Neural Networks and Deep Learning
Machine Learning Optimization Algorithms

Project Report On

Novel View Synthesis and 3D Reconstruction with Neural Radiance Fields

Authors

Vansh Shah
C093

Siddh Sanghvi
C079

Under the Mentorship of

Prof. Archana Nanade

Prof. Abhay Kolhe

Department of Computer Engineering
Mukesh Patel School of Technology Management & Engineering
NMIMS (Deemed-to-be University), Mumbai

22nd March 2024

1. Introduction

Scene understanding is a computer vision task that involves implementation of algorithms that allow a computer to represent and process a 3D environment. The field of scene understanding and novel view synthesis has witnessed significant advancements with the involvement of deep learning techniques. This research project explores the capabilities of Neural Radiance Fields (NeRF) for reconstructing 3D scenes and explores complementary optimized methods for further analysis. NeRF, introduced in 2020, revolutionized the field by leveraging deep learning MLP based architectures to efficiently learn and represent a scene's radiance and geometry from a set of 2D images. This research project implements and evaluates NeRF for its effectiveness in reconstructing scenes. This project also explores and compares the potential of K-Planes, a recent approach that complements NeRF by offering a more optimized and sophisticated representation technique. To understand the core functionalities and practical implementations behind NeRF and other related methods, this project also explores the process and techniques of ray tracing and volumetric rendering. Ray tracing simulates the path of light through a scene, allowing for realistic image generation and also facilitating novel view synthesis. Volumetric rendering is a density based differentiable rendering technique that builds upon ray tracing by considering the opacity and color information within a 3D volume, leading to more accurate reconstructions. Furthermore, this research explores the application of ray marching, a technique for efficiently traversing a 3D scene along rays, to extract surface meshes from the reconstructed volume. This allows for the creation of a tangible 3D model from the NeRF representation.

By combining NeRF with K-planes, ray tracing, volumetric rendering, and ray marching, this research project aims to achieve a comprehensive understanding of scene reconstruction and explore the potential for generating high-fidelity 3D representations and extractable meshes from sparse input data.

Keywords
Neural Radiance Fields, MLPs, Novel View Synthesis, Ray Tracing, Volumetric Rendering, Differentiable Rendering, K-Planes, Ray Marching, Mesh Extraction, 3D Mesh, 3D Reconstruction.

2. Literature Survey

3D Scene interpretation and novel view synthesis have long been difficult problems in computer vision, with researchers exploring various techniques to reconstruct 3D environments from sparse input data. Traditional techniques, such as multi-view stereo [9] attempted to extract depth information from multiple overlapping images. However, these approaches frequently struggled with occlusions, texture-less regions, and scenes with complex geometry.

Structure from motion [1] approaches aimed at estimating both camera poses and 3D scene geometry from a collection of unsorted images. While effective for certain situations, these methods required significant manual involvement and often failed in extracting fine details or handling challenging overexposed images.

Another technique which was involved in scene representation was voxel-based representations [8]. In this technique the 3D scene was divided into a regular grid of voxels, each comprising information about the occupancy and color. These representations were computationally expensive and suffered from quantization artifacts, making them unsuitable for high resolution scenes.

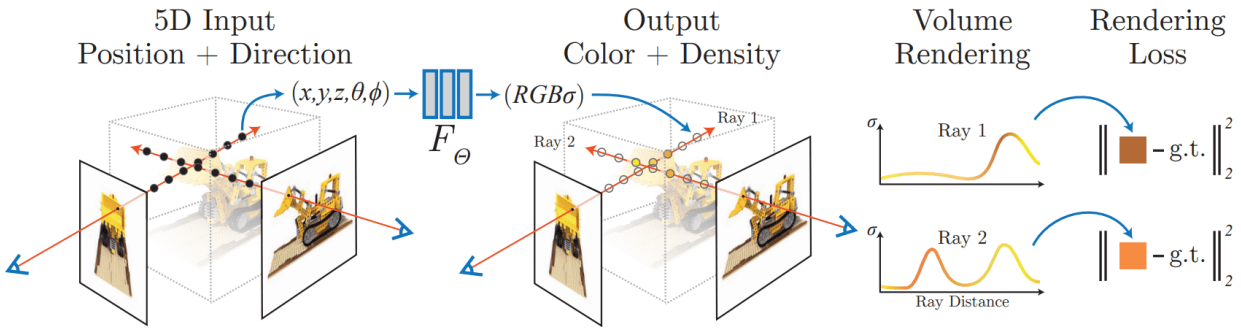
Prior to NeRF, multiple techniques were explored for understanding 3D scenes and generating novel views, but these methods often struggled with capturing fine details and required manual interference with complex scenes.

Neural Radiance Fields (NeRF) [2] transformed the field by introducing deep learning and neural networks to represent a scene's radiance and geometry from 2D images. Researchers have improved and developed variants of NeRF, e.g., PixelNeRF (Yu et al., 2021) for unbounded scenes and NeRF++ (Zhang et al., 2020) for improved rendering quality and convergence.

