Report for Computer GraphicII
Final Report

Jiang Xinhong , Tan Shitao

December 19, 2022

1 Overall Introduction

The goal of NeRF's research is to synthesize images from different perspectives of the same scene. The method is very simple, based on a number of images of a given scene, reconstruct the 3D representation of the scene, and then input different perspectives when reasoning can synthesize (render) the image from this perspective.
3D representation comes in many forms. NeRF uses radiation fields and then renders them into an image with a technique called Volume Rendering, given a camera Angle of view. The reason for choosing radiation field and volume rendering is very simple, the whole process is differentiable. It's a very interesting process, and you can think of it as flattening a space in one direction, and the weighted sum of the colors in the space gives you the colors on the plane.

2 The principle

2.1 Core ideas of NeRF

The process of observing a 3D scene by the human eye or camera is that, given the pose (position and rotation) of a camera, a projected image can be rendered according to the parameters of the 3D scene. The NeRF implementation is actually such a process, the 3D scene is represented by MLP, the forward network calculation is the same as the human eye or camera to observe the 3D scene process, when the whole calculation process is micro, through the supervision of the rendering image, the MLP can be optimized, "learn" the "implicit" parameters of the 3D scene.

2.2 Field of radiation

The radiation field can be regarded as a function: if we radiate a ray from an Angle into a static space, we can query the density σ of this ray at each point

(x, y, z) in the space, and the color c ($c = (R, G, B)$ of this position under the ray Angle (θ, ϕ), that is, F: $(x, y, z, \theta, \phi) \rightarrow (R, G, B, \sigma)$. The density is used to calculate the weight, and the weighted sum of the colors on the points can render the pixel colors. In fact, for the radiation field we just need to maintain a table, given a (x, y, z, θ, ϕ) directly look up the table to get the RGB value and density, for the volume rendering method is good. NeRF proposes a better way to model this mapping: using neural networks.

2.3 Field of radiation

Volume rendering, intuitively speaking, we know that the focus of the camera, the line between the focus and the pixel can lead to a ray, and we can do some kind of summation of the colors of all the points on that ray to get the color value of that pixel. In theory, we can integrate the density (dependent only on the spatial coordinates) and the color (dependent on both the spatial coordinates and the Angle of incident) of the ray through each point in space to get the color of each pixel. When the color of each pixel is calculated, the image from this perspective is rendered. As shown below:

$$F_\theta : (x, y, z, \theta, \phi) \rightarrow (R, G, B, \sigma)$$

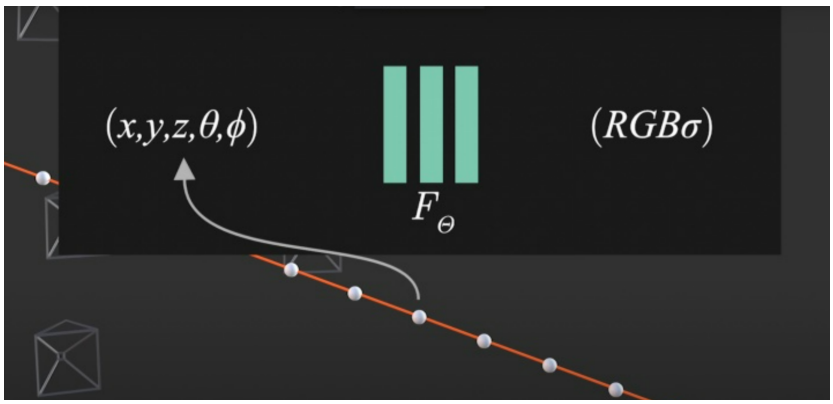

Figure 1: Volume rendering formula

To do that, we'll need to have a Tensor that represents the radiation field, a tensor that takes RGB and density. One of the problems here is that the table can be as big as the space, and it can only be discrete. What NeRF does is model the radiation field with a neural network so that no matter how big the space is, it doesn't affect the amount of memory we need to represent the radiation field, and the radiation field representation is continuous:

If you get an output image from NeRF's model with a given pose, the key is to get the pixel value of each pixel coordinate for each image. In NeRF's

paper, given a camera pose, pixels of a certain pixel coordinate (x,y) should be calculated. Generally speaking, the pixel calculation method of this point is as follows: a camera ray from the optical center of the camera passes through the pixel coordinate, passing through many points in the three-dimensional scene. The accumulation of these "path points" or "sampling points" determines the final color of this pixel. This process, also known as "volume rendering," is visually shown in the image below.

