Computer Vision - Project 2: Buildings built in minutes - SfM and NeRF

USING 3 LATE DAYS

Jesdin Raphael
Worcester Polytechnic Institute
Worcester, MA, USA
Computer Science
Email: jraphael@wpi.edu

Harsh Verma
Worcester Polytechnic Institute
Worcester, MA, USA
Robotics Engineering
Email: hverma@wpi.edu

Muhammad Sultan
Worcester Polytechnic Institute
Worcester, MA, USA
Robotics Engineering
Email: msultan@wpi.edu

Abstract—**Project 1: 'Buildings built in minutes' focuses on creating a 3D reconstruction of the scene using multiple images. This is implemented in two ways: The first is SfM (Structure from motion), the traditional approach, and the second is NeRF, the deep learning approach. In Phase 1 of the project, we implement the SfM approach. Here we create the entire rigid structure from a set of images with different viewpoints.**

I. Phase 1: Structure From Motion with Traditional Approach

In this section, we describe the Traditional approach: SfM (Structure from Motion) to create a 3D scene from a given set of images.

The data consists of 5 images which were taken using a Samsung S22 Ultra's primary camera at f/1.8 aperture, ISO 50, and 1/500 sec shutter speed. The camera is calibrated after resizing using a Radial-Tangential model with 2 radial parameters and 1 tangential parameter using MATLAB R2022a's Camera Calibrator Application. The images provided are already distortion-corrected and resized to 800×600px. The feature-matching points were provided in files name matching$< i >$.txt where i is the image name. For Example, matching3.txt will contain matches from image 3 to image 4 and image 5.

The steps for SfM can be summarized in the following points.

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

A. Estimaating Fundamental Matrix

The F (Fundamental) matrix (rank 2) is only an algebraic representation of epipolar geometry and can both geometrically (constructing the epipolar line) and arithmetically. As a result, we obtain: $x\prime_i^T F x_i = 0$, where i=1,2,…,m. This is known as epipolar constraint or correspondence condition (or Longuet-Higgins equation). Since, F is a 3×3 matrix, we can set up a homogeneous linear system with 9 unknowns:

$$\begin{bmatrix} x'_i & y'_i & 1 \end{bmatrix} \begin{bmatrix} f_{11} & f_{21} & f_{31} \\ f_{12} & f_{22} & f_{32} \\ f_{13} & f_{23} & f_{33} \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} = 0$$

Thus we get

$$\begin{aligned} & x_i x'_i f_{11} + x_i y'_i f_{21} + x_i f_{31} + y_i x'_i f_{12} \\ & + y_i y'_i f_{22} + y_i f_{32} + x'_i f_{13} + y'_i f_{23} + f_{33} = 0 \end{aligned} \tag{1}$$

Using Eqn (1) we can construct Matrix A_i. For m correspondences, we can simply stack it depthwise.

For F matrix estimation, each point only contributes to one constraint as the epipolar constraint is a scalar equation. Thus, we require at least 8 points to solve the above homogenous system. That is why it is known as the Eight-point algorithm [1]. We can now solve using SVD for $Ax = 0$ to get the F estimate.

B. Feature Matching and Outlier rejection using RANSAC

Since there is noise in the data the matchings would have outliers. To remove these outliers and get a better F estimate we use the RANSAC algorithm. Below is the pseudo-code for the RANSAC implemented.

```
$n = 0$;
for $i = 1 : M$ do
    Choose 8 correspondences, $\hat{x_1}$ and $\hat{x_2}$ randomly.
    $F = EstimateFundamentalMatrix(\hat{x_1}, \hat{x_2})$
    $S = \emptyset$
    for $j = 1 : N$ do
        if $|x_{2j}^T F x_{1j}| < \epsilon$ then
            $S = S \cup \{j\}$
    if $n < |S|$ then
        $n = |S|$
        $S_{in} = S$
```

