RBE 549: Project3 Phase 2 - Deep Learning Approach - NeRF (Using 1 late day)

Smit M Shah
Email: smshah1@wpi.edu
Worcester Polytechnic Institute

Rigved Sanku
Email: rsanku@wpi.edu
Worcester Polytechnic Institute

***Abstract*—This project focuses on implementing computer vision techniques to reconstruct 3D scenes and derive camera poses using a method known as Structure from Motion (SfM). SfM leverages a sequence of 2D images to reconstruct the three-dimensional structure of a scene. Similar to LiDAR, SfM can generate 3D models based on point clouds. The method employs the principle of stereoscopic photogrammetry, utilizing triangulation to compute the relative 3D poses of objects from stereo pairs of images.**

I. Phase 1 SfM

A. Introduction

In the 3D scene reconstruction process, a crucial step within the Structure from Motion (SfM) pipeline is feature matching, where common points in the scene are identified and outliers are removed using the RANSAC algorithm. Following this, the Fundamental matrix is estimated, establishing the relationship between corresponding points in two images captured from different viewpoints. This matrix is then used to calculate the Essential Matrix. Camera poses are determined and the correct one is chosen based on cheirality constraints using Triangulation. This process is repeated for multiple perspectives, and ultimately, the re-projection error is computed and minimized through bundle adjustment to refine the reconstruction.

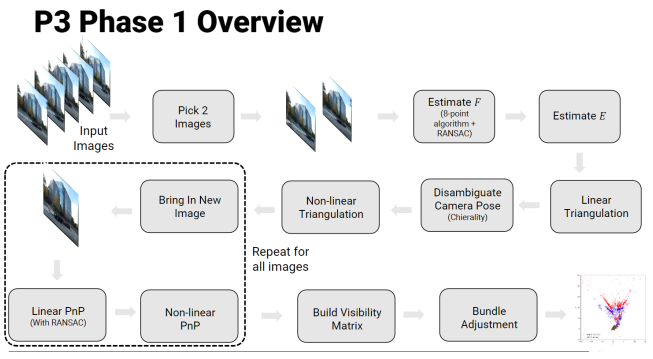

Fig. 1. Overview of SfM Pipeline

The traditional method consists of following steps:

1) Feature Matching and Outlier rejection using RANSAC
2) Estimating Fundamental Matrix (F)
3) Estimating Essential Matrix (E)
4) Estimate Camera Pose from Essential Matrix
5) Check for Cheirality Condition using Triangulation
6) Perspective-n-Point
7) Bundle Adjustment

B. Feature MAtching and RANSAC using Fundamental matrix

High-quality features are crucial for the effectiveness of computer vision algorithms. In this context, the SIFT feature descriptor stands out for its robustness, particularly in structure of motion problems. These features are provided in the 'matching.txt' file for all five images, with each file containing information about the number of feature points in the respective image and their matches across images. The data includes the RGB values and coordinates of these features, as well as a feature-flag map indicating matches with other images.

Fig. 2. Images of Unity Hall

Images of Unity Hall To handle noise in the data introduced by the SIFT descriptor, the RANSAC algorithm is employed with the fundamental matrix to identify the maximum number of inliers. The normalized 8-point algorithm is used for calculating the fundamental matrix. Normalization is necessary because the epipolar lines may not precisely intersect at the center of the point correspondences. After normalization, the original fundamental matrix is retrieved. However, due to noise in the correspondences, the fundamental matrix may have a full rank of 3. To address this, the rank is reduced to 2 by setting the last diagonal element to zero, thereby obtaining the epipoles.

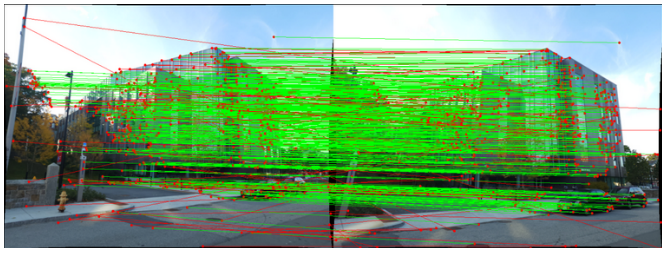
Fig. 3. Feature Matching for 2nd and 3rd image

