Understanding and Using the

Neural Radiance Fields

By

Yash Gupta

yashguptanotes@gmail.com

Under the Supervision of

Spandan Madan

spandan_madan@g.harvard.edu

Hanspeter Pfister

pfister@g.havard.edu

Visual Computing Group, Harvard University

Cambridge, Massachusetts

March 2022

Table of Contents

Introduction

The fundamental motivating question behind understanding and experimenting with the NeRF family of models is: *is it possible to give to a neural network, or a mapping in general, a point in a 3D scene and a direction along which it is viewed, and ask the network to predict the color and transparency at that position?*

To appreciate the problem, let us take a step back and consider 2D scenes, or our run-of-the-mill images, as we know them. *Is it possible for a neural network, or a function in general, to output the color at an input pixel in an image?* A naive solution could be to memorize the entire image of interest; then a query on a pixel position would simply be a retrieval of the pixel value from the memorized image.

Extending this strategy to 3D scenes would entail memorizing each voxel, or volume pixel, of the scene. Such an approach would clearly necessitate a very dense set of images to learn from in addition to exorbitant memory expenses [1]. Can we perform better? Specifically, can we learn the scene, given only a sparse set of images, say 50 images for recreating a 3D cube with 1000 voxels along each edge (for a total of 10^9 voxels).

Neural Radiance Fields, or NeRF, due to Ben Mildenhall et. al [2], provide an elegant and minimal solution to learn a 3D scene using a few images of the scene. The model learns the volumetric scene by using an equivalent radiance field representation.

The present report briefly reviews the fundamentals of NeRF and some of the prominent NeRF variants, before moving on to Lightning Fast NeRF, based on Instant Neural Graphics Primitives, and our experiments utilizing the same. Also, reviewed is the NeRF coordinate system and our module to control rendering of previously unseen views based on the same.

1. The Basic NeRF Model

The section briefly covers the basic architectural and theoretical aspects of the original (2020) NeRF, beginning with a basic review of the theory of radiance fields, required to fully appreciate the model.

1.1 Radiance Fields

Light - which is central to all of computer graphics - is a form of electromagnetic wave, which allows us to see because EM waves scatter off the atoms. When light touches an atom, the electrons oscillate in response to the oscillating EM field. Since individual atoms are too small for us to care about, we think of matter as a continuous medium with certain scattering properties at each point, implying that all light scattering is volumetric and can be dealt rigorously using wave optics.

However, since most real-world obstacles are large with respect to the wavelength of visible light, and light sources encountered are incoherent, a simplifying geometric optics approximation (allowing us to view light as rays) and surface-level modeling (simplifying the process of specular reflection) is permissible.

In Computer Graphics, we may view light as a continuous entity flowing through space in straight lines, having color, and not interacting with itself [3]. Thus, the electromagnetic field may be represented mathematically as a much simpler *radiance field:*

$$L: R^3 \times S^2 \rightarrow R^3$$

Here, L is a mapping from the 3D position (x, y, z) of point of interest and a 2D viewing direction (θ, ϕ) to the RGB color value at that point; to put it simply L is a 5D viewing function: $L(x, y, z, \theta, \phi) = (r, g, b)$.

Radiance fields are vector spaces on which integrals and differentials can be treated as linear transformations - due to the superposition property of the

underlying electromagnetic field. This enables us to write the famous rendering equation as $L = L_e + TL$, where L is the radiance field of the scene we want to solve for, L_e is the radiance field emitted by objects in the scene, and T is the scattering operator - a linear transformation [3].

1.2 The Problem Addressed by NeRF

NeRF takes up the problem of synthesizing novel views of complex scenes by optimizing an underlying continuous volumetric scene function using a sparse set of input views. The scene is learnt with a fully-connected deep neural network, which takes in the 5D coordinates (x, y, z, θ, ϕ), and outputs the volume density σ - can be thought of as the 3D analogue of the 2D transparency α - and view-dependent - dependent on the view direction (θ, ϕ) - emitted radiance at that spatial location (x, y, z). New views are synthesized by querying 5D coordinates along camera rays and using classic volume rendering techniques to project the output colors and densities into an image [1].

Figure 1.1 below shows the basic conceptual pipeline of the NeRF input and output. The training input, in practice, is a set of camera poses with corresponding images as shown in the figure.

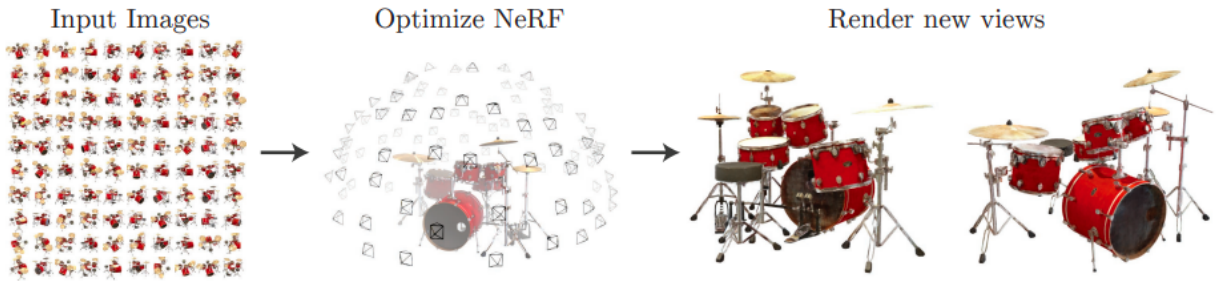


Figure 1.1: Conceptual Pipeline of NeRF
Image Credit: *Please see Reference 1.*

1.3 The NeRF Pipeline

We wish to approximate the 3D scene as a continuous 5D function with a neural network F_Θ: $(x, d) \rightarrow (c, \sigma)$, where $x = (x, y, z)$ is the 3D position in the scene, $d = (\theta, \phi)$ is the viewing direction, $c = (r, g, b)$ is the emitted color, and σ is the volume density.

To train - or optimize - this 5D representation, we supply a sparse set of images. Consider one such image: we shoot a ray $r(t) = o + td$ through each pixel and sample some points on the ray (Figure 1.2). These sample points each have a unique location (x, y, z), and the ray has an associated viewing angle (θ, ϕ). Note that we can shoot a ray through a single pixel in several different ways, each with a unique viewing angle. These sampled points act as the input to the NeRF model, which is asked to predict the RGB color and the volume density at that point. The loss on these predictions are minimized to learn the scene.

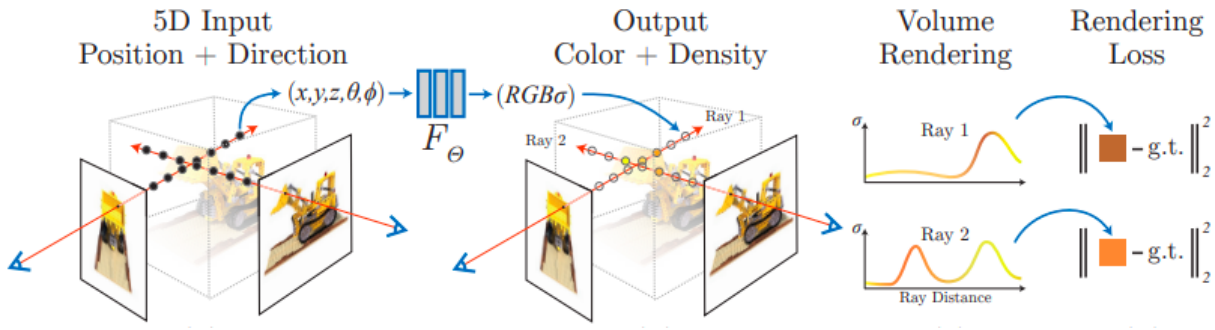


Figure 1.2: The pipeline to train and render with NeRF.
Image Credit: *Please see Reference 1.*

1.4 Positional Encoding

This particular implementation detail of NeRF is of particular interest to us, for it will be changed when moving to the Instant-NGP based NeRF we used. Having the network F_Θ directly operate on (x, y, z, θ, ϕ) input coordinates results in renderings that perform poorly at representing high-frequency variation in color and geometry (Figure 1.3).

