NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis

Anirudh Gupta Saksham Agrawal Shivey Ravi Guttal Aditya Arsh
UMC 203 AI-ML
Indian Institute of Science
{anirudhgupta, sakshama, shiveyravi, adityaarsh}@iisc.ac.in

Abstract

Neural Radiance Fields (NeRF) represent a state-of-the-art deep learning approach for synthesizing novel views of complex 3D scenes from a set of 2D images. In this project, we explore and gain an understanding of the original NeRF implementation (NeRF-Orig) using both publicly available datasets and our own collected data. The algorithm produces a scene using a Multilayer Perceptron (MLP), whose inputs are the spatial location (x, y, z) and the viewing direction (θ, ϕ), with the output being the volume density and (view-dependent) emitted radiance at that spatial location. Although NeRF-Orig demonstrates impressive results, it suffers from significant limitations, most notably extremely long training times (15–20 hours) and suboptimal rendering quality for larger datasets. We document our work and store our collected datasets and saved outputs in the NeRF-UMC203 GitHub repository.

1 Introduction

NeRF (Neural Radiance Fields) is a deep learning model that generates high-quality 3D scenes from 2D images. It represents a scene as a continuous function, mapping 3D coordinates and viewing directions to color and density values. NeRF is trained using a set of posed images and can synthesize novel views with realistic lighting and details. It works by optimizing a volumetric rendering function using a neural network, making it particularly useful for applications like 3D reconstruction, virtual reality, and synthetic view generation. A more detailed description of the technique is as follows.

1.1 Neural Network and Parameters

A 5D continuous scene is approximated to a **Multi-Layer Perceptron**, which takes as input the 3D location $\mathbf{x} = (x, y, z)$, and 2D viewing direction $\mathbf{d} = (\theta, \phi)$. The MLP outputs a color $\mathbf{c} = (r, g, b)$ and the volume density σ.
The volume density is calculated solely using the spatial coordinates while the color is a function of both spatial and directional coordinates. The MLP consists of 8 fully connected layers that use ReLU activations and have 256 channels per layer that output σ and a 256-dimensional feature vector. This is passed on to another fully connected layer with ReLU activation and 128 channels that outputs **c**, as show in Figure4. An overfitting of the MLP is done according to the given specific view.

1.2 Ray Marching

Ray Marching is a technique used to identify colors of different points in the scene. A ray from some pixel of a fixed camera image is projected to the 3D scene. A set of points are sampled along the ray. The coordinates of a point on the ray are passed to the MLP function F_Θ to get the values of the color and density of that point. This technique is used for all the points on the ray, and the corresponding data obtained is passed for volume rendering.

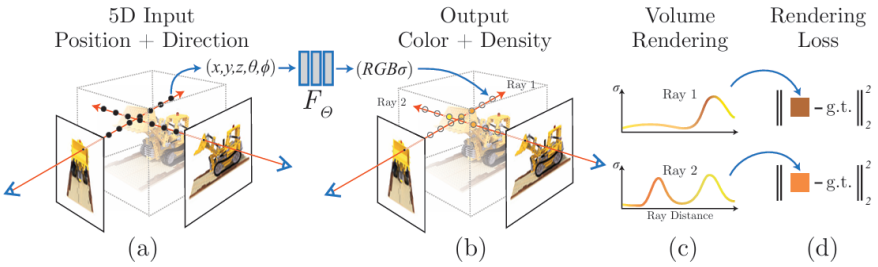


Figure 1: Overview of Ray Marching and Volume Rendering [7]

1.3 Volume Rendering

The color of any ray passing through the scene is rendered using the principles from classical volume rendering [4].
The volume density $\sigma(\mathbf{x})$ can be interpreted as the probability of a ray terminating at an infinitesimal particle at location $\mathbf{x}$. The expected color of the $C(\mathbf{r})$ of the camera ray $\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$ with near and far bounds t_n and t_f is:

$$C(\mathbf{r}) = \int_{t_n}^{t_f} T(t)\sigma(\mathbf{r}(t))\mathbf{c}(\mathbf{r}(t), d)dt, \quad \text{where} \quad T(t) = \exp\left(-\int_{t_n}^{t} \sigma(\mathbf{r}(s))ds\right) \tag{1}$$

where $T(t)$ is the probability of a ray not hitting any particle from t_n to t. The above integration can be interpreted as the weighted sum of the color of each point. Each point has two contributions, the probability that the ray reaches the point t, and the volumetric density.

