Novel-View Synthesis with NeRF and TensoRF

1st Nan Xiao
University of California, San Diego
Jacobs School of Engineering
La Jolla, USA
naxiao@ucsd.edu

This report acts as the solution report to HW3 of CSE 275 3D Deep Learning, on NeRF [3] implementations and optimizations, at University of California, San Diego.

Codes' link: Codes.

Synthesized images' link: Results.

I. Questions

A. (a)

The radiance field can be regarded as a five-dimensional function $L(x, y, z, \theta, \phi) = (r, g, b, \sigma)$. It maps the three dimensions of position (x, y, z) and two of viewing/emitting directions (θ, ϕ) to the RGB colors (r, g, b) emitted from that scene point to the direction, as well as the volume density σ characterizing the opacity at that location. "radiance" in physics refers to the amount of light energy emitted from a given point to a given direction. Hence, a radiance field encodes the color information of all the points in a given 3D scene. Neural Radiance Field (NeRF) encodes the scene radiance field information into a neural network, representing the scene in a "neural" way, and that's why it's called a "neural" radiance field.

B. (b)

Ray marching is a 3D scene rendering method. It divides 3D rays into smaller ray segments, traverses them, and combines all samples' radiance (color) values to render the 2D pixels' color values under the given camera pose. To compute pixel color, the camera will cast a ray from itself through the pixel to the scene, and then sample some points within a near-far range. After that, It performs ray-marching to combine all 3D points' colors (fetched from the radiance field) and volume densities to get the pixel's color.

The rendering equations are given below:

$$L_o(x, \omega_o, \lambda, t) = L_e(x, \omega_o, \lambda, t) + L_r(x, \omega_o, \lambda, t)$$

$$L_r(x, \omega_o, \lambda, t) = \int_\Omega f_r(x, \omega_i, \omega_o, \lambda, t) L_i(x, \omega_i, \lambda, t)(\omega_i \dot{n}) d\omega_i$$

Where ω_i, ω_o denote the incoming and outgoing lights' directions respectively. λ denotes the light's wavelength. x is the position where the radiance is emitted. t is the time. n is the surface normal. L_o denotes the total spectral radiance of wavelength λ directed outward along ω_o at time t from position x. L_e denotes the radiance emitted at x. f_r denotes the bi-directional reflectance distribution function (BRDF) getting the proportion of light reflected from ω_i to ω_o at x, at time t on light wavelength λ. L_i denotes the spectral radiance of the wavelength λ coming inward toward x from direction ω_i at time t. L_r is the reflected radiance at x. It is computed by integrating over Ω (unit hemisphere centered around n containing all possible values of ω_i) the total radiance reflected to ω_o ($f_r(x, \omega_i, \omega_o, \lambda, t) L_i(x, \omega_i, \lambda, t)$) multiplied by a weakening factor ($\omega_i \dot{n}$) caused by the incident angle of ω_i (making the light flux not perpendicular to the surface).

C. (c)

Positional encoding refers to a function mapping positions (x, y, z) and viewing directions (θ, ϕ) to a higher dimensional space so that high-frequency details of the position signals are more separable. NeRF uses positional encoding to enable the MLP to more easily approximate a higher frequency function and to learn and render high-frequency details better.

Figure 1 gives comparisons for test 0000 and 0016 rendered images with and without positional encoding. We see that the images without encoding are more blurred (having fewer high-frequency details) than the ones with encoding. It shows how positional encoding can learn high-frequency radiance details of the scene more effectively.

(a) 0000 test with positional encoding

(b) 0000 test without positional encoding

(c) 0016 test with positional encoding

(d) 0016 test without positional encoding

Fig. 1: Visualizations on poses

D. (d)

Yes, it is possible to extract scene geometry information, such as depth maps, from a trained NeRF model. We can first estimate the depth maps of the training images to get the ground truth depth labels. We then train NeRF using still (x, y, z, θ, ϕ) as inputs supervised by the depth labels together with previous RGB and volume density values, to encode the depth information in the NeRF's MLP. Then during inference, we perform ray-marching from the test cameras, infer depths using the trained NeRF, and aggregate those depths based on

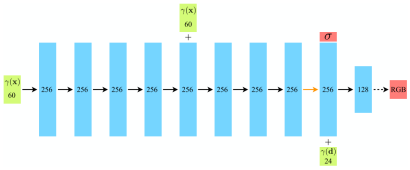
the volume density. I believe that's a brilliant idea on NeRF-based novel view depth estimation though I don't have time to implement that.

