density $\sigma(x)$ is interpreted as the differential probability of a ray terminating at an infinitesimally small particle at location x. The expected color $C(r)$ of a camera ray $r(t) = o + td$, with near and far bounds t_n and t_f, is defined as:

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

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

Here, $T(t)$ represents the cumulative transmittance along the ray from t_n to t, indicating the probability that the ray travels from t_n to t without interacting with any particle. To render a view from our continuous neural radiance field, it is necessary to numerically approximate the integral $C(r)$ for each ray corresponding to a pixel in the virtual camera. This is achieved by applying quadrature through stratified sampling: the interval $[t_n, t_f]$ is divided into N equal bins, and one sample is uniformly drawn from each bin:

$$t_i \sim U\left(t_n + \frac{i-1}{N}(t_f - t_n),\, t_n + \frac{i}{N}(t_f - t_n)\right).$$

This stratified approach allows the MLP to be evaluated at continuously varying positions during optimization, leading to:

$$\hat{C}(r) = \sum_{i=1}^{N} T_i\,(1 - \exp(-\sigma_i\,\delta_i))\,c_i,$$

where $\delta_i = t_{i+1} - t_i$ denotes the distance between consecutive samples. This formulation, which computes $C(r)$ from the set of (c_i, σ_i) values, is fully differentiable and corresponds to standard alpha compositing with $\alpha_i = 1 - \exp(-\sigma_i\,\delta_i)$.

B. Dataset

We employ the dataset provided by the original authors. A parser function is used to read data from a JSON file, process and load the images along with their associated metadata, and prepare the information for further analysis. This function outputs details about camera parameters, images, poses, focal lengths, and similar information for the test data.

C. *Pixel To Ray*

A pinhole camera model is utilized, where the camera matrix and the transformation between the camera pose and the world coordinate system are used to compute the origin and the unit direction vector for each ray. NeRF requires that each pixel in the image correspond to a ray. In our implementation, we downscale the original 800×800 images to 400×400, resulting in $160\,000$ rays per image.

D. *Volume Rendering*

The network outputs RGB values and volume densities for multiple sample points along each ray. For each sample, the NeRF model is queried to obtain the corresponding color and density, after which volume rendering techniques are applied to compute the final RGB image. This process involves calculating alpha values based on density and inter-sample distance, which are then used as weights to blend the RGB values along the ray, ultimately forming the rendered image.

E. *Model Architecture*

Our model follows the architecture detailed in the original paper, consisting of eight fully connected layers. A skip connection is introduced by concatenating the original input with the activation of the fifth layer. Following an additional four fully connected layers, the network produces both the sigma output and a feature map. This feature vector is then concatenated with the positional encoding of the viewing direction, $\gamma(d)$, and processed through an extra fully connected ReLU layer with 128 channels. Finally, a sigmoid-activated layer outputs the emitted RGB radiance at position x for a ray directed along d. Consistent with the original setup, the positional encoding dimensions are set to 10 for the ray query points and 4 for the ray directions. The overall architecture is depicted in Figure 2.

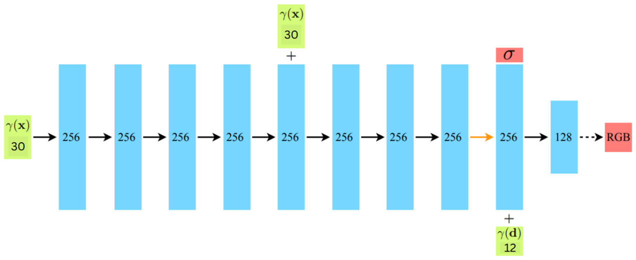


Fig. 2. Network Architecture

F. *Training Parameters*

During training, rays are generated for all images in the dataset. A random sample of 4096 rays (the batch size) is selected from the complete set, ensuring that each batch contains rays from multiple viewpoints, which helps prevent overfitting to any single image. Training over $100\,000$ iterations required roughly 24 hours per dataset, with Pytorch's automatic mixed precision enabled for memory efficiency. The hyperparameters used for training NeRF are as follows:

- Learning Rate: 5×10^{-4}
- Optimizer: Adam
- Near bound (t_{near}): 2
- Far bound (t_{far}): 6
- Batch Size: 4096
- Number of Query Points per Ray: 192

G. *Custom Dataset*

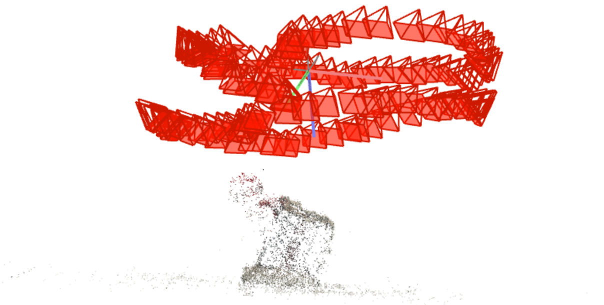


Fig. 3. Colmap for Custom Dataset

III. Results

The best performance was achieved at $95\,000$ iterations for the LEGO dataset and $69\,000$ iterations for the SHIP dataset. The evaluation of the trained model on the test set is summarized below in terms of PSNR and SSIM:

1) **LEGO:**
    - Average PSNR: 25.6451
    - Average SSIM: 0.8939
2) **SHIP:**
    - Average PSNR: 27.0990
    - Average SSIM: 0.8414
3) **SPIDEY:**
    - Average PSNR: 31.3224
    - Average SSIM: 0.8715

Comparision of Ground Truth and NeRF Renderings are given in the below tables.