Note: Certain optimization techniques have been applied and introduced by NeRF-Orig whose details are provided in Section 5.1.

NeRF-Orig calculates MSE loss between the actual color of a ray to the output color given by the function for a fixed camera image. Refer Section 5.2 for more details.

2 Methodology

Here we describe the process employed in running NeRF for datasets.

2.1 Data Collection

The data used for experiments includes pre-existing datasets (Fern and LEGO), and self-collected data. For the latter, our data was of LLFF type since it is used to represent real images of natural scenes. Note that we have not created any blender or deepvoxels type datasets. The self-made datasets include **Wifiroom Dataset**, **Humanities Project Dataset (HMB)** and **JNT Statue Dataset (JNTS)**. A detailed note about these datasets are mentioned in Section 5.3.

2.2 Post Processing Images

To obtain input (**x, y, z,** θ, ϕ) for NeRF (only LLFF type dataset), we use COLMAP which performs Structure-from-Motion (SfM). The output is a 3D reconstruction of the object, and the reconstructed intrinsic [1] and extrinsic [2] camera parameters of all images. Figure 5 shows the point cloud obtained for HMB Dataset. More details on COLMAP are in the Appendix Section 5.4.
Nerf-Orig also requires `poses_bounds.npy` (or `transforms.json`) which is pre-processed version of the camera poses and intrinsics. This is a `(num_images, 17)` numpy array containing extrinsic and intrinsic values in a flattened matrix, along with near and far bounds required during volume rendering.

[1] intrinsic values include focal length, image resolution, number of features, etc.
[2] extrinsic values contain coordinates of camera including viewing directions, position.

For Blender type dataset, we use `transforms.json` which contains camera extrinsics (as `transform_matrix`) and metadata like file paths and `camera_angle_x`. NeRF reads these to know where each camera was in 3D space and how it was oriented. These matrices are used to cast rays correctly for rendering and training.

2.3 Training Models

Using the obtained output, we trained NeRF-Orig on our datasets. We obtained two videos, a depth video showing the render using only volume density and one using depth and RGB values obtained from the network. Due to limited GPU resources (6GB VRAM), we ran the models for nearly 250k iterations, but lower hyperparameter values (including batch size, N_c, N_f) which resulted in a lower quality render, each taking around 18-24 hours. This is because lowering the number of rays marched decreases the time taken at the cost of less accurate volume density and RGB values. We have run the code upto 250-300K iterations since the algorithm convergence by then, which can be seen from Figure 2a This issue is observed for all experiments below and mentioning the same will be omitted.

3 Experiments

We ran NeRF-Orig on five datasets, which include Fern, Lego, Wifiroom, HMB, and JNTS-one-side. NeRF runs on mainly 3 types of datasets, which include LLFF, Blender and deepvoxel. We have only run the model on the first two due to ease of availability and capturing data.

3.1 FERN and LEGO

For the Fern and Lego datasets, pre-processed data was available, and we attempted to replicate the rendered videos shown in the original NeRF paper. Due to less complexity of the dataset, the quality of videos came out to be better than other ones. LEGO dataset shows a clean 360 degree view of a vehicle created using blender.

3.2 Wifiroom and HMB

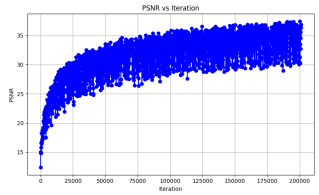
For the Wifiroom and HMB datasets, we performed post-processing using camera intrinsics, extrinsic, and the point cloud data. These scenes were trained using the same configuration as Fern. The rendered videos successfully captured all major objects in the scene without significant distortions. Reflections and shadows were rendered realistically and are visibly dynamic in the final outputs. These results highlight the strengths of the ray marching algorithm used in NeRF for photorealistic rendering.