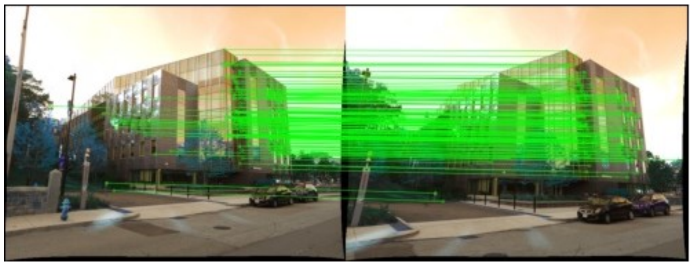
Fig. 4. Outlier Rejectiong - RANSAC (using Fundamental matrix)

We randomly sample 8 points to estimate the fundamental matrix F. Subsequently, we count the number of points satisfying the epipolar constraint $x'^T F x \approx 0$. Finally, we select the fundamental matrix that yields the largest number of inlier correspondences. The resulting fundamental matrix estimated using this method is as follows:

$$F = \begin{bmatrix} -1.475\times10^{-6} & 2.738\times10^{-5} & 3.774\times10^{-3} \\ -3.015\times10^{-5} & 6.873\times10^{-7} & 2.192\times10^{-2} \\ -4.182\times10^{-3} & -2.057\times10^{-2} & 1.000 \end{bmatrix}$$

C. Estimating Essential Matrix (E)

We estimate the essential matrix from the fundamental matrix using the equation $E = K^T F K$, where K represents the camera calibration matrix or intrinsic matrix. Similar to the computation of the fundamental matrix, the singular values of E may not necessarily be $(1, 1, 0)$ due to noise in K. To address this, we reconstruct E with singular values of $(1, 1, 0)$ using Singular Value Decomposition (SVD). The resulting essential matrix is evaluated and presented below:

$$E = \begin{bmatrix} -0.04214669 & -0.92090234 & 0.14438325 \\ 0.91820955 & -0.03036483 & -0.35526598 \\ -0.15619302 & 0.36683896 & 0.01020847 \end{bmatrix}$$

D. Estimate Camera Pose from Essential Matrix

The first camera serves as the origin of the global or world coordinate system. Utilizing the data from the second camera, four distinct configurations of the second camera relative to the first are computed in terms of translation vectors (C) and rotation matrices (R), based on the essential matrix. These configurations are determined using singular value decomposition, where $E = UDV^T$, and W is defined as:

$$W = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The four configurations are calculated as follows:

$$\begin{aligned} C1 &= U(:,3) \quad R1 = UWV^T \\ C2 &= U(:,3) \quad R2 = UWV^T \\ C3 &= U(:,3) \quad R3 = UW^TV^T \\ C4 &= U(:,3) \quad R4 = UW^TV^T \end{aligned}$$

E. Cheirality Condition using Triangulation

Our task is to calculate a unique camera pose out of 4 by removing the ambiguity. This can be done using cheirality conditions, i.e., reconstructed points should be in front of cameras and $r_3(X - C) > 0$, whereas r_3 is the third row of the rotational matrix.

After obtaining linearly triangulated 3D points, we aim to minimize the reprojection error of the location of 3D points between actual points and re-projected points. In linear triangulation, we minimize algebraic error, and in non-linear triangulation, we attempt to minimize geometric error, also called reprojection error, which is more meaningful. So, when we try to minimize the reprojection error, we refine the location of 3D points. We obtain an initial guess from linear triangulation. We use the `scipy.optimize` function and trust region field as optimization methods. The reprojection error is expressed as

$$\text{minimize} \sum_{j=1}^{2} \left((u_j - \frac{P_j^T \tilde{X}}{P_j^T X_3^T})^2 + (v_j - \frac{P_j^T \tilde{X}}{P_j^T X_3^T})^2 \right)$$

