CS172 Computer Vision I: assignment3 NeRF

Shouchen Zhou
Student ID: 2021533042
zhoushch@shanghaitech.edu.cn

Abstract

In this assignment, we implemented the NeRF and its accelerate method to do 3D novel view synthesis. And we tested the methods on our own datasets, and try to merge the different accelerate methods to chase for a better performance.

And the following tasks were finished.

- *Task1: Implementation of NeRF accelerate method (TensoRF and NGP)*
- *Task2: Processing of our own 3 custom dataset and dataloader of it, then test the performance.*
- *Task3: improve the performance by reasonably combining NGP and TensoRF.*

1. paper reading

1.1. background

The volume rendering process requires some concepts as followed:

For each ray start at $\mathbf{0}$ and on the direction $\mathbf{d}$, the ray $\mathbf{r}$ could be described as $\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$.

For each point x, the color of it is represented as $c(\mathbf{x}) = c(\mathbf{r}(t)) = c(t)$.

The occupation a.k.a. the volume density is the probability that the ray hit a particle at the position $\mathbf{x}$ is represented as $\sigma(\mathbf{x}) = \sigma(\mathbf{r}(t)) = \sigma(t)$.

The transmittance is the probability of a ray travelling from time 0 to time t without hitting any particles, and it is represented as $T(t) = \exp(-\int_0^t \sigma(t)dt)$.

So the expectation of the color that the ray $\mathbf{r}$ throw volume rendering is that:

$$C(\mathbf{r}) = \int_{t_n}^{t_f} T(t)\sigma(t)\mathbf{c}(\mathbf{r}(t), \mathbf{d})dt + T(t_f)C_{bg} \quad (1)$$

Where C_{bg} is considered as the color of the background, usually take $(0,0,0)^\top$ or $(1,1,1)^\top$.

The NeRF's pipeline could be seen in figure 1. With the formula of volume rendering formula 1, we could train

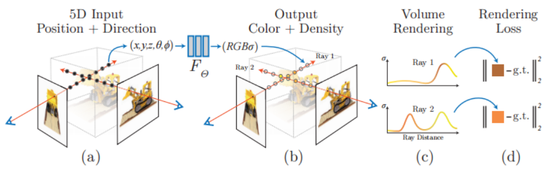

Figure 1. pipeline of NeRF

the network using the trained implicit function F_Θ, with input (x, y, z, θ, ϕ), then output the (R, G, B, σ) for the correspondence input, where σ is the volume density.

With the volume rending formula 1, we could computer the image with given (x, y, z, θ, ϕ). Then we can compute the rendering loss 2, in order to train a implicit model F_Θ for a certain scenario.

The loss function is defined as:

$$\mathcal{L} = \sum_{\mathbf{r}\in\mathcal{R}} \left[\|\hat{C}_c(\mathbf{r}) - C(\mathbf{r})\|_2^2 - \|\hat{C}_f(\mathbf{r}) - C(\mathbf{r})\|_2^2\right] \quad (2)$$

The NeRF method also takes some tricks to improve the performance, such as the positional encoding

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

But the vanilla NeRF has some problem and trained slowly, so the TensoRF 1.2 and Instant-NGP 1.3 and etc. methods were proposed.

1.2. TensoRF

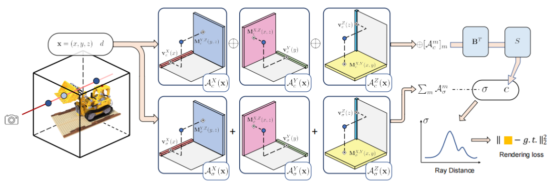

Figure 2. pipeline of TensoRF

The work TensoRF(Tensorial Radiance Fields) [1] proposed a fast training method that does not require a large amount of storage space, comparable to SOTA in rendering quality.

The pipeline of TensoRF could be seen in figure 2.

The vanilla NeRF is based on the coordinates, which uses voxel grid. This requires a large GPU memory to store them, increase in the space complexity of $O(n^3)$, n is the scenario's size.