3.2.1 PSNR evaluation for HMB Datasets

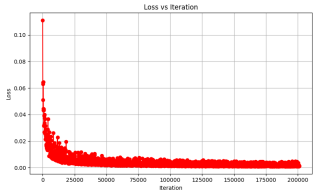
The original implementation fo Nerf-Orig uses $L = 10$ for $\gamma(x)$, $L = 4$ for $\gamma(d)$. We changed these values to $(20, 4)$ respectively to understand if these increase the quality of our video. The increase in quality was not very significant. We used PSNR[3] value to plot the graph and notice a small difference of average $0.10dB$ according to Figure 6.

3.2.2 Convergence of Algorithm

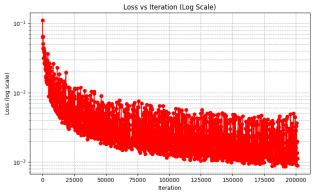
To check for convergence of NeRF-Orig, we calculate the PSNR value(Refer Section5.5) w.r.t ground-truth for HMB dataset, which decreases to give a band of average value 35 (by observation). We also observe the loss value to gradually decrease to constant band. Thus, the plot of loss signifies that the model will converge in finitely many iterations.



(a) The plot of PSNR vs Iteration for HMB Dataset, set at (20,6)

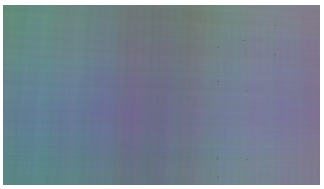


(b) Plot of Actual Loss vs Iterations of HMB Dataset, set at $L_{\gamma(x)}, L_{\gamma(d)} = (20, 6)$



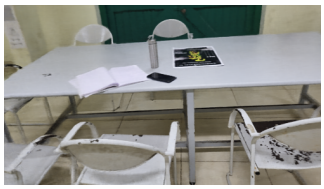
(c) Plot of Log Loss vs Iterations of HMB Dataset, set at $L_{\gamma(x)}, L_{\gamma(d)} = (20, 6)$

The trend of images obtained on the Wifiroom Dataset with respect to the number of iterations showing improvements in quality of images rendered are shown in Figure 3.

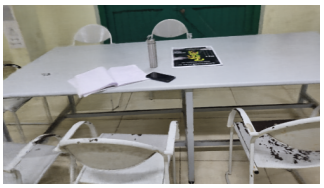


(a) 0 iterations

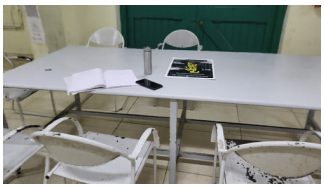
(b) 1,000 iterations



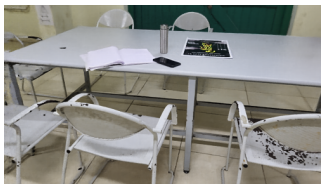
(c) 10,000 iterations



(d) 100,000 iterations



(e) 200,000 iterations



(f) 250,000 iterations

Figure 3: Images obtained at different iterations of NeRF

3.3 JNTS

Using the same configuration as Fern, we trained NeRF-Orig on both versions of our custom JNTS dataset. Due to memory limitations, we were unable to run the full 360-degree version. However, the one-sided dataset was successfully rendered within the same time constraints.
We observed distortions at the extreme left and right ends of the video. This is likely due to insufficient image coverage in those areas, combined with the complexity of the scene (trees, bushes, and buildings at the edges). As a result, NeRF struggled to accurately learn the depth and color information in these regions. In contrast, previous datasets containing well-defined solid objects with consistent volume and density showed significantly fewer such distortions. A metric of lower quality can be visualized w.r.t the Ground-truth from Figure 6.