NeRF's [2] representing dense volumetric output can be inefficient leading to a considerable computation time to render the output. Researchers have explored improvements, e.g., Kernel Predicting Neural Rendering (Srinivasan et al., 2021) for complex light transport, and Scene Representation Networks (Sitzmann et al., 2019) for single-viewpoint rendering. However, integrating complementary techniques like K-Planes [3], ray tracing, volumetric rendering, and ray marching for efficient scene reconstruction and mesh extraction is still a relatively newer practice.

3. Proposed Work

We propose Neural Radiance Fields [2] (NeRF) models for scene understanding. NeRF models represent a scene using deep learning. By overfitting to the scene and using positional encoding, a NeRF model can capture and represent complex lighting effects and finer details. Unlike the earlier methods that rely on multi-view stereo reconstruction, NeRFs have the capabilities to learn from sparse views and generalize well to novel view points, making them a powerful tool for creating realistic and detailed 3D scene understanding.

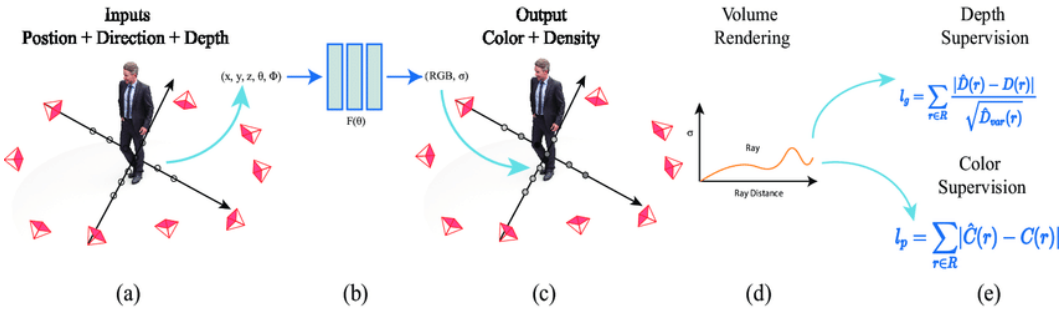


1. NeRF Model Architecture Source

We also explore the implementation of K-Planes NeRF [3], an optimized version of NeRF that uses explicit planes to represent 3D scenes. This superior method allows us to gain an approximate 1000x performance compared to the traditional NeRF approach. We now discuss the entire pipeline and implementation of these models.

3.1. Understanding NeRF Model

The NeRF model has a rather simple architecture. The core model consists of a fully connected deep and dense network having around 256 hidden layers. The earliest implementations of NeRF used Multilayer Perceptrons as their hidden units. The model takes in a 5D vector (x, y, z, θ, φ) as input. The (x, y, z) component are coordinates representing a point in 3D space. This point is usually the origin point for a ray in a w x h x 3 dimensional plane. The θ component represents a directional vector of a ray with respect to the centre of the 3D plane. x, y, z and θ can be used with another time component t to trace the movement of the ray across a plane for a given timeframe t' essentially giving us a simplified version of ray tracing. The φ component is the angle of elevation of the ray relative to the x, y plane.



2. Inputs and Outputs of NeRF Source

The NeRF model overfits to the scene data in order to memorize the entire 3D scene representation. This means there is only one NeRF model per scene and it does not follow the traditional deep learning concept of generalization. The output from the NeRF model is a 4D vector (R, G, B, alpha). The R G B component of the output represents the Red Green and Blue value for a point predicted by the NeRF model in the particular scene. The alpha component represents the accumulated transmittance which is used t represent the opacity of that point and occlusion produced by that point. The final image or novel view synthesis process of NeRF s performed through volumetric rendering [4] wherein a novel 5D input that represents a non-existent view point is used as input. This point casts rays onto the representation and the model predicts the output and accumulations. This output is then rendered using volumetric rendering to reconstruct a novel image. Volumetric rendering [4] can be mathematically represented as:

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

Where $C(r)$ is the expected colour of camera ray $r(t) = o + td$ and $\sigma(x)$ is the differential probability of a ray terminating at an infinitesimal particle at location x. $T(t)$ is the accumulated transmittance of the traced ray. This rendering function can be differentially represented as:

$$\hat{C}(r) = \sum_{i=1}^{N} T_i \left(1 - \exp(-\sigma_i\delta_i)\right)ci \text{ where } T_i = \exp\left(-\sum_{j=1}^{i-1} -\sigma_i\delta_i\right)$$

Now we discuss the various data acquisition and preparation methods involved in training a NeRF model.

3.2. Data Acquisition: Structure from Motion Approach using Colmap

Colmap is an open-source software used to create sparse and dense 3D reconstructions from multiple images of a scene. It uses the photogrammetric process of extracting structure from motion using extraction and feature matching to provide sparse scene reconstructions.

