RBE/CS 549 Computer Vision

P1 - SfM and NeRF

Fig. 1. DataSet

Abstract—In this project, we are reconstructing a Building's 3D structure from different views using epipolar geometry calculations, Non-linear Triangulation, Perspective N Points and Bundle Adjustment and Synthesizing novel views of complex scenes by optimizing an underlying continuous volumetric scene using NeRF. We have used tiny NeRF for our implementation.

Index Terms—SfM, NeRF

I. PHASE 1 : BUILDINGS BUILT IN MINUTES - STRUCTURE FROM MOTION

The most important phase in the 3D reconstruction of scenes, or SfM pipeline, is feature matching from common spots in the scene and removing outliers using RANSAC algorithm. The next step is to estimate the fundamental matrix, which connects the corresponding points of two images taken from different perspectives. The essential matrix is then calculated using this matrix. Linear and Non Triangulation is used to estimate camera postures and choose the best one based on cheriality restrictions. We repeat this process for n perspectives, at which point we compute the re-projection error and attempt to use bundle adjustment to reduce this non-linear re-projection error.

A. Dataset

The data given is a collection of five pictures of Unity Hall, WPI. The five photos were captured by the main camera of the Samsung S22 Ultra at f/1.8, ISO 50, and 1/500 sec.

With two radial parameters and one tangential parameter,

the Ran-Tan Model is used to calibrate this camera. The images have been distortion corrected and have been scaled to 800×600 pixels.

B. Feature Extraction

For a computer vision algorithm to function, a good feature is still essential. SIFT, a feature descriptor with strong robust- ness in structure of motion problems, is used. Fig. 2 presents

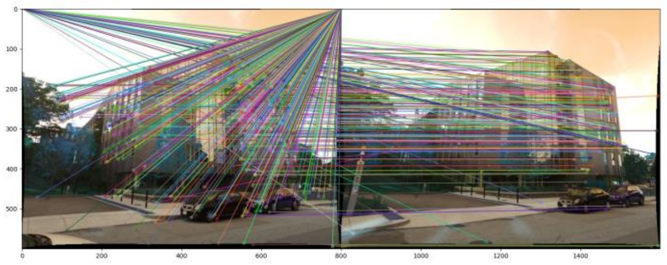

Fig. 2. Feature Matching before RANSAC

the information below: Images of the matching.txt file for all 5 pictures of Unity Hall. There are four ".txt" files and five photos total. It includes nFeatures: (the ith image's number of feature points; the next row's specification of matches between photos based on an ith image's feature location; and

In each row: (Jth feature matches as a percentage) Red, Green, and Blue values are represented by (ucurrent image), (vcurrent image), (image id), and (uimage id image) (vimage id image). From the ".txt" file, we must extract these values.

C. Fundamental Matrix

After SIFT feature descriptor, data becomes noisy, so RANSAC is employed with a basic matrix that has the most Inliers possible. To generate the fundamental matrix, we apply the normalized 8-points algorithm. We normalize it since epipolar lines don't always coincide with correspondence centers of points. With these normalized points, we compute the fundamental matrix, and then we extract the original fundamental matrix. Owing to noise in correspondances, F can have full rank, or 3, but we must reduce it to rank 2 by giving the final diagonal element a value of zero, and this is how we obtain the epipoles. Nevertheless, we must first comprehend epipolar geometry in order to comprehend what a fundamental matrix is. The intrinsic projective geometry