4 Conclusion

NeRF proved to be a novel method to synthesize 3D scenes from images. Although we were able to produce reasonable results, NeRF required a significant amount of training time. We were able to achieve good quality videos on standard datasets (FERN and LEGO) but it could not produce satisfactory results for larger datasets.
To overcome the shortcomings, other versions of NeRF(Mip-NeRF[1], NeRF-SH[8], ZipNeRF[2], etc.) were created to observe faster running time and similar or better quality of rendered videos. Gaussian Splatting, a newer and faster method as compared to NeRFs which runs in real time. This addressed the deficiency of using NeRF to represent object and scenes as continuous functions.

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. *ICCV*, 2021.

[2] Jonathan T. Barron, Ben Mildenhall, Dor Verbin, Pratul P. Srinivasan, and Peter Hedman. Zip-nerf: Anti-aliased grid-based neural radiance fields. *ICCV*, 2023.

[3] Fernando A Fardo, Victor H Conforto, Francisco C de Oliveira, and Paulo S Rodrigues. A formal evaluation of psnr as quality measurement parameter for image segmentation algorithms. *arXiv preprint arXiv:1605.07116*, 2016.

[4] James Kajiya and Brian von herzen. Ray tracing volume densities. *ACM SIGGRAPH Computer Graphics*, 18:165–174, 07 1984.

[5] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering, 2023.

[6] Ruilong Li, Matthew Tancik, and Angjoo Kanazawa. Nerfacc: A general nerf acceleration toolbox, 2023.

[7] 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.

[8] Alex Yu, Ruilong Li, Matthew Tancik, Hao Li, Ren Ng, and Angjoo Kanazawa. PlenOctrees for real-time rendering of neural radiance fields. In *ICCV*, 2021.

5 Appendix

NeRF-Orig uses various optimization at volume rendering and ray marching steps. Below is an account for the optimization used by NeRF-Orig.

5.1 Optimization

5.1.1 Positional Encoding

It was observed that applying the neural network function directly to the 5D image coordinates (x, y, z, θ, ϕ) yielded poor results in regions with higher details. To obtain finer accuracies at places where the frequency of the variations is high, the features are encoded to a higher dimensional space, i.e., from $\mathbb{R}$ to $\mathbb{R}^{2L}$ with an encoding function:

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

The function $\gamma(.)$ is applied to each of the three coordinate values in $\mathbf{x}$.
These encoded features are then passed to the MLP function instead of the original coordinates, with $L = 10$, thus having 60 input features for NeRF-Orig. Notice that the viewing (i.e. the directions) are added in the 9^{th} layer of the MLP. This is because the RGB colors are view dependent meanwhile the volume density is view independent. The view dependence helps NeRF capture details like reflections and shadows.

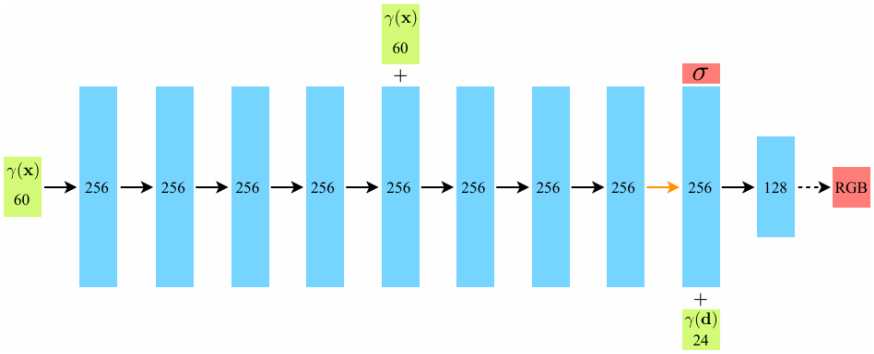


Figure 4: MLP used in NeRF-Orig

5.1.2 Hierarchical Volume Sampling

Densely evaluating the network at N query points is inefficient as free space and regions obscured from the view are sampled repeatedly.
The scene is represented using a coarse and a fine network. Sampling N_c points from the coarse network first gives an estimate of where points are more likely to be located, and a probability distribution is thus produced.
N_f points are sampled from this distribution, and the fine network is evaluated on $N_c + N_f$ points. This speeds up volume rendering without compromising on the quality of images.

