Neural Radiance Field (NeRF)

Ajith Kumar Jayamoorthy
Robotics Engineering Department
Worcester Polytechnic Institute
Worcester, MA, U.S.A.
ajayamoorthy@wpi.edu

***Abstract*—This document consists of the project implementation of the 3D reconstruction of a scene from multiple 2D images. Only the second approach, deep learning using the Neural Radiance Fields (NeRF) has been implemented. The explanation and the outputs have been recorded in this document.**

I. Introduction

The main objective of the project is to generate a 3D model of a particular scene from multiple 2D images captured from different perspectives. In the case of the second method, a new deep learning approach called the Neural Radiance Field (NeRF) has been used to train a model that helps in the evaluation of coordinates for the reconstruction of the scene in 3D. NeRF aims at optimizing an underlying continuous volumetric scene function.

II. Data

The data for NeRF is given from the original authors link. [2] The input data consist of three files, namely, train, test, and validation with images consisting of a lego model of an object. A sample image is shown in figure 1. Each Camera parameter for each image has been further given by .json files to be used for the training purpose.

Fig. 1. Example train image used by the NeRF network [1]

III. NeRF

The Neural Radiance Field or simply NeRF is the state-of-the-art method that generates a complex novel view of a scene by optimizing the underlying volumetric scene function using a sparse set of input images. Figure 2 shows an example of how NeRF builds a 3D volumetric model from the images.

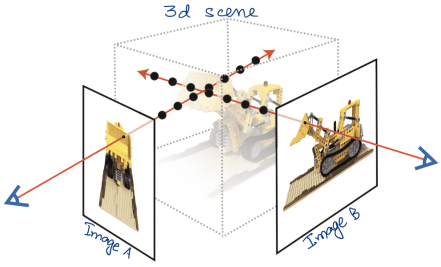

Fig. 2. Example train image used by the NeRF network [6]

NeRF can be visualized in the following steps:

1) Generate Rays
2) Sample Points
3) Deep Learning
4) Volume Rendering
5) Photometric Loss

A. Generate Rays

In the context of NeRF, we generate rays by taking the origin of the ray as the pixel position of the image plane and the direction as the straight line joining the pixel and the camera aperture. This is done for each image and the corresponding pixel rays intersect in the 3D space. Figure 3 shows an instance of the intersection of the two rays from the corresponding pixel from two different images.

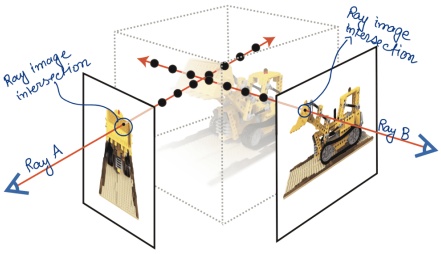

Fig. 3. Example train image used by the NeRF network [6]

A ray in computer graphics can be parameterized as

$$\vec{r}(t) = \vec{o} + t\vec{d}$$

where, $\vec{r}(t)$ is the ray,$\vec{o}$ is the origin of the ray, t is the time parameter and $\vec{d}$ is the unit vector in the direction of the ray.

pixel position based on the camera coordinate frame can be calculated using the following equations:

$$x_c = z_c \frac{u - o_x}{f}$$

$$y_c = z_c \frac{v - o_y}{f}$$

Next, we have the camera-to-world transformation matrix given in the .json file. We can find the location of the pixel points by the following method:

$$\tilde{X}_w = T_{cw} \tilde{X}_c$$

And we can calculate the direction of the rays as follows:

$$\vec{d} = \frac{R'_{cw} \times X_c}{\text{mod } R'_{cw} \times X_c}$$

The ray's origin will be the translation vector of the camera-to-world matrix:

$$t'_{cw} = \begin{bmatrix} t'_x \\ t'_y \\ t_z \end{bmatrix}$$

B. Sample Points

In our implementation, we are sampling points at random positions of the ray. This is done in order to expose the model to new data from the random sampling thus generalizing it more. The following equation defines the random sampling process:

$$t_i = U \left[t_n \frac{i-1}{N} (t_f - t_n), t_n + \frac{i}{N} (t_f - t_n) \right]$$

where U represents the uniform sampling space.

C. Deep Learning

1) Architecture: The input is provided as a 5D function with each image, where each pixel point co-ordinates in 3D space is provided as (x,y,z) co-ordinates followed by the direction of each scene with respect to the global co-ordinates as (θ; ϕ). [4]These 5D points serve as the input to the MLP. This field of rays with 5D points is referred to as the neural radiance field in the paper.