The key idea is to use a Vector Matrix (VM) decomposition algorithm. Using fewer component tensors to represent a scene and rendering images faster and better. The vanilla NeRF did not effectively use the voxel grid, so each feature grid could be seen as a 4 dimension tensor. The first 3 dimension is the space coordinate x, y, z, and the 4-th dimension is the feature. Then the decomposition algorithm could be used greatly reduced to memory consumption. And it has universality and can be innovated using various tensor decomposition algorithms.

The pipeline figure 2 shows that the neural radiance field expressed as a set of vectors v and matrix M, calculate the volume density and color from this. For each spatial coordinate, we use linear or bilinear sampled values to simulate trilinear interpolation. Add up the volume density along the rays to obtain a density value of A_σ. Stack the color values together to obtain a vector, then multiply it with the appearance matrix B, and use the decoding function S to obtain the color c.

So we can see that the TensoRF could greatly improve the training efficiency of of the NeRF's process.

1.3. Instant-NGP

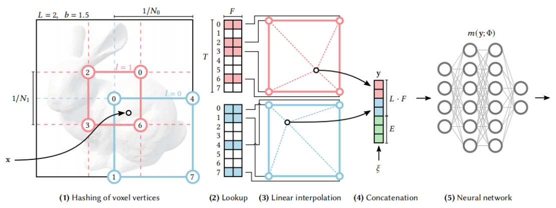

Figure 3. hash encoder

The work Instant-NGP(Instant Neural Graphics Primitives) [3] proposed a way that with hash encoding in figure 3. It firstly converts the coordinate (x, y, z) into the index of the hash table. And the different resolution level has different results(red and blue in the figure). Then do the tri-linear interpolation, concat the result vectors as the feature vector. Then the feature vector is sent to the neural network.

The Instant-NGP mainly used 2 MLP, one is similar to the vanilla NeRF, the density feature encoding at multiple resolutions corresponding to the sampling point is output by the multi solution hash encoding, mixed according to different weights, and decoded to obtain the true density value.

Compared dimensions of MLP in NGP and the number of layers in the network is also much smaller than the vanilla NeRF. This makes Instant-NGP run much faster than the vinalla NeRF.

1.4. Nerfstudio

The work Nerfstudio [4] proposed an integration work, it integrate various NeRF technologies into reusable modular components. And it did real time visualization of NeRF scenes through rich controls. Which makes users much easier to use the tools and the pipeline.

2. Task1: Implementation of NeRF accelerate method

We took 3 videos in different scenarios, then we can implement the dataloader of them, and put them into the vinilla NeRF, the NGP, and the TensoRF. Then we could compare the difference among them.

2.1. Scenarios

To make the testing diverse, the scenarios were taken indoor and outdoor, big and small.

The scenario 1 is a small object stationery.

The scenario 2 is a bigger object chair.

The scenario 3 is also a chair, but with matting preparation in order to chase for a better behavior. The code part for the matting can be seen in to folder '*matting tool*'.

The 3 scenarios look like in figure 4. From left to right are scenario 1, scenario 2, scenario 3 respectively.

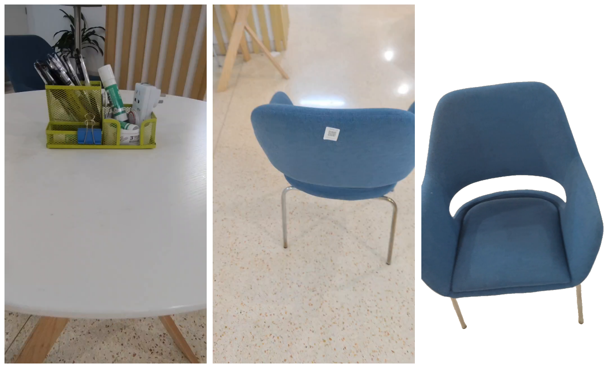

Figure 4. the 3 scenarios

2.2. dataloader

Since we took the videos of the 3 scenarios, but the NeRFs' input should be the images and the intrinsic parameters of the cameras.

With the help of the nerfstudio's integration, and the usage of colmap, we can input the video, and generate the internal and external parameters of cameras. Which is exactly the input for NeRF.