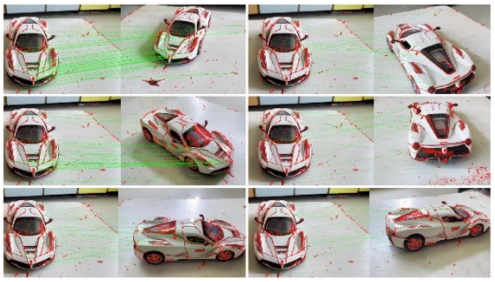
3.2.1 Feature Extraction

Colmap [5] utilizes a detector-descriptor apporach for feature extraction.

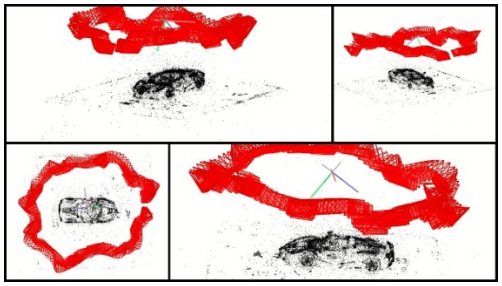
Detection: COLMAP leverages corner detectors like Harris corner detection or SIFT (Scale-Invariant Feature Transform). These algorithms identify points in the image where there's significant variation in intensity across neighboring pixels in multiple directions. These points often correspond to edges, corners, or blobs and are considered informative for reconstruction.

Descriptor Extraction: COLMAP employs a descriptor like SIFT or SURF (Speeded Up Robust Features) to create a unique mathematical representation of each feature's local neighborhood. This descriptor captures the key characteristics of the region around the feature point, including its intensity distribution and gradients. It's crucial for matching these features across different images despite variations in lighting, viewpoint, or even small occlusions.

3. Feature Extraction



4. Exhaustive Feature Mapping



5. Sparse 3D Reconstruction

3.2.2. Feature Mapping

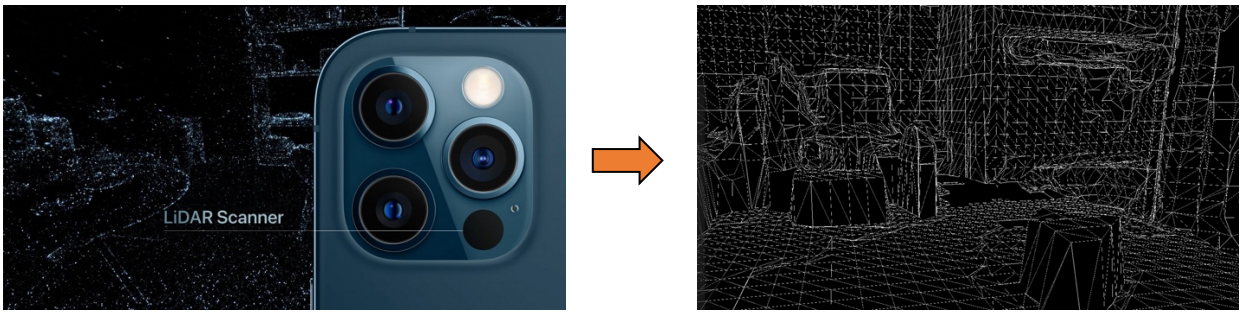
The extracted features are then exhaustively matched and overlapping images are found. This allows analysing the feature disparities across the various frames allowing the determination of structure of the scene based on feature disparity.

3.2.3. Sparse Reconstruction

After establishing feature correspondences across images, it estimates the 3D locations of these features (points) and the camera poses (position and orientation) for each image. This is achieved through iterative bundle adjustment, which minimizes the reprojection error - the difference between the locations of the features projected back from the estimated 3D points and camera poses, and their actual positions in the images. This process refines the initial estimates, resulting in a sparse 3D model consisting of these reconstructed 3D points and camera locations.

3.3. Data Acquisition: LiDAR Assisted

Light Detection and Ranging is a mapping technology that can capture the 3D structure of an object by calculating the time taken by a shot laser pulse to return to origin. LiDAR technology has been adopted by iPhones since the iPhone 12 line-up allowing convenient pose extraction and data creation of 3D objects. We used the application Polycam which utilises the LiDAR sensor to scan a 3D object and export it in the form of poses (x,y,z) and angles.



