Project 2 - Buildings built in minutes - SfM and NeRF
Phase 2

Manoj Velmurugan*, Rishabh Singh†
Robotics Engineering
Worcester Polytechnic Institute
Email: *v.manoj1996@gmail.com, †rsingh8@wpi.edu

***Abstract*—This project introduces an effective implementation of Neural Radiance Fields (NeRF) for 3D scene reconstruction using 2D images. Utilizing deep learning, NeRF synthesizes photorealistic scenes by interpreting light and color data. The approach combines spatial details with color information, leading to high-quality renderings.**

I. Introducion

In recent years, the field of computer vision has witnessed remarkable advancements, with deep learning playing a pivotal role in these developments. One of the most significant breakthroughs is the concept of Neural Radiance Fields (NeRF), introduced in the groundbreaking paper "NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis." This technique revolutionizes the way we reconstruct and understand 3D scenes from 2D images.

In this project, we understand NeRF by implementing the methodology as described in the original paper. Our objective is to not only replicate but also to understand and validate the viability of this approach in a real-world context. To achieve this, we utilize two types of datasets: the widely recognized synthetic LEGO dataset to verify and compare our work with others and then to use our custom dataset collected for this project.

II. Summary

- NeRF algorithm was implemented from scratch. Our implementation is slightly different and uses direction vector instead of using azimuth and elevation.
- The very first implementation of our NERF did raytracing batchwise for each image. It should run faster theoretically but it required more than 24 GB VRAM. Hence we had to breakdown (chunk) the rendering process of each image and stick them later (for loop).
- Training loss and images were logged using wandb to get realtime stats.
- Different learning rates, encoding (positional and no positional encoding), sampling (uniform and uniform random) were tested for ship and lego datasets. Decent performance was obtained for test dataset.
- **(Bonus)** Lastly we performed the training on a custom dataset of a flip-flop, collected using phone and colmap.

III. Dataset

In this project, we initially utilized a synthetic dataset created with Blender, which was provided by the original authors of the NeRF technique. To evaluate our model, we selected two distinct datasets from this collection. The first one we chose was the "LEGO" dataset. Following that, we opted to work with the "Ship" dataset. We wrote our own parser to work with this dataset.

Fig. 1: LEGO Dataset

Fig. 2: Ship Dataset

IV. Generating Rays

We employ a traditional volume rendering technique, viewing each pixel in an image as a ray within the 3D space. We begin by transforming the pixel coordinates (u, v) into

Parameter	Value
Optimizer	Adam
Learning Rate	5e-4
Image Size	(100, 100)
Frequency Count	64
Loss	MSE
GPU	H100 (turing), RTX3090
Batch Size	10000

TABLE I: Network Training Parameters

normalized coordinates (X, Y, 1), relative to the camera's center.

The formula for a standard ray can be expressed as: $r(t) = o + td$. In this equation, 'o' represents the ray's origin (the position of the image's pixel in the 3D world), and 'd' signifies the ray's direction (a unit vector from the camera's center to the pixel on the image). We sample this ray in discrete points.

Once we have established the ray's direction in relation to the camera frame, we apply a rotation matrix to get the ray in 'world frame'. We then normalize to a unit vector for further processing.

V. Position Encoding

Directly inputting point coordinates into the network leads to sub-optimal outcomes. This issue arises because the network tends to focus on learning low-frequency features, often overlooking higher frequencies. To counteract this, we employ positional encoding - the sines and cosines of the point coordinates at various frequencies - as inputs, rather than the raw coordinates themselves.

VI. Implementing the Network

The network architecture is provided in Fig. 3.

Dropout (p = 0.25) was later added. But it did not make a big difference. The hyper-parameters that were found to work well are given in TABLE I.

For any higher learning rate, the network was not generating anything.

The network used to get stuck without learning anything. Correcting the coordinate frames as used in the original dataset/paper helped. Additionally, to avoid image pixels getting stuck at 0 after relu, a small positive offset was added. Noise was added to σ estimates as done in nerf-pl work. We used Leaky-ReLU to allow for good gradient propagation when the values are negative.