5.2 Loss NeRF-Orig

Here, loss is the total squared error between true pixel color and rendered pixel color, for both coarse and fine renderings.

$$\mathcal{L} = \sum_{r\in\mathcal{R}} \left[\left\| \hat{C}_c(\mathbf{r}) - C(\mathbf{r}) \right\|_2^2 + \left\| \hat{C}_f(\mathbf{r}) - C(\mathbf{r}) \right\|_2^2 \right] \tag{3}$$

where: $\hat{C}_c(\mathbf{r}), \hat{C}_f(\mathbf{r}), C(\mathbf{r})$, are coarse volume predicted, fine volume predicted and ground truth respectively.

This function is differentiable, which enables backpropagation for optimization. Loss corresponding to $\hat{C}_c(\mathbf{r})$ is also minimized. Weights of MLP are iterated until loss converges.
We use the Adam optimizer, with a learning rate that decreases from 5×10^{-4} to 5×10^{-5} with time.

5.3 Dataset Collection

We collected 3 main datasets for our own testings. Below is a brief of description of our custom datasets formed.

1. **Wifiroom Dataset:** Depicts a bottle, a notebook and a Rhythmica Poster on the table. We took 45 images for a small angle range.

2. **Humanities Project Dataset (HMB):** We took a video covering 90 degree around the model. Note that we couldn't take 360 degree due to exhibition presentation limitations. We then used **ffmpeg** to create images at **3 fps** thus creating **90** images.

3. **JNT Statue Dataset(JNTS)**: Using Burst Mode of camera, we created 1400 images of 4000×2304 resolution images 360 degrees around the statue. This data was reduced in resolution and size to create two smaller sets, a 300 images 360-degrees dataset and a 100 image front-face of the statue.

A detailed description of the methodology of creation of both these datasets is present in Releases of our GitHub repository.

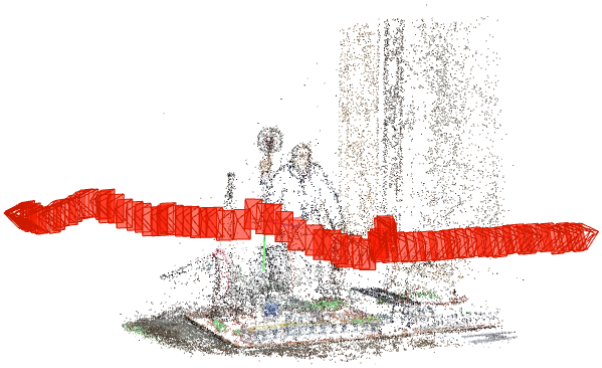
5.4 Brief Working of COLMAP

COLMAP is an automatic image-based 3D reconstruction tool that simply takes a folder of input images and produces a sparse and dense reconstruction in a workspace folder. It involves the use of Structure-from-Motion (SfM), which is the process of reconstructing a 3D structure from its projections into a series of images. The input is a set of overlapping images of the same object, taken from different viewpoints. The output is a 3D reconstruction of the object, and the reconstructed intrinsic and extrinsic camera parameters of all images. Typically, Structure-from-Motion systems divide this process into three stages:

- Feature detection and extraction
- Feature matching and geometric verification
- Structure and motion reconstruction

COLMAP reflects these stages in different modules, that can be combined depending on the application. After passing the input images in sequential order to COLMAP, we obtain -

- `colmap_output.txt` - contains the camera intrinsic values.
- `database.db` - Database containing feature matches, keypoints, descriptors, camera parameters, etc.
- `sparse/` - contains `camera.bin, points3D.bin, images.bin`. The point cloud for our 3D representation and the camera intrinsics which any model uses (including Gaussian Splatting, Other NeRF implementations, etc.) are contained in this.
- Compressed and Optimised Images for faster processing while maintaining quality of output to a good extent.



(a)

(b)

Figure 5: We depict the point cloud obtained by using COLMAP. Figure (a) shows the generated point clouds along with marked camera poses depicting position and angle of our camera. Figure(b) is a reference image from a similar angle as point cloud. The camera poses are very accurate to how the images were taken, with a very accurate point cloud representation of HMB dataset.