6. LiDAR Scan

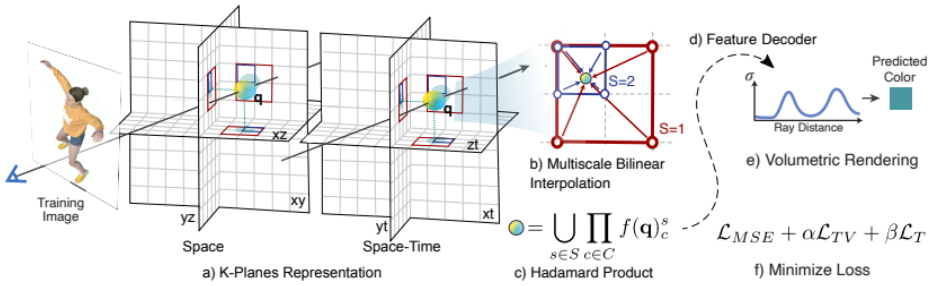
3.4. Optimization with K-Planes

K-Planes [3] is an optimized approach to the NeRF proposed above. In this technique a d-dimensional scene is translated into K 2-dimensional planes i.e. $k = {}^{d}C_{2}$ which enables us memory efficient storage and faster rendering for the scene

3.4.1. Scene Representation

In K-Planes as said above the d-dimensional is factorized into K 2D planes where each plane is a combination of 2 dimensions from dimensional scene. For static 3-dimensional

scenes are factorized into triplanes $k = {}^3C_2 = 3$ spatial planes i.e. P_{xy}, P_{yz}, P_{xz}. For 4-dimensional dynamic scenes are factorized into hex planes $k = {}^4C_2 = 6$ spatial-temporal planes where, including the spatial only planes three new planes are introduced i.e. P_{xt}, P_{yt}, P_{zt}



7. *K-Planes Architecture*

3.4.2. Bilinear Interpolation

Features of the D-Dimensional coordinate q is obtained which is normalized and projected onto the K-Planes. Let's take an example of dynamic scene where $q = (i, j, k, t)$ and is normalized between $[0, n)$ projecting it onto 6 planes

$$f(q)_c = \psi(P_c, \pi_c(q))$$

Here the coordinate q is projected by π_c onto the c^{th} planes and the φ denotes the billinear interpolation of a point in 2-dimensional grid. This is repeated for each plane $c \in C$ and obtain a feature vector $f(q)_c$. This featconda ure vector is concatenated using the Hadamard produce a final feature vector

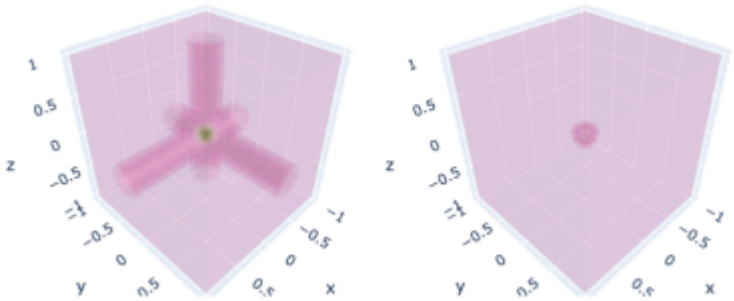
$$f(q) = \prod_{c \in C} f(q)_c$$

A Multilayer Perceptron is used for the interpreting the feature vector into colour and density

3.4.3. Hadamard Product

Hadamard Product [3] has special properties like feature selection where some values are effectively "turned off" or on based on their values and it also allows spatial localization

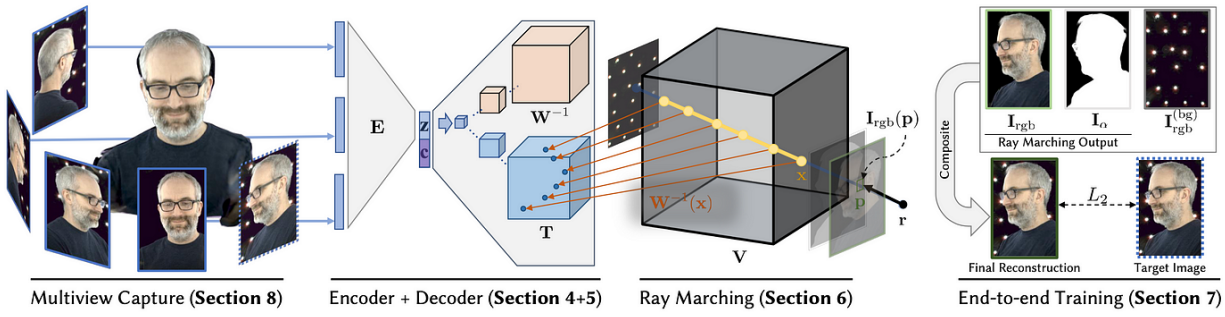
which is not possible if we simply add the products. This lowers the burden while decoding the feature vector causing improvements while rendering the scenes



8. Hadamard vs Addition

3.5. Mesh Extraction and Marching Cubes

Once a NeRF model understands a scene, it creates an internal representation of that scene, representing it as a scene of density fields. Methods like Ray Marching are used to traverse these density fields for novel view synthesis. A similar approach is also used for mesh extraction. Marching cubes algorithm is a polygonal mesh extraction algorithm which interprets the density field as a grid of voxels and using thresholding determines whether the point is dense enough to be a solid voxel. Marching Cubes utilizes a specific density value (like 0.5) as a threshold. Marching also Cubes analyses each voxel and its neighbours. If a voxel's density is above the threshold and its neighbour's density is below it indicates the surface intersects that voxel. Based on the intersection patterns within each voxel, Marching Cubes generates triangles at the exact intersection points, effectively forming the 3D mesh representing the object's surface.



