Novel View Synthesis with NeRF - CSE291E 22Fall HW3 Report

Tianyang Liu
UC San Diego
til040@ucsd.edu

1. Question Answering

What is a radiance field? What information is included in a radiance field? How is neural radiance field (NeRF) different?

A radiance field is a parametric function $(\mathbf{c}, \sigma) = f_\theta(\mathbf{p}, \mathbf{d})$, whose input is a single continuous 5D coordinate: $\mathbf{x} \in \mathbb{R}^3$ is the 3d point (x, y, z) and $\mathbf{d} \in \mathbb{R}^2$ is the viewing direction (θ, ϕ), and whose output is a 4D vector: $\mathbf{c} \in \mathbb{R}^3$ is the color (i.e., RGB value) and σ is the volume density. Neural Radiance Fields refers to fitting Radiance Fields with a neural network (Multi-layer Perceptron MLP), which is capable of implicit 3D scenes representation, thus rendering a clear picture of the scene from any view. The biggest difference of NeRF is that it bypasses the manual design of 3D scene representation by implicit representation and is able to learn the 3D information of the scene from higher dimensions.

What is ray marching? Given a radiance field, how is each pixel calculated? (This is called the render equation.) Write down your render equation in a concrete math expression with clarified notations.

Ray marching is the process of casting a ray through the scene and generating discrete samples along the ray. The rendering equation [6] is as follows:

$$L_o(\boldsymbol{x}, \boldsymbol{d}) = L_e(\boldsymbol{x}, \boldsymbol{d}) + \int_\Omega f_r\left(\boldsymbol{x}, \boldsymbol{d}, \boldsymbol{\omega}_i\right) L_i\left(\boldsymbol{x}, \boldsymbol{\omega}_i\right) \cos\theta d\boldsymbol{\omega}_i$$

in which,

- $\mathbf{x}$ is the location in space (light source).
- $\mathbf{d}$ is the direction of the outgoing light.
- $L_e(\mathbf{x}, \mathbf{d})$ is the emitted spectral radiance towards direction $\mathbf{d}$ at position $\mathbf{x}$.
- $\mathbf{w_i}$ is the negative direction of the incoming light
- $f_r\left(\boldsymbol{x}, \boldsymbol{d}, \boldsymbol{\omega}_i\right)$ is the bidirectional reflectance distribution function, the proportion of light reflected from $\mathbf{w_i}$ to $\mathbf{d}$ at position $\mathbf{x}$.
- $L_i\left(\boldsymbol{x}, \boldsymbol{\omega}_i\right)$ is spectral radiance toward $\mathbf{x}$ from direction $\mathbf{w_i}$.
- θ is the angle of $\mathbf{w_i}$ and the surface normal at $\mathbf{x}$.

The rendering equation expresses the radiation $L_o(\boldsymbol{x}, \boldsymbol{d})$ of the 3D spatial position $\mathbf{x}$ in the direction $\mathbf{d}$. This radiation is expressed as the radiation (emitted light) from the point itself outward $L_e(\boldsymbol{x}, \boldsymbol{d})$ and the sum of the radiation (reflected light) reflected from the outside by the point.

What is positional encoding? What is the purpose of positional encoding in NeRF models? Train your model without positional encoding and compare the results. You need to show at least two pairs of examples.

Position encoding (PE) is defined as the following function:

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

With this function, we can modify the F to $F = F' \circ \gamma$ (γ is the above encoding function from $\mathbb{R}$ to $\mathbb{R}^{2L}$ and F' is MLP network).

Generally speaking, the position encoding is used to map the position information to high frequencies to improve the clarity. Specifically, the position encoding makes it easier for the network to understand and model the position information in the form of a sine and cosine periodic function similar to the one used in Transformer [4] (It is stated here that it is only similar to Transformer's, but their purpose is intrinsically different. Position Encoding in transformer is used to represent the sequence information of the input, but position encoding in NeRF is applied to the input, mapping the input to higher dimensions thus allowing the network to better learn the high frequency information).

The comparison is shown in Figure 1.