And the frames of the videos were split into the training dataset, validation dataset, and the testing dataset. Each dataset include the images and the correspondence cameras' intrinsic parameters.

Then with the prepared datasets, we could implement the methods in different scenarios.

2.3. metrics

The training information training time(minute:second), GPU memory consumption(MB), and the training metrics are PSNR, SSIM, LPIPS. The methods' metrics comparison are shown in the section 5.

Where PSNR, SSIM and LPIPS are metrics to evaluate image quality.

- PSNR(Peak Signal-to-Noise Ratio) is defined as:

$$\text{PSNR} = 10 \cdot \log_{10} \frac{\text{MAX}_I^2}{\text{MSE}}$$

Where MAX_I is the max value of the pixels in image I, and the MSE of two images I, K is:

$$\text{MSE} = \frac{1}{mn} \sum_{i=0}^{m-1} \sum_{i=0}^{n-1} \|I(i,j) - K(i,j)\|^2$$

- SSIM(Structural Similarity Index Measure) is used for comparing the similarity of two images from 3 key features: Luminance, Contrast, and Structure.

1. Luminance is the average gray scale measurement, obtained by averaging the values of all pixels. Defined as:

$$\mu_x = \frac{1}{N} \sum_{i=1}^{N} x_i$$

And its contrast function is:

$$l(x,y) = \frac{2\mu_x \mu_y + C_1}{\mu_x^2 + \mu_y^2 + C_1}$$

2. Contrast is easured by the grayscale standard deviation. Unbiased estimation of standard deviation:

$$\sigma_x = \left(\frac{1}{N-1} \sum_{i=1}^{N} (x_i - \mu_x)^2 \right)^{\frac{1}{2}}$$

And its contrast function is:

$$c(x,y) = \frac{2\sigma_z \sigma_y + C_2}{\sigma_z^2 + \sigma_y^2 + C_2}$$

3. Structure is the comparison is made after normalization $x - \frac{\mu_x}{\sigma_x}$ and $\frac{y - \mu_y}{\sigma_y}$, i.e. can be measured by correlation coefficient:

$$s(x,y) = \frac{\sigma_{xy} + C_3}{\sigma_x \sigma_y + C_3}$$

$$\sigma_{xy} = \frac{1}{N-1} \sum_{i=1}^{N} (x_i - \mu_x)(y_i - \mu_y)$$

So SSIM is the combination of Luminance, Contrast, and Structure:

$$\text{SSIM}(x,y) = l(x,y)^\alpha \cdot c(x,y)^\beta \cdot s(x,y)^\gamma$$

Where α, β, γ is the ratio of each of the 3 parts of the features.

- LPIPS(Learned Perceptual Image Patch Similarity) is an indicator for measuring image similarity. It learns an image similarity measurement method through deep learning that can better simulate the human visual system. The lower it is, the two images are more likely to be the same.

3. Task2: Processing of our own 3 scenarios of the accelerate methods

We can implement the accelerate methods(NGP, TensoRF) on 3 scenarios, which were introduced in 2.1.

With the dataloader 2.2, we can use our own scenarios to run the accelerate methods and test the behaviors.

The vanilla NeRF without any acceleration [2] were also implemented as the baseline to compare with.

3.1. basic settings

All the code were run under the environment of python3.10.

For the training mode of vanilla-NeRF and TensoRF, we choose the blender mode, and modify the correspondence dataloader.

Since our dataset is somehow bigger than the default once, so the scale should be modify by timing 4, i.e. the parameter '*aabb_scale*' is set to be 4, the '*scene_box*' is modifed from $[-1.5, -1.5, -1.5], [1.5, 1.5, 1.5]$ to $[-6, -6, -6], [6, 6, 6]$.

The training iterations for the vanilla NeRF, TensoRF, NGP, and the improvement method are all set to be default iterations.

And other settings are by default.

3.2. way to run the code

To run the code and see the results, the details are in the jupyter notebook *NeRF.ipynb*, just run all modules in it would work. The running records are all in *NeRF.ipynb*.