TABLE I
Comparison of Ground Truth and NeRF Renderings: LEGO

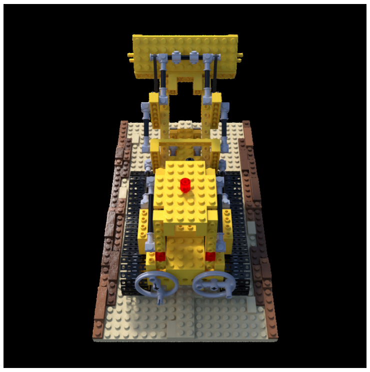
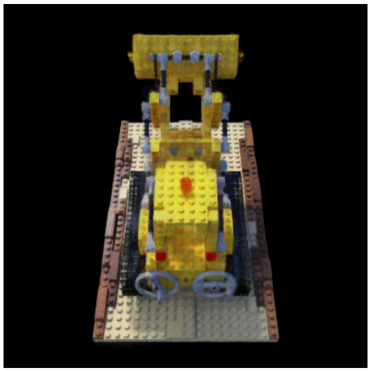
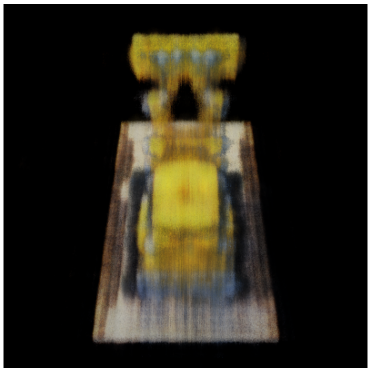
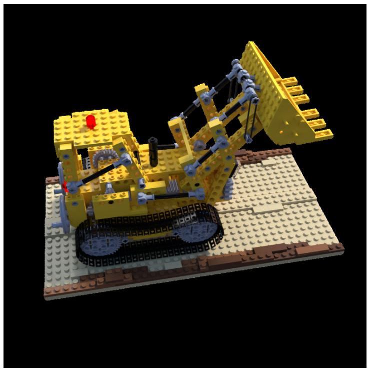
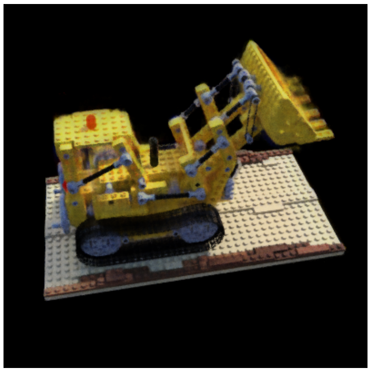
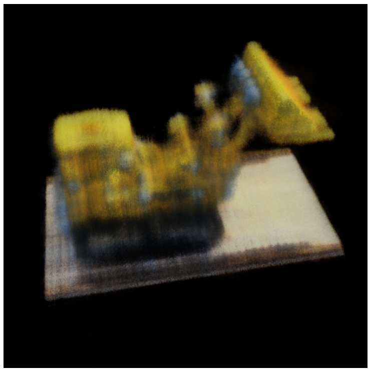
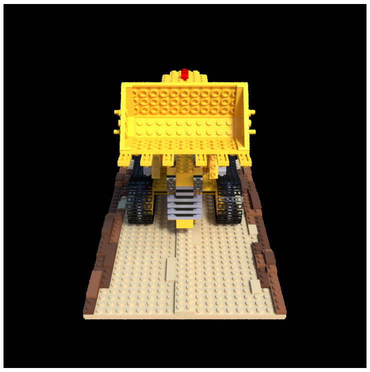
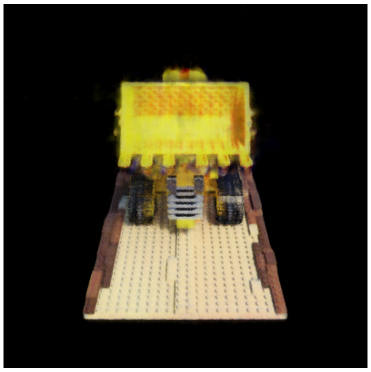
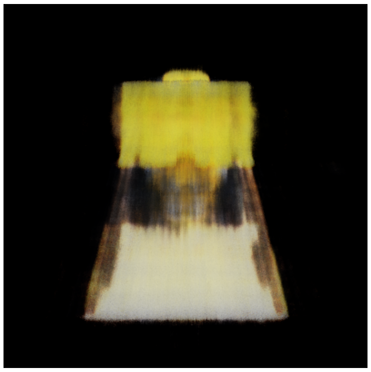
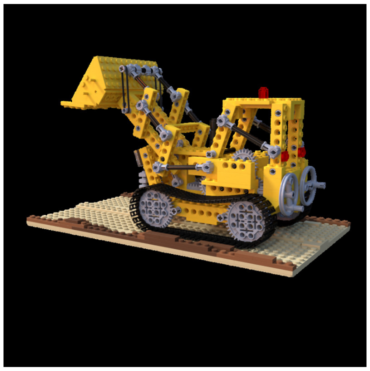
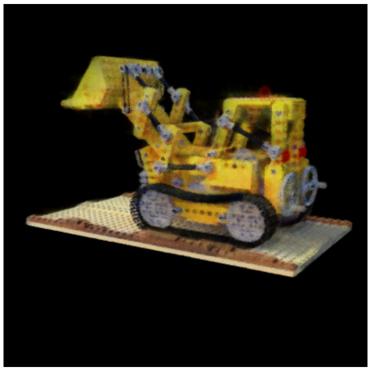
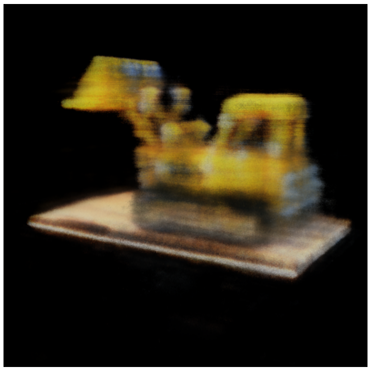
Ground Truth	NeRF (with positional encoding)	NeRF (without positional encoding)
		
		
		
		