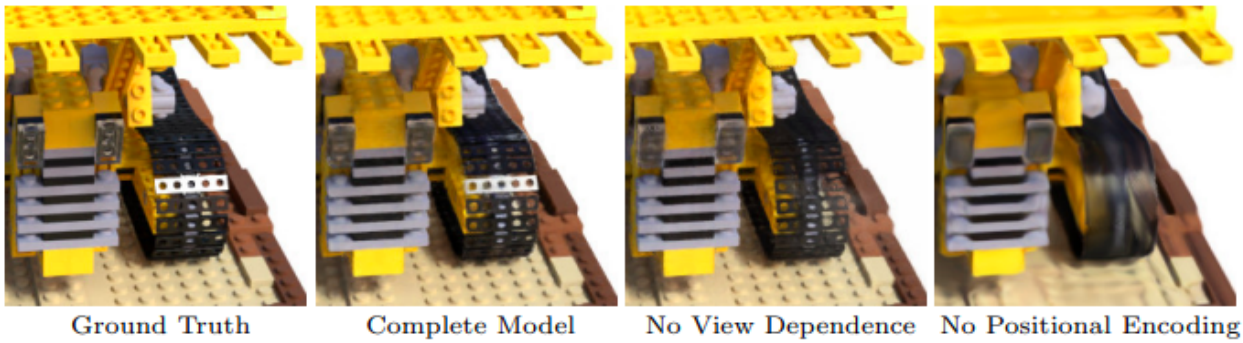


Figure 1.3: The effect of positional encoding. Note that the model is not able to represent the perforated wheel chain of the lego toy satisfactorily without the encoding.

To overcome this problem, the input is mapped to a higher dimensional space using high frequency functions before passing them to the network. NeRF authors, in particular, use the positional encoding mapping γ, where $\gamma(p) = (sin\ 2^0\pi p,\ cos\ 2^0\pi p,\ ...,\ sin\ 2^{L-1}\pi p,\ cos\ 2^{L-1}\pi p)$, where L is a hyperparameter. This encoding is applied to each of x-, y-, and z-coordinate inputs before passing them to the network [1].

1.5 Comparison with Previous Models

The original NeRF authors compared their model's performance against the pre-existing models for rendering from sparse samples of images - specifically, Neural Volumes (NV), Scene Representation Networks (SRN), and Local Light Field Fusion (LLFF). Figure 1.4 shows the results obtained for rendering synthetic and real-world objects, where NeRF outperforms the other models.

	Diffuse Synthetic 360° [41]			Realistic Synthetic 360°			Real Forward-Facing [28]		
Method	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑	LPIPS↓
SRN [42]	33.20	0.963	0.073	22.26	0.846	0.170	22.84	0.668	0.378
NV [24]	29.62	0.929	0.099	26.05	0.893	0.160	-	-	-
LLFF [28]	34.38	0.985	0.048	24.88	0.911	0.114	24.13	0.798	**0.212**
Ours	**40.15**	**0.991**	**0.023**	**31.01**	**0.947**	**0.081**	**26.50**	**0.811**	0.250

Figure 1.4: Comparison of performance of NeRF against SRN, NV, and LLFF for synthetic scenes (first 2 columns) and real-world scenes (last column).
Image Credit: *Please see Reference 1.*

2. Prominent NeRF Variants

Before moving to the NeRF variant we used in our experiments - chiefly due to its remarkably fast training - let us briefly review two important NeRF variants, which might be combined with the presently used Instant-NGP NeRF to better handle rendering foreground without adversely affecting the background rendering quality and to handle different lighting conditions.

2.1 NeRF++

When using NeRFs for real-world scenes, the depth of the scene captured in any image might vary in a very wide range. For instance, the background like the mountains might be well over fifty kilometers away from the camera when the foreground - say a person - is standing just a meter away. Such variations adversely affect the resolution of images rendered by NeRF due to the limitations of the integral used to render in original NeRF: $C(r) = \int_{t=0}^{\infty} \sigma(o + td) \bullet c(o + td, d)\, e^{-\int_{s=0}^{t} \sigma(o+sd)ds}\, dt$, where $C(r)$ is the color seen along the ray when rendering [4]. This problem is encountered frequently enough in real-world scenes (Figure 2.1) to warrant a modification to the original NeRF of Section 1.

(a) bounding volume for the truck only (b) bounding volume for the entire scene

Figure 2.1: (a) Background suffers when bounding the truck with a box. *Box* represents the bounded volume of interest. (b) Foreground suffers if extending the bounding box to include the distant background.
Image Credit: *Please see Reference 4.*

The authors suggest considering the foreground and background components separately by constructing a sphere large enough to include the camera (Figure 2.2). Whatever lies within the sphere is the foreground - parameterized as $(x,\ y,\ z)$ - and whatever lies outside becomes the background - parameterized as $(x',\ y',\ z',\ 1/r)$, where $r\ =\ \sqrt{x^2 + y^2 + z^2}$ and $x'^2 + y'^2 + z'^2\ =\ 1$.

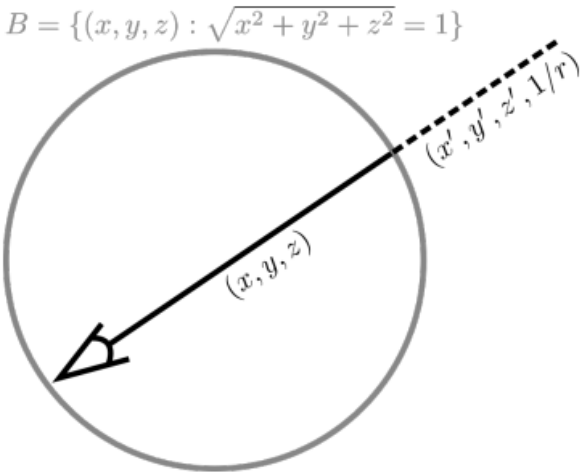


Figure 2.2: Inverted Sphere Parameterization to separate foreground and background. The sphere should be large enough to include the camera.
Image Credit: *Please see Reference 4.*

This allows us to express our volumetric rendering integral as a sum of two integrals for $t \in (0,\ t')$ - inside the sphere - and $t \in (t',\ \infty)$ - outside the sphere:

$$C(r)\ =\ \int_{t=0}^{t'} \sigma(o + td) \bullet c(o + td,\ d)\ e^{-\int_{s=0}^{t} \sigma(o+sd)ds}\ dt\ +$$

$$e^{-\int_{s=0}^{t'} \sigma(o+sd)ds}\ \int_{t=t'}^{\infty} \sigma(o + td) \bullet c(o + td,\ d)\ e^{-\int_{s=0}^{t} \sigma(o+sd)ds}\ dt,$$

thereby allowing us to combine the predicted foregrounds and backgrounds into one better rendered image (Figure 2.3).