#	Output w/o Postion Encoding	Output with Postion Encoding
Pair 1		
Pair 2		

Figure 1. The comparison of NeRF output without Position Encoding (2nd column) and with Position Encoding (3rd column).

Is it possible to extract scene geometry (e.g. depth) information from a trained NeRF model? Describe your method in detail. Implement your method and show two depth maps generated by your method.

First, we know that the volume rendering formula for discrete points in space is expressed as:

$$\hat{C}(\mathbf{r}) = \sum_{i=1}^{N} T_i \left(1 - \exp\left(-\sigma_i \delta_i\right)\right) \mathbf{c}_i, \text{ where } T_i = \exp\left(-\sum_{j=1}^{i-1} \sigma_j \delta_j\right)$$

in which actually, $T_i \left(1 - \exp\left(-\sigma_i \delta_i\right)\right)$ represents the weight of each sampled point along the rays, which is important. And when we sample points along the rays, we can easily get the depth value for each point, which is the distance of the sampled point and the original point of the corresponding ray. Then just simply multiply the weight and the depth value of each point and make a sum, that will result in the depth of the pixel you are querying. And accordingly, the depth map can be obtained.

Two depth maps are shown in Figure 2.

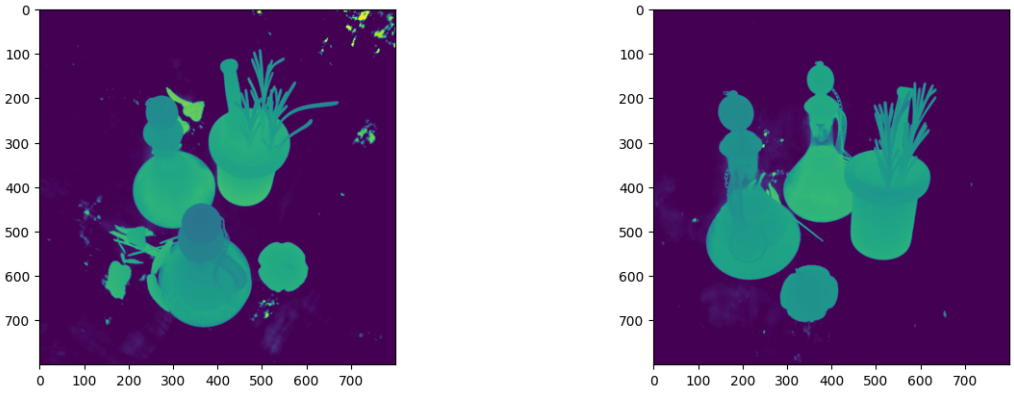

Figure 2. Two example depth maps (val_{13} and val_{44}).

What are the major issues you find when using NeRF? List at least 2 drawbacks. For each of them, propose a possible improvement. You are encouraged to check follow-up papers of NeRF, but you should cite these works if borrowing their ideas.

The drawbacks from my side are listed as follows.

1. **NeRF is slow to train and render.** NeRF suffers from a very slow training and rendering due to its sampling strategy. Even though hierarchical sampling strategy is proposed, most of the points are still sampled at the empty places. A possible solution (I am not sure whether this is feasible) is that we can first try to use certain methods to construct a rough 3d points cloud. For example, normally, for a $100 \times 100 \times 100$ points cloud, actually we may only have 20% points have certain density, if we have a method to estimate a rough points cloud, for example, we set a threshold of 30% (maybe 50% or 60%) to drop some points, so that we can sample points along the ray more efficiently.

2. **NeRF has no generalizability.** This is the fatal drawback of NeRF. NeRF tries to overfit a scene, which leads to the need to retrain for a new scene and cannot be directly extended to scenes that have not been trained before. The possible solution is trying to use Convolutional Neural Networks for feature extraction instead of use MLPs to overfit one scene. (But apparently simply subtitute the MLP with CNN will not work, so some more tricks may need to be applied). Some subsequent research work focuses on handling this drawback with the same idea with me. For example, GRF [3], PixelNeRF [7] and IBMNet [5] both learns local features for each pixel in 2D images and to then project these features to 3D points, thus yielding general and rich point representations.

