RBE/CS549: P2 - Buildings built in minutes - SfM and NeRF

Shrishailya Chavan
WPI Robotics Engineering
Worcester Polytechnic Institute
schavan@wpi.edu

Sreejani Chatterjee
WPI Robotics Engineering
Worcester Polytechnic Institute
schatterjee@wpi.edu

***Abstract*—In Phase 1 we implement the Structure from Motion(SfM) procedure where we reconstruct a 3D scene and obtain camera poses with respect to the given scenes. We start by rejecting outlier correspondences between pairs of images using the RANSAC algorithm. The fundamental matrix and essential matrix are then estimated for the first two images, followed by linear triangulation to estimate the 3D position of the points in the scene. The camera pose and 3D points are then refined using non-linear triangulation, and the process is repeated for the rest of the images. Camera registration is performed using PnP RANSAC, followed by non-linear PnP, and new 3D points are added to the reconstruction. A visibility matrix is built to capture the visibility of each 3D point from each camera, and bundle adjustment is performed to refine the camera poses and 3D points in the scene. The final output is a 3D reconstruction of the scene with high accuracy and completeness. In Phase 2, we have implemented Neural Radiance Fields (NeRF) by synthesizing novel views of complex scenes by optimizing an underlying continuous volumetric scene function using a sparse set of input views. The algorithm represents a scene using fully connected(non- convolutional) deep network, whose input is a single continuous 5D coordinate(spatial location (x,y,z) and viewing direction(theta, phi) and whose output is the volume density and view independent emitted radiance at spatial location. Furthermore, we also describe how to effectively optimize neural radiance fields to render photorealistic novel views of scenes with complicated geometry and appearance, and demonstrate results that outperform prior work on neural rendering view synthesis.**

***Index Terms*—SfM, NeRF, Radiance Fields, Photorealistic, 3D Recontruction**

I. Phase I: Classical Structure from Motion(SfM) Pipeline:

A. Overview:

In the 3D reconstruction of scenes using the Structure-from-Motion (SfM) pipeline, the first critical step we take is matching features between common points in the scene, followed by eliminating any outliers using the RANSAC algorithm. Once this is done, the Fundamental matrix is estimated to relate the corresponding points of two images captured from different views. The Essential matrix is then computed using this matrix, and camera poses are estimated and refined. The correct camera pose is selected using the cheirality constraints through triangulation. This process is repeated for n perspectives, and the re-projection error is computed and minimized using bundle adjustment to refine the 3D reconstruction. Refer Fig. 1 for an overview of the entire pipeline.

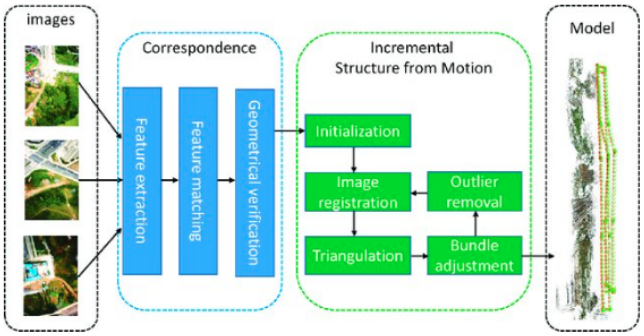

Fig. 1. SfM Pipline Architecture

Fig. 2. Images of Unity Hall to perform SfM

B. Dataset:

The dataset had been provided for the assignment and consists of a set of five images captured of Unity Hall. Fig. 2 depicts these images, which were captured using the Samsung S22 Ultra's main camera at an f/1.8 aperture, ISO 50, and a shutter speed of 1/500 seconds. The camera used for capturing these images was calibrated using the Ran-tan model, which includes two radial parameters and one tangential parameter. Therefore, the images in the dataset are distortion-corrected and have been resized to a resolution of 800x600 pixels.