(a) NeRF++ prediction (b) predicted foreground (c) predicted background

Figure 2.3: Combining the predicted background and foreground to a single NeRF++ prediction.
Image Credit: *Please see Reference 4.*

2.2 NeRF-W

The 2021 NeRF in the Wild paper - due to R. Brualla et al - addresses the problem of application of NeRFs to images in the wild - such as those obtained online through Flickr - that is, images not in a traditional controlled setup. A model might need to operate on such images when generating novel views of a tourist destination, where a field visit might not be feasible. In such cases, the traditional NeRF, which assumes a static world, fails [5]. By a static world, we mean an invariant scene is invariant with no moving objects so that identical photographs from the same position and viewing direction at any time. It fails for outdoor photography where we need to deal with photographs with photometric variations - such as, variable illumination and variable camera filters - and transient objects in the scene - such as, tourists and festive decorations.

To improve the performance on such unconstrained, or dynamic, data, the authors of the paper suggest a NeRF-W, or NeRF in the Wild, model which uses special embedding vectors to capture photometric variations and transient objects (Figure 2.4). These vectors - appearance vector, $l^{(a)}$, for photometric variations and transient head, $l^{(\tau)}$, for taking into account transient objects - are input alongside (x, y, z, θ, ϕ) to the network and are optimized alongside the weights Θ of the network with backpropagation.

The model - and its ablations with only appearance or transient embeddings - show better results on unconstrained datasets of monuments than the traditional NeRF, as shown in Figure 2.5. Further, by controlling the appearance embedding vector, it is

possible to interpolate for different illumination conditions in the scene - for instance, Taj Mahal at evening using images for the same at midnight and noon.

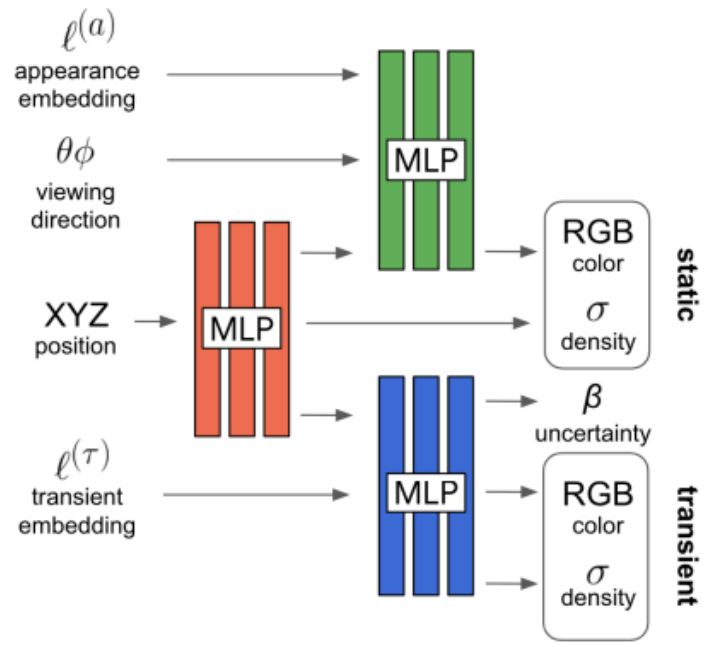


Figure 2.4: The NeRF-W model with additional embedding vector inputs.
Image Credit: *Please see Reference 5.*

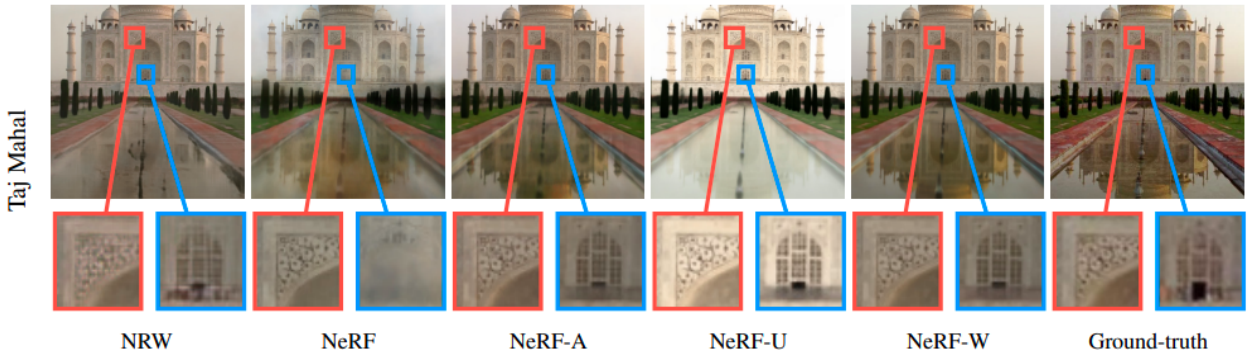


Figure 2.5 NeRF-W and its ablations - NeRF-A and NeRF-U - outperform the traditional NeRF and NRW models on unconstrained images of Taj Mahal.
Image Credit: *Please see Reference 5.*

3. Our Pipeline

This section explores our pipeline for training and rendering with NeRF, using the Instant Neural Graphics Primitives backbone. Also, we briefly mention the generation of new home-brewed datasets for training NeRF. Finally, we look into our module to control the rendering images by allowing controlled translatory and rotatory movements of the camera to generate desired novel views.

However, let us briefly review the fundamental ideas of the Instant NGP upon which the rest of the pipeline is built.

3.1 Instant NGP

NeRFs, albeit able to synthesize a 3D scene from a sparse sample of images, present a huge training time expense - about a day to two. This applies equally to the improved NeRF variants, such as NeRF++ and NeRF-W. The Depth Supervised NeRF, or DS-NeRF, brings down this training time to about 4 - 12 hours [6], which is a major improvement, but still quite expensive in absolute terms.

The 2022 Instant Neural Graphics Primitives - or Instant NGP - paper due to Mueller et al at NVIDIA labs provides an ingenious solution to this problem, bringing the training time to a mere minute, hence quite aptly claiming the name *Lightning Fast NeRF.* They reduce the training cost with a versatile new input encoding scheme that permits the use of a smaller network without sacrificing quality, thereby significantly reducing the number of floating point and memory access operations. A small MLP is augmented by a multiresolution hash table of trainable feature vectors, whose values are optimized through stochastic gradient descent and which is easily parallelized on NVIDIA GPUs. They leverage this parallelism by implementing the whole system using fully-fused CUDA kernels with a focus on minimizing wasted bandwidth and compute operations [7].

Recall the position encoding scheme to deal with high frequency images from Section 1.4. The authors replace this encoding scheme with a multiresolution hash

encoding scheme. Given an input coordinate x, we find the "surrounding" voxels at L resolution levels and assign indices to their corners by hashing their integer coordinates (step 1 in Figure 3.1).

