Final Project
Buildings built in minutes - Classical - SfM and Deep Learning - NeRF
ENPM673

Abubakar Siddiq
M.Eng Robotics
UID: 120403422
UMD College Park
Email: absiddiq@umd.edu

Gayatri Davuluri
M.Eng Robotics
UID: 120304866
UMD College Park
Email: gayatrid@umd.edu

***Abstract*—In this project, we reconstructed a building's 3D structure from multiple viewpoints by employing classical computer vision techniques and modern deep learning approaches. We utilized Fundamental and Essential matrix calculations, non-linear triangulation, Perspective-N-Points (PnP), and bundle adjustment to establish a framework for 3D reconstruction through the classical Structure from Motion (SfM) approach this generated a 3d point cloud of the structure.**

Following the classical reconstruction, we further enhance our project by synthesizing novel views of the complex scene using Neural Radiance Fields (NeRF). Here, we optimize an underlying continuous volumetric scene representation through a deep neural network. For our implementation, we utilize Tiny NeRF, a scaled-down version of the standard NeRF model.

I. Structure from Motion

The pipeline that we followed is similar to a project that we did in a different subject. The objective of this part is to obtain a 3D point cloud of the structure and obtain the camera poses simultaneously.

The folllowing steps collectively form the SfM pipeline:

- Feature Detection and Matching
- Outlier rejection using RANSAC
- Fundamental Matrix Estimation
- Esential Matrix Estimation
- Camera Pose extraction
- Triangulation
  - Linear Triangulation
  - Disambiguate Camera Pose
  - Non-Linear Triangulation
- PnP (Perspective-n-Point)
- Bundle Adjustment

A. Feature detection and Matching

The dataset we have utilized is of a building in-front of Levine Hall at UPenn, using a GoPro Hero 3 with fisheye lens distortion corrected. The dataset consists of 6 images and the intrinsic camera matrix (K) of that GoPro Hero 3 is also given.

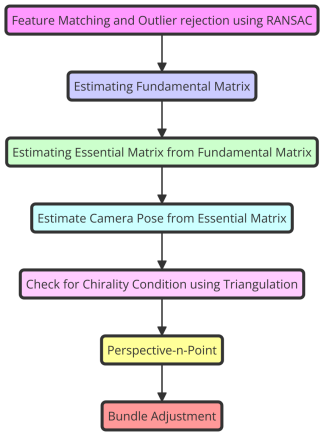

Fig. 1. SfM Pipeline

We have utilized SIFT feature detector and a FLANN matcher with KNN to detect feature points and match them with the consecutive images. One of the matching is visualized and plotted in Figure 2.

B. RANSAC

Accurately identifying and matching features across multiple images is crucial. However, these matches often include outliers that can significantly distort the final reconstruction. To address this challenge, we used the Random Sample Consensus (RANSAC) algorithm, a statistical method designed to identify inliers among a dataset contaminated with outliers.

RANSAC operates by randomly selecting a minimal set of feature matches and estimating the transformation (e.g., homography, fundamental matrix) that would align the features. This model is then tested against the entire dataset to classify the inliers that fit well with the model and outliers that do not. This process is iterated multiple times to maximize the number

of inliers and ensure the most reliable geometric interpretation. By using RANSAC, we significantly enhanced the accuracy of our feature matching process, thereby improving the stability and quality of the subsequent 3D reconstruction.

Example feature matching between some images after RANSAC is shown below.

Fig. 2. Feature matching after RANSAC between images 1 and 3

C. Fundamental Matrix Estimation

Following the feature detection and matching, we proceed to estimate the fundamental matrix. The fundamental matrix encapsulates the epipolar geometry between pairs of images taken from different viewpoints. This matrix facilitates the understanding of how points in one image relate to lines (epipolar lines) in another.

We estimated the fundamental matrix using the cv2.findFundamentalMat function from OpenCV. This function implements several algorithms to compute the fundamental matrix, but we specifically utilize the robust method of RANSAC included within it to further refine our estimates and reject outliers. The cv2.findFundamentalMat function not only provides the matrix but also helps in fine-tuning it by optimizing the fitting of epipolar lines to the matched points. This optimization reduces the reprojection error, enhancing the accuracy of the fundamental matrix.