3. **NeRF needs images from multiple views.** Although NeRF successfully completed the perspective synthesis, more than 100 images from different views are required to achieve good results, which greatly limited

its application in the real world. This problem will build on the previous drawback because if one wants to achieve the synthesis of fewer views or even one view, the model has to have generalization capabilities, i.e., the model needs to be able to extract features and predict the shape of the objects at locations that are not visible from the view, which may require pre-trained models with generalization capabilities.

4. **NeRF's performance in complex scenes is inconsistent.** This is what I found in my experiments, for example, there are some 3D scenes with some small and scattered items. These items can easily not be rendered in NeRF, and also NeRF is not good at rendering transparent objects. The problem of this is still mainly in the sampling, the solution should be a more efficient and accurate sampling method. In addition to the one mentioned in (1), Mip-NeRF [1] enlightened me with such an idea, which is to render with multiple rays from per pixel.

2. Model Structure

This paper mainly uses two model structures, which are essentially based on NeRF. One is using a single MLP, so I named it SingleNeRF, and the other is using two MLPs, one as a coarse model and the other as a fine model, which names DoubleNeRF.

2.1. SingleNeRF

The structure of SingleNeRF strictly follows the description from NeRF [2]. See the model structure in Figure 3, Single NeRF is a fully-connected neural network. The positional encoding of the input location is passed through 8 fully-connected ReLU layers, each with 256 channels, also with a skip connection that concatenates the input to the fifth layer's activation. An additional layer outputs the volumn density (δ), which uses a ReLU to ensure a non-negative density and a 256 dimensional feature vector. Then the feature vector is concatenated with the positional encoding of the input viewing direction (θ, ϕ), and is processed by an additional fully-connected ReLU layer with 128 channels and a sigmoid activation, which outputs the RGB radiance.

2.2. DoubleNeRF

The structure of DoubleNeRF also follows the description from NeRF [2]. Instead of using single NeRF for rendering, we can simultaneously optimize two networks: one coarse SingleNeRF and one fine SingleNeRF. Specifically, we first sample N_c points for each ray using stratified sampling and evaluate the coarse model. Then we do hierarchical volume sampling to sample N_f points on the rays using the density output from coarse model and optimize the fine model.

3. NeRF Pipeline

Abbr.	Description
N_s	The number of sample points on a ray for SingleNeRF
N_c	The number of sample points on a ray for coarse model in DoubleNeRF
N_f	The number of sample points on a ray for fine model in DoubleNeRF
L_p	Position encoding parameters for points (Sign as 10 in all experiments)
L_d	Position encoding parameters for directions (Sign as 4 in all experiments)
$\#Iter$	The number of iterations
LR	The value of Initial Learning Rate during the training
γ	Decay rate value for StepLR
$size$	Step size for learning rate decay for StepLR

Table 1. A description of all the abbreviation (Abbr.) use in paper.