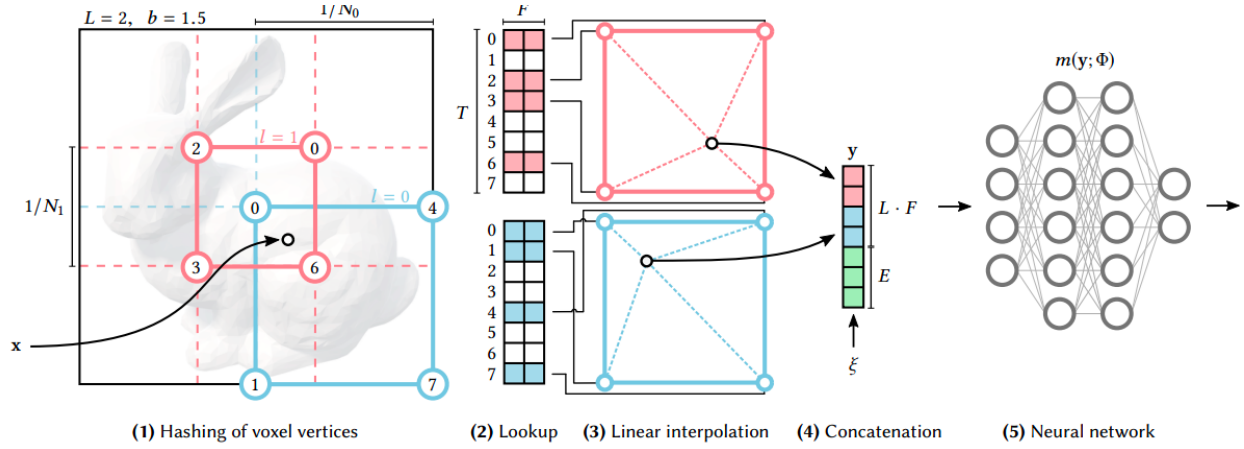


Figure 3.1: Multiresolution Hash Encoding Scheme for Instant Neural Graphic Primitives.
Image Credit: *Please see Reference 7.*

For all resulting corner indices, look up the corresponding F dimensional feature vectors from the hash tables (step 2 in Figure 3.1) and linearly interpolate them according to the distance of x from the farthest vertices chosen (step 3 in Figure 3.1). Concatenate the F dimensional vector for each level, along with some auxiliary vectors (step 4 in Figure 3.1) - not required for NeRF, but for other neural graphic primitives like SDF and NRC - and feed this as input to the NN (step 5 in Figure 3.1).

Hash table entries are optimized during backpropagation, along with the MLP weights. As such, even if the MLP is removed, the hash table itself can somewhat recreate a scene in case of NeRF [7]. The total size of this hash table is $L \cdot T \cdot F$ (Figure 3.1), and limit the size or depth of the 3D scene we can model with a fixed amount of main memory.

The hash encoding scheme, and subsequent C++/CUDA implementation of the network, brings down the training time to ~30 seconds to 2 minutes for most scenes, without significant loss of quality as compared to the original NeRF. It may be noted, however, that the improvements devised in NeRF++/NeRF-W are not incorporated into this NeRF model currently.

3.2 The Structure of Datasets

As mentioned in Section 1, our objective is to reconstruct a 3D scene given a sparse sample of 2D images representing the same in order to synthesize new - interpolated or extrapolated - views of the scene. Therefore, it makes sense that the dataset would be a collection of images with some way to derive the 3D scene from them.

The datasets we used contain around 50 to 150 images of a scene - synthetic (Blender based) or real-world (derived from a video) - and a JSON format dictionary with camera parameters and transformation matrices for each, as shown below along with a brief description of each parameter.

```
{
  "camera_angle_x": 0.5327525694415512,        ←Camera field of view in x-direction
  "camera_angle_y": 0.8894693216170686,        ←Camera field of view in y-direction
  "fl_x": 1979.0314037752323,                  ←Effective Focal length along x-axis
  "fl_y": 2014.362758698270,                   ←Effective Focal length along y-axis
  "w": 1080.0,                                 ←Width of the images used
  "h": 1920.0,                                 ←Height of the images used
  "aabb_scale": 1,                             ←Size of the bounding box separating
                                                    foreground and background
  "frames": [                                  ←A list of images
    {
      "file_path": "./drive/MyDrive/gokuImagesBox1/0078.jpg",     ←image file path
      "sharpness": 8.810492852592493,                             ← image sharpness
      "transform_matrix": [                    ←a 4x4 transformation matrix to go from world
                                                    coordinates to coordinates in the image
        [
          -0.9411432208990053,
          0.3289832941735656,
          -0.07758498517253742,
          -0.4512808166097412
        ],
        [
          -0.13454907898623603,
          -0.15406675691603736,
          0.9788564653502481,
          4.040612178543337
        ],
        [
          0.31007415744310546,
```

```
        0.9316831148956737,
        0.18926328307033463,
        0.5588347017560679
      ],
      [
        0.0,
        0.0,
        0.0,
        1.0
      ]
    ]
  },
    ...
```

The camera parameter `aabb_scale` is of particular importance, for it decides the size of the bounding box which will contain the background. As explored in Section 2.1, the traditional NeRF - and even its faster counterparts, including the Instant NGP NeRF - fail at scenes where dynamic range varies heavily. So this parameter captures our decision of how much importance we give to the quality of foreground rendering at the cost of the quality of the background.

Let us also look more closely at the transformation matrices. As shown in Figure 3.2, the camera may be regarded as a mapping to go from the 3D coordinates in the scene - or the world coordinates - to the 2D pixel coordinates on the image via coordinates in the camera coordinate frame (also in 3D). By augmenting the 3 x 3 system of coordinates and linear transformations to 4 x 4 matrices, we obtain a homogeneous coordinate system, where each rotation and translation can be represented as a linear transformation, hence a matrix multiplication.

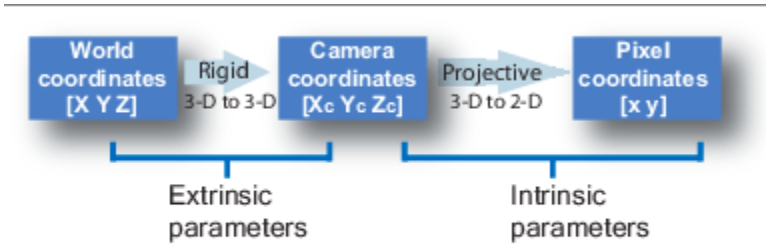


Figure 3.2: Camera as a transformation from 3D word coordinates to the 2D image coordinates.
Image Credit: *Please see Reference 2.*

Hence each transformation matrix P in the `frames` list is a transformation to go from the real world coordinates X of the points captured in the image to their 2D pixel coordinates x, that is, $x = PX$.

It is interesting to note that the structure of the JSON format dictionary remains the same for rendering. We use the same camera and scene parameters as used in the training dataset. Only the transformation matrices - and the file paths - for the images to be generated are to be changed as required.

3.3 Setting up the Instant NGP Codebase

To get started with Instant NGP codebase, we need an NVIDIA GPU with a driver of version 460+. CUDA 11.2+ needs to be installed to execute the efficient Tiny-CUDA components of the project [8]. Further, CMake 3.21 needs to be installed to compile the code available open-source at https://github.com/NVlabs/instant-ngp. The model compilation takes ~15 minutes on a standard Google Colab GPU down to ~3 minutes on a high RAM Colab GPU.