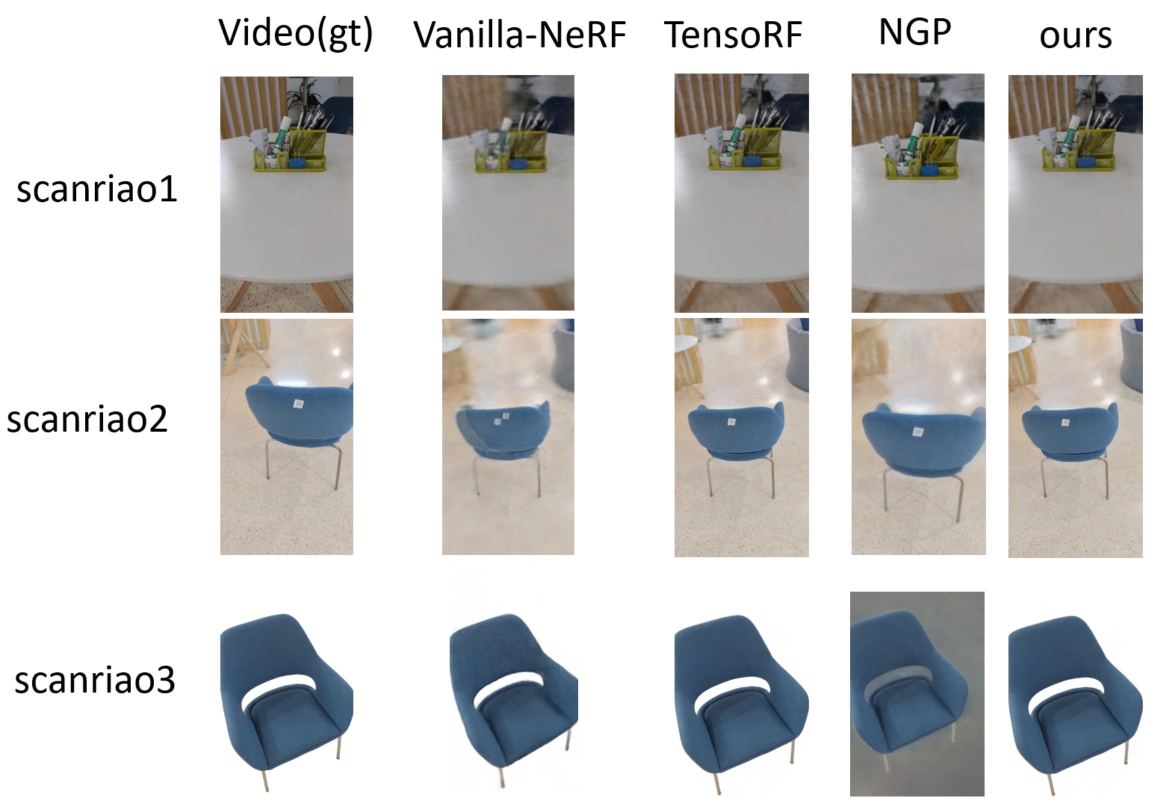

Figure 5. novel synthesis view of the 3 scenarios

3.3. results

The running records are all in *NeRF.ipynb*.

The metrics mentioned in 2.3 would be shown in section 5. And the figures 5 are one of the results of the novel synthesis view of the 3 scenarios.

Since the dataset are seperated into training, testing, validation dataset. So all the novel synthesis view of the 3 scenarios are chosen from the testing dataset, so the novel view exist ground truth for comparison.

4. Task3: Improve the performance by reasonably combining NGP and TensoRF

We can merge the accelaration methods such as TensoRF and the Instant-NGP, take the advantages of them in order to chase a better performance. Such as a faster training time, or lower memory consuption, or better behavior(the metrics PSNR, SSIM, LPIPS, etc.).

4.1. implement method

Since TensoRF is implementing VM decomposing algorithm to decompose tensors, and Instant-NGP with hash encoding also generate a feature, with could be seen as a tensor. So with such inspiration, we can try to add method to decompose feature vector that were generated by Instant-NGP with VM mentioned in TensoRF.