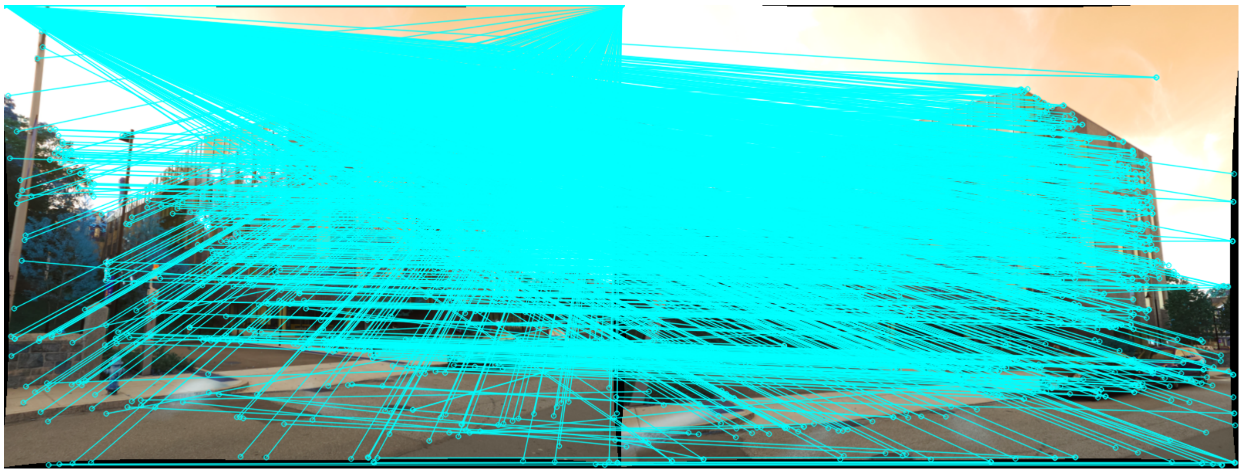

Fig. 1: Matched features before 8-point RANSAC

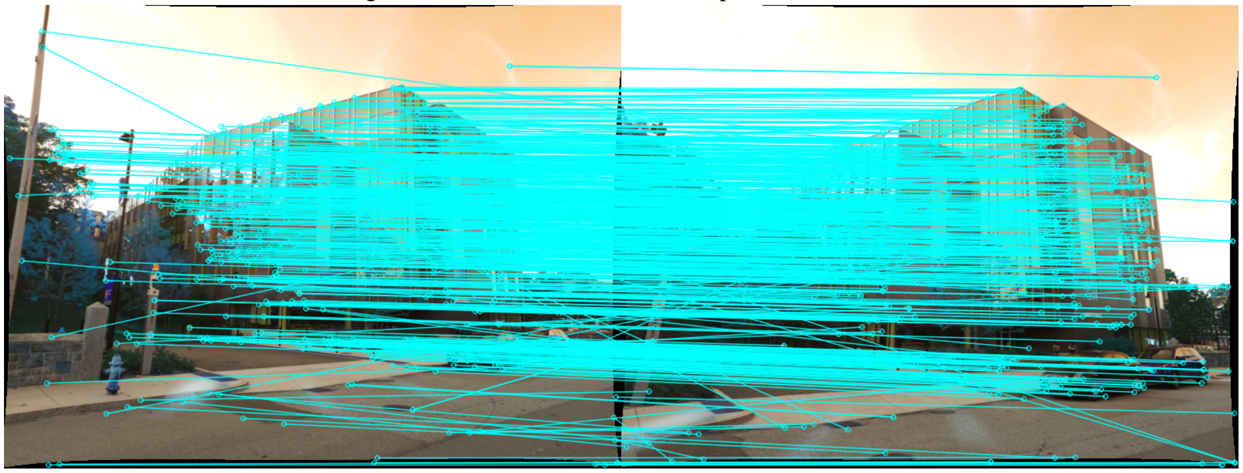

Fig. 2: Matched features after 8-point RANSAC

C. Estimating Essential Matrix from Fundamental Matrix

Now that we have calculated the Fundamental Matrix F we can find the relative pose between two images. This can be computed by the Essential Matrix which is also a 3x3 matrix. It has additional properties that relate to the corresponding points assuming that the cameras obey the pinhole mode. It can be calculated by $E = K_T F K$ where K is the Camera Calibration / Intrinsic Matrix. The singular values of E are not necessarily $(1, 1, 0)$ due to the noise in K. This can be corrected by reconstructing it with $(1, 1, 0)$ singular values, i.e.

