RBE/CS549: Computer Vision
Project 2 - Building built in Minutes: SfM and NeRF

Shreya Bang
M.S. Robotics Engineering
Email: srbang@wpi.edu
Using 3 Late Day

Rajus Nagwekar
M.S. Robotics Engineering
Email: rmnagwekar@wpi.edu
Using 3 Late Day

***Abstract*—Phase 1 focuses on estimating a three dimensional structure from two-dimensional image sequences which are related to each other by change in camera motion, which is referred to as Structure from Motion.**

In Phase II, we have implemented Nerf: Neural Radiance Field for synthesizing novel views of complex scenes by optimizing a continuous volumetric scene given a sparse set of input views.

Index: SfM, Nerf, Volume Rendering

I. PHASE 1: TRADITIONAL APPROACH

A. Overview

The goal of the project is to understand how to create 3D structures from a dataset of 2D images using traditional methods of Structure from Motion (SfM). The project pipeline involves several steps, such as finding matching features, removing outliers using RANSAC, and estimating the Fundamental Matrix. Other steps include estimating the Essential Matrix based on Epipolar geometry, refining camera pose, performing Perspective-n-point (PnP) estimation, building a visibility matrix, and Bundle Adjustment. Each step is implemented in separate modules, and the program flow is controlled by the wrapper.py module.

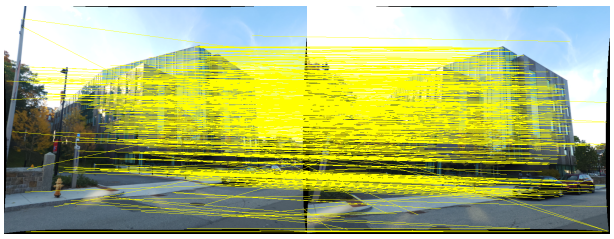

Fig. 1: Feature Matching Using SIFT

B. Fundamental Matrix

The feature point correspondences from the text files were converted to a format suitable for use in the pipeline. Using the epipolar constraint, we estimated the inlier-feature correspondences by estimating the Fundamental matrix and performing RANSAC. The resulting inliers were shown in the output of the inliers function. We used the normalized 8-point algorithm to compute the Fundamental matrix for images 1 and 2, which involves solving a linear solver matrix generated by stacking the product of 8-point correspondences using singular value decomposition. RANSAC was used to obtain a better estimate of the Fundamental matrix.

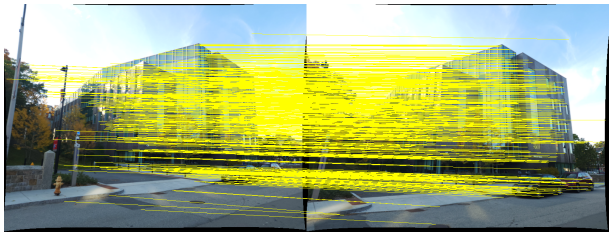

Fig. 2: Feature Matching After RANSAC

C. Essential Matrix

After obtaining the Fundamental matrix and the camera intrinsic matrix, we used them to estimate the Essential matrix E. We decomposed E to obtain four possible poses mathematically, consisting of rotation R and translation C matrices. These poses correspond to the pose of the second image, with the first image considered to be in reference alignment. In the reference alignment, R is a 3x3 identity matrix and C is a 3x1 zero matrix.

D. Linear Triangulation

Linear Triangulation involves finding the 3D coordinates of the world points from feature correspondences, camera matrix, and camera poses. One 2D-3D correspondence provides us with two equations. A single 3D point and its projections in both images form two lines in 3D space. Ideally, these lines should intersect at a point, which is the solution to the triangulation problem. However, if the lines are non-coplanar, which is usually the case, we cannot find an exact solution, so we try to find the best approximation instead. Each perspective projection gives us two equations, resulting in a total of four equations, but these equations have an unknown scale parameter unique to each projection. Therefore, we have five unknowns (X, Y, Z, s1, s2) in an under-determined system

that cannot be solved. To eliminate the scale parameters, we use the cross product of a vector with itself, which is equal to zero. We create a system of linear equations and solve it using SVD.