The training loss curve is given in fig. 4

VII. Volumetric Rendering

The Neural Radiance Fields (NeRF) network produces outputs that include RGB values and the volume density based on the given input location and camera direction. These outputs are then integrated into the volume rendering formula to determine the color values at specific world coordinates. The standard formula for volume rendering is presented as follows:

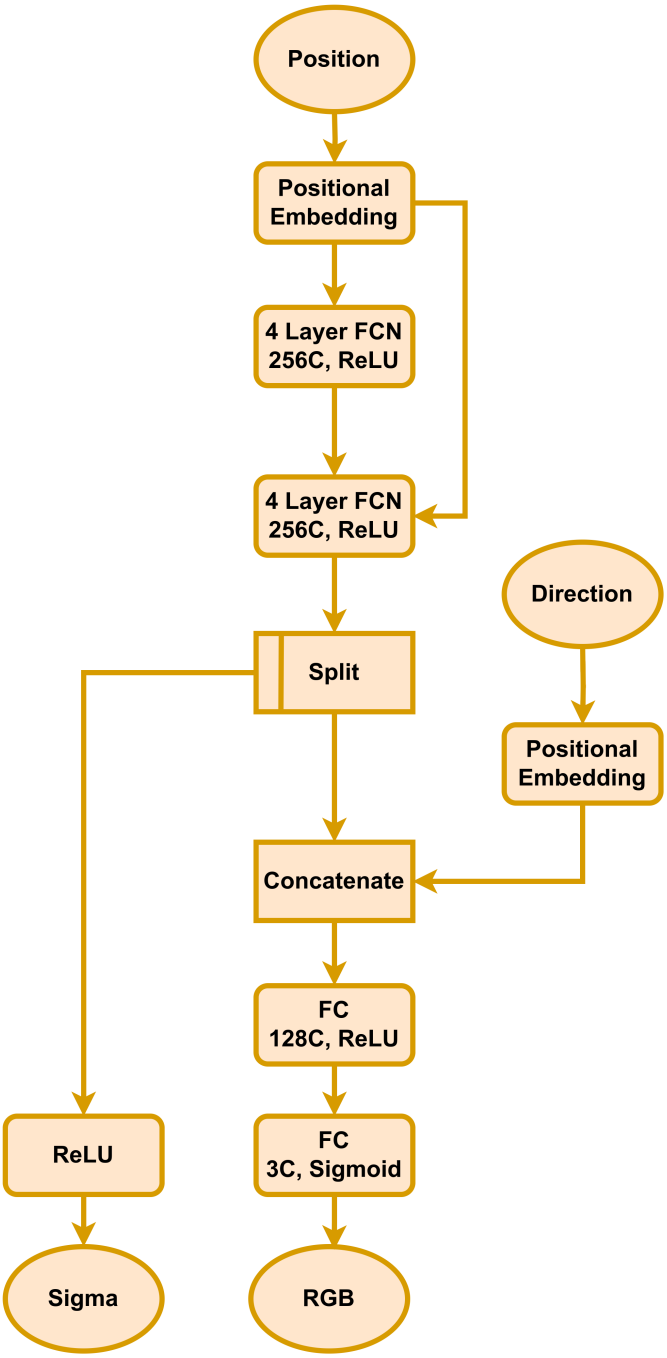

Fig. 3: NeRF Network Architecture

$$C \approx \sum_{i=1}^{N} T_i \alpha_i c_i \tag{1}$$

where T_i represent the weights and c_i the colors at each point. The weights are computed using the equation:

$$T_i = \prod_{j=1}^{i-1} (1 - \alpha_j) \tag{2}$$

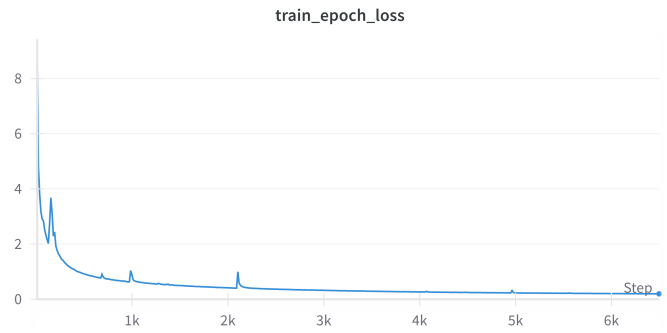

Fig. 4: NeRF training loss per epoch. X axis is approximately 12 x epoch count.