$$E = U \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} V^T$$

D. Estimation of Camera Pose from Essential Matrix

The camera pose consists of 6 degrees-of-freedom (DOF) Rotation (Roll, Pitch, Yaw) and Translation (X, Y, Z) of the camera with respect to the world. Since we have estimated the E matrix, the four camera pose configurations: (C_1, R_1), (C_2, R_2), (C_3, R_3), and (C_4, R_4) where $C \in \mathbb{R}^3$ is the camera center and $R \in SO(3)$ is the rotation matrix, can be computed. Thus, the camera pose can be written as:

$$P = KR[I_{3\times3} - C]$$

These four pose configurations can be computed from E matrix. Let $E = UDV^T$ and $W = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$. The four configurations can be written as:

$$\begin{aligned} C_1 &= U(:,3) \text{ and } R_1 = UWV^T \\ C_2 &= -U(:,3) \text{ and } R_2 = UWV^T \\ C_3 &= U(:,3) \text{ and } R_3 = UW^TV^T \\ C_4 &= -U(:,3) \text{ and } R_3 = UW^TV^T \end{aligned}$$

E. Triangulation Check for Cheirality Condition

We have Estimated four Camera Poses $P_1 - P_4$. Using these Poses we will triangulate the 3D Points **X** between two camera poses and identify the best unique Pose.

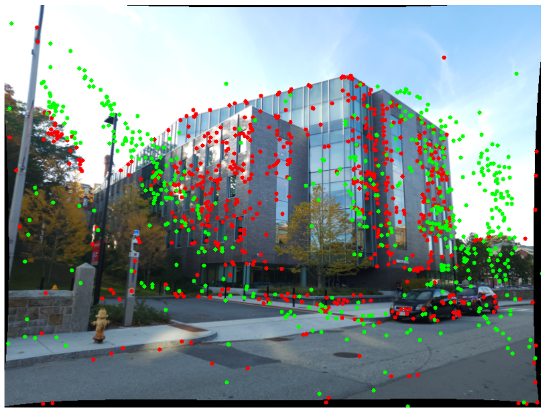

Fig. 3: Reprojection with Linear Triangulated Points

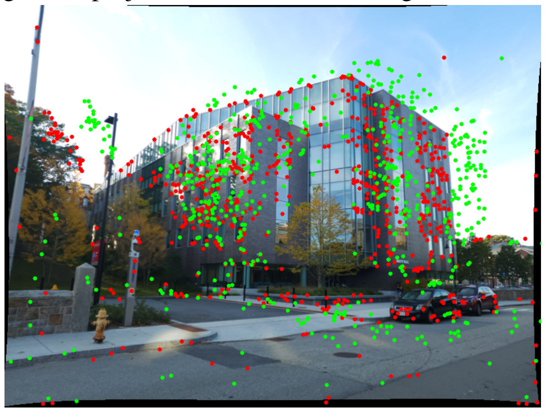

Fig. 4: Reprojection with Non Linear Triangulated Points

1) Linear Triangulation: Given Two Camera Poses and the Points We can calculate the Projected 3D points using the following Steps

```
for $i = 1 : N$ do
    $x1_i = skew_matrix(x1_i)@P1$
    $x2_i = skew_matrix(x2_i)@P2$
    $x = [x1_i, X2_i]$
    $\_, \_, V = SVD(x_P)$
    $X_{non\_homogeneous} = Vt[-1]$
    $X_i = X_{non\_homogeneous}/X_{non\_homogeneous}[-1]$
```

Here X_i is in homogeneous coordinate i.e $(x, y, z, 1)$.

2) Cheirality Check: To check the Cheirality condition, triangulate the 3D points (given two camera poses) using linear least squares to check the sign of the depth Z in the camera coordinate system with respect to the camera center. A 3D point $\mathbf{X}$ is in front of the camera if and only if:

$$\mathbf{r_3}(\mathbf{X} - \mathbf{C}) > 0$$

where $\mathbf{r_3}$ is the third row of the rotation matrix (representing the z-axis of the camera). Not all triangulated points satisfy this condition due to the presence of correspondence noise. The best camera configuration, $(\mathbf{C}, \mathbf{R}, \mathbf{X})$, is the one that produces the maximum number of points satisfying the Cheirality condition.

3) Non Linear Triangulation: After obtaining the 3D points X from the best Pose we can further refine the points by using nonlinear optimization. This can be done by minimizing the error between actual and reprojected points (Reprojection Error) as shown in Eqn (3).

$$\min_X \sum_{j=1}^{2} \left((u_j - \frac{P_1^{jT}\tilde{X}}{P_3^{jT}X})^2 + (v_j - \frac{P_2^{jT}\tilde{X}}{P_3^{jT}X} \right) \quad (3)$$

Fig 5 shows the triangulated points between the first and second Image. The red points show the triangulated points using Linear Triangulation. The Blue points show the refined triangulated points by using non-linear optimization.