To build one's own datasets - the images and associated JSON file - COLMAP (https://github.com/colmap/colmap) and Ceres-Solver (https://github.com/ceres-solver/ceres-solver), further need to downloaded and installed, which can only be performed on a high RAM GPU environment and takes ~20 minutes in total. If the images are to be extracted from a video first, COLMAP will invoke FFMPEG utility, which can be installed through `apt install.`

3.4 Training and Rendering with NeRF

The principal advantage of using the Instant NGP NeRF in favor of the other NeRF variants is its remarkably fast training time. Training with ~100 images - ~1000 x ~1000 pixels each - would take around 3 minutes of training time with 10,000 epochs (just 30 s for a 1000 epochs). The user needs to specify the path to the training JSON file and the number of epochs, and run the authors' driver script (run.py) in NeRF mode - authors have actually developed fast modules for Gigapixels and SDF models as well.

The snapshot of the trained model may be saved or can be used immediately for rendering by specifying a transformations JSON file for rendering, a directory to store the rendered images, and the size of each rendered image. It takes around 1 to 2 minutes to render a ~1000 x ~1000 pixels image on a high RAM GPU Colab environment.

3.5 Creating Your Own Datasets

Creating your own datasets for training and rendering with NeRF is unexpectedly simple. Just take a minute long video of the scene, capturing different views of interest of an object. This video is fed into COLMAP, which uses FFMPEG to extract frames of the video to use as images sampling the scene. Author's recommended rate of frame extraction for a minute long video is 2 fps to get around 100 - 120 frames [8]. COLMAP further generates a JSON format dictionary for the scene, which may now be used by NeRF. Overall, the process of going from video to JSON file should take around 15 minutes for a minute long video.

It is important to set the `aabb_scale`, or bounding box size, parameter very carefully during dataset generation, for it can have a profound effect on the quality of rendering. Other parameters like camera FOV and focal length might be left at their default values - they were not found to be as important (Section 4) factors for the rendering quality.

3.6 Module for Moving the Camera

Since we began with the objective of synthesizing novel views from a sparse set of images, it only makes sense that we need a module to move our camera - both translate it and revolve it - in the scene so that we may be able to capture previously unseen views in the image. It would be convenient to be able to specify the amount of translation along and the angle of rotation about each of the x-, y-, and z-axes from a starting camera position and get a transformation matrix for the image for that view.

We developed a module which takes the transformation matrix of any starting image (could be any frame in a Blender scene or video), and allows us to specify the translation along the three axes relative to the bounding box size and rotation about each axes in radians to generate a new world-to-camera-to-image transformation matrix.

If $x_0 = PX_0$ is the transformation equation for the original image, where P is the transformation matrix recorded in the dataset, after moving Δx, Δy, Δz along the coordinate axes and revolving the camera θ_x, θ_y, θ_z about the axes, the new image is given by $x_1 = MPX_0$ where,

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta_x) & -\sin(\theta_x) & 0 \\ 0 & \sin(\theta_x) & \cos(\theta_x) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos(\theta_y) & 0 & \sin(\theta_y) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\theta_y) & 0 & \cos(\theta_y) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos(\theta_z) & -\sin(\theta_z) & 0 & 0 \\ \sin(\theta_z) & \cos(\theta_z) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & \Delta x \\ 0 & 0 & 0 & \Delta y \\ 0 & 0 & 0 & \Delta z \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

4. Some Experiments of Interest

This final section of the present report explores some of the interesting experiments we performed with our pipeline to form a better understanding of the capabilities of NeRF and how to harness them. The experiments should be reproducible using Google Colab high RAM environment with NVIDIA P100 series GPU (CUDA version 11.2 and driver 460.32).

4.1 NeRFs and Synthetic Scenes

Before moving to the more complex problem of representing real-world 3D scenes with NeRF, we took up the more controlled synthetic scenes - generated in the Blender environment - available due to the authors of [1]. Specifically we trained our model on the 100 images of a synthetic Lego tractor. Hence, we rendered 10 new views of the object using the transformation matrices in the test partition of the authors for the same scene.

Figure 4.1: Two different 800x800 views rendered by NeRF trained on 100 images after 250 iterations.

The model took only 32 s to train on 100 images for 250 epochs, and was able to render 800 x 800 pixel images of the likes shown in Figure 4.1.

The effect of increasing iterations showed significant improvements (Figure 4.2) upto 1000 iterations for this synthetic scene, after which it plateaued. [The quality keeps improving upto ~5000 to ~10,000 iterations for real-world scenes, in contrast.] The multiresolution hash encoding size was noted to be $F = 2$, $T = 2^{19}$, $L = 16$, and model training loss was 0.00101.

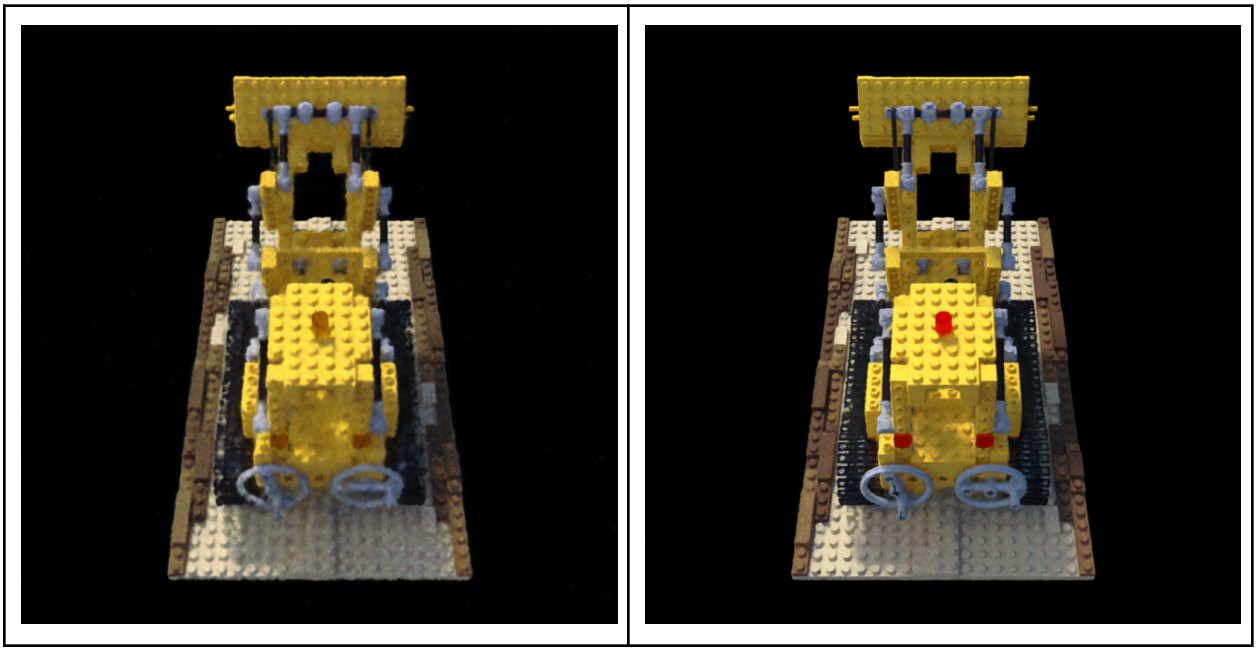


Figure 4.2: Rendered after 250 epochs (left) and 1000 epochs (right).