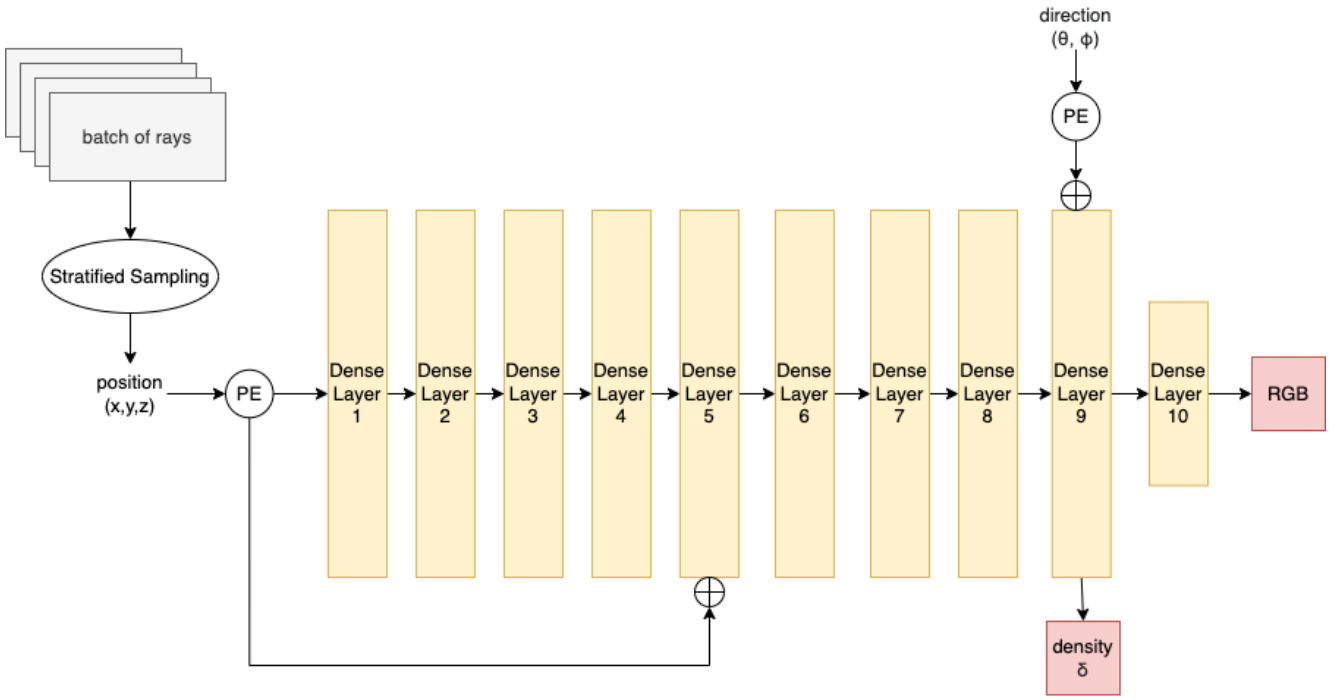

Figure 3. The model structure of a single NeRF.

In order to increase the stability and accuracy of rendering, I propose a NeRF rendering pipeline, divided into the following four modules:

1. **Train SingleNeRF with Data Argumentation.** Data augmentation is done by doubling the probability that a pixel with color (not the background color) is sampled, which successfully increasing the probability of rendering small, slim and scattered objects (such as garlic and leaves) in the image. We first train SingleNeRF on the training data with data augmentation for few iterations until it roughly converges.

2. **Train SingleNeRF without Data Argumentation.** Although the data augmentation can effectively ensure the successful rendering of the slim objects, however, the weight of colored pixels is increased, resulting in the appearance of a small amount of noise in the rendered white background, so we continue training on the un-augmented training set until roughly convergence.

3. **Hard-copy SingleNeRF and train DoubleNeRF.** When SingleNeRF can roughly render new scene with acceptable accuracy, we hard copy it so that we get two identical SingleNeRFs, which we directly splice into a DoubleNeRF, and thus we change the loss function simply to the sum of the MSE of the rbg images rendered by the two models with the ground truth, and train again until converge.

4. **Fine-Tune DoubleNeRF.** Finally, in order to further improve the accuracy, we increase the number of samples on each ray and perform the final fine-tuning with a lower learning rate.

4. Implementation Details

First, I split the data into training set and validation set. Note that there are 200 data in total, among which val_0 and train_0 are duplicate data, so I drop one of them, and finally I randomly select three data (val_23, val_39, val_44) as the validation set and the remaining 196 data as the training set.

To avoid the Out Of Memory (OOM) error, in all the training process, I sample 4,096 rays from all the rays ($196 \times 800 \times 800$) each iteration, while in all the validation process, I crop the 800×800 image into 16 parts (each is 200×200), then rendered them separately and merge them to a complete image subsequently.

Then, in the next two subsections, I summarize all the implementation details into tables, one for the proposed pipeline and another for the ablation study, all the descriptions of the abbreviations are given in details (referring to Table 1).

4.1. Details of the Pipeline

The Table 2 summarizes the implementation details of the pipeline.