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

where,

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

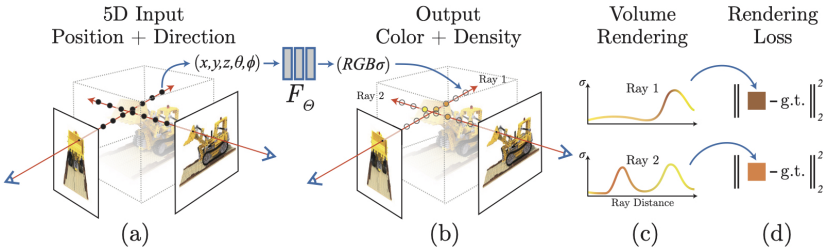

Figure 2: Pipeline

Mathematically, its color is calculated from the following "volume rendering formula", where c represents the color, $/sigma$ represents the density,r and d represent the distance and direction on the camera ray respectively, and t represents the distance of the sampling point on the camera ray from the camera's optical center.

3

We have a general understanding of how the volume rendering process works. But how do you integrate the colors in space along the rays? If we think of rays as rays, we can intuitively get two conditions for this integral:

1. The higher the density of a point, the weaker the ray passing through it, and the density is inversely proportional to the transmittance
2. The higher the density of a point, the more weight the color of the point in this ray reflects on the pixel

Therefore, if the ray connected from the focal point to a pixel is: r (t) = o+td, where o is the origin and t is time. The time starting point (near bound) and time ending point (far bound) are tn and tf. We have the formula of integrating along this ray to get the pixel color as follows:

4 Implement

We try to get the following results through the original author's code.

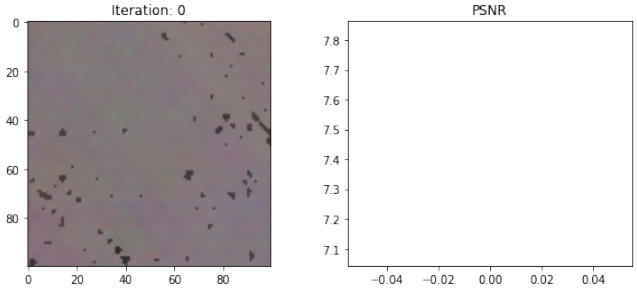

Figure 3: Iteration 0

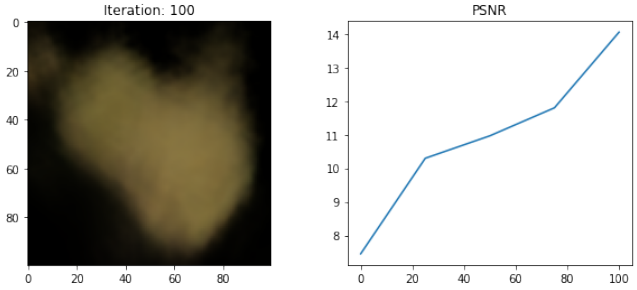

Figure 4: Iteration 100

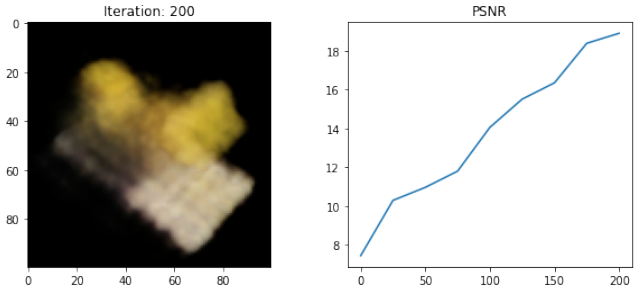

Figure 5: Iteration 200

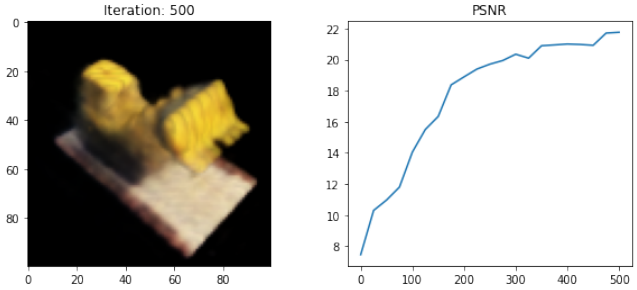

Figure 6: Iteration 500

5 Make dataset based on COLMAP

We tried to make our own LLFF format dataset based on COLMAP. We took series photos around a Hatsune miku figures, and used COLMAP to perform feature matching on the image to obtain the camera pose, then convert the matched pose to LLFF format.