C. Feature Matching, Fundamental Matrix and RANSAC:

Having good features is critical for computer vision algorithms to work accurately. The SIFT feature descriptor is commonly used because of its high robustness in solving the structure of motion problem. The dataset provided in Fig. 2 includes images of Unity Hall, along with a matching.txt file

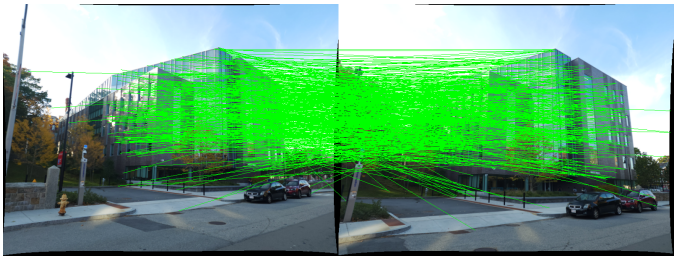

Fig. 3. Feature matching between image 1 and 2 with Selected Inlieres using RANSAC

for all five images. There are five images and four matching .txt files, and each row specifies the matches across images given a feature location in the ith image. The file format can be describes as nFeatures: (the number of feature points of the ith image - each following row specifies matches across images given a feature location in the ith image and Each Row: (the number of matches for the jth feature) (Red Value) (Green Value) (Blue Value) (ucurrent image) (vcurrent image) (image id) (uimage id image) (vimage id image) (image id) (uimage id image) (vimage id image). The row includes the number of matches for the jth feature, RGB values, and image IDs. These values need to be extracted from the .txt files and stored in a list of features in x, features in y, RGB values, and feature flag map. The feature flag map contains 0s and 1s, with 1 indicating that a point in the ith image matches with other images. As the data can become noisy after applying the SIFT feature descriptor, the RANSAC algorithm is used with the fundamental matrix to eliminate outliers. The normalized eight-point algorithm is used to calculate the fundamental matrix, which is then retrieved from the normalized points. To eliminate noise, the fundamental matrix F is made rank 2 by assigning zero to the last diagonal element, which helps in obtaining the epipoles. The process of calculating the fundamental matrix is shown in eq. 1, which is necessary because the epipolar lines do not pass through the center of point correspondences. Fig. 3 depicts the matching inlieres between image 1 and 2 after RANSAC.

$$\begin{bmatrix} x_1x_1' & x_1y_1' & x_1 & y_1x_1' & y_1y_1' & y_1 & x_1' & y_1' & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_mx_m' & x_my_m' & x_m & y_mx_m' & y_my_m' & y_m & x_m' & y_m' & 1 \end{bmatrix} \begin{bmatrix} f_{11} \\ f_{21} \\ f_{31} \\ f_{12} \\ f_{22} \\ f_{32} \\ f_{13} \\ f_{23} \\ f_{33} \end{bmatrix} = \mathbf{0} \tag{1}$$

D. Essential Matrix Estimation:

To determine the relative camera poses between two images, we use the Fundamental matrix and K matrix provided, which contains the camera's intrinsic parameters. The Fundamental matrix is used to compute the Essential matrix, which is decomposed using SVD. To enforce the diagonal elements to be 1, 1, 0, we modify the Essential matrix. This provides us with the relative camera poses between the two views, which are in image coordinates and have been normalized previously. Unlike the Fundamental matrix, which is in pixel coordinates, these relative camera poses are in normalized image coordinates.

E. Camera Pose Estimation from Essential Matrix:

The Essential matrix E is decomposed using SVD to obtain camera poses with 6 degrees of freedom: 3 rotational and 3 translational. The camera center C and camera rotation R are calculated using the formula presented in eq. 2.

$$\begin{aligned} C_1 &= \mathbf{U}(:,3) \\ R_1 &= \mathbf{U}\mathbf{W}\mathbf{V}^T \\ C_2 &= -\mathbf{U}(:,3) \\ R_2 &= \mathbf{U}\mathbf{W}\mathbf{V}^T \\ C_3 &= \mathbf{U}(:,3) \\ R_3 &= \mathbf{U}\mathbf{W}\mathbf{V}^T\mathbf{V}^T \\ C_4 &= -\mathbf{U}(:,3) \\ R_4 &= \mathbf{U}\mathbf{W}\mathbf{V}^T\mathbf{V}^T \end{aligned} \tag{2}$$