#	*Step*	*#Iter*	L_p	L_d	N_s	(N_c, N_f)	*LR*	*Sche*	γ	*size*
1	SingleNeRF w D.A.	10,000	10	4	256	-	5e-4	StepLR	0.9	1000
2	+ SingleNeRF w/o D.A.	10,000	10	4	256	-	5e-4	StepLR	0.9	1000
3	+ DoubleNeRF	20,000	10	4	-	(96,128)	1e-4	StepLR	0.9	1000
4	+ Finetune	30,000	10	4	-	(256,96)	1e-5	StepLR	0.99	1000

Table 2. The implemention details of the pipeline given in Section 3. '+' indicates that the model of current step is based on the previous one. 'D.A.' indicates Data Augumentation.

4.2. Details of Ablation Studies

The Table 3 summarizes all the implementation details of the ablation study:

#	**Model Description**	*#Iter*	L_p	L_d	N_s	(N_c, N_f)	*LR*	*Sche*	γ	*size*
1	SingleNeRF w/o PE	20,000	-	-	256	-	5e-4	StepLR	0.9	1000
2	DoubleNeRF	50,000	10	4	-	(96,128)	5e-4	StepLR	0.9	1000

Table 3. The implementation details of the ablation studies, including two models. 'SingleNeRF w/o PE' indicates a single NeRF model without position encoding; 'DoubleNeRF' indicates a DoubleNeRF model without any pre-training.

5. Results

This section contains two parts, the first is a report of the PSNR values of the NeRF model on the validation set, and the other is visual analysis, including a figure of the trend of PSNR change with increasing number of iterations, and several figures of the change of image details during the rendering process.

5.1. PSNRs

Table 4 summarizes the PSNRs of each step in the pipeline, and the ablation studies.

	Model	**PSNR ↑**	**Improvement**
Ablation Study	SingleNeRF w/o P.E.	22.78	
	DoubleNeRF	28.56	
Pipeline	SingleNeRF w D.A.	26.64	
	+ SongleNeRF w/o D.A.	27.35	+ 0.71
	+ DoubleNeRF	29.04	+ 1.69
	+ Finetune	30.12	+ 1.08

Table 4. PSNRs of ablation studies and every step in NeRF Pipiline.

It is worth noting that the low PSNR of training directly with DoubleNeRF is due to the rendering failure of small objects (see later for visualization). Additionally, the effectiveness of my proposed pipeline can also be seen, as each step improves on the previous one for the effect eventually to over 30.

5.2. Visualization

5.2.1 PSNR vs Number of Iterations

First, I plot the variation of PSNR with the increasing of the number of iterations for the four steps in my pipeline. See the Figure 4, the PSNR value rises gradually.

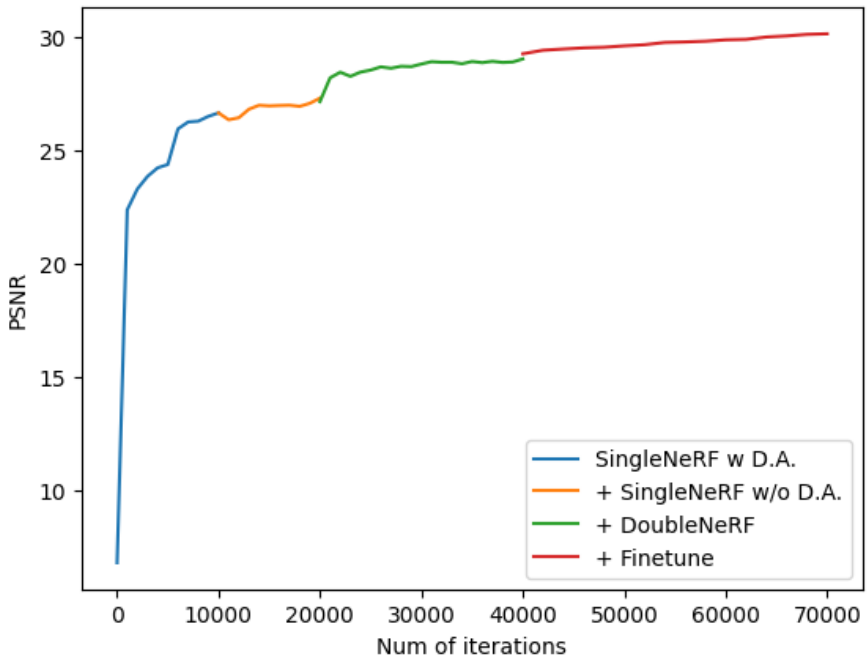

Figure 4. The PSNR variation against the number of iterations of four steps in NeRF pipeline.