9. Complete Pipeline Source

3.6. NeRF Studio and Open-Source Assistance

NeRF studio [6] is an open-source, modular and optimized framework that can be used to pre-process data, train and load different NeRF architectures, research and experiment with checkpoints and also visualize training and extract 3D mesh or point cloud representations from NeRF models. One of our main limitations when dealing with high fidelity data such as real world data was the memory constraints. A typical orbital data consisting of 223 images of an object with a FHD resolution (1920 x 1080) requires approximately 8,32,34,30,400 bytes (223 x 1920 x 1080 x 3 x 6) that roughly translates to around 8.32 GB of graphics (accelerated) or RAM memory. This much overhead is required just for loading the data, processing and training will further require computational resources. Fortunately software like NeRF Studio has various optimizations made to tackle these constraints such as chunk data processing, lazy loading and also leverages Nvidia's proprietary tinycudann [7] library that gave us further increase in performance and scalability. This allowed us to train NeRF models over high fidelity scenes.

We have also used other open-source software such as Colmap [5] which was used to pre-process raw data. Colmap offers both a command line as well as GUI option to process the data and extract features such as camera poses, angle and elevation. We also use ffmpeg to process videos to individual frames, MeshLab and Blender to visualise extracted meshes.

4. Results and Discussions

In this research report we have implemented and trained both NeRF and K-Planes models. To perform measured inferences, we have trained our models over labelled synthetic data from the NeRF Synthetic Blender dataset. We have used the Lego example as our target scene for training and testing.

4.1. Synthetic Dataset

The motive behind use of synthetic data is the availability of accurately calculated and controlled intrinsic parameters such as focal length of camera, position of camera, projection vectors and more. In order to calculate intrinsics of real world data, specialized hardware sensors are required. Synthetic data therefore provides us a quantified target which we can use to accurately evaluate the performance of our model. We have used the Lego data from the Blender Dataset.


```
"camera_angle_x": 0.6911112070083618,
"frames": [{"file_path":
"./train/r_0","rotation":0.012566370614359171,"transform_matrix": [[-
0.9999021887779236,0.004192245192825794,-0.013345719315111637,-
0.05379832163453102],[-0.013988681137561798,-
0.2996590733528137,0.95394366979599,3.845470428466797],[-4.656612873077393e-
0,0.9540371894836426,0.29968830943107605,1.2080823183059692],[0.0,0.0,0.0,1.0]]},
```


10. Synthetic Data Sample

4.2. Training And Parameters

Both the NeRF and K-planes models have been trained on the same dataset using the same optimizers and the same loss functions for 16 epochs. The training methodology involves backpropagation in order to optimize the differentiable rendering function.

4.2.1. Loss Function

We use the Mean Squared Error (MSE) Loss function. The core objective is to minimize the difference between the rendered image by the NeRF model and the actual ground truth image of the scene captured from the corresponding camera pose. MSE therefore facilitates pixel by pixel gradient calculation.

$$\frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

4.2.2. Optimizer

Due to its adaptive nature, ease of use and state of the art performance, the preferred optimizer was the Adam optimizer. We start with a relatively low learning rate of $\alpha=5^{-4}$ and use a multistep scheduler to further optimize the learning rate.