COLMAP is widely used for LLFF type datasets mainly to provide camera views and point clouds. It is not used for Blender and deepvoxel type datasets since they are not captured by any camera, instead, they themselves or a separate script outputs the required `transforms.json`.

5.5 PSNR-Loss Evaluation for NeRF-Orig

For HMB dataset, we perform the following analysis. The aim is to compare the difference in the video quality using PSNR[3] for two videos formed for $L_{\gamma(x)}, L_{\gamma(d)} = (10, 4), (20, 6)$ which represent the dimension for positional encodings.

5.5.1 Brief Introduction to PSNR

PSNR is a measure of the quality of image reconstruction, expressed in decibels (dB). It quantifies how close the rendered image is to the ground-truth image.

$$\text{PSNR} = 10 \log_{10} \left(\frac{\text{MAX}^2}{\text{MSE}} \right)$$

where MAX is maximum possible pixel value, MSE is mean-squared error between rendered image and ground-truth image.
A high PSNR implies rendered image is close to ground-truth image, whereas a low PSNR indicates lower quality of synthesis. PSNR represents a measure of the peak error.

5.5.2 Effects of Increasing dimension of Positional Encodings

The original paper uses $L_{\gamma(x)}, L_{\gamma(d)} = (10, 4) =: L_1$. We experimented to see if the quality of the video improves on increasing the value of these positional encodings to $(20, 6) =: L_2$. We obtained that increasing the value of $L_{\gamma(x)}, L_{\gamma(d)}$ does increase the video quality, but only by a very small amount of $0.10dB$.

Average PSNR of Groundtruth (GT) vs Video_104(L_1): 11.35 dB
Average PSNR of Groundtruth (GT) vs Video_206(L_2): 11.45 dB
Average PSNR of Video_206 vs Video_104: 0.10 dB

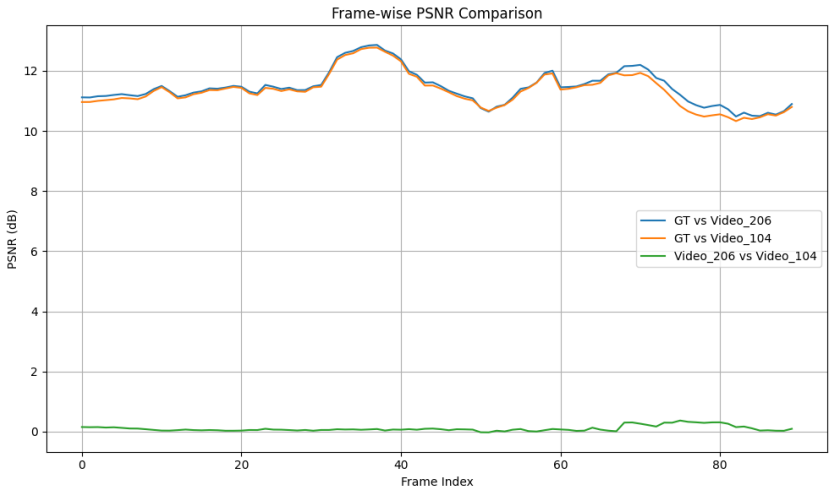


Figure 6: Plot between PSNR values of Groundtruth and Videos at L_1, L_2 as well as their relative. It shows the small increase in PSNR on changing L. It also shows quality of groundtruth is much higher than videos rendered by NeRF-Orig.

5.6 Improvements on NeRF-Orig

5.6.1 Mip-NeRF

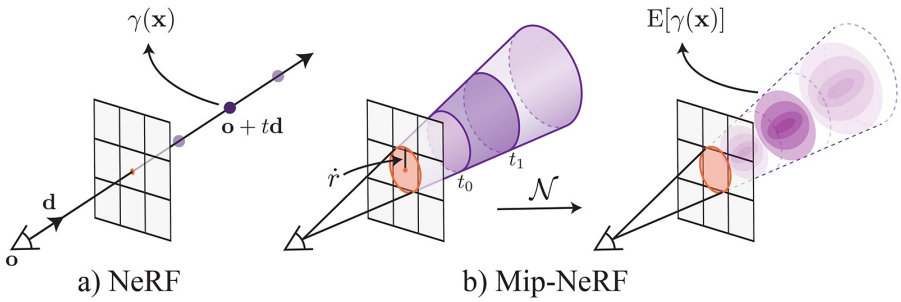


Figure 7: Comparison of NeRF and Mip-NeRF

This modification of NeRF takes as the input a Gaussian region of space instead of a point. The ray-marching step involves a cone instead of a ray. Sampling is done by dividing the cone into multiple conical frustums, and evaluating the model on gaussians approximating each one.