The MLP network predicts each input point's color c and volume density σ. Color refers to the (r, g, b) content of the point. The volume density can be interpreted as the differential probability of a ray terminating at an infinitesimal particle at that point. The figure 4 shows the flow of the network from the input data to the output estimation error function.The Deep learning architecture of the multi-layer perceptron used in the above model can be further explored as shown in the figure5

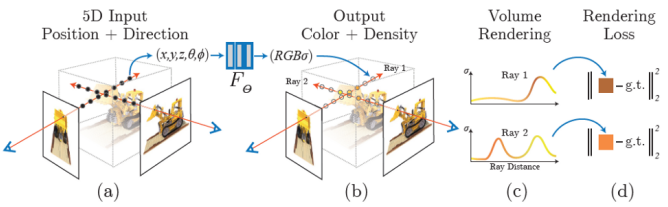

Fig. 4. (a) Sampling 5D coordinates (location and viewing direction) along camera rays; (b) feeding those locations into an MLP to produce a color and volume density; c)using volume rendering techniques to composite these values into an image; (d) optimize the scene representation by minimizing the residual between synthesized and ground truth observed images [3]

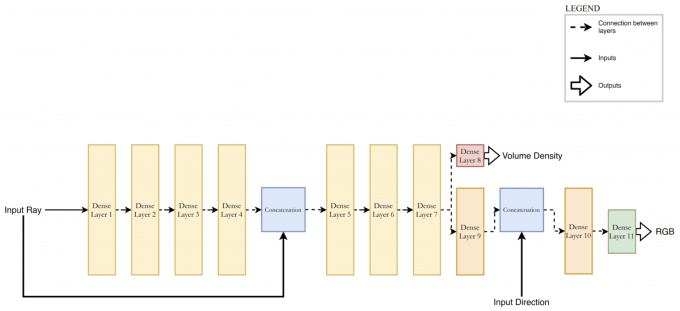

Fig. 5. Network Architecture used in the training process of NeRF model [6]

D. Volume Rendering

In this method we used the predicted color and volume density from the MLP to render the 3D scene. The output from the network is used as an input to the traditional volumetric equation to derive the colour of one particular point. The form of the equation is as given below:

$$C(r) = \int_{tn}^{tf} T(t)\sigma(r(t))c(r(t), d)dt$$

where,

- C(r) is the colour of the point of the object
- r(t) = 0+td is the ray that is fed into the network
- σ(r(t))is the volume density which can also be interpreted as the differential probability of the ray terminating at the point t.
- c(r(t)) is the color of the ray at the point t
- $T(t) = \exp\left(-\int_{tn}^{t} \sigma(r(s))ds\right)$ represents the transmittance along the ray from the near point tn to the current point t. This is kind of a measure of the penetration power of the rays in 3D space.

E. Photometric Loss

The general loss function used by the NeRF network is known as the photometric loss. This is computed by comparing

the colors of the synthesized image with the ground-truth image. This can be expressed as follows

$$L = \sum_{i}^{n} \| I_i - \hat{I}_i \|_2^2$$

$$\theta = argmin_\theta L$$

where I is the real image and is the synthesized image. This function, when applied to the entire pipeline, is still fully differentiable. This allows us to train the model parameters (θ) using back-propagation.

IV. Results

The model is initiated for training, but after an iteration it is getting terminated. The problem with the code is yet to be rectified. The following is the output of the program:

```
ajith@ajith-AORUS-5-SE:~/Downloads/NeRF/custom_functions$ python Wrapper.py
Starting training
Iteration number:  0
/home/ajith/.local/lib/python3.8/site-packages/torch/functional.py:478: UserWarning:
./aten/src/ATen/native/TensorShape.cpp:2894.)
  return _VF.meshgrid(tensors, **kwargs)  # type: ignore[attr-defined]
Prediction loss: 0.04984418302774429
prediction_loss: 0.10962028056383133
Iteration number:  1
Killed
```

Fig. 6. Output of the NeRF model training [6]

References

[1] https://rbe549.github.io/fall2022/proj/p3/
[2] https://drive.google.com/drive/folders/1lrDkQanWtTznf48FCaW5lX9ToRdNDF1a
[3] Mildenhall,B., Srinivasan,P., Tancik, M., Barron, J.T., Ramamoorthi,R., & Ng,R. (2020, August 3). Nerf: Representing scenes as neural radiance fields for view synthesis. arXiv.org. Retrieved November 10, 2022, from https://arxiv.org/abs/2003.08934v2
[4] https://github.com/yenchenlin/nerf-pytorch
[5] https://pyimagesearch.com/2021/11/10/computer-graphics-and-deep-learning-with-nerf-using-tensorflow-and-keras-part-1/
[6] https://pyimagesearch.com/2021/11/17/computer-graphics-and-deep-learning-with-nerf-using-tensorflow-and-keras-part-2/#blog_title
[7] https://pyimagesearch.com/2021/11/24/computer-graphics-and-deep-learning-with-nerf-using-tensorflow-and-keras-part-3/