D. Essential Matrix Estimation

Once the fundamental matrix (F) is estimated, we proceed to compute the essential matrix (E) using the relationship $K^T.F.K$, where K represents the camera's intrinsic matrix. This step transforms the fundamental matrix from pixel coordinate space to a normalized coordinate system that is intrinsic to the camera.

To refine the essential matrix, we apply Singular Value Decomposition (SVD) to E decomposing it into its constituent components U, D, and V^T. We adjusted the singular values in D to be $diag[1, 1, 0]$. This adjustment ensures that E has two non-zero singular values that are equal, and a third that is zero, which is a necessary condition for a valid essential matrix.

E. Camera Pose Estimation from the Essential Matrix

Once we have accurately estimated the essential matrix E, determining the camera pose is the subsequent critical step in our 3D reconstruction process. The camera pose includes six degrees of freedom: three for rotation (roll, pitch, yaw) and three for translation (X, Y, Z), which define the position and orientation of the camera relative to the world.

The decomposition of the essential matrix E into UDV^T via Singular Value Decomposition (SVD) allows us to explore the four possible camera pose configurations. This is achieved by incorporating matrix W, defined as:

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

and its transpose W^T.

The four possible configurations for the camera poses are as follows:

1) $C_1 = U(:, 3)$ and $R_1 = UWV^T$
2) $C_2 = -U(:, 3)$ and $R_2 = UWV^T$
3) $C_3 = U(:, 3)$ and $R_3 = UW^TV^T$
4) $C_4 = -U(:, 3)$ and $R_4 = UW^TV^T$

In these configurations, C_i represents the camera center, and R_i denotes the rotation matrix. It is crucial to ensure the correctness of the rotation matrix, R, which must belong to the special orthogonal group SO(3), implying $\det(R) = 1$. If the determinant of R is found to be -1, the configuration is corrected by negating the camera center C and the rotation matrix R.

The correct estimation of camera poses from these configurations is fundamental for accurate 3D reconstruction and critical for subsequent steps like triangulation and scene reconstruction.

F. Triangulation

Using the relationship between points in the real world we can identify the 3D world point X. Triangulation is preferred because we have the direction and distance from the camera to the world point.

1) Linear Triangulation: The point in the image should correspond to the world point projection. Using this and the camera poses we compute the world coordinate X. SVD is then performed on the equation x X P * X and the first camera is considered to be the origin to compute the triangulation. The plot of Initial triangulation with disambiguity is shown below.

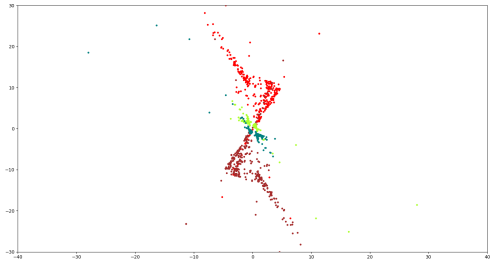

Fig. 3. Initial triangulation with disambiguity

2) Disambiguate Camera Pose: The Cheirality condition, the coordinate of the predicted world points must be in front of the camera must be met otherwise the camera poses will not be considered. Taking C as the camera origin (translation vector), X as the world coordinate and r3 as the third row of the rotation matrix R the Cheirality condition is given as $(r3(X - C) > 0)$.

3) Non-Linear Triangulation: We reduce the error between the reprojected world point and the detected image point once we find the actual pose of the camera. The impact of linear error is less on the algebraic error in the 3D space, we compute the non-linear error and converge the reprojected point and find a more accurate world point.

The reprojection error is a geometrically meaningful error and can be computed by measuring the error between the measurement and the projected 3D point:

$$\min_{X} \sum_{j=1}^{2} \left(u_j - \frac{P_j^T 1X}{P_j^T 3X} \right)^2 + \left(v_j - \frac{P_j^T 2X}{P_j^T 3X} \right)^2$$

Here, j is the index of each camera, X is the homogeneous representation of X. P_i^T represents each row of the camera projection matrix, P. This minimization is highly nonlinear due to the divisions. The initial guess of the solution, X_0, is estimated via linear triangulation to minimize the cost function. This minimization can be solved using nonlinear optimization functions.