Mip-NeRF[1] substantially improves upon the accuracy of NeRF, and this benefit is even greater in situations where scene content is observed at different resolutions (i.e. setups where the camera moves closer and farther from the scene). On one benchmark, mip-NeRF is able to reduce error rates relative to NeRF by 60% on average.

5.6.2 NeRF-SH and PlenOctrees

NeRF-SH[8] is a modification of NeRF-Orig which changes the output from the view-dependent radiance values (r, g, b) to a view-independent set of spherical harmonic coefficients. The model input also changes from (x, y, z, θ, ϕ) to only (x, y, z). An additional sparsity term is added to the loss which penalizes volume density in inaccessible areas.

The set of spherical harmonic functions so obtained encodes the radiance emitted in all directions. An advantage of this is that the output for a specific point needs to be calculated only once,

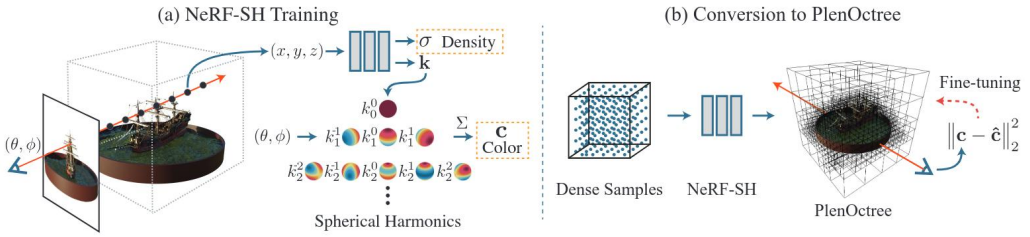


Figure 8: Architecture of NeRF-SH+PlenOctrees
[**?**]

after which it can be cached, as opposed to having to be calculated for each viewing direction in NeRF-Orig.

This model is converted to a sparse voxel representation (**PlenOctree**). First, the model is evaluated on a grid of points. Then, the low-density points are eliminated, and the remaining points are associated with a voxel each. Random points inside each voxel are sampled to reconstruct the spherical harmonic coefficients. This set of voxels is then converted to an **octree** for efficient ray intersection calculation. The final octree can then directly be queried, removing the need for performing inference for the MLP altogether.

Through this approach, the output performance has been improved to **nearly 3000 times** that of conventional NeRF, making it real-time, while also maintaining the final output quality. The training time can also be improved – by cutting off training early and then fine-tuning the resultant octree itself. In many cases, this gives a superior output compared to waiting for the NeRF training to complete.

5.6.3 Our Experiments with Improved NeRF Algorithms

We used NeRFStudio for using these algorithms. Due to lesser GPU vRAM, we were not able to run models but **nerfacto** which is NeRF version by NeRFStudio itself, combining best ideas from NeRF and instant-NGP (Instant Neural Graphics Primitives) to achieve faster training, better quality, and support for real-world datasets.
We ran the above for Wifiroom and Fern Datasets. We obtained a much clear and better quality video, properly showing reflections, shadows and objects as NeRF-Orig shows. The time taken for each running was about 3 hours, which is significantly lesser than NeRF-Orig. This shows the increase in accuracy, improvement in time taken and better quality of newer and improved models.

5.7 3D Gaussian Splatting [5]

Although NeRF and it's variants can achieve high quality images and videos, they are either hindered by high costs for training and rendering, or trade off quality for speed. 3D Gaussian Splatting addresses these issues and provides State-of-the-Art visual quality along with real-time rendering.
3D Gaussian Splatting renders scenes using tiny ellipsoids called *Gaussian Splats* that contain information about it's position (x, y, z), color (r, g, b), size, and transparency (α).