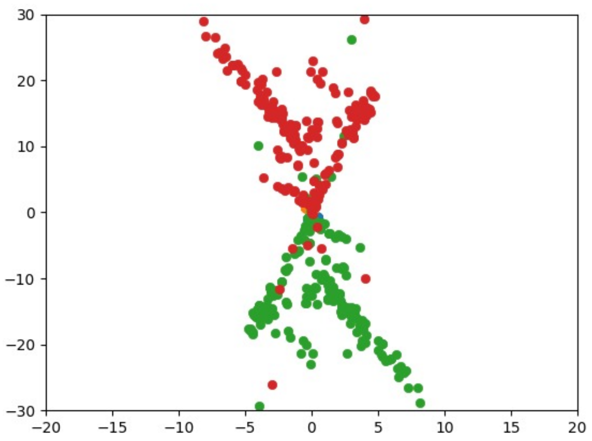

Fig. 5. Cheirality check

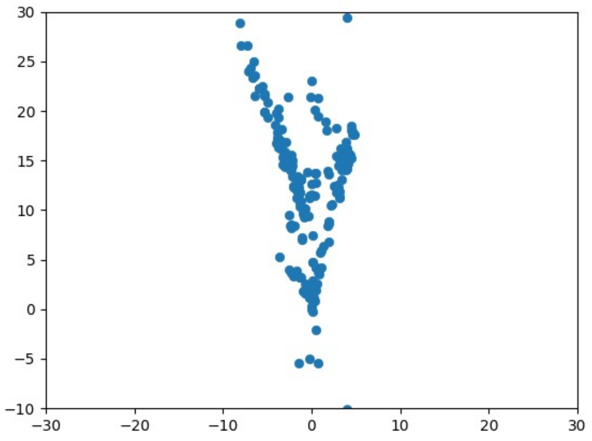

Fig. 6. Triangulation(1 & 2) using Correct Camera pose

F. Perspective-n-Points (PnP)

The Perspective-n-Points (PnP) problem involves estimating the 6 degrees of freedom camera pose based on the 2D projections of a set of 3D points in the world, along with the intrinsic camera parameters. This problem is fundamental in computer vision and is typically solved using linear least squares techniques.

1) **Linear Camera Pose Estimation (Linear PnP):** Given the correspondence between the image points (x) and the world points (X), as well as the intrinsic camera parameters (K), we normalize the image points by calculating the inverse of the intrinsic parameter matrix. Then, we solve the system of equations using the $Ax = 0$ method to obtain a linear least squares solution with Singular Value Decomposition (SVD).

2) **PnP RANSAC:** Linear PnP can be sensitive to outliers in the point correspondences. To address this issue, RANSAC (Random Sample Consensus) is employed to make the camera pose estimation more robust to outliers in the data.

3) **Non-Linear PnP:** In the Non-Linear PnP approach, we aim to further refine the camera pose estimation by minimizing the projection error. This involves optimizing the rotation matrix and translation vector using the given correspondences. The optimization process is similar to that of Non-Linear triangulation, and it utilizes methods such as `scipy.optimize.least_squares` for optimization. Additionally, the rotation matrix may be converted into a quaternion representation to maintain orthogonality during optimization.

G. Bundle Adjustment

Bundle Adjustment (BA) is a technique used to refine camera poses and 3D points in a scene simultaneously. The process involves two main steps:

1) Visibility Matrix: A visibility matrix is created based on the camera poses and the visibility of the 3D world points from each camera's perspective. Each row of the matrix represents a camera pose index, and each column represents a real-world coordinate index. For every world point index, the matrix contains a binary value (0 or 1) indicating whether the point is visible from a particular camera index.

2) Bundle Adjustment Implementation: After obtaining initial estimates of camera poses and 3D points, typically through methods like Non-Linear PnP, the next step is Bundle Adjustment. In Bundle Adjustment, the new rotation matrices and translation vectors obtained are further optimized to minimize errors. The optimization process involves refining the rotation matrices (R) and translation vectors (C) using the least squares method. The input to the bundle adjustment function includes the world coordinates (X), the new camera coordinates (x), camera parameters, rotation matrices (R), translation vectors (C), and the visibility matrix (V). The optimization is performed based on an error function, typically the re-projection error, to align the observed 2D image points with the corresponding 3D world points. The refined camera poses and 3D points are obtained as outputs of the optimization process.

Bundle Adjustment helps refine the location of 3D points and camera poses, leading to increased accuracy and consistency in the reconstructed scene. It uses the visibility matrix to establish relationships between cameras and points, enabling the refinement process to optimize the entire reconstruction pipeline effectively.