4.2 Using Instant NGP Author's Fox Dataset

Having witnessed a satisfactory performance from NeRFs on highly controlled synthetic scenes, we proceeded to experiment with a real-world dataset provided by the author of the Instant-NGP repository - a Fox head hanging on a wall.

Figure 4.3 shows the results: even after 5000 epochs, NeRF could not satisfactorily render the image of the fox. The performance suffers further if the camera parameters are tweaked to values different from what the authors originally created the dataset with, implying that all hyperparameter optimizations on the dataset must be handled when generating the dataset, and not at a stage afterwards.

Figure 4.3: From left to right: the ground truth; rendered image after 5000 epochs; image rendered after training with hand-modified camera parameters (cropped due to memory constraints).

4.3 Building Our Own Dataset

A 58 s video of a Goku action figure posed before a switched on laptop screen on a wooden table was captured using an MiA2 mobile phone camera. The camera was moved along all three axes and tilted while keeping it in front of the laptop, that is, the back view and extreme side views of the action figure were omitted.

117 images were extracted from the video with FFMPEG, which were used by COLMAP to generate the JSON transforms file. The bounding box size was chosen to be 16, considering that we wanted to capture the background details like the laptop screen to the greatest detail as well, if possible. However, any box size above and including 8 exceeds the main memory size provided in even high RAM Colab environments. Therefore, the hash table size was set manually for training with this dataset.

Figure 4.4 shows the results of rendering on the training dataset, which clearly outline the problem of underfit - the performance on even the previously seen images shows the room for improvement.

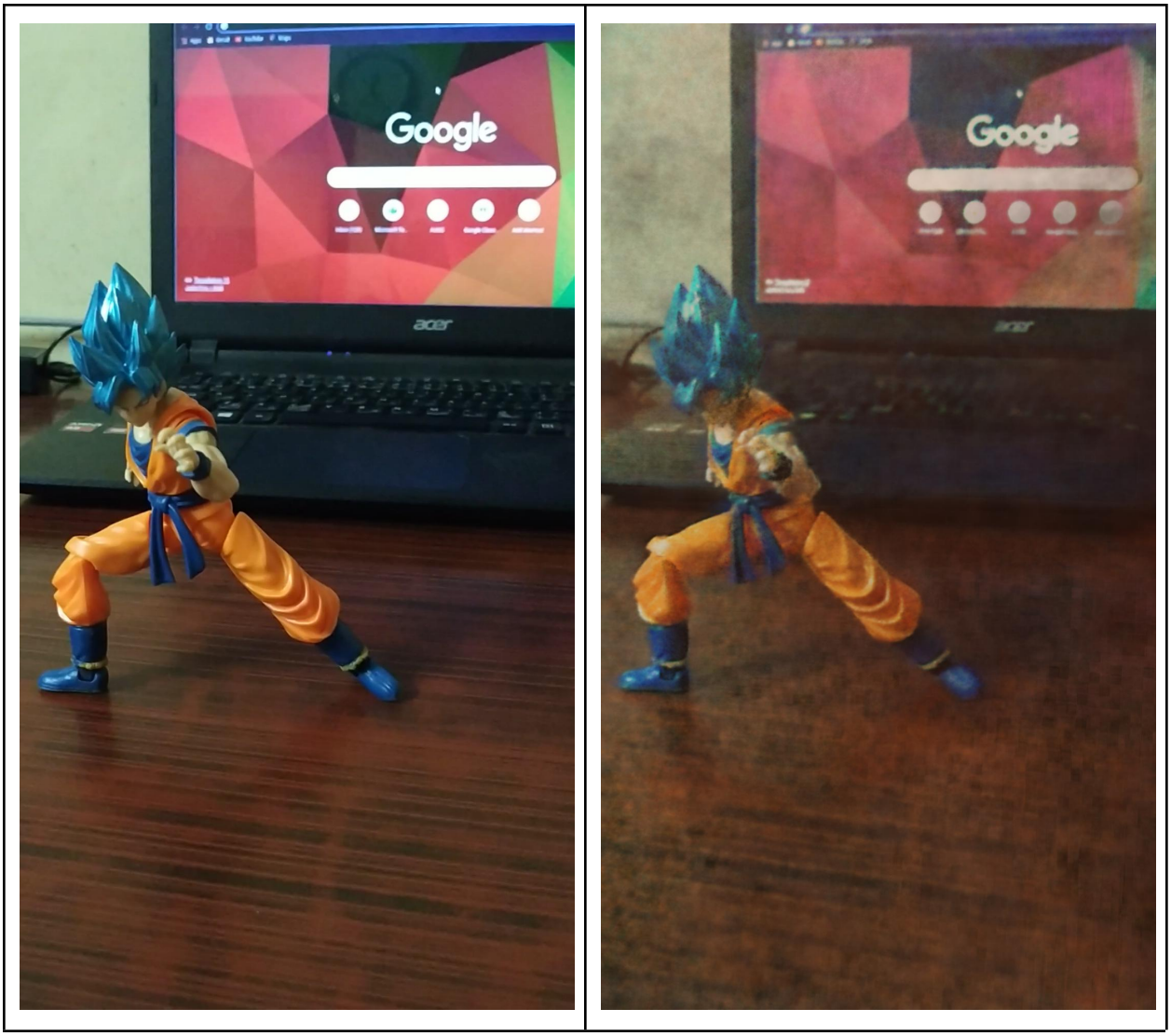


Figure 4.4 Ground truth (left) vs rendered image (right) shows evidence of underfit.

4.4 Solving the Problem of Underfit

To solve the problem of underfit, we tried increasing the number of epochs and then tweaked the hyperparameters while generating new JSON dictionaries for the same set of images. The camera parameters - focal length and field of view - and the number of epochs showed no significant effect on the model performance.

The deciding factor for the quality of rendering was found to be `aabb_scale`, or the bounding box size. The underfit was significantly reduced (Figure 4.5) on

decreasing the box dimensions from 16 to 4, indicating the importance of prudent estimation of scene size.