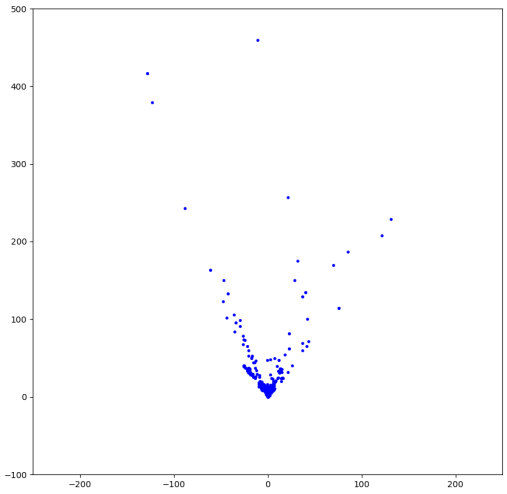

Fig. 3: Linear Triangulation

E. Non-Linear Triangulation

After obtaining 3D points from linear triangulation, we can improve their accuracy by applying non-linear optimization. The initial estimates from linear triangulation are used to minimize the reprojection error between the actual points and the reprojected points. We use the trust region field method, which is implemented in the scipy.optimize function. We found that running the optimizer for each point correspondence leads to faster convergence than running it once for all the estimated 3D points.

F. PnP

To include features from additional images in the scene, we used PnP RANSAC to eliminate outlier features and determine the estimated camera pose of the third image using the remaining inliers. However, the estimated camera poses were inaccurate due to the nonlinearity of division and reprojection in our system. To obtain more accurate estimations of camera poses, we used the estimated camera poses from linear PnP as initial guesses and performed non-linear optimization to minimize the geometric reprojection error. The reprojected features were much closer to the measured features after applying nonlinear PnP.

G. Bundle Adjustment

After registering all the images to the scene using both linear and nonlinear PnP, we proceeded to refine the camera poses and 3D points using bundle adjustment. We utilized the initial guesses for the camera poses and 3D points that we obtained from the previous steps, and utilized scipy.optimize.leastsq to minimize the reprojection error.

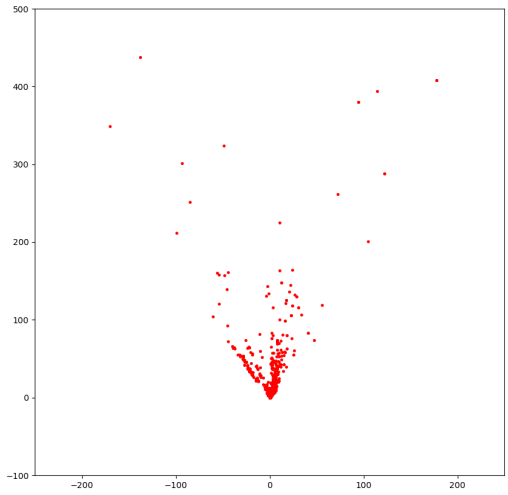

Fig. 4: Non Linear Triangulation

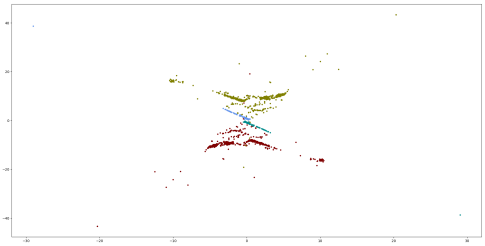

Fig. 5: Camera Pose Disambiguation

II. PHASE 2:DEEP LEARNING APPROACH

A. Overview

In Phase II, we have implemented Neural Radiance Fields (NeRF) to synthesize novel views of complex scenes by optimizing an underlying continuous volumetric scene function using a sparse set of input views. NeRF takes input as a single continuous 5D coordinate (spatial location (x, y, z) and viewing direction (θ, ϕ) and provides the output as the volume density and emitted radiance which depends on the view. Further, We have synthesized views by querying 5D

Step	**Error**
Linear Triangulation	211.37
Non-Linear Triangulation	137.71
Linear PnP	5102.10
Non-Linear PnP	4517.68
Bundle Adjustment	732.22