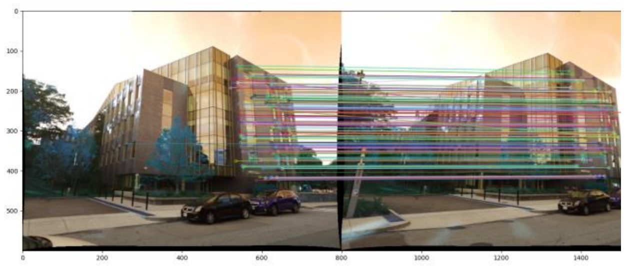

Fig. 3. Feature Matching after RANSAC

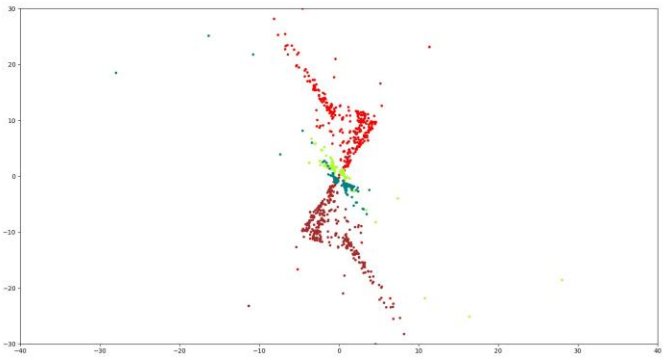

Fig. 4. Initial Triangulation with dis-ambiguity

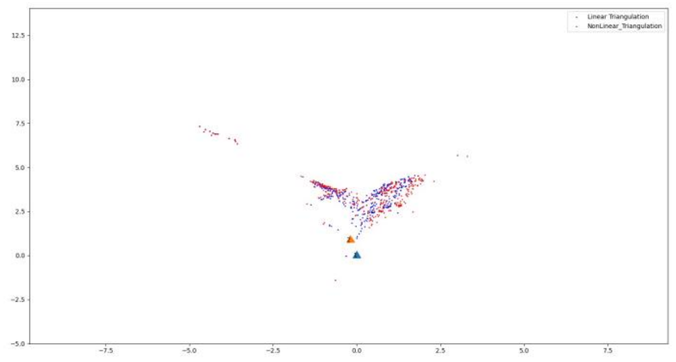

Fig. 5. Linear Triangulation vs Non Linear Triangulation -1

separating two viewpoints is known as the epipolar geometry. It is independent of the scene structure and only depends on the internal parameters (K matrix) of the cameras and the relative position.

D. Essential Matrix

Relative camera Poses needs to be found between two images and using Fundamental matrix computed above and K matrix given which has the intrinsic values of camera in it Essential matrix is computed and is Decomposed using SVD. It's diagonal elements are again enforced to 1,1,0 due to this. This gives us the relative camera poses between two views. Assuming that the cameras adhere to the pinhole paradigm, the essential matrix is another 3*3 matrix with some additional attributes that connect the appropriate spots (unlike F).

E. Camera Pose Estimation and Cheriality Condition, Triangulation

The six degrees of freedom (DOF) in the camera pose are rotation (Roll, Pitch, Yaw) and translation (X, Y, Z) of the camera with respect to the environment. The four camera pose configurations (C1,R1), (C2,R2), (C3,R3), and (C4,R4), where CR3 is the camera center and RSO(3) is the rotation matrix. Hence, the camera pose can be expressed as P=KR[I3×3C].From the E matrix, these four pose configurations can be calculated.

Linear triangulation is used to discover the X (3D-point) in the world using two camera postures and point correspondence. To identify the X (3D point in front of the camera having a positive Z value), we do this for all camera postures. Depth positive limitations refer to this.

By reducing the dis-ambiguity, our objective is to determine the unique camera posture out of 4. You can achieve this by applying the cheirality criteria, which state that the reconstructed points must lie in front of the cameras and that r3(X-C) 0. R3 is the third row of the rotational matrix.

We attempt to reduce the re-projection inaccuracy of the position of 3D points between actual points and re-projected points after obtaining linear- triangulated 3D points. In non-linear triangulation, we attempt to minimize geometric error, also known as re-projection error, which is more significant than algebraic error, which is minimized in linear triangulation. So, we adjust the placement of 3D points in an effort to minimize the re-projection error. The linear triangulation provides us with a first estimate. Scipy.optimize function is used to enhance using trust region field optimization.

F. PnP

Given a collection of n 3D points in the real world, along with their 2D image projections and intrinsic parameters, we can use linear least squares to estimate the 6 DOF camera posture. This is known as the Perspective-n-Point problem (PnP). Once we have 2D-3D correspondences (X-x), we can register a new image through nonlinear optimization.

To solve the RT (3X4) matrix using SVD, we require at least 6 2D-3D correspondences. The resulting RT matrix has rotational and translational elements, and the 3 columns of the RT matrix represent the rotational elements, which are orthonormal. To prevent errors, we enforce this orthonormality by decomposing the SVD and only multiplying U and V. We

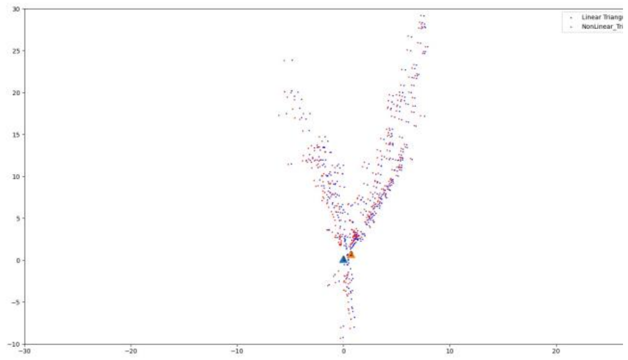

Fig. 6. Linear Triangulation vs Non Linear Triangulation -2

also check the determinant of the new R matrix, and if it is
-1, we multiply the R matrix by -1. The third column of the RT matrix represents the translation vector.