The plot of Comparison between non-linear vs linear triangulation is shown below.

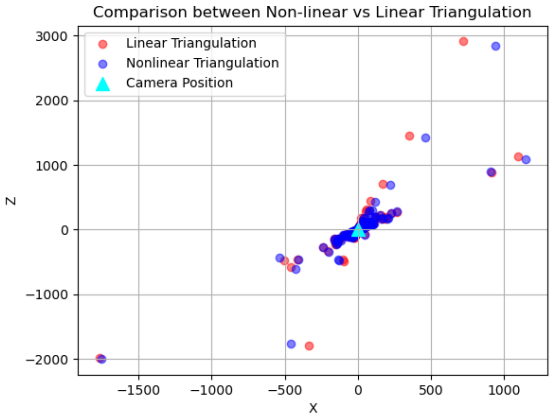

Fig. 4. Comparison between non-linear vs linear triangulation

G. PnP (Perspective-n-Point)

The Perspective-n-Point (PnP) problem is pivotal in 3D reconstruction for estimating the camera pose from known 3D points in space and their corresponding 2D projections on the image plane. This problem involves determining the rotation matrix R and translation vector t that align the 3D points to their 2D projections optimally.

Given a set of n 3D points $\{P_i\}$ and their corresponding 2D projections $\{p_i\}$, the objective is to find R and t such that the reprojection error is minimized. The reprojection error is defined as:

$$\text{error} = \sum_{i=1}^{n} \|p_i - \pi(K(RP_i + t))\|^2$$

where π denotes the projection function from 3D to 2D through the camera, and K is the camera's intrinsic matrix.

We utilize the Direct Linear Transform (DLT) method within a RANSAC framework to enhance robustness against outliers. This iterative algorithm selects subsets of point correspondences to hypothesize potential solutions, validating them against a set of inliers based on the reprojection error threshold.

Mathematically, the Rodrigues formula is employed to convert the rotation vector obtained from the solution into a rotation matrix. If $rvec$ is the rotation vector obtained, the rotation matrix R is computed as:

$$R = \exp([\omega]_\times)$$

where $\omega = rvec$ and $[\omega]_\times$ is the skew-symmetric matrix of ω. The exponential of a skew-symmetric matrix, which represents a rotation, is given by:

$$\exp([\omega]_\times) = I + \sin(\theta)[\omega]_\times + (1 - \cos(\theta))[\omega]_\times^2$$

with θ being the magnitude of ω. This Rodrigues' rotation formula transforms the compact axis-angle representation of the rotation into the corresponding rotation matrix effectively.

The estimated R and t effectively describe the camera's pose, allowing us to proceed with accurate 3D reconstruction based on the alignment of the real-world points and their imaged counterparts while minimizing the reprojection error.

H. Bundle Adjustment

1) Visibility Matrix: Considering a scene with I cameras and J points of interest, the camera-point visibility relationship can be represented by a binary matrix $V \in 0, 1$ I×J. Each element V_{ij} indicates whether the jth point is visible by the ith camera: V_{ij} =1 if the point falls within the camera's field of view, and V_{ij} =0 otherwise. This V matrix effectively acts as a map, capturing which cameras have a "line of sight" to specific points in the scene.

2) Bundle Adjustment: The purpose of bundle error is to minimize the reprojection error over $C_{ii=1}^I, q_{ii=1}^I, and X_{jj=1}^J$.

$$\min_{C_i, q_i \in \mathbb{R}^3} \sum_{i=1}^{I} \sum_{j=1}^{J} V_{ij} \left(\left(u_j - \frac{P_j^T \hat{X}_i}{P_j^T 3\hat{X}_i} \right)^2 + \left(v_j - \frac{P_j^T \hat{X}_i}{P_j^T 3\hat{X}_i} \right)^2 \right) \quad (1)$$

where V_{ij} is the visibility matrix.