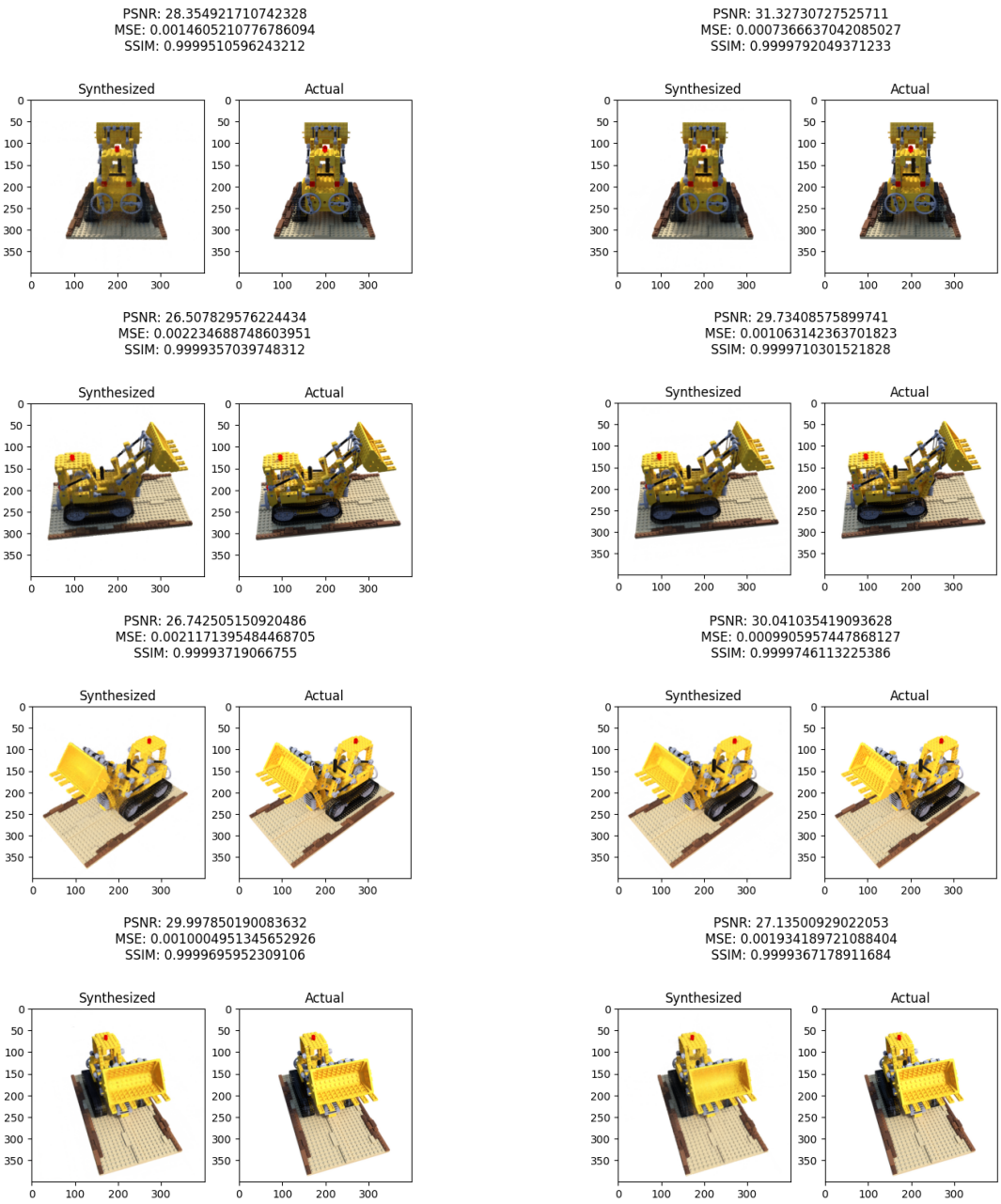
$$w_{t+1} = w_t - \frac{\alpha_t}{(v_t - \varepsilon)^{0.5}} * \left[\frac{\delta L}{\delta w_t}\right] \text{ where } v_t = \beta v_{t-1} + (1-\beta) * \left[\frac{\delta L}{\delta w_t}\right]^2$$

4.3. Performance and Comparisons

	K-Planes	NeRF
MSE	**26.63**	29.92
PSNR	0.0022	**0.0010**
SSIM	0.99	**0.99**
Render Time(s)	**3.90s**	10.13s

11. Averaged Metrics

The above table compares the averaged performance of both models trained on the same dataset. As we can observe, the K-Planes model has a more superior MSE and also 2.5x faster average render time compared to the traditional NeRF. NeRF has a better Peak Signal to Noise Ratio (PSNR) and Structural Similarity (SSIM) however these superiorities are marginal.