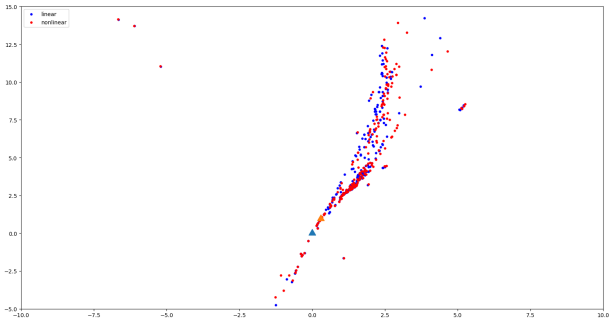

Fig. 4. Comparison between linear vs non-linear triangulation for 1st two images

F. Linear and Non-Linear Triangulation:

Linear triangulation is used to find the 3D point X in the world from two camera poses and point correspondences, and this process is repeated for all camera poses while ensuring that the reconstructed 3D points have positive Z values, also known as depth positivity constraints. Our goal is to calculate the unique camera pose out of four by removing ambiguity, which is achieved through cheirality conditions: the reconstructed points must be in front of the cameras and satisfy $r_3(X - C) < 0$, where r_3 is the third row of the rotational matrix. After linear triangulation, we aim to minimize the re-projection error between the actual and re-projected 3D point locations. Algebraic error is minimized in linear triangulation, while geometric error (i.e. re-projection error) is minimized in non-linear triangulation, which is more meaningful. We use the $scipy.optimize.least\_squares$ function with the trust region field to refine the location of 3D points based on the initial guess from linear triangulation. The re-projection error is defined as shown in eq. 3. Fig. 4 depicts the comparison between linear and non-linear triangulation between image 1 and 2 from the dataset.

$$\sum_{j=1,2} \left(u^j - \frac{P_1^{jT}\widetilde{X}}{P_3^{jT}X} \right)^2 + \left(v^j - \frac{P_2^{jT}\widetilde{X}}{P_3^{jT}X} \right)^2 \tag{3}$$

G. Linear, RANSAC, Non-Linear Perspective-n-Points(PnP):

Given a set of corresponding 2D image points ($\mathbf{x}$) and correpsponding 3D points($\mathbf{X}$), along with the intrinsic camera matrix K, we can estimate the camera pose parameters R and C through the perspective-n-point problem. To remove the influence of K on the 2D points, we normalize them using $K^{-1} * \mathbf{x}$. To obtain a linear PnP solver matrix for SVD decomposition, we need at least six 2D-3D point correspondences. The resulting 3×4 pose matrix RT consists of the rotation and translation components, where the first three columns correspond to the rotation matrix. Ideally, this matrix should be orthonormal, but practical errors may lead to deviations. To enforce orthonormality, we use SVD decomposition and multiply only the U and V matrices. The translation vector t is obtained by computing $-R^{-1}(RT_4)$, where RT_4 corresponds to the fourth column of RT. However, PnP estimation with just six points can be highly erroneous, so we use RANSAC with a reprojection error threshold of 5 to eliminate outliers. After obtaining the pose estimates from RANSAC, we refine the results using non-linear least squares optimization with the trust region method in the $scipy.optimize.least\_squares$ library. To speed up the optimization, we convert the rotation matrix R to a 1×4 quaternion vector and concatenate it with the translation matrix t (1×3) to create a 1×7 parameter vector for the optimizer.