F. Perspective-n-Points

Now, since we have a set of n 3D points in the world, their 2D projections in the image, and the intrinsic parameters; the 6 DOF camera pose can be estimated using linear least squares. This fundamental problem, in general, is known as Perspective-n-Point (PnP). For there to exist a solution, $n \geq 3$. There are multiple methods to solve the PnP problem, and most of them have assumptions that the camera is calibrated.

We register a new image given 2D-3D correspondences, i.e. $X \leftrightarrow x$, followed by nonlinear optimization.

1) Linear PnP: 2D points can be normalized by the intrinsic parameter to isolate camera parameters, (C, R), i.e. $K^{-1}x$. A linear least squares system that relates the 3D and 2D points can be solved for (t, R) where $t = -R^T C$. Since the linear least square solve does not enforce orthogonality of the rotation matrix, $R \in SO(3)$, the rotation matrix must be corrected by $R = UV^T$ where $R = UD^VT$. If the corrected rotation has a determinant of -1, $R = -R$. This linear PnP requires at least 6 correspondences.

We can solve the Equation $Ax = 0$ to find the Pose where A is given by Eqn (2)

2) PnP RANSNAC: PnP is prone to error as there are outliers in the given set of point correspondences. To overcome this error, we use RANSAC again to make our camera pose more robust to outliers. Below is the pseudo-code for the implemented PnP RANSAC.

$$A = \begin{bmatrix} X & Y & Z & 1 & 0 & 0 & 0 & 0 & -xX & -xY & -xZ & -x & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & X & Y & Z & 1 & 0 & 0 & 0 & 0 & -yX & -yY & -yZ & -y \end{bmatrix} \tag{2}$$

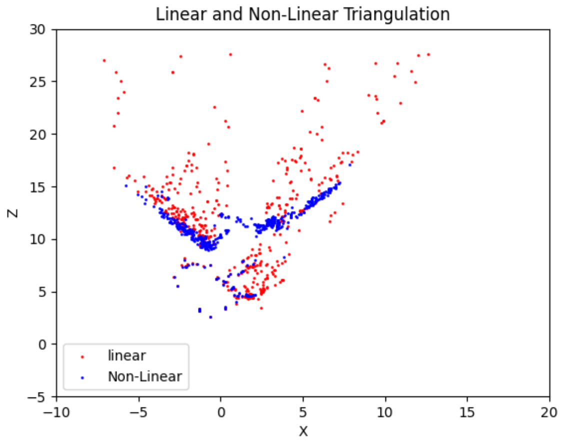

Fig. 5: Triangulated points using Linear and Non-Linear Triangulation

```
$n = 0;$
for $i = 1 : M$ do
    Choose 6 correspondences, $\hat{X}$ and $\hat{x}$ randomly.
    $[C, R] = LinearPnP(\hat{X}, x, K)$
    $S = \emptyset$
    for $j = 1 : N$ do
    $e = reprojectionerror$
    $e = \left( (u - \frac{P_1^T \tilde{X}}{P_3^T X})^2 + (v - \frac{P_2^T \tilde{X}}{P_3^T X} \right)$
        if $e < \epsilon$ then
            $S = S \cup \{j\}$
    if $n < |S|$ then
        $n = |S|$
        $S_{in} = S$
```

3) Nonlinear PnP: A compact representation of the rotation matrix using quaternion is a better choice to enforce orthogonality of the rotation matrix, $R = R(q)$, where q is a four-dimensional quaternion, i.e.,

$$\min_{C,q} \sum_{j=1}^{2} \left((u_j - \frac{P_1^{jT} \tilde{X}}{P_3^{jT} X})^2 + (v_j - \frac{P_2^{jT} \tilde{X}}{P_3^{jT} X} \right) \tag{4}$$

This minimization is highly nonlinear because of the divisions and quaternion parameterization. The initial guess of the solution, $(\mathbf{C}_0, \mathbf{R}_0)$, estimated via the linear PnP is needed to minimize the cost function.

We minimize Eqn (4) using $scipy.optimize.least_squares$ to get optimal Pose.