Due to the time limit, the method were implemented with a naive implement, it may include much bugs or parameters to adjustment to achieve better performance.

4.2. basic settings and way to run code

Since the improved method was implemented on the base of Instant-NGP, so all settings were same with the previous work 3.1. And to run the code, it is in the same jupyter notebook mentioned in 3.2. Just run the module of the *improved method* would work.

4.3. results

The result images were also shown in figure 5. And the training time, memory usage, metrics(PSNR, SSIM, LPIPS) were also shown in the table of section 5.

5. conclusion

From the implements above, we can compare the accelerate effect of the vanilla NeRF, tensoRF, and NGP.

All the tasks' experiments were down in the following hardware configuration:
GPU:NVIDIA GeForce RTX 4090.
CPU: 13th Gen Inter(R) Core(TM) i9-13900KF of 32 cores.

And the comparison of different methods in the different scenarios are in the following tables, we could discover that the both TensoRF and Instant-NGP has the better performance in acceleration. Which is much faster than the Vanilla NeRF.

The Instant-NGP has the lowest training time, and TensoRF has the lowest rendering time, the different implement methods has their own advantages. The rendering result for the different method is close, and perhaps the vanilla-NeRF requires more training times, so it is worse than others.

Since the background are the noise, so for the scenario 1 and scenario 2, we can see some foggy blur in the images, and the performance metrics are lower. With the help of matting the images, the model will not be effected by the background noise, so we can see the scenario 3 has better performance, and has much less foggy blur.

For the our method trying to improve effects, we tried to merge the advantages of the Instant-NGP and TensoRF method by applying VM decomposing to the hash encoded feature tensor. Since it was modified based on Instant-NGP, so their metrics are close.

There may be further methods under exploration, but due to the time limitation, the effect achieved by us was not quite well, there are much more things need to learn, and lots of experiments need to do, which attract us to put more effort in later studying and researching. Hope we could learn more and chase for better performance.

References

[1] Anpei Chen, Zexiang Xu, Andreas Geiger, Jingyi Yu, and Hao Su. Tensorf: Tensorial radiance fields. In *European Conference on Computer Vision (ECCV)*, 2022. 2, 5

[2] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis, 2020. 3, 5

[3] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Trans. Graph.*, 41(4):102:1–102:15, 2022. 2, 5

[4] Matthew Tancik, Ethan Weber, Evonne Ng, Ruilong Li, Brent Yi, Justin Kerr, Terrance Wang, Alexander Kristoffersen, Jake Austin, Kamyar Salahi, Abhik Ahuja, David McAllister, and Angjoo Kanazawa. Nerfstudio: A modular framework for neural radiance field development. In *ACM SIGGRAPH 2023 Conference Proceedings*, 2023. 2

Method	training time↓	rendering time↓	Size↓	PSNR↑	SSIM↑	LPIPS↓
vanilla NeRF[2]	317:26	—	10874	27.825	0.784	0.308
TensoRF[1]	69:10	103:39	**8255**	31.422	**0.897**	0.215
NGP[3]	**5:12**	**55:19**	11031	30.99	0.83	**0.174**
our method	6:18	65:21	12063	**31.493**	0.886	0.195

Table 1. Scenario 1

Method	training time↓	rendering time↓	Size↓	PSNR↑	SSIM↑	LPIPS↓
vanilla NeRF[2]	279:19	—	5422	31.13	0.902	0.174
TensoRF[1]	70:51	60:29	15376	35.319	0.938	**0.043**
NGP[3]	5:20	**40:44**	**2630**	**36.54**	0.893	0.112
our method	**5:00**	67:46	7625	35.896	**0.943**	0.138

Table 2. Scenario 2

Method	training time↓	rendering time↓	Size↓	PSNR↑	SSIM↑	LPIPS↓
vanilla NeRF[2]	303:49	—	5445	32.518	0.914	0.128
TensoRF[1]	77:12	**10:27**	5910	**37.876**	0.941	**0.022**
NGP[3]	4:31	79:13	**2689**	37.561	**0.955**	0.057
our method	**3:33**	45:26	6913	37.642	0.944	0.042

Table 3. Scenario 3