The input is a set of images. COLMAP is used to estimate a 3D point cloud from this set of 2D images.
3D Gaussians are used for scene representation as they are differentiable, can be easily projected to 2D splats, and allow fast α-blending. Each point from the point cloud is converted into a Gaussian that is described by a full 3D covariance matrix Σ.

Optimization of position p, α, Σ and spherical harmonics, which represent the color, is done using stochastic gradient descent. First, the Gaussians are rasterized, that is, projected into 2D splats, sorted by depth and blended together.
Next L_1 loss is calculated based on the difference between the rasterized image and original image. The Gaussian parameters are then adjusted according to loss,

- If the Gaussian is too small, clone it and adjust parameters
- If the Gaussian is too large, split it and adjust parameters

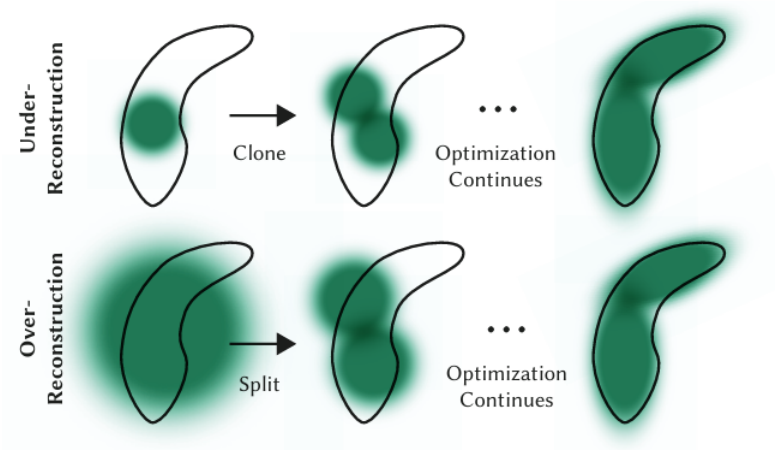


Figure 9: Adjusting Gaussian Parameters to fit Image

Gaussians with very low values of α are removed. Once trained, the renderer allows real time navigation of scenes, that can be accessed via interactive viewers like SIBR.

5.7.1 Our Experiments with Gaussian Splatting

We used Gaussian Splatting for all 3 of our custom dataset. We obtained a very clear and high quality 3D view of the scene. Gaussian Splatting GitHub Repository provides an inbuilt 3D real-time viewer of the rendered model which has been visualized and outputs are attached in the Releases.

5.8 Contribution of Team Members

1. Anirudh Gupta:
   (a) Reading Paper NeRF-Orig.
   (b) Collection of all self-made 3 Datasets and Post-Processed them.
   (c) Implementation of NeRF-Orig on all datasets.
   (d) PSNR metric analysis for HMB Dataset.
   (e) Implemented Gaussian Splatting on 3 mentioned datasets.
   (f) Report writing(Methodology, Experiments, Appendix - 5.1,5.2,5.3) and proof reading.
   (g) Video and Presentation Making.
2. Saksham Agrawal:
   (a) Reading Paper NeRF-Orig.
   (b) Collection of all self-made 3 Datasets.
   (c) Implementation of NeRF-Orig.
   (d) Implementation of Nerfacto from NeRFStudio on mentioned datasets.
   (e) Report Writing(Introduction) and proof reading.
   (f) Video and Presentation Making.
3. Shivey Ravi Guttal:
   (a) Reading Paper NeRF-Orig.
   (b) PSNR metric analysis for HMB Dataset.
   (c) Report writing(Introduction, Appendix - 5.3, 5.5) and proof reading.
   (d) Video and Presentation Making.
4. Aditya Arsh:

(a) Reading Paper NeRF-Orig
(b) Reading papers on improvements to NeRF (PlenOctrees and Mip-NeRF).
(c) Report writing (Appendix - 5.6) and proof reading.
(d) Video and Presentation making.

****