G. Bundle Adjustment

Once all the Camera Poses P and 3D points X we need to refine them together. This can be done by solving the optimization problem shown in Eqn (5)

$$\min_{\substack{\{C_i, q_i\}_{i=1}^{i} \\ \{X\}_{j=1}^{J}}} \sum_{i=1}^{I} \sum_{j=1}^{J} \left(\left(V_{ij} (u^j - \frac{P_1^{jT} \tilde{X}}{P_3^{jT} X})^2 + (v^j - \frac{P_2^{jT} \tilde{X}}{P_3^{jT} X} \right) \right) \tag{5}$$

Where V_{ij} is the Visibility Matrix

Computing the Jacobian of the above minimization function is cumbersome, thus we will rely on the finite difference approximation. To make this process time feasible we provide Jacobian sparsity structure (i.e. mark elements that are known to be non-zero) using $scipy.sparse$ [2].

Implementing the Bundle Adjustment helps reduce the residuals. Fig 6 and Fig 7 show the residuals of the image before and after Bundle Adjustment. The Orange curve shows that minimizing (5) helps improve the residuals.

H. Results

From Table I, we see that the magnitude of errors is huge. However, the there is a significant reduction in errors (by powers of 10) after each optimization step, such as non-linear triangulation, non-linear PnP, and bundle adjustment, showing the effectiveness of the optimizations. The results can be further improved by implementing RANSAC for Homography matching at the start to reduce the outliers.

TABLE I: Reporjection Error after Every Step

Ig Id	Linear PnP	Non-Linear PnP	Linear Triangulation			
			1	2	3	4
2	-	-	5.1×10^6	-	-	-
3	5.9×10^3	3.6×10^3	9.7×10^5	7.0×10^5	-	-
4	6.0×10^3	4.4×10^3	$5.006\,892\,478 \times 10^6$	1.5×10^6	5.5×10^6	-
5	2.9×10^4	9.1×10^3	3.5×10^7	5.3×10^6	1.8×10^5	1.7×10^5

	Non linear Triangulation				Bundle Adjustment				
Image Ids	1	2	3	4	1	2	3	4	5
2	9235.604	-	-	-	-	-	-	-	-
3	5521.278	3864.327	-	-	3,547,929.11	3,722.60	2910.147	-	-
4	5573.156	3217.052	1058.0952	-	3119151.85	1150194.954	1377.5225	1438.2166	-
5	7034.604	6056.601	2324.468	2242.181	2333744.667	2108322.257	12699.373	7949.147	5858.621

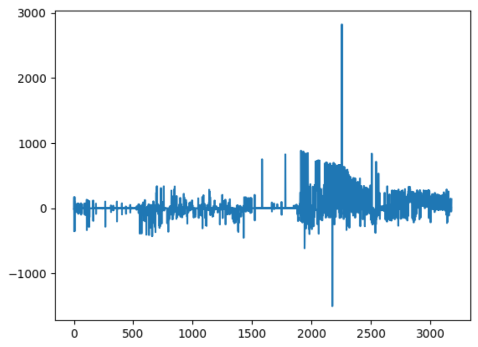
Fig. 6: Initial Residual

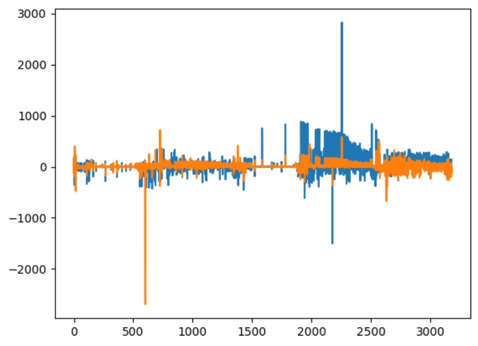
Fig. 7: Residuals after Bundle Adjustment

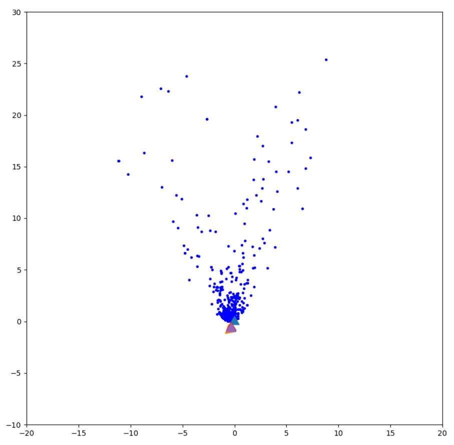
Fig. 8: Final Feature World Coordinates in 2D

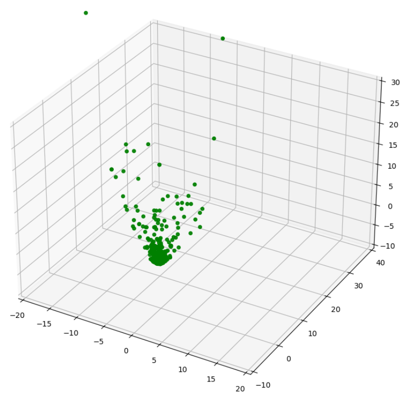
Fig. 9: Final Feature World Coordinates in 3D

II. Phase 2 - NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis

In Phase 2 we use the Deep learning approach to create a 3D scene given a set of images. Here we implement the paper 'NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis' by Mildenhall Et al. for scene construction.