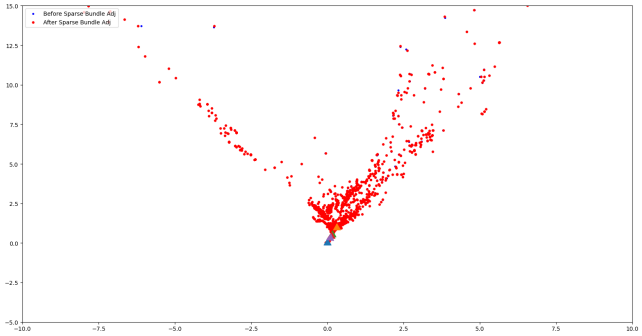

Fig. 5. 3D Reconstruction before and after Sparse Bundle Adjustment

H. Bundle Adjustment:

We observed that the non-linear optimization of the camera pose and 3D points tended to yield coarse-level refinement, as larger reprojection errors resulted in more effective optimization. To achieve more accurate 3D reconstruction, we performed bundle adjustment to further optimize all estimated camera pose parameters and 3D points together until the specific camera under registration was considered. Bundle adjustment requires creating a sparsity matrix that records whether a 2D point observation belongs to a particular parameter. For instance, we had N image points, N_{3d} world points, and n_C cameras, where each camera had six extrinsic parameters (i.e., Rotation: roll, pitch, yaw; Translation: c_x, c_y, c_z). The sparsity matrix M_{ba} has dimensions of $2N \times (N_{3d} \times 3 + n_C \times 6)$, where if the image point at index 12 in N corresponds to the world point at index 12 in N_{3d}, then the elements of matrix M_{ba} that relate them will be 1. The sparse matrix is then sent as the Jacobian sparsity parameter to the least squares optimizer in the scipy library using the 'trf' method. Fig. 5 shows the final 3D reconstruction of the pipeline before and after bundle adjustment. We can see the points(blue) obtained before bundle adjustment are sparsely distributed. On the other hand the points (red) obtained after bundle adjustment are more accurate and compactly distributed.

I. Results and Analysis:

	Linear PnP	Non-Linear PnP	Linear Triangulation					Non-Linear Triangulation					Bundle Adjustment				
ImageId			1	2	3	4	5	1	2	3	4	5	1	2	3	4	5
1	NA	NA															
2	NA	NA	16058.96					242.5					1076.90				
3	2170.98	1333.81	150.94		4973.9			76.8		485.32				86.45			
4	30380.64	442.7	58935.75	3685 32.5		1037 89.79		403.842	198.46		461.66		222.4	819 577.47	187.83		
5	23512.47	788.38	17819.87	62 31 4.81	1004 4.27		821.68	155.87	219.02	278.6		740.10	326 4.54	156.76	155.92	134.16	

Fig. 6. Reprojection Error after each step

- Obtaining good features is the most critical step for any computer vision pipeline. In this case noisy feature will end up with noisy F matrix. RANSAC is used here to curb the noise in features, but we observed RANSAC does not always generate the same result. After running for 2000 iterations we did not find any consistency in the output.
- We observed the optimization time for least square to be pretty slow. This may be improved by using other optimization algorithms.
- Fig. 6 shows the reprojection errors in each step

II. Phase II: Neural Radiance Fields

A. Input Values to Model

The Dataset that we have is of images of a lego structure as seen in Figure below. The given Dataset also provides us with camera poses of those images (i.e. camera to world transformation matrices).

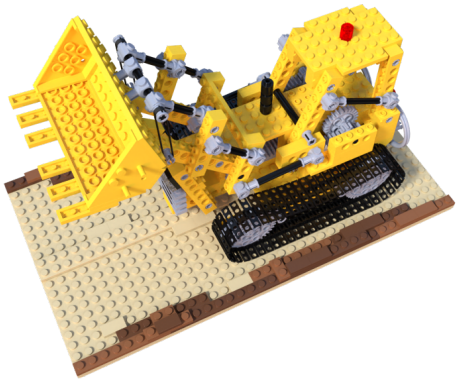

Fig. 7. Image from Lego Dataset

Now, in here we have classic volume rendering. Therefore, due to this each image pixel in the image is treated as a ray in real world and further we will sample points on that ray to get out input for the model. However, befor that we need to convert everything to world co-ordinates, this can then help us to specify the ray direction and find 3D spatial co-ordinates.

1) Ray Generation: We are supposed to generate rays from each pixel of the image as mentioned above. A typical formulation of ray looks like below equation:

$$\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$$

Fig. 8. Ray Equation

In above equation 'o' is the origin, 't' is the sampling parameter and 'd' is the direction.

In our case our origin for the ray will be the pixel position of the image. The direction is a unit vector along the vector joining the camera center and the particular pixel position. However, initially all are values are in 2d image plane and in pixel coordinates.

We need to follow below steps to get the rays:-

• Firstly, we need to convert the image pixel coordinates to normalized coordinates with respect to to camera center. The COLMAP frame is (X,-Y,-Z). Additionally, we have Z = -1 has been assumed.

• Secondly, we now have a ray direction with respect to the camera frame.When we multiply the ray direction vector by the camera to world transformation matrix (only rotation part), we will have this ray vector converted to world frame and finally, we get the ray direction by dividing by the magnitude of the vector.