TABLE II
Comparison of Ground Truth and NeRF Renderings: SHIP

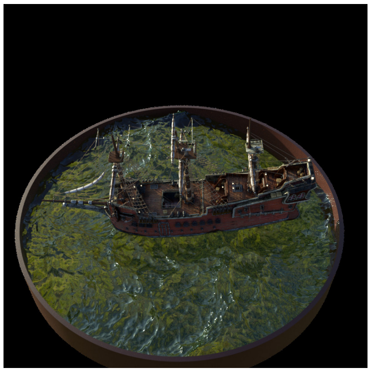
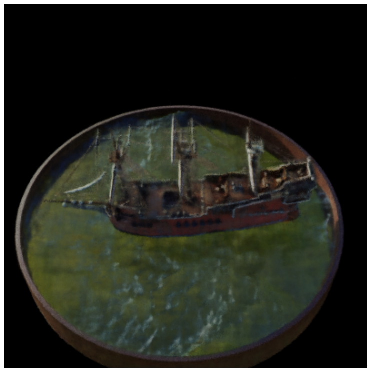
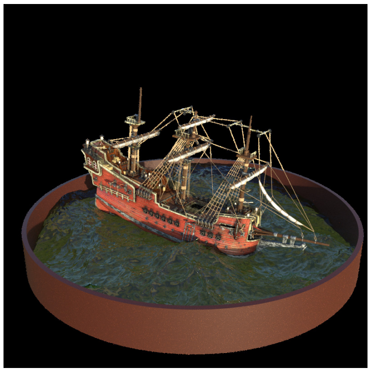
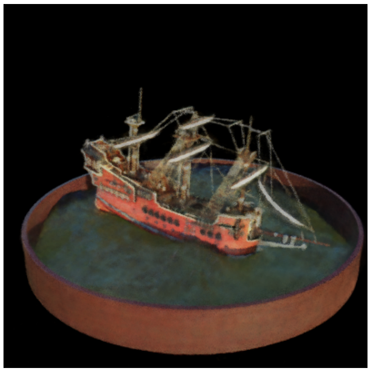
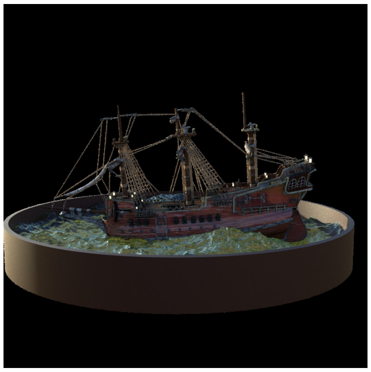
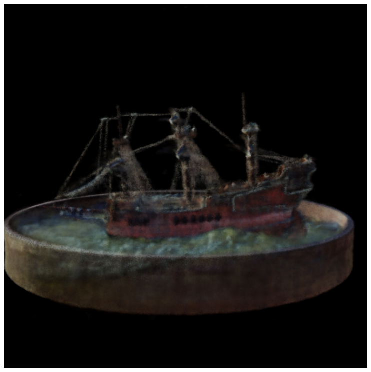
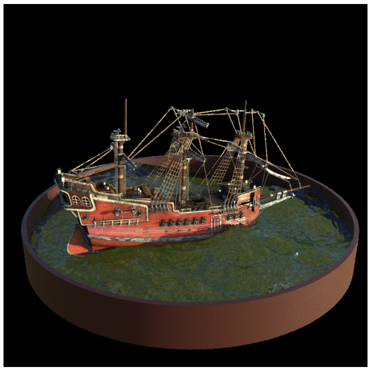
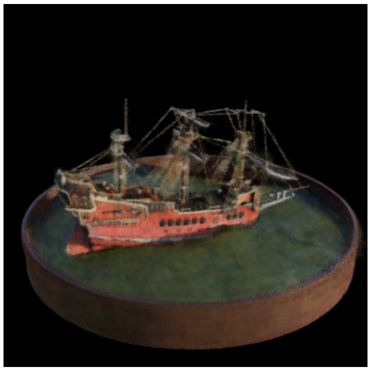
Ground Truth	NeRF
	
	
	
	

TABLE III
Comparison of Ground Truth and NeRF Renderings: SPIDEY

Ground Truth	NeRF