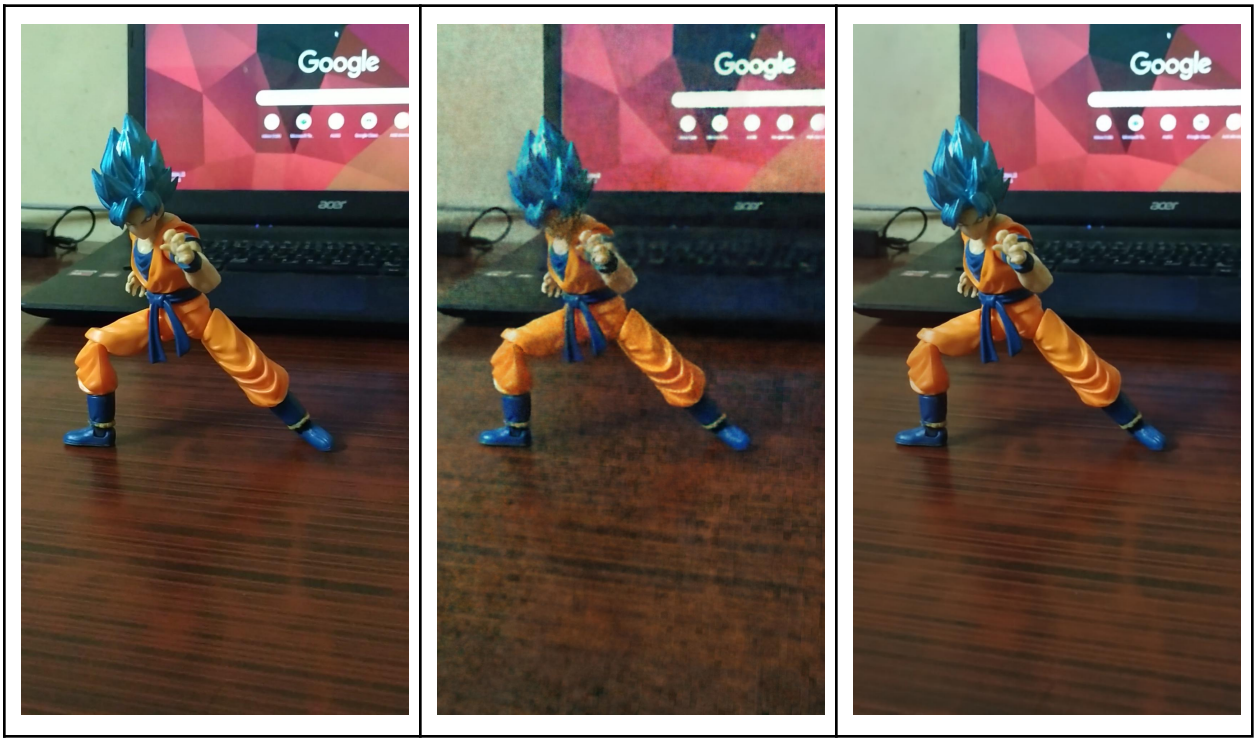


Figure 4.5: From left to right: the ground truth, the image with aabb=16 before improvement, the image after improvement.

4.5 Simpler Background

A 1 min 7 s video of the same action figure with the same camera was captured - this time with a white table and a white wall in the background, allowing us to focus on the object itself (without worrying about the background). The dataset was generated with `aabb_scale=1`, reflecting our wish to better learn the object, that is, the action figure.

Figure 4.6 shows the results of rendering images from the test partition when the dataset - of 135 images - was split in an 80:20 training:testing ratio randomly.

Figure 4.6 Ground truth (left) versus rendered image (right) - previously unseen image (from the 20% testing partition)

It was surmised that NeRF is *good at interpolation*, that is, filling in gaps between images when given a sparse sample of the same. However, its performance on extrapolation still remained to be seen.

4.6 Extrapolation - Reserving Images at Boundaries

The first 20 images from the beginning of the video and the last 20 images from the end of the video were reserved for testing - the rest 95 images were used for training the model. It was hoped that this would serve as a test for NeRF's abilities for extrapolation.

Figure 4.7 shows the results of the trial. NeRF shows *limited* extrapolation abilities, as evidenced by the blackened out portion of the white table in the rendered image. Further, the frames in the video were similar enough to warrant further investigation in case of better results in extrapolation.

Figure 4.7: Ground truth (leftmost); the rendered image (middle); closest training image in terms of similarity to the ground truth.

4.7 Very Sparse Training Sample

Till now we have been using a sample of more than 50 images for training, just as the Instant-NGP authors recommended. It would be an intriguing experiment, however, to check if NeRFs could learn the scene effectively with a much limited - or very sparse - sample, say only 10 images.

We use only 13 images - distributed evenly throughout the video - for training now - the rest 122 images belong to the testing partition. Even with such an arrangement, NeRF could effectively render the testing images, as shown in Figure 4.8. The performance appears to be comparable to the case when the model is trained on the entire dataset.

Figure 4.8: Ground truth (leftmost); Rendered image after training on the entire dataset (middle); Rendered image (in testing partition) after training on only 13 images.

4.8 Generating Novel Views

Section 3.6 described the module we built to control camera movements - translation and rotation - thereby viewing the scene from previously unoccupied vantage points to generate novel views. We put the learning abilities of NeRF to test by using this module. Figure 4.9 shows the results of such movements - one angle or direction at a time, with the coordinate system shown in Figure 4.10.

Rotations around the z-axis have been omitted from Figure 4.8 and explored in Section 4.8 instead. Figure 4.8 highlights the limited abilities of NeRF when it comes to extrapolation.

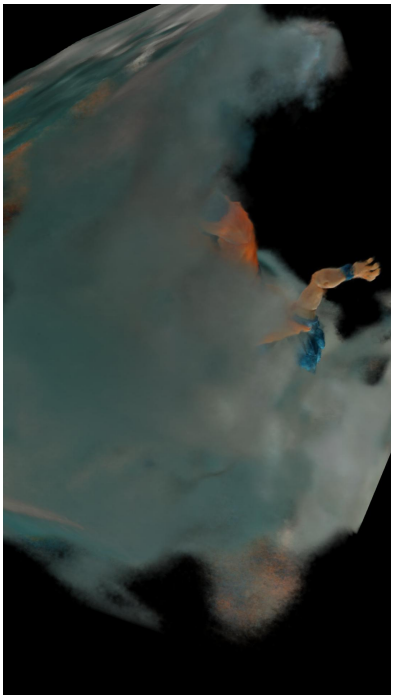
Starting Image (from the video)

Translation along x-axis=+1

Translation along y-axis=+5

Translation along z-axis=+2

Rotation about y-axis=+pi/12



Rotation about x-axis=+pi/4

Figure 4.9: Novel views using modules to control camera movements.

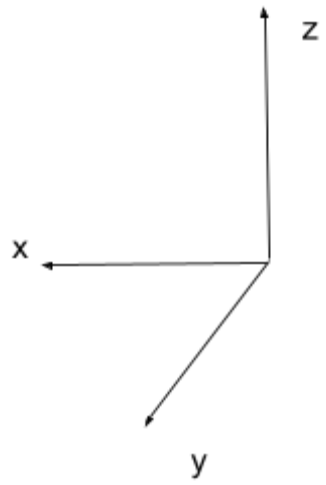


Figure 4.10: The world coordinate axes system used.

4.9 Generating a 180 Degree View

Using the module to control camera movements (Section 3.6), a 180 degree view of the action figure was generated by rotating the camera - terms of matrix transformations - about the z-axis $\frac{\pi}{24}\ rad$ at a time in a counterclockwise manner.

Figure 4.10 shows the 180 degree view thus generated. It clearly shows how NeRF is unable to extrapolate to views which are more than 7.5 degrees outside the last scene of the video (starting at 67.5 degrees in this case). This leads us to believe that NeRF is more suited to interpolation than extrapolation.