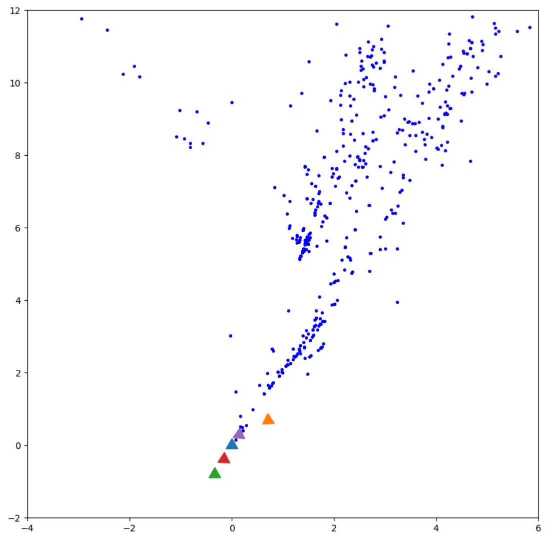

Fig. 7. Reconstructed scene after Sparse Bundle Adjustment (SBA) for images 1 to 5.

The final reconstructed scene after Sparse Bundle Adjustment (SBA) for images 1 to 5.

H. *Results Analysis*

Several observations arise from the comparison between the refinement before and after Bundle Adjustment, as well as the overall Structure of Motion pipeline:

- As emphasized in the Feature Matching section, the quality of features plays a crucial role in computer vision tasks. Poor quality data can lead to significant problems and inaccuracies, particularly in feature matching, which directly impacts the calculation of the fundamental matrix (F).
- Due to the algorithm's heavy reliance on optimization, the least squares method employed in the optimization process can be slow. Class discussions have highlighted the existence of more efficient methods for implementing nonlinear optimization, which can enhance both accuracy and speed.

The initial estimates for both intrinsic and extrinsic parameters are crucial for the optimization process to converge efficiently. The checkerboard pattern serves as a reference grid during calibration, allowing accurate correspondences between real-world points and their image pixels.

II. Phase 2 NeRF - Deep Learning Approach

A neural radiance field (NeRF) is a method based on deep learning for reconstructing a three-dimensional representation of a scene from sparse two-dimensional images. The NeRF model enables learning of novel view synthesis, scene geometry, and the reflectance properties of the scene. Additional scene properties such as camera poses may also be jointly learned.

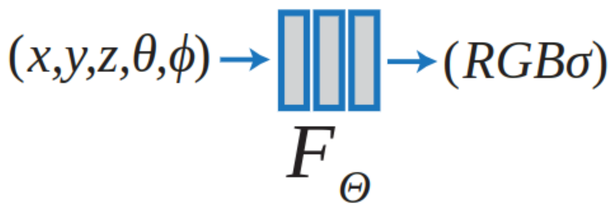

Fig. 8. Input and Output of NeRF

Traditional 3D reconstruction methods often struggle with scenes that have complex geometry or lighting conditions. NeRF addresses this by representing a scene as a continuous 5D (spatial (x,y,z) and viewing direction (θ,ψ))function that maps 3D spatial coordinates and viewing directions to radiance values (color and intensity of light) at those points. This allows for high-fidelity rendering of novel views of the scene from any viewpoint.

The key idea behind NeRF is to train a neural network to approximate this 5D function using a large set of images captured from different viewpoints. During training, the network learns to predict radiance values for any given 3D point and viewing direction by aggregating information from the input images.

A. *Getting Rays*

The Neural Radiance Fields (NeRF) method relies on tracing rays from each pixel in the input images, using a pinhole camera model. These rays' directions are calculated relative to the camera frame and then transformed into the world coordinate system using the camera poses' rotation matrices.

B. *Positional and Directional Encoding*

Passing the coordinates of points directly to the network results in poor performance, especially when only a single image is used. This is because the network tends to focus on learning low-frequency features and ignores higher frequencies. To address this, Neural Radiance Fields (NeRF) use positional encodings, which are the sines and cosines of point coordinates at various frequencies, as inputs to the network instead of the raw coordinates. Similarly, directional encodings, which are the sines and cosines of directional inputs at different frequencies, are used instead of the raw directional inputs.

C. *Network*

The NeRF model is 8 layers deep with feature dimension of 256 for most layers. A residual connection is placed at layer 5. After these layers, the RGB and values are produced. The RGB values are further processed with a linear layer, then concatenated with the view directions, then passed through yet another linear layer before finally being recombined with at the output.

However, our results are based on a tiny variation of the NeRF model called Tiny NeRF, which decreases the size of input images from 800x800 to 100x100 and the model is decreased to a 3-layer deep neural network. These step was taken to decrease processing time and computational power required. We take 3000 rays and from each ray 64 samples are taken. The whole process is iterated 1000 times with batch size of 64, learning rate 0.006.