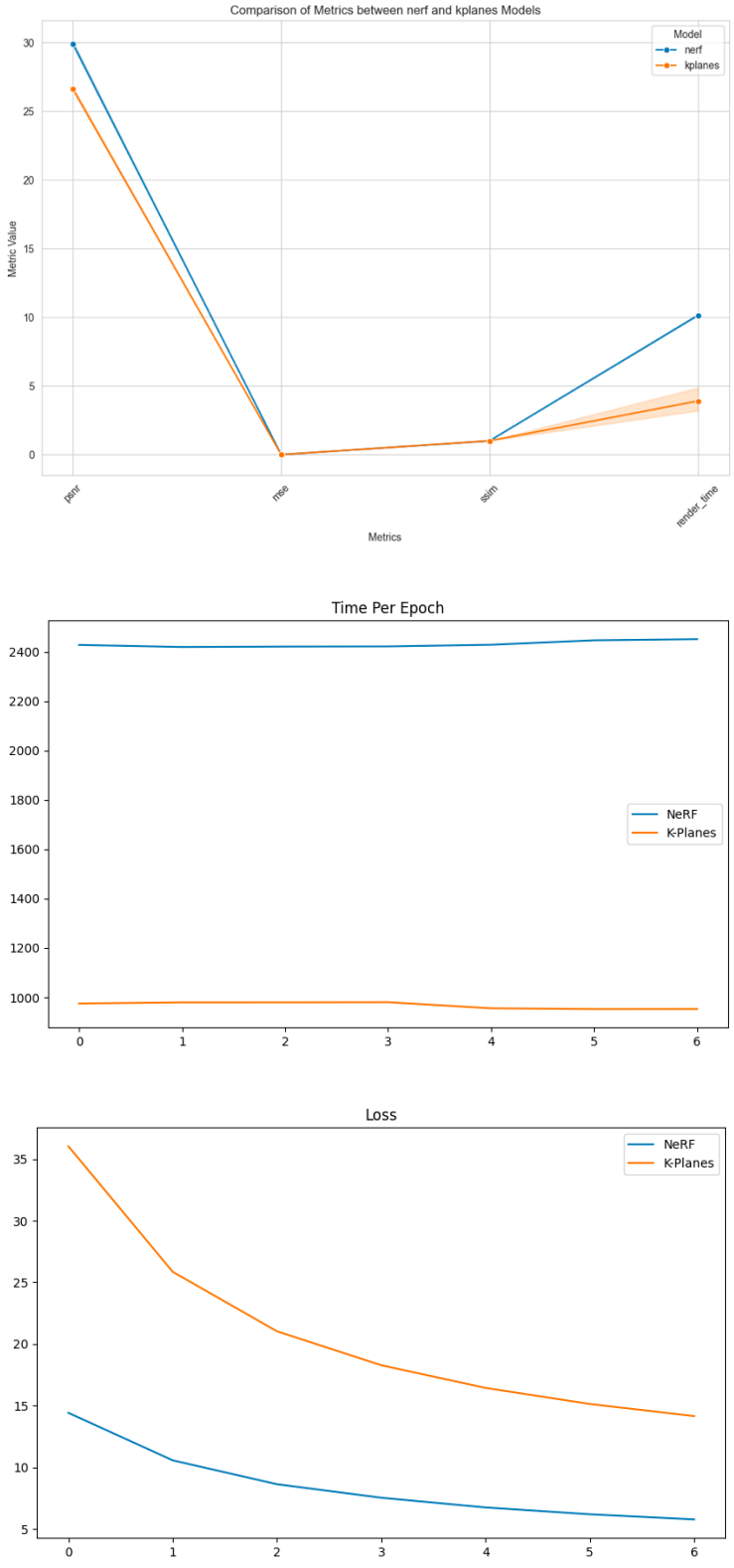


12. Novel Views

We can also assess the performance of NeRF and K-Planes by visualizing their recorded training data.

As indicated by the graphs, the K-Planes model has a much shorter training time taking an average of **17 minutes** (967s) per epoch. This means it is 60% faster compared to the NeRF model that averages around **41 minutes** (2431s) per epoch. The k-planes model also has a much shorter average render time whilst still delivering equivalent if not identical performance compared to NeRF. As seen in the first figure, NeRF and K-Planes have almost identical MSE. The K-Planes model however does converge slower than NeRF, however due to its much shorter training time, the increase in epochs is a fair compromise and the model will always fit faster compared to NeRF.

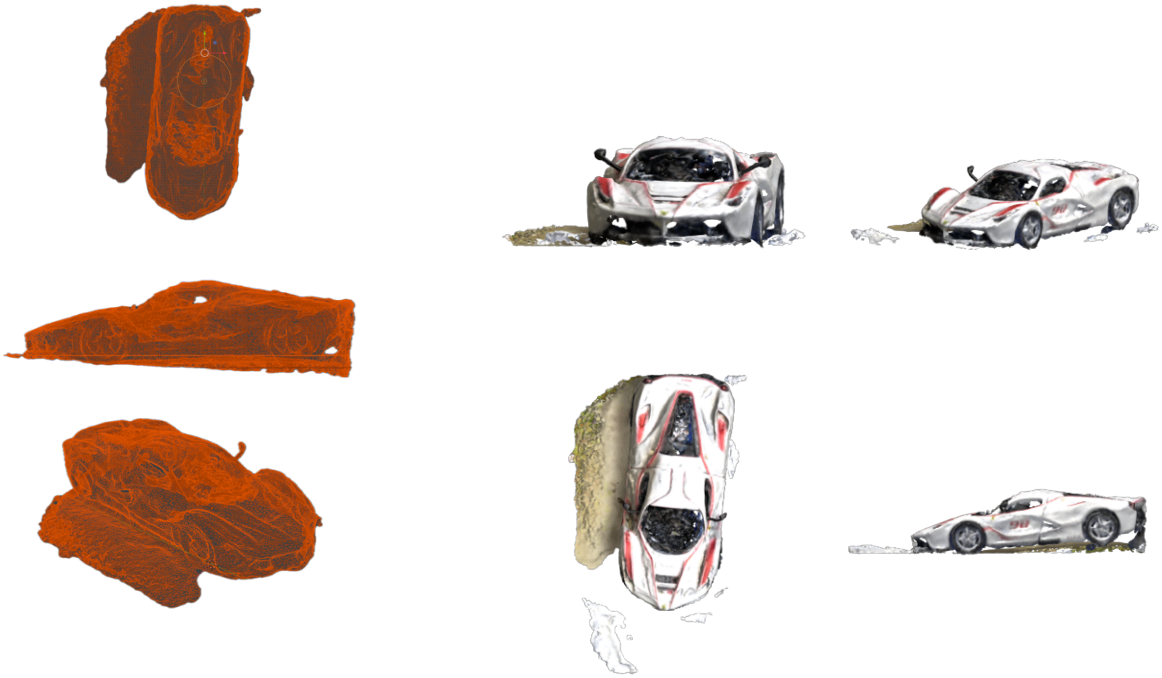
All models have been trained and tested using GPU acceleration. The GPU used is a single Nvidia RTX 2070 Super with 8 GB VRAM. The models have been implemented using PyTorch.