Bundle Adjustment aims to optimize the alignment of world points, pixel coordinates in images, camera poses (represented by translation and quaternion rotations), and a visibility matrix. The visibility matrix streamlines computations by indicating which Jacobians are necessary for optimization, thereby enhancing efficiency.

I. Results of 3D Reconstruction

The final point cloud of the reconstructed building is shown below, The shape of the point cloud after bundle adjustment somewhat reflects the shape of the building but the point cloud is not perfect and sparse because of many reasons like initial feature mapping is sensitive to RANSAC and perspective N points (PnP) and also the dataset is small to reconstruct the whole structure.

The point cloud obtained is plotted in the below figure.

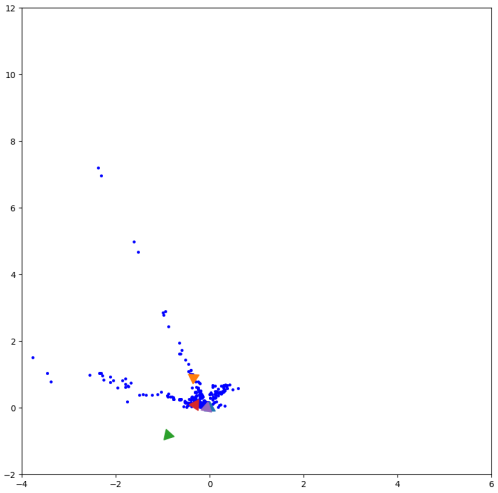

Fig. 5. Top view of the reconstructed scene

II. Neural Radiance Fields (NeRF)

Neural Radiance Fields (NeRF) represents a groundbreaking deep learning approach designed to synthesize photorealistic images from sparse and irregularly sampled views of a 3D scene. NeRF employs a continuous, differentiable function that maps 3D coordinates and viewing directions to color and density. This modeling capability allows NeRF to render complex scenes with high fidelity and detailed view-dependent effects.

A. Overview of NeRF

NeRF utilizes a fully connected deep neural network to learn a volumetric scene representation. Inputs to the network include 3D coordinates (x, y, z) and 2D viewing angles (theta, phi), which are processed to predict the RGB color and density at each point. The training data comprises a set of 2D images of the scene, along with their associated camera parameters.

This approach diverges from traditional 3D reconstruction techniques by integrating the volume density and color directly through the network, enabling the rendering of scenes with intricate details and realistic lighting effects. The model operates by simulating the way light travels through space and interacts with surfaces, which is crucial for achieving photorealism in generated views.

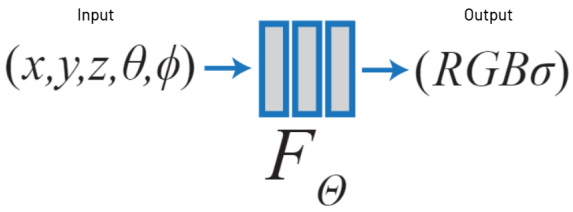

Fig. 6. NERF Model

B. Tiny NeRF

Tiny NeRF is a scaled-down version of the original Neural Radiance Fields (NeRF) designed to provide an efficient yet effective solution for 3D scene reconstruction from a sparse set of input views. Tiny NeRF makes it feasible to render high-quality images with reduced computational requirements. Below is a detailed description of the Tiny NeRF architecture:

- **Input Encoding:** Each input image, along with its camera pose and intrinsic parameters, is processed by a lightweight encoder network. This network is tasked with extracting a set of feature vectors from the images. The encoder uses a simplified architecture to ensure minimal computational load while capturing essential image characteristics.
- **Feature Aggregation:** The feature vectors extracted from all input images are combined to form a unified scene representation. This aggregation is accomplished using a max-pooling operation, which selects the maximum value for each feature from across the images, resulting in a single, compact feature vector that captures the most significant features from the entire dataset.
- **Network Body:** The aggregated feature vector serves as the input to a multi-layer perceptron (MLP). This MLP consists of several layers of fully connected neurons with ReLU activations, optimized to learn the 3D geometry and appearance attributes of the scene effectively. The design of this network ensures that it can generalize well from minimal data while still capturing complex spatial relationships.
- **Radiance Estimation:** The final output of the MLP is fed into another layer that predicts the RGB color and opacity for each 3D point in the scene. A softplus activation function is used in this layer to ensure that the estimated radiance and density values remain non-negative, contributing to the physical realism of the rendered images.