A. Input

We first load the images and Camera Poses of each image. We calculate the camera's Focal Length by the equation $F = 0.5 * image\_width / tan(0.5 * cam\_angle\_x)$. The images are of the same object but from different camera angles and position.

We start off by generating rays (as ray directions and ray origins) through each pixel. We then sample points on those rays. We have set the clipping threshold for the depth values to be 2 for the Near Threshold and 6 for the far threshold. These are used to set the min-max values of the ray (z-axis) between which we will sample points for querying the model, which then produces an output. This is then converted to RGB and α values using the render function.

The reference paper uses Positional Encoding to improve the resolution of the resulting image (model higher frequencies better). Hence, we use 10 frequencies for the positional encoding (L=10) to encode the ray samples. Eqn. (6) shows the equation for the positional encoding function [3].

$$\gamma(p) = \sin(2^0\pi p), \cos(2^0\pi p), \ldots, \sin(2^{L-1}\pi p), \cos(2^{L-1}\pi p) \tag{6}$$

B. Model Architecture

To save processing time and to work with the available hardware capabilities, we implemented a smaller version of NeRF called 'TinyNeRF' (for the same purpose we resized all images to 100 x 100, otherwise our GPU was running out of memory). However, we did implement Hierarchical Volume Sampling in the TinyNeRF, to get slightly better results. Hierarchical Sampling basically samples more around some of the previously sampled points which are more likely to contribute to the view (are not empty space or occluded by the object itself).

We have two similar models, one handles coarse samples, that are sampled initially, and the other model handles fine samples (obtained using hierarchical sampling). The loss of both models is calculated and added together to be used for backward propagation.

Fig. 10 shows the coarse and fine models' architecture (same model). The input to the model are the positional encoded samples along the rays (ray directions not used since this is TinyNeRF). For training we have set $learning\_rate = 5e-3$, $number\_iterations = 4100$. We used the Adam optimizer to facilitate the training process. Moreover, we used Mean Squared Error as the loss function of choice.

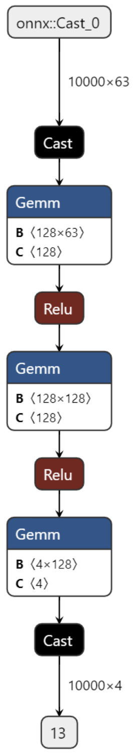

Fig. 10: Visualization of the Fine / Coarse model architecture

C. Results

From the results, we can see that the model performed comparatively well. The images are pixelated and have some pixels missing in a few novel views (black pixels). The PSNR scores per epoch while training for both the datasets (Fig. 11, Fig. 12, Fig. 13) reached a max value at roughly 4100 epochs after which the PSNR stayed roughly the same, so we clipped the epochs at 4100 and resulted in a PSNR score of approximately 18 for the lego set and approximately 20 for the ship dataset on one holdout image that we isolated from the training images.

Comparing the results of using positional encoding and not using it for the lego dataset (Fig. 15), we can see the images are much sharper with positional encoding (high-frequency features are easily captured) than without the use of positional encoding.

The average PSNR and SSIM scores for testing are shown in Table II. Both the PSNR and SSIM scores for the Lego set (with and without positional encoding) are higher than the scores for the ship dataset.

Overall, the lego dataset gave decent results, while ship dataset also performed well.