Figure 7: The Hatsune miku we would like to reconstruct

We regrettably found that after training our dataset, we could only get a random messy image. This may be related to the number of training (limited by the performance of the personal computer). At the same time, we also tried to train for several hours on Google's colab platform, but the result still made us very disappointed.

Figure 8: A disappointing result

We guess that this may be related to the calibration error of the data set. It should be that after we used the classic image set "fern" to perform data calibration according to the above process, we got satisfactory results in just ten minutes.

Figure 9: The result we got by "Fern", which looks well

6 Discussion

6.1 Slow speed

The NeRF method required nearly 200 forward predictions of the MLP depth model per pixel to produce the raw image. Although the single computation is small, the amount of computation required to complete the entire image rendering by pixel computation is still considerable. Second, NeRF takes a very slow time to train for each scenario

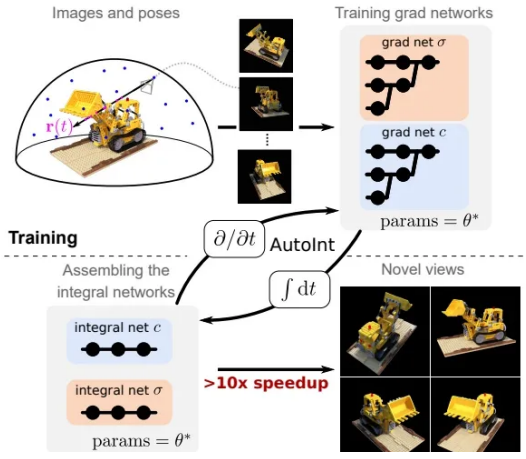

Figure 10: There are many researches on slow reasoning time.

6.2 Static scenarios

The NeRF method only considers static scenarios and cannot be extended to dynamic scenarios. This problem is mainly combined with monocular video to learn the implicit representation of scene from monocular video. Neural Scene Flow Fields[5] Model dynamic scenes as time-varying continuous functions of appearance, geometry and 3D scene movement. This method requires only a monocular video with a known camera position as input.

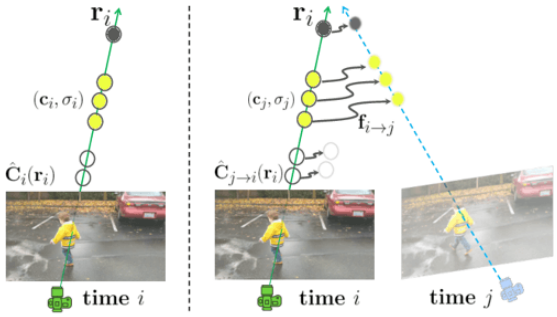

Figure 11: Mapping of Neural Scene Flow Fields in time dimension

7 optimization of nerve radiation field

Similar to Transformer, use higher-dimensional representations of coordinates and perspectives as network inputs to solve the problem of fuzzy rendering.

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

Because the density distribution in the space is not uniform, the rendering efficiency will be low if the ray is sampled uniformly and randomly: the ray shot for half a day may pass through fewer points of high density. From the above analysis we can see that the whole rendering process is nothing more than a weighted summation of the colors of the sample points on the shooting line. Weight of which

$$\omega_i = T_i(1 - \exp(-\sigma_i\delta_i))$$

We can use the color weighted weight w_i in the rendering formula as the probability of sampling in the corresponding interval. We trained two radiation field networks, one Coares and one Fine. Rough network is a network that is rendered and trained at the points where less ${}_N c$ is obtained from uniform sampling, which is used to output ω_1 for sampling probability estimation. Normalize $w_i i$ and think of it as a probability value:

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