E. (e)

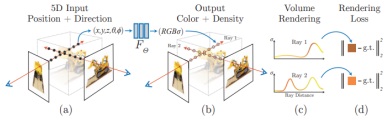
Issue 1: Training is too slow due to the volumetric renderings which gives too many samples for inference on MLP. It often takes over 6 hours to train 200k epochs on a strong GPU like T100. A potential improvement is to represent the radiance field using a set of voxel-bounded implicit fields organized in a sparse voxel octree [1], providing fewer inputs to the MLP.

Issue 2: NeRF might face challenges in generalizing well to scenes that significantly differ from the training data. It might struggle with novel scenes or scenes with distinct characteristics not present in the training set. A potential improvement is to sample points from an optical flow trajectory that limits the relations on the point trajectories and hence shares similar point features [2].

II. Model details and architecture



(a) NeRF network architecture



(b) NeRF model architecture

Fig. 2: Visualizations on poses

The details of the NeRF model [3], as well as the architecture of the MLP inside, are shown in figure 2. NeRF takes the position (x, y, z) and the viewing direction (θ, ϕ) as inputs. It first performs encoding on both inputs to facilitate high-frequency feature extractions. After that, the encoded features are fed into an MLP. To predict the volume density, the MLP uses 8 dense layers each having 256 neurons and 1 output layer to predict the resulting σ with only the encoded position $\gamma(x)$ as inputs. To predict the RGB values, it feeds the input encoded positions into the shared first 8 dense layers and another 1 dense layer of size 256 to extract the points' features. After that, it concatenates those features with the encoded viewing directions $\gamma(d)$ and feeds the combination to another 1 dense layer with size 128 to extract the combined features. Finally, it forwards the combined features to the RGB output layer to infer the RGB values. In this sense, we can see that only the encoded positions participate in estimating σ, whereas both encoded positions and viewing directions contribute in the RGB values' estimations. After network passing, the image on novel view is rendered using volumetric rendering and the pixel-wise Mean-Squared Error (MSE) is computed to supervise training.

III. Train and test settings

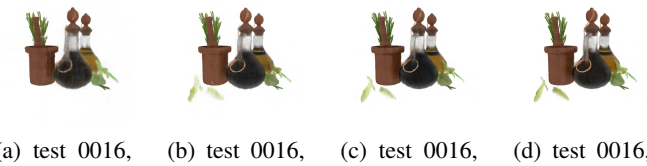
In training, I set the number of epochs to 200k for the tradeoff between performance and speed. I set the near plane to 0 and the far plane to 5 to incorporate the scene more realistically. In testing, similarly, I set the near plane to 0 and the far plane to 5, which gives optimal results in comparison to the Lego settings of 2 and 7 respectively.

IV. Validation scores

Validation performances		
	PSNR	V100 training time (h)
TensoRF	34.0913	Around 2
NeRF05+Encoding	30.3321	Around 6
NeRF05+No encoding	25.2187	Around 6
NeRF27	18.8231	Around 7

The table above gives the training time and validation PSNR of different models (NeRF05: Choosing near=0 and far=5 as cropping range. NeRF27: Choosing near=2 and far=7 as cropping range). We see that TensoRF **??** trained with 100k epoches reaches the optimal performance on the validation set with the highest PSNR and the lowest time complexity for training, showing the effectiveness of the tensor decompositions in representing the scene. NeRF has a better performance in terms of PSNR after adding positional encoding, demonstrating the effectiveness of the high-frequency positional encoding. Choosing near=0 and far=5 also helps the model learn the scene more effectively by looking at it more comprehensively.

V. Visualizations of the training process



(a) test 0016, epoch=10k (b) test 0016, epoch=20k (c) test 0016, epoch=30k (d) test 0016, epoch=40k

(e) test 0016, epoch=50k (f) test 0016, epoch=100k (g) test 0016, epoch=150k (h) test 0016, epoch=200k

Fig. 3: Visualizations on poses

The training process of the NeRF model with near=0, far=5 is shown in figure 3. We see that the model is learning the high-frequency details of the scene (like the cornors and edges) more and more effectively as the training process goes on.

REFERENCES

[1] Liu, L., Gu, J., Lin, K.Z., Chua, T.S., Theobalt, C.: Neural sparse voxel fields. NeurIPS (2020)

[2] Fengrui Tian, Shaoyi Du, and Yueqi Duan. Mononerf: Learning a generalizable dynamic radiance field from monocular videos. In Proceedings of the IEEE/CVF International Conference on Computer Vision, pages 17903–17913, 2023.

[3] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, "NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis," in ECCV, 2020.

[4] A. Chen, Z. Xu, A. Geiger, J. Yu, and H. Su, "Tensorf: Tensorial radiance fields," in Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXXII. Springer, 2022, pp. 333–350.