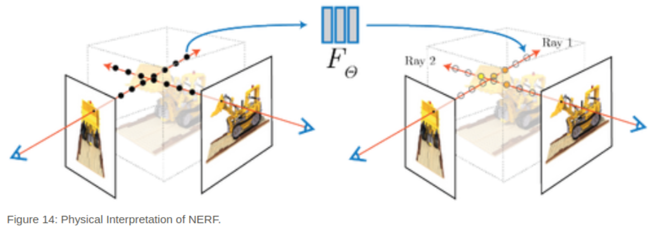

Fig. 9. Physical interpretation of NeRF

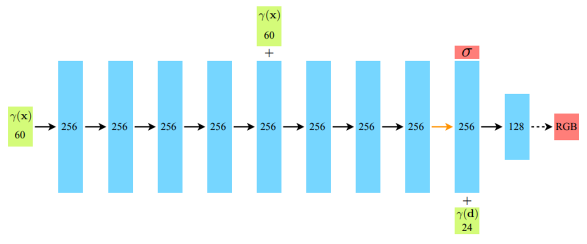

Fig. 10. Network architecture of NeRF

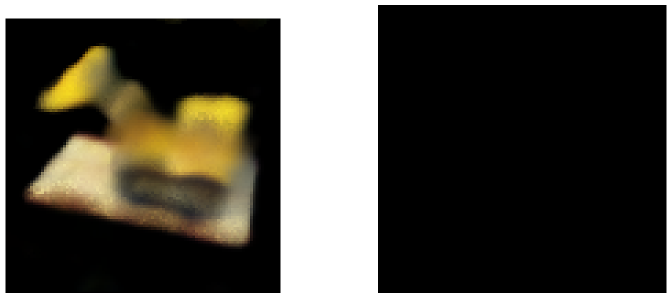

Fig. 13. With and Without Positional Encoding for Lego

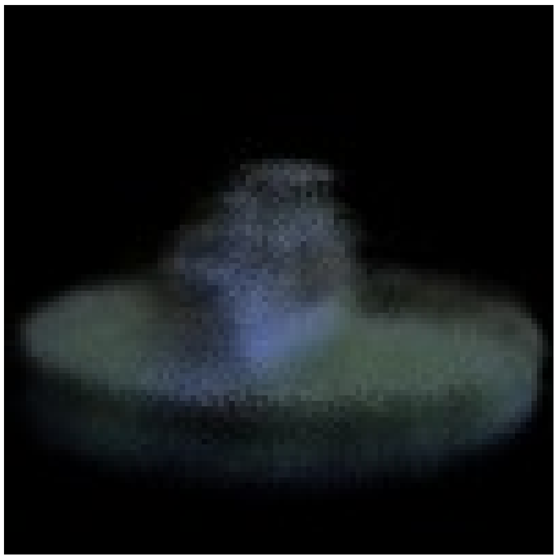

Fig. 11. Frame from rendered gif

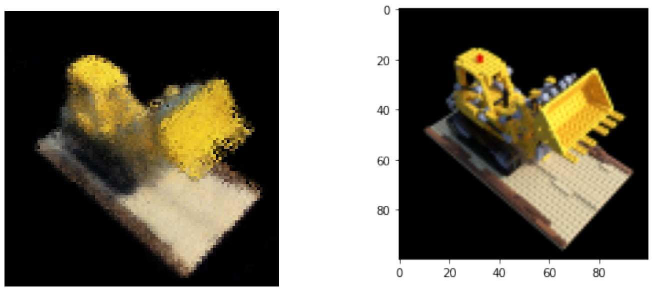