TABLE I: Reprojection Errors

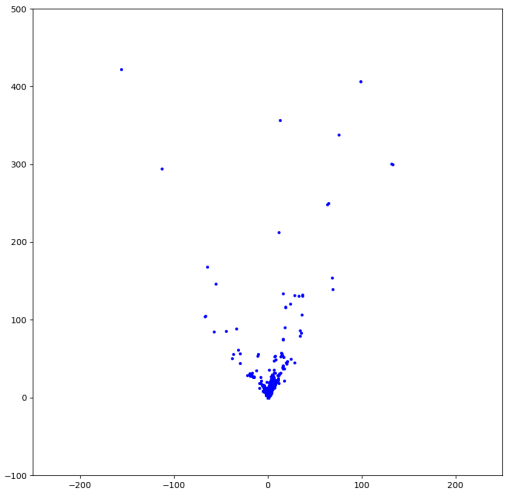

Fig. 6: Bundle Adjustment

coordinates along camera rays and using volume rendering techniques to project the output emitted color c = (r, g, b) and volume density σ into an image.

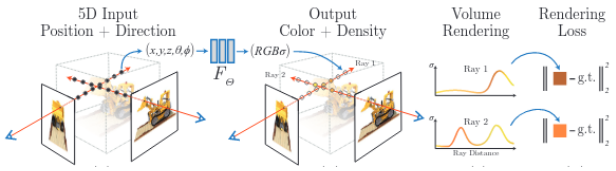

Fig. 7: An overview of Neural Radiance Field Scene Representation

B. Ray Generation and Stratification

In order to generate rays for a given camera position and orientation, NeRF uses the pinhole camera model. Rays are cast from the camera through each pixel in the image plane, and their directions are calculated based on the camera position and orientation. Then, for each ray, NeRF samples a set of points along the ray and calculates the radiance at each point. For sampling parameter, we are doing uniform sampling along the ray with some added noise which results in better results as model gets exposed to more data.Then, the radiance values are used to generate a color value for the corresponding pixel in the final image.

C. Positional Encoding

Positional encoding is used to embed the spatial information into the input to the network in a way that allows the network to get better results and render high frequency features. This is achieved by encoding the coordinates of each point as a series of sine and cosine functions of different frequencies, which are added to the input features before being fed into the network. This encoding allows the network to better capture the complex spatial patterns.

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

D. Network

The architecture used is shown in figure 08. This is a deep fully connected neural network without any convolutional layers which can be referred to as a multilayer perceptron. NeRF takes spatial location (x, y, z) and viewing direction (θ, ϕ) as inputs and generates a color (R, G, B) and volume density σ for each point. The positional encoding of the input is passed through 8 fully-connected ReLU layers, each with 256 channels. A final layer outputs the emitted RGB radiance and volume density at position x, as viewed by a ray with direction d.

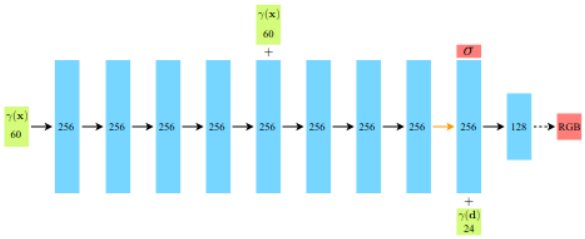

Fig. 8: Fully-connected Network Architecture

E. Volume Rendering

After processing the input through the network, the output consists of RGB color values and volume density for a given location in the 3D scene. These predictions are then used to render the scene by plugging them into the volume rendering equation. The equation takes into account the predicted color and density values to derive the final color for a specific point in the scene. After performing 3D volume rendering and obtaining the RGB color values, photometric loss was calculated by comparing the predicted color values with the actual image values.

$$\hat{C}(\mathbf{r}) = \sum_{i=1}^{N} T_i(1 - \exp(-\sigma_i\delta_i))\mathbf{c}_i\,, \text{ where } T_i = \exp\left(-\sum_{j=1}^{i-1}\sigma_j\delta_j\right)$$

F. Implementation

Following are the parameters used to train NeRF model:

1) Epochs = 3000
2) Mini Batch size = 4096
3) Optimizer = Adam
4) Learning rate = 5e-3

Due to the period required to train the model, we could train it for 3000 iterations. Also considering training time, we resized the input images as 100x100.