Overall, Tiny NeRF offers a compact and efficient alternative to the full NeRF model, making it particularly suitable for applications where computational resources are limited or rapid rendering is required. Despite its reduced complexity, Tiny NeRF is capable of producing high-quality 3D images, leveraging advancements in neural network design and scene understanding.

C. Dataset and Workflow

The dataset comprises a series of photographs capturing a Lego structure from various angles, accompanied by precise camera poses for each image. This comprehensive dataset enables the detailed reconstruction of the Lego model using the Tiny NeRF architecture.

Fig. 7. Lego Data set images from multiple view points

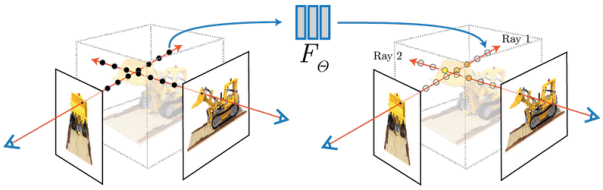

Fig. 8. NERF Workflow from original Paper

Below is an outline of the workflow and key steps involved in processing this dataset:

1) **Ray Generation:** Each pixel in an image corresponds to a ray in 3D space, originating from the camera center and passing through the pixel. The process of generating these rays involves several transformations:
   - Convert image pixel coordinates to normalized device coordinates by adjusting for the camera center and assuming $Z = -1$. This places the coordinates in a frame where X is positive rightward, Y is positive upward, and Z is negative forward, aligning with the COLMAP convention $(X, -Y, -Z)$.
   - Multiply the normalized coordinates by the camera's world transformation matrix (rotation component) to orient the ray in the world frame.
   - Normalize the resulting vector to obtain the ray direction, and use the translation component of the camera-to-world matrix as the ray origin.
2) **Sampling Along the Ray:** To simulate the continuous nature of the scene, we sample points along the ray. This is performed uniformly with some added random noise to expose the model to new data variations, which enhances the learning process.
3) **Positional Encoding:** The positional encoding step is crucial for learning high-frequency details. The encoding function used in NeRF, which includes six frequency terms, is applied as follows:

   $$\gamma(p) = (\sin(2^0\pi p), \cos(2^0\pi p), \ldots, \sin(2^5\pi p), \cos(2^5\pi p))$$

   This function significantly increases the model's capacity to represent complex spatial variations.
4) **Neural Network Processing:** The encoded position and direction data are input into the Tiny NeRF network, a series of fully connected layers that predict the RGB color and volume density at each sampled point along the ray.
5) **Volume Rendering:** The outputs (color and density) from the network are integrated to render the final image using the volume rendering equation:

   $$C(r) = \int_0^\infty T(t)\sigma(r(t))c(r(t), d)\, dt$$

   where $T(t) = \exp\left(-\int_0^t \sigma(r(s))\, ds\right)$ represents the accumulated transmittance along the ray, σ is the volume density, and c is the predicted color. This equation models the light absorption and scattering across the medium, producing the final color of each pixel.
6) **Loss Calculation:** The rendered image is compared against the actual image to compute the photometric loss, which drives the training process. This loss measures the difference between the predicted and actual pixel values, facilitating the refinement of the network's parameters for more accurate scene reproduction.
7) **Loss Function:** The loss function used in NeRF is defined as the squared L2 norm between the predicted colors and the true colors of the pixels in the input images. This is formally expressed as:

   $$\mathcal{L} = \sum_{r \in \mathcal{R}} \|C(r) - \hat{C}(r)\|^2$$

   where r represents a ray passing through a pixel, $C(r)$ is the color computed by the volume rendering integral of the scene as predicted by the model, and $\hat{C}(r)$ is the observed color from the dataset. The sum runs over all rays $\mathcal{R}$ cast through the image pixels.