Fig. 12. Good Example for Lego

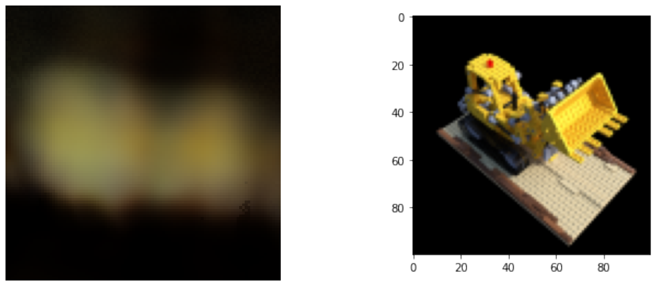

Fig. 14. Bad Example for Lego

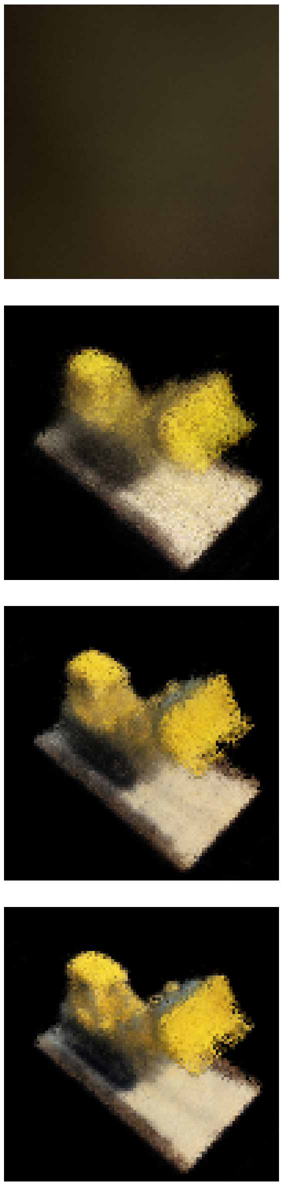
Fig. 15. Lego result when trained for (a) 100 (b) 700 (c) 2000 and (d) 4000 iterations

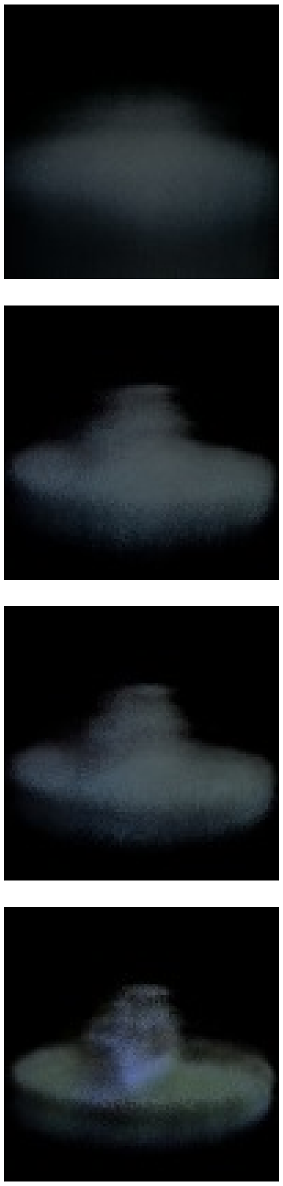
Fig. 16. Ship result when trained for (a) 200 (b) 700 (c) 1500 and (d) 4000 iterations