• Finally, the origin of the ray is the translation part of the camera to world transformation matrix.

2) Sample Points: Now, after that we have the direction and origin of the ray and now we need to decide only sampling parameter to generate a ray. Furthermore, for sampling parameter, we will perform uniform sampling along the ray by adding some noise so that the model is exposed to new data and which will help us to get better results.

3) Positional Encoding: Further, for us to get better results and render high frequency features we will be using positional encoding. Below is given encoding function that has been used in NeRF official paper. In our function we have 6 terms for encoding. All the input values are encoded separately before we input it to the network for training.

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

Fig. 9. Positional Encoding Equation

B. Network- Multi Layer Perceptron

The data which is 5D is given to our Network as input which is a fully connected layers(Multi-Layer Perceptron) and giving us the output as volume density with RGB value at the particular sample point. The network architecture can be seen in the below picture.

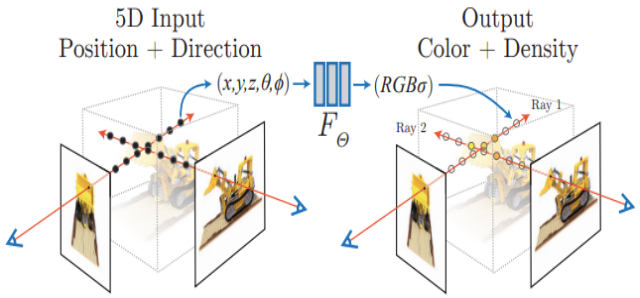

Fig. 10. Neural Network Structure

In our Network implementation we have used a small network relative to the one mentioned in official paper. The input to the network are both spatial location and direction unlike the one mentioned in paper where direction input is given later while training only for the RGB values. This was done to get better results.

C. Volume Rendering

The output we get from the Network is RGB color value and volume density at a different locations. Due to this we tend to use these predicted values for rendering of a 3D scene. Furthermore, the prediction values that we get from the network are given into the classical volume rendering equation to derive the color of one particular point. The equation for the following is mentioned below:-

Additionally using values of volume densities, we initially calculate the transmittance until we reach a particular sampling position and then multiply it with predicted color at that particular position to get the final color in the radiance field. We repeat this similar process for all the pixels in the images.

D. Loss Function

After we get the RGB values by performing 3D volume rendering we can just find Photometric loss between the Predicted Color values and Actual Image Values.

E. Network training parameters

Following are the parameters used to train Train NeRF deep learning model:

- Epochs = 100000
- Mini Batch size = 40
- Near point = 2
- Far point = 2
- Number of samples on 1 ray = 32
- Learning rate = 5e-3
- Number of terms in Encoding Function = 6
- Image input size = 100 x 100

F. Results

The Loss plot of the network with 100000 epochs has shown been below. Also, a sample rendered image can be seen after 100000 epochs. A sample gif has been attached with the results made with the predictions on the test set.

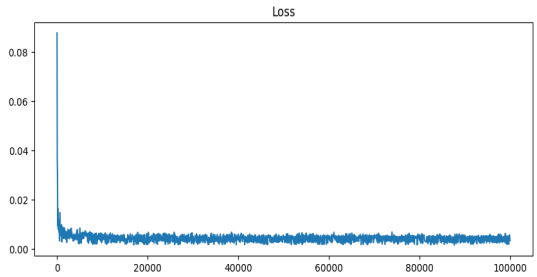

Fig. 11. Loss Plot

III. Conclusion

We recreate the SfM Pipeline in this project for Phase I where we reconstruct a 3D scene while obtaining camera poses for each scenes. We observe the importance of good features in the pipeline and the advantage of optimization. We also implement the NeRF pipeline in Phase II. If we train the model with 1000 epochs as well we get good results, as we increase the number of epochs we get much more better results. Furthermore, if hierarchical sampling is used along with a network suggested in the original paper, we can obtain much more better results. If we use full sized images (800 x800), and use more number of samples along the ray, we can get sharp results.

References

[1] https://arxiv.org/pdf/2003.08934.pdf
[2] https://github.com/facebookresearch/neuralvolumes
[3] https://rbe549.github.io/spring2023/proj/p2/