This loss function effectively guides the training of the NeRF model by penalizing deviations from the observed

data, thus driving the optimization process towards generating images that are visually indistinguishable from the real ones.

This structured approach, from ray generation to photometric loss calculation, enabled Tiny NeRF to learn and render the Lego structure by leveraging the neural representation of radiance fields for complex 3D scene synthesis.

D. Results

We have trained the scaled down Tiny Nerf model for 25 epochs first and 1000 epochs next to observe the behaviour of the model with the input images of size 800×800 pixels. The overall results from both the 25 and 1000 epochs didn't had a drastic difference. Reason is, we simplified the model because of computational constraints and the network became too simple to be trained for too many epochs so the 25 epochs and 1000 epochs didn't made much of a difference in the quality of results.

The training parameters used are:

1) Epochs - 1000
2) Image size - 800×800
3) Learning Rate - 0.005
4) Number of Encoders - 6

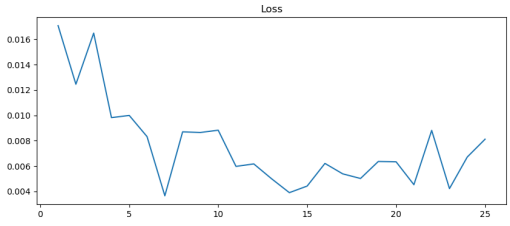

Fig. 9. NERF Workflow from original Paper

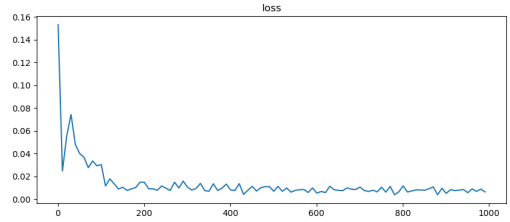

Fig. 10. NERF Workflow from original Paper

Finally, the rendered output video is obtained and the results are as follows:

III. Conclusion

In this report, the classical Structure from Motion (SfM) approach and Deep Leaning approach using Neural Radiance Fields (NeRF) are implemented, highlighting their distinct outputs and applications. SfM generates a 3D point cloud with precise coordinates for each point, ideal for applications requiring accurate spatial measurements. In contrast, NeRF

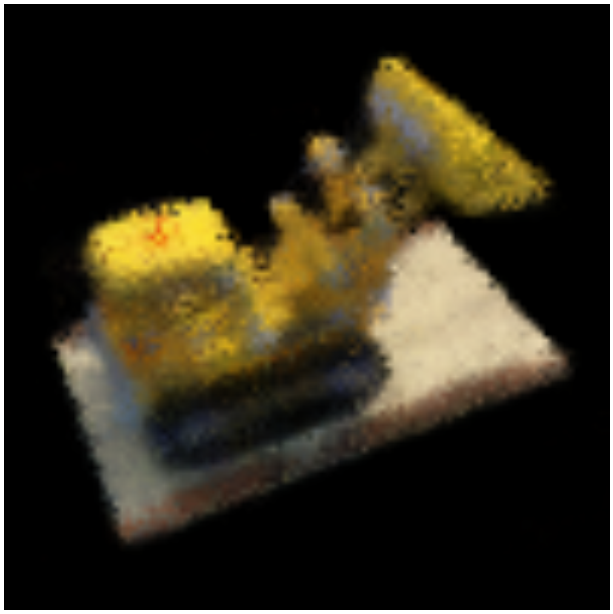

Fig. 11. One of the frames from rendered output video

synthesizes photorealistic images from a learned volumetric scene without providing explicit spatial information, favoring visual realism over geometric precision. These differences underscore the importance of selecting the appropriate method based on our priority for spatial accuracy or visual representation.

References

[1] https://cmsc733.github.io/2022/proj/p3/
[2] https://www.matthewtancik.com/nerf