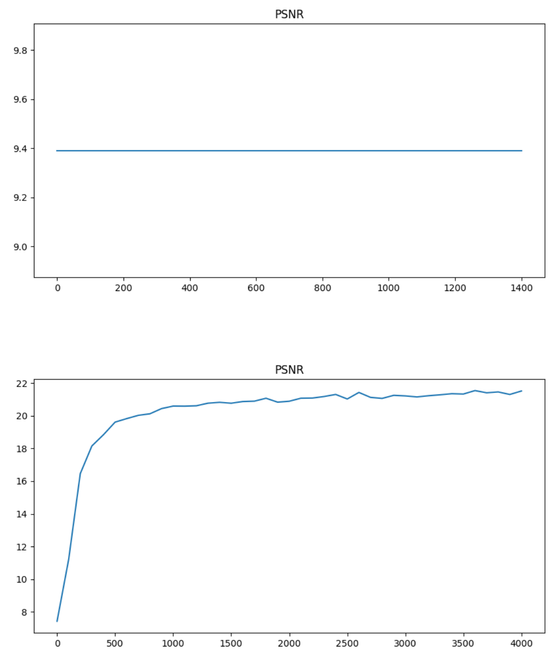

Fig. 17. PSNR for Lego Without and With Positional Encoding

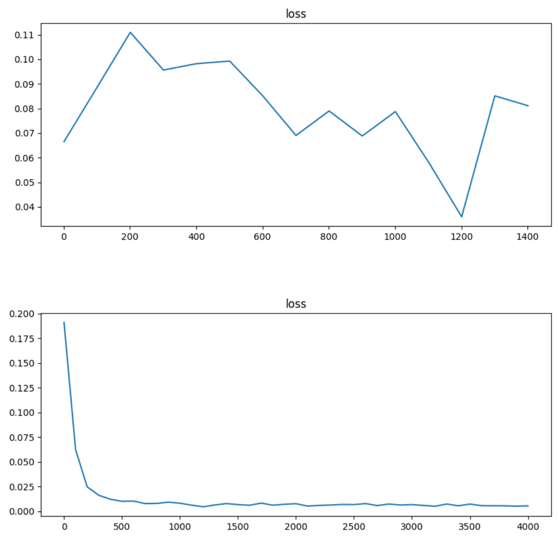

Fig. 19. MSE Loss for Lego Without and With Positional Encoding

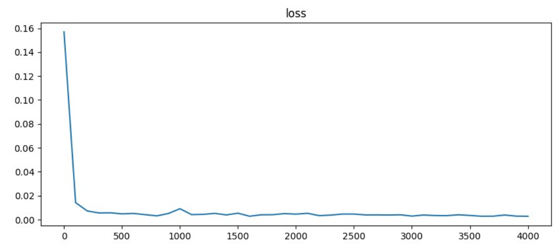

Fig. 20. MSE Loss for Ship Dataset

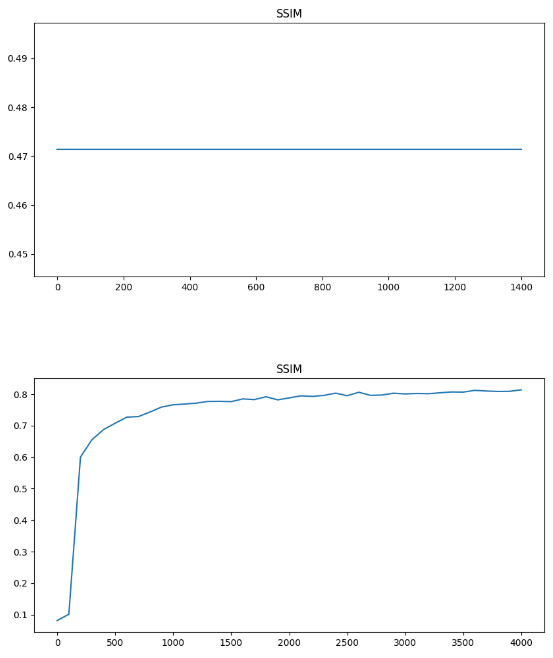

Fig. 18. SSIM for Lego Without and With Positional Encoding

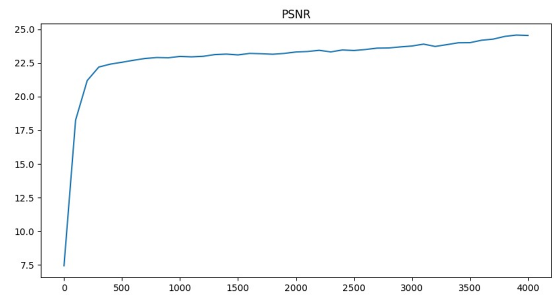

Fig. 21. PSNR for Ship Dataset

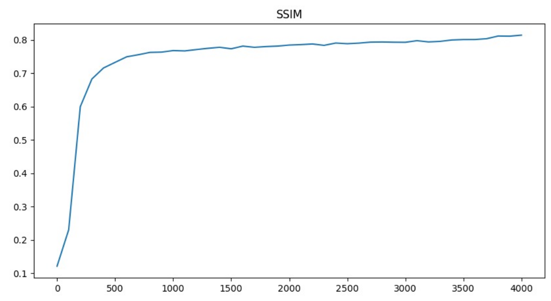

Fig. 22. SSIM for Ship Dataset

Metric	Lego	Ship
Avg PSNR	20.361	24.111
Avg SSIM	0.7562	0.5888

TABLE I
PSNR AND SSIM ERROR.

REFERENCES

[1] Bundle Adjustment:
https://scipy- cookbook.readthedocs.io/items/bundle_adjustment.html

[2] Upenn Project:
https://www.cis.upenn.edu/ cis580/Spring2015/Projects/proj2/proj2.pdf

[3] Repository Reference:
https://github.com/Prasannanatu/sfm_and_nerf

[4] Tiny Nerf Repository:
https://github.com/bmild/nerf