13. Training Plots

4.4. High Fidelity Outputs from NeRF Studio

We use NeRF Studio to process and train high fidelity data due to its optimized features such as lazy loading and its compatibility with Nvidia GPUs due to its integration of tinycudann.



14. Generated Meshes and Rendered Objects

5. Conclusion

In conclusion in this project, we have explored the domain of 3D reconstruction of sparse inputs using Neural Radiance Fields (NeRF). We have attempted to achieve a 3-dimensional representation and an extractable mesh of a scene from sparse input using deep learning and volumetric rendering. Furthermore, we attempted to use K-Planes to optimize the existing NeRF model and have achieved modest results while rendering the output. The results give us an understanding of the potential of these techniques for improving the efficiency of 3D modelling and computer vision applications.

6. Future work

Integrating knowledge distillation and optimizing the NeRF model to develop smaller and faster NeRF models. We aim to create models that could be deployed in real time applications and devices like mobile phones with constraint of resources. Knowledge distillation is an optimization technique that is used transfer knowledge to a model from a large complex model so that we could develop more efficient versions of the model.

Development of a robust architecture for so that it could dynamic scenes with high efficiency. In the future we aim to develop a deep learning architecture that would interpret and render 4-dimensional dynamic with moving objects or a moving camera.

Development of a user-friendly tool captures real world scenes from sparse input. One of the challenges we faced while training nerf is that it would require large number of images to train and reconstruct a scene. In the future we wish the optimize the nerf for fewer no of images to make it accessible for a wide array of applications.

Development of personalized avatars and living spaces with unique styles and preferences. By using user-specific data we can develop customized avatar in living spaces. In the future we aim to t and pretrain NeRF model to generate real life avatars that reflect unique styles and preferences of individual.

Integrating Open AI's SORA into our pipeline to develop a tool that generates scenes from text. Open AI has developed a new model SORA that generates videos from text. By integrating this into our pipeline we could develop 3D scenes from the textual description of a scene.

Development of computer vision applications that create 3D maps and integrate them into robots for better navigation. In the future we aim to develop 3d maps of

environments and integrate them with autonomous vehicles and robots to avoid obstacles and better navigation.

Incorporating physics simulation into our model so that it could capture the properties of objects into our scene. In the future in our simulation, we want to capture the physical properties of an object like mass, friction and how they interact with objects. This could lead to development of more realistic renders and especially in applications that require object manipulation.

7. References

[1] J. L. S. J.-M. Frahm, “Structure-From-Motion Revisited,” in IEEE Conference on Computer Vision and Pattern Recognition, 2016.

[2] B. M. P. P. S. M. T. J. T. B. R. R. R. Ng, “NeRF: representing scenes as neural radiance fields for view synthesis,” Communications of the ACM, vol. 65, no. 1, p. 25, 2020.

[3] S. F.-K. G. M. F. R. W. B. R. A. Kanazawa, “K-Planes: Explicit Radiance Fields in Space, Time, and Appearance,” in IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2023.

[4] M. N. L. M. M. O. A. Geiger, “Differentiable Volumetric Rendering: Learning Implicit 3D Representations Without 3D Supervision,” in IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2020.

[5] T. J. P. I. B. W. R. M. P. Hanrahan, “Ray tracing on programmable graphics hardware,” in SIGGRAPH'05, 2005.

[6] M. T. E. W. E. N. R. L. B. Y. T. W. A. K. J. A. K. S. A. A. D. M. J. K. A. Kanazawa, “Nerfstudio: A Modular Framework for Neural Radiance Field Development,” in SIGGRAPH'23, 2023.

[7] T. M. A. E. C. S. A. Keller, “Instant neural graphics primitives with a multiresolution hash encoding,” in ACM Transactions on Graphics, 2022.

[8] R. Szeliski, "Stereo Algorithms and Representations for Image-based Rendering.," in BMVC, 1999, 1999.

[9] S S. B. C. J. D. D. S. R. Szeliski, "A Comparison and Evaluation of Multi-View Stereo Reconstruction Algorithms," in CVPR , 2006.