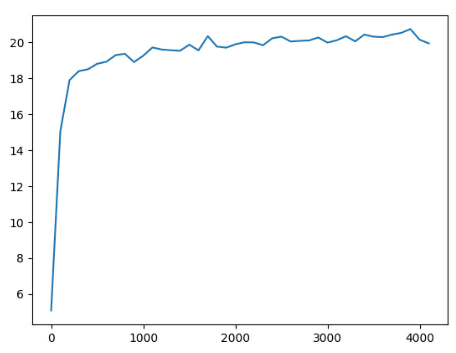

Fig. 11: PSNR per epoch of training ship

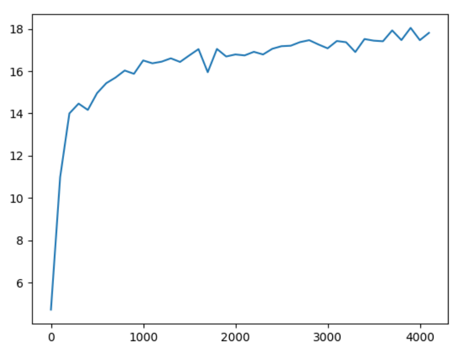

Fig. 12: PSNR per epoch of training lego

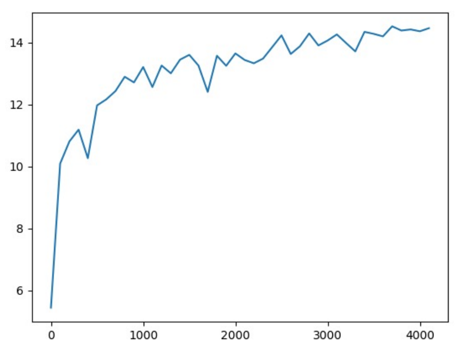

Fig. 13: PSNR per epoch of training lego without positional encoding

TABLE II: Comparison of PSNR and SSIM values for different toys on Test Data.

Toy	PSNR	SSIM
Lego (encoded)	17.326	0.794
Lego (un-encoded)	15.800	0.681
Ship (encoded)	14.129	0.585

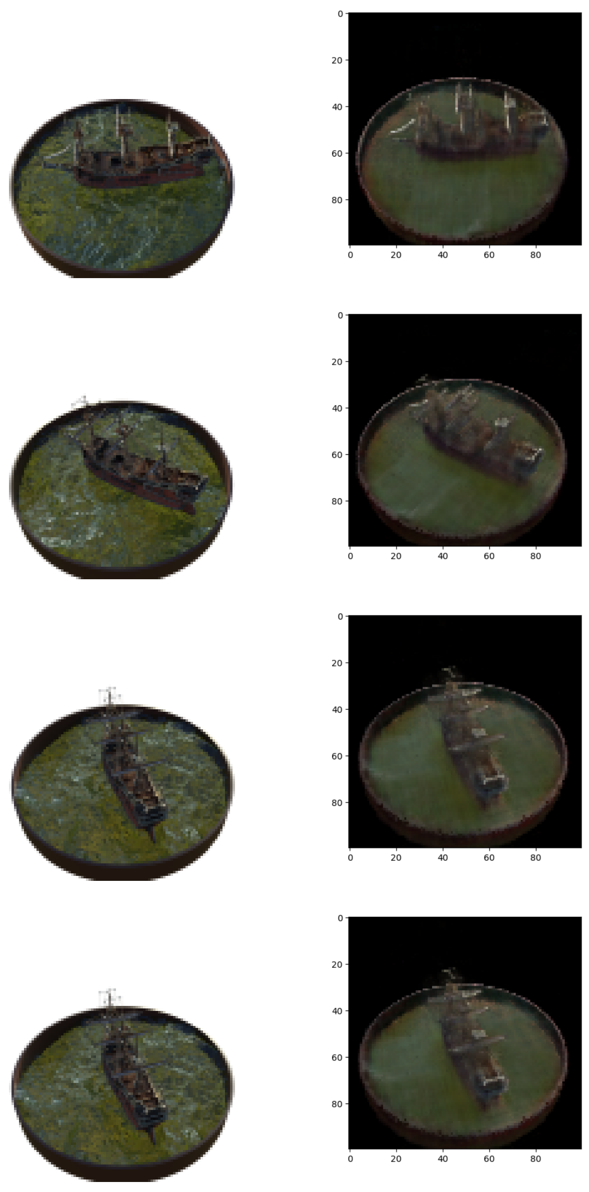

Fig. 14: Visualization of the ship images. The leftmost column displays the ground truth images, and the rightmost column shows the encoded images.

D. Problems Faced

While training the dataset, one problem we faced consistently was the GPU running out of memory. to solve this, we adopted the following steps:

1) Reduced the number of rays in a given mini batch.
2) Resized the image to a lower dimension of 100x100 pixels from 800x800.