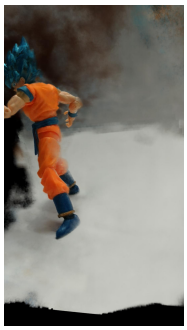
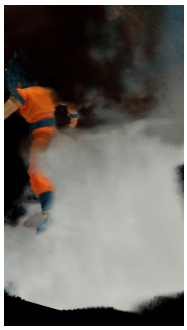
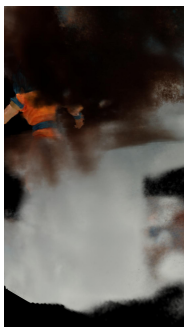
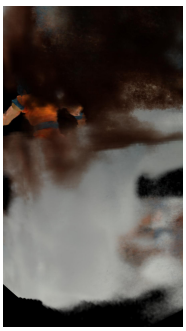
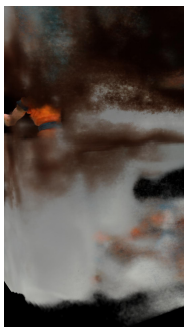
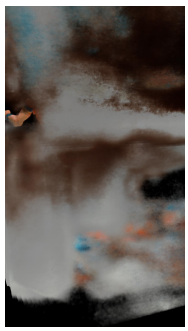
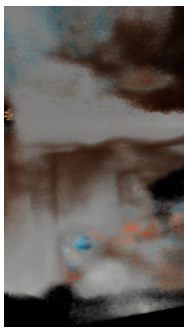
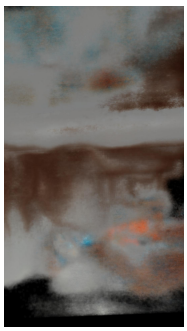
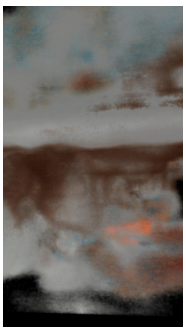
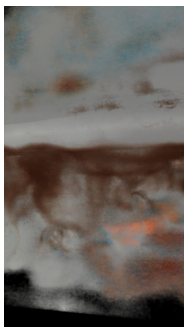
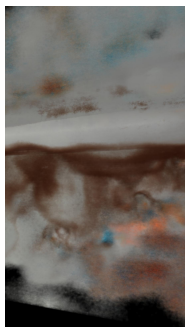
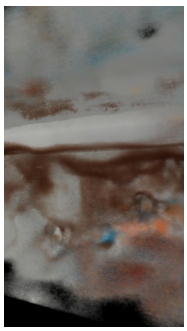
					
0 deg	7.5 deg	15 deg	22.5 deg	30 deg	37.5 deg
					
45 deg	52.5 deg	60 deg	67.5 deg	75 deg	82.5 deg
					
90 deg	97.5 deg	105 deg	112.5 deg	120 deg	127.5 deg
					
135 deg	142.5 deg	150 deg	157.5 deg	165 deg	172.5 deg

Figure 4.11 180 degree view of the scene consecutive images differ by pi/24 radians.

4.10 Epic Kitchens

NeRF has so far been fairly successful in learning 3D scenes of real-world objects from suitable videos. However, the videos that have been considered were restricted to a single object and its hemispherical neighborhood. We decided to stress-test NeRF with more complex videos - specifically, ego-centric videos of a person washing his dishes from the Epic Kitchen dataset [10].

The results stood in contrast with those in Section 4.5 to 4.10. The objects in the renderings were not visible clearly and certain frames from the video could not be recreated at all. This might be due to the presence of too many transient and moving objects and too few samples for certain scenes (like a laundry machine at one end of the kitchen or the window pane). The results are displayed in Figure 4.12. Further, it may be noted that increasing the number of epochs did not lead to any visible improvement in performance.

Figure 4.12: The ground truth images (left) and the rendered image (right) for the Epic kitchen dishwashing video.

4.11 Some Interesting Questions

The experiments done so far highlight the profound interpolation abilities of NeRF with a remarkably short training time. Further, rendering an ~1000 x ~1000 image can be done in ~1 minute, which is comparable to other state-of-the-art rendering techniques. The section, however, also brings to light the limitations of NeRF when it comes to extrapolating to previously unseen views of the scene. This leaves us with several interesting questions to explore via further experimentation, a few of which are highlighted below.

It was observed that setting `aabb_scale` to 1 results in outstanding rendering quality of the foreground, or the object of interest, as in Section 4.5. However, some of the finer background details - for instance, a white switchboard on the wall - were lost. *Would it be possible to optimize the performance on both foreground and background simultaneously?* Possible attempts could be to incorporate the inverted sphere parameterization model of NeRF++ with Instant-NGP or develop a

different fusion scheme to combine results of two separate NeRFs for foreground and background.

The use of NeRFs to combine the different scenes remains to be explored. For instance, *would it be possible to render an image - half-cat and half-dog- by combining NeRFs trained on images of cats and dogs separately?*

The experiments have been conducted so far on either synthetic scenes or scenes captured in videos taken in one sitting. It would be interesting to see the performance of NeRFs on scenes where the sample of images comes from different times of the day, different lighting conditions, and possibly with transient objects. This would bring us closer to the '*in the wild'* setting NeRF-W has been designed to handle.

The authors hope that NeRFs would prove to be a convenient tool for purposes both within and beyond the original objective they were trying to achieve (Section 1), and would serve as useful alternatives to other machine learning and rendering techniques.

References

[1] A. R. Gosthipaty and R. Raha, “Keras Documentation: 3D volumetric rendering with Nerf,” *Keras*. [Online]. Available: https://keras.io/examples/vision/nerf/. [Accessed: 25-Mar-2022].

[2] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. 2020. NeRF: Representing Scenes as Neural Radiance Field for View Synthesis.. arXiv:2003.08934.

[3] N. Reed, “Nathan Reed,” *The Radiance Field – Nathan Reed's coding blog*. [Online]. Available: https://www.reedbeta.com/blog/the-radiance-field/. [Accessed: 25-Mar-2022].

[4] Kai Zhang, Gergnot Riegler, Noah Snavely, and Vladlen Koltun. 2020. NeRF++: Analyzing and Improving Neural Radiance Fields. arXiv:2010.07492.

[5] R. Martin-Brualla, N. Radwan, M. S. M. Sajjadi, J. T. Barron, A. Dosovitskiy and D. Duckworth, "NeRF in the Wild: Neural Radiance Fields for Unconstrained Photo Collections," 2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2021, pp. 7206-7215, doi: 10.1109/CVPR46437.2021.00713.

[6] Kangle Deng, Andrew Liu, Jun-Yan Zhu, and Deva Ramanan. 2021. Depth-supervised NeRF: Fewer Views and Faster Training for Free. arXiv:2107.02791.

[7] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. 2022. Instant Neural Graphics Primitives with a Multiresolution Hash Encoding. arXiv:2201.05989.

[8] NVlabs, “NVlabs/instant-NGP: Instant Neural Graphics Primitives: Lightning Fast Nerf and more,” *GitHub*. [Online]. Available: https://github.com/NVlabs/instant-ngp. [Accessed: 26-Mar-2022].

[9] “Installation¶,” *Installation - COLMAP 3.7 documentation*. [Online]. Available: https://colmap.github.io/install.html. [Accessed: 26-Mar-2022].

[10] Dima Damen, Hazel Doughty, Giovanni Maria Farinella, Antonino Furnari, Evangelos Kazakos, Jian Ma, Davide Moltisanti, Jonathan Munro, Toby Perrett, Will Price, and Michael Wray. 2021. “Rescaling Egocentric Vision”. arXiv:2006.13256.

External Datasets Used

Synthetic Lego Tractor: : Available along with several other synthetic Blender based datasets on the original NeRF paper authors' google drive - https://drive.google.com/drive/folders/1JDdLGDruGNXWnM1eqY1FNL9PlStjaKWi.

Fox Dataset: Provided in the public Github repository of the Instant NGP authors under https://github.com/NVlabs/instant-ngp/tree/master/data/nerf/fox.

Epic Kitchen Dish Washing: The video used may be accessed at https://data.bris.ac.uk/data/dataset/41a435694f6f990ddd922a5f6f1875b8 as P11_01.MP4.