5.2.2 Quality Improvement during Training

Then, I visualize the rendered image output by the model from each step as shown in Figure 5. Note that 10k iterations with SingleNeRF with data argumentation can already give a great rendering effects in general. However, if we zoom in and check the details in the image, we can notice that the detail part of the rendering is relatively poor. So for each scene, I randomly truncate three small detail parts in the Figure 5 to highlight the differences in rendering that my model made during training for each step in the proposed pipeline. It can be found that before the hierarchical sampling with DoubleNeRF, the chains and the branches of the grass are blurred. After hard copy singleNeRF and construct a doubleNeRF with hierarchical sampling, the details of the chains can be well rendered, but the branches of the grass still have white noise. Thus, finally we increase the number of samples for fine-tuning, which shows that the details are rendered more clearly and almost without the appearance of any white noise.

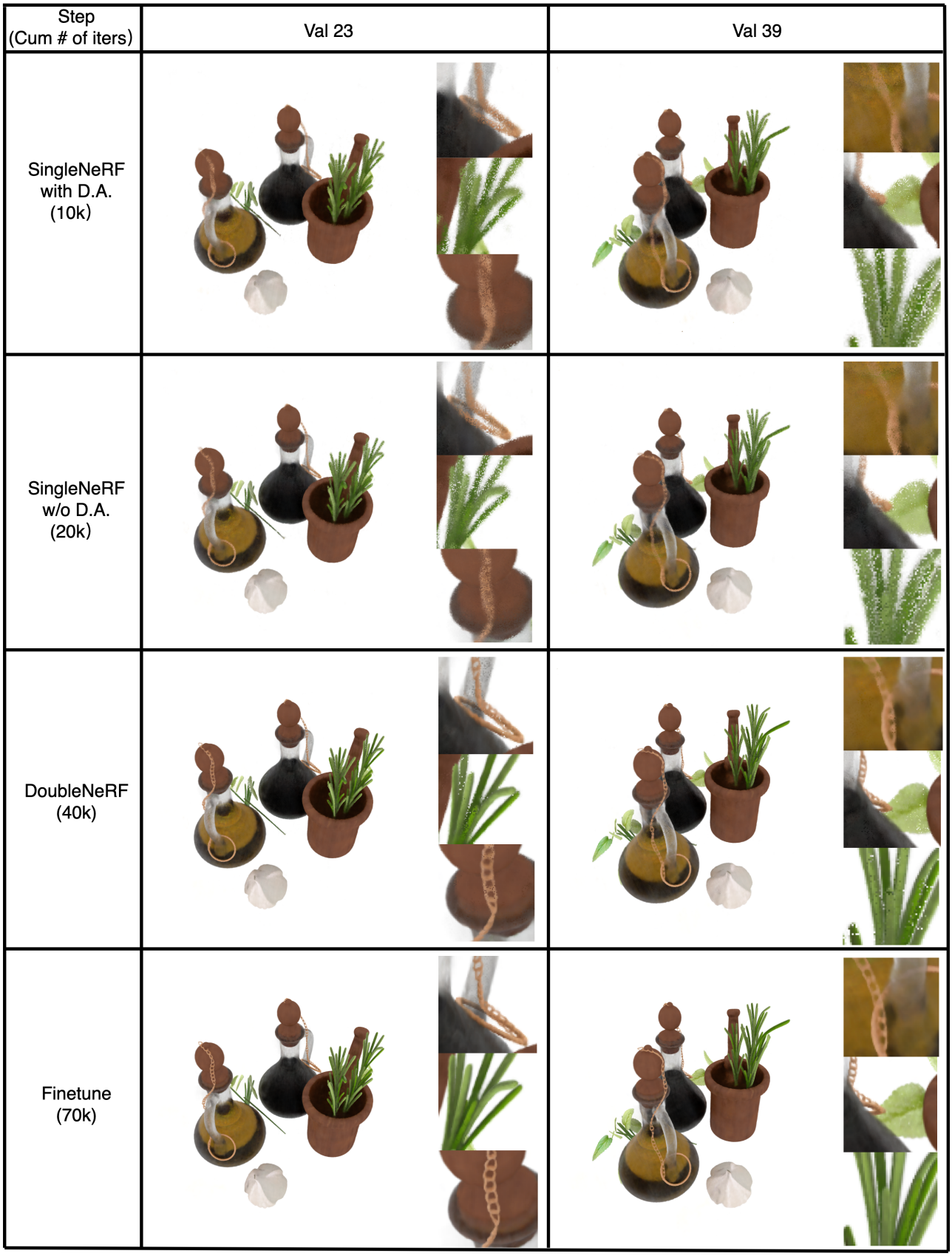

Figure 5. Visualization of two example rendered image output by the model from each step. ('Cum # of iters' in the figure is short for 'cumulative number of iterations').

5.2.3 Ablation Study Visualization

Also, we can know that directly using the DoubleNeRF model for a relatively higher number of iterations can already give to reach a PSNR of more than 28, but its fatal drawback is the inability to render all objects, which is also common in the training of models without positional encoding, which is why I proposed data augmentation before, see the following three examples in Figure 6.

Figure 6. The rendered images from SingleNeRF w/o PE (left column), DoubleNeRF (middle column) and my proposed pipeline (right column).

Now I give descriptions of the figures above in the following itemizations:

- **SingleNeRF w/o PE** In the left top figure, the garlic and leaves behind the brown flowerpot are not successfully rendered. In the left bottom figure, the leaves at the bottom are not successfully rendered.
- **DoubleNeRF (directly)** In the middle top figure, the leaves behind the brown flowerpot are not successfully rendered. In the middle bottom figure, the leaves at the bottom are not successfully rendered.