G. Results

The training loss is plotted in Fig.09. The rendered image after 3000 epochs has been illustrated in Fig.10. Further, we tested rendering for one image, which is illustrated in Fig.12 against the ground truth shown in Fig.11.

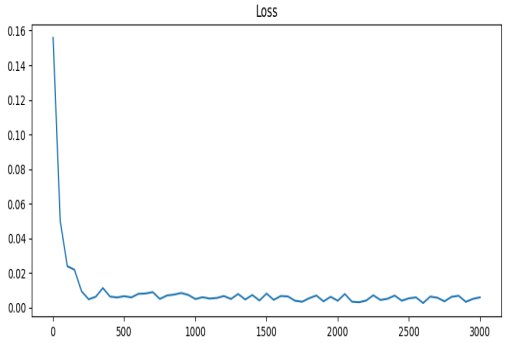

Fig. 9: Loss

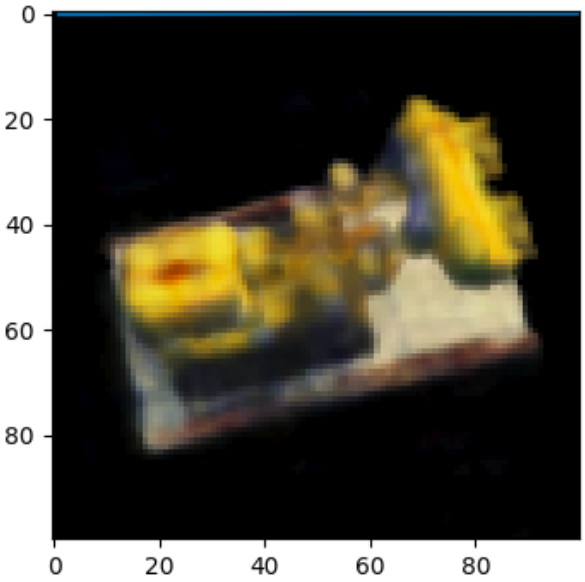

Fig. 10: Image Output after training

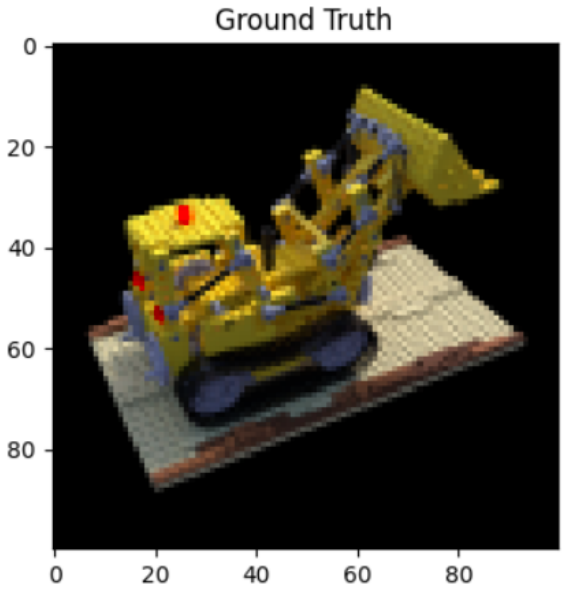

Fig. 11: Ground Truth

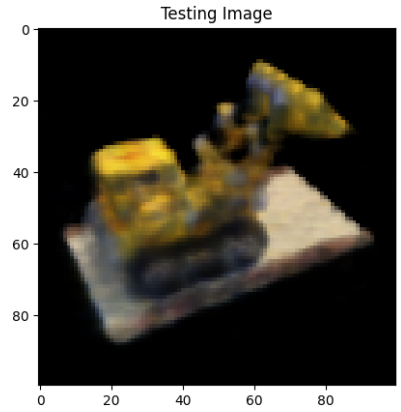

Fig. 12: Test Image Output