The opacity for each point is determined by:

$$\alpha_i = 1 - \exp(-\sigma_i \delta t_i) \tag{3}$$

In this process, we calculate the transmittance of specific sample point using the density values. This is then used to modify the RGB colors at that location, resulting in the final RGB values for the image.

VIII. Custom Dataset

For the purpose of evaluating our network, we assembled a custom dataset taking hints from the guidelines, which emphasized the importance of uniform soft lighting for optimal scene capture. After we collected 176 images (as shown in 5) from all around the object, we required accurate camera pose estimates, a critical input for our network. To achieve this, we explored different software solutions. Initially, we utilized COLMAP , which employs traditional Structure from Motion (SfM) techniques. This approach proved effective in providing precise camera pose estimates, which can be seen in figure 6. Subsequently, we experimented with Polycam, a tool that not only facilitated the capture of pose data but also allowed us to run the entire NeRF sequence, which was helpful to compare the output with our network. A sample output can be seen in figure 7 Polycam provided us with the raw data (such as camera poses) directly from the process, which was very similar to the output given by COLMAP. We then wrote a simple script to convert the output of these software into a a JSON structure similar to the dataset we were already using.

IX. NERF Output and Observations

A. MSE, PSNR and Avg. SSIM Metrics

The Mean Square Error, Signal to Noise Ratio and the Structural similarity index are presented for 3 scenarios for the lego dataset in TABLE II.
As shown by the last row of TABLE II, when positional embedding is absent, MSE and PSNR goes bad. But you may wonder why the SSIM improves. That's because, training actually breaks without positional encoding and a white image was obtained; For white images, since our ground truth contains mostly white pixels, it gives a better match via SSIM.

Fig. 5: Example of our custom dataset

Fig. 6: COLMAP SFM output

SSIM is a tricky metric to use for multiple images, because its values can be negative. Still, in this work, we simply did an average SSIM for 50 images in test dataset for lego.

B. Novel Views

Novel views (transform matrices were taken from test set) were obtained for ship and lego dataset for this step [fig. 9, fig. 8].

C. Failure Cases

For certain views, the network did not generalise well as seen in this fig. 10. Increasing the resolution or different sampling techniques can help.

D. Custom Dataset Views

A custom dataset was prepared as mentioned in the summary section. This dataset trained faster than the other dataset

Scenario	MSE	PNR	Avg. SSIM
Pos. Encoding + Ray dist. perturb.	0.6696	1.7438	0.0086
Only Pos. Encoding	0.6612	1.7995	0.0073
Only Ray dist. perturb.	0.798	0.9803	0.0126

TABLE II: NeRF metrics on the test dataset for the Lego model

Fig. 7: Polycam NERF output

Fig. 8: Ship - Novel NeRF Views (left), Ground Truth (right)

likely owing to the lack of textures. The generated images are given in fig. 11.

X. Conclusion

In conclusion, we successfully integrated the sophisticated technique of NeRF, utilizing synthetic datasets such as the

Fig. 9: Lego - Novel NeRF Views (left), Ground Truth (right)

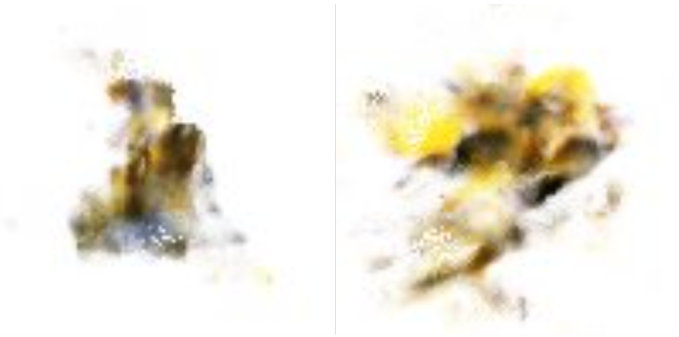

Fig. 10: Failed Views

LEGO and Ship models to evaluate our model's effectiveness. Our experiments demonstrated the network's proficiency in rendering complex 3D scenes from 2D images, showcasing detail and realism. We then collected a custom dataset to further test our network in real world data. Through our work, we've gained valuable insights into the capabilities and limitations of NeRF, laying a foundation for future exploration and improvement in the field of 3D scene reconstruction.

Fig. 11: Crocs dataset generated from colmap