- **Mine** Everything is well rendered.

6. Conclusion

I use a pipeline based on NeRF model, with four steps. Specifically, I first train a single NeRF model with data argumentation for 10k iterations, then drop the data argumentation and train it for another 10k iterations. Thirdly, I hard copy the single NeRF model to construct a NeRF system with one coarse model and one fine model, and

train it for 20k iterations. Finally, I increase the sample number to fine-tune the NeRF system. The final PSNR of 30.12 was achieved on the validation set.

References

[1] Jonathan T. Barron, Ben Mildenhall, Matthew Tancik, Peter Hedman, Ricardo Martin-Brualla, and Pratul P. Srinivasan. Mip-nerf: A multiscale representation for anti-aliasing neural radiance fields, 2021. 4

[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. In *ECCV*, 2020. 4

[3] Alex Trevithick and Bo Yang. Grf: Learning a general radiance field for 3d representation and rendering, 2020. 3

[4] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017. 2

[5] Qianqian Wang, Zhicheng Wang, Kyle Genova, Pratul Srinivasan, Howard Zhou, Jonathan T. Barron, Ricardo Martin-Brualla, Noah Snavely, and Thomas Funkhouser. Ibrnet: Learning multi-view image-based rendering. In *CVPR*, 2021. 3

[6] Wikipedia. Rendering equation — Wikipedia, the free encyclopedia, 2022. [Online; accessed 22-November-2022]. 1

[7] Alex Yu, Vickie Ye, Matthew Tancik, and Angjoo Kanazawa. pixelNeRF: Neural radiance fields from one or few images. In *CVPR*, 2021. 3

Acknowledgement

I am really grateful to Jaidev Shriram. We discussed a lot and shared our perspectives with each other, which helped me tremendously. Also, many thanks to the TAs, Simon Li, Jiayuan Gu and Zhan Ling for their patient question answering and homework guidance. Finally, I would like to thank Professor Hao Su for the painful but interesting course, which benifits me greatly.

Appendix

A. Synthesized Images on Test Set

In this section of appendix, I show the results on the given set of five black box tests. Not surprisingly, they should have an average PSNR of at least 29 or more.

Figure 7. 2_test_0000.png

Figure 8. 2_test_0016.png

Figure 9. 2_test_0055.png

Figure 10. 2_test_0093.png

Figure 11. 2_test_0160.png