References

[1] RBE549, “Spring 2024 project 2,” https://rbe549.github.io/spring2024/proj/p2/, 2024, [Online; accessed February 2024].

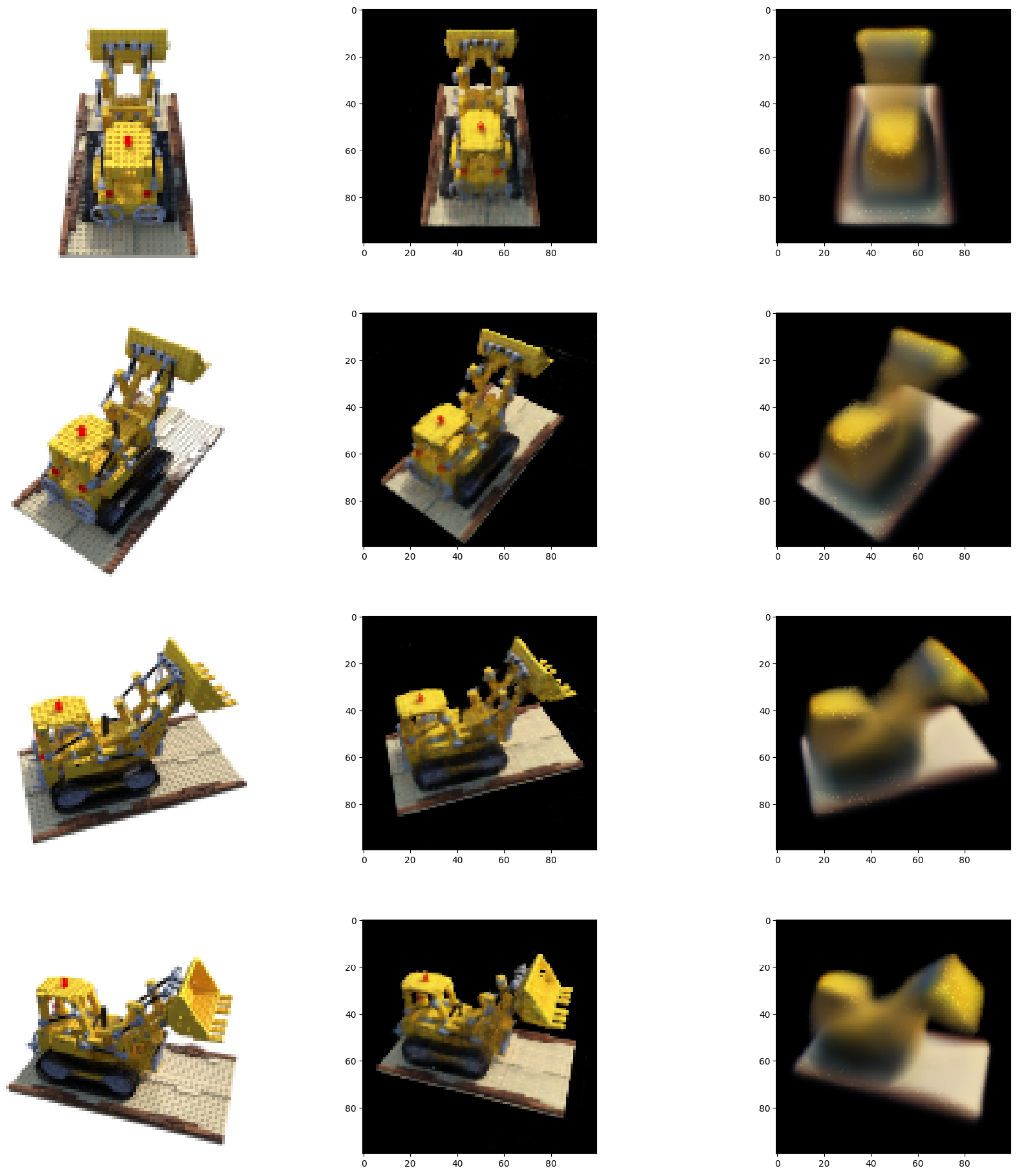

Fig. 15: Visualization of the images. The leftmost column displays the ground truth images, the middle column shows the encoded images and the rightmost column shows the decoded images.

[2] SciPy Cookbook, "Bundle adjustment," https://scipy-cookbook.readthedocs.io/items/bundle_adjustment.html, accessed on: Insert Access Date.

[3] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, "Nerf: Representing scenes as neural radiance fields for view synthesis," 2020.