We use RANSAC to reduce outliers by minimizing the reprojection error. Once the linearly estimated camera pose is obtained, we may fine-tune the posture to further minimize the reprojection error, similar to triangulation. We refine the camera position using Non-Linear PnP optimization through Scipy.optimize. The Rotation matrix is converted into a Quaternion to maintain orthogonality and translation vector when passed for optimization.

G. Visibility Matrix and Bundle Adjustment

Bundle adjustment is a technique used in computer vision and photogrammetry to simultaneously refine the 3D point locations and camera poses that were estimated using the previous steps of the pipeline. It is essentially an optimization problem where the goal is to find the parameters that minimize the difference between the observed image points and the corresponding points projected from the estimated 3D locations and camera poses.

The first step in bundle adjustment is to construct the visibility matrix Vij that relates each camera i to the 3D point j. This matrix indicates whether a particular point is visible in a particular camera or not. Once this matrix is constructed, it is used to construct a sparse matrix Mba of size 2N x (N3d3
+ nC6), where N is the number of image points, N3d is the number of 3D points, and nC is the number of cameras. The elements of Mba are set to 1 if a particular image point is related to a particular 3D point by the visibility matrix, and 0 otherwise.

The goal of bundle adjustment is to find the parameters that minimize the difference between the observed image points and the corresponding points projected from the estimated 3D locations and camera poses. This can be done using the trust region reflective algorithm method of least squares, which is more robust to sparse problems.

Once the bundle adjustment is completed, the 3D point locations and camera poses are refined, resulting in higher

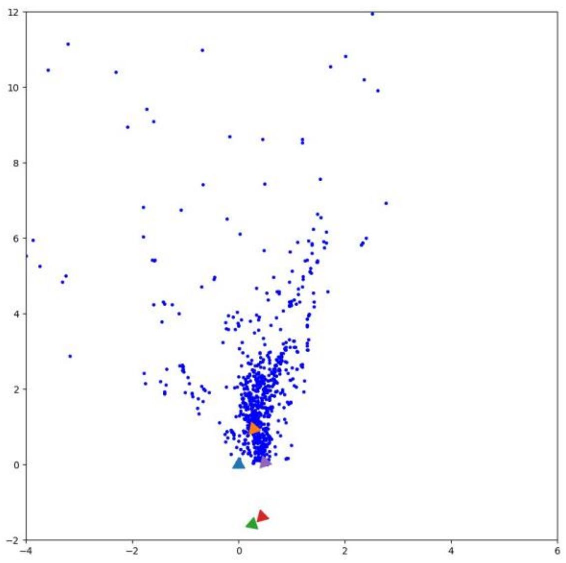

Fig. 7. Sparse Bundle Adjustment for all 5 camera using Scipy

accuracy and more optimal values. The refinement can be compared before and after bundle adjustment to evaluate the effectiveness of the technique.

H. Conclusion

The shape of the point cloud after bundle adjustment is still not perfect. There are many reasons, initial feature mapping is sensitive to RANSAC and perspective N points with more than 6 pairs might provide with better results.

II. PHASE 2 : BUILDINGS BUILT IN MINUTES - NEURAL RADIANCE FIELDS

In the Deep Learning portion, Neural Radiance Fields (NeRF) will be implemented to create unique views of com- plex scenes by refining a continuous volumetric scene function using a small number of input views. A 5D continuous array serves as the input for NeRF, with the first three elements representing the spatial location's 3D coordinates and the final two providing the direction of the ray created by connecting the specific picture pixel to the camera's center. The RGB color (radiance field) of the particular pixel and the volume density at that exact spatial location are the NeRF's outputs. We have used tiny Nerf for our implementation.

A. Tiny NeRF

TinyNerf is a lightweight variant of the Neural Radiance Fields (NeRF) architecture, designed to render high-quality 3D images from a sparse set of input views. Here's an overview of the TinyNerf architecture:

- Input Encoding: The input to the network consists of a set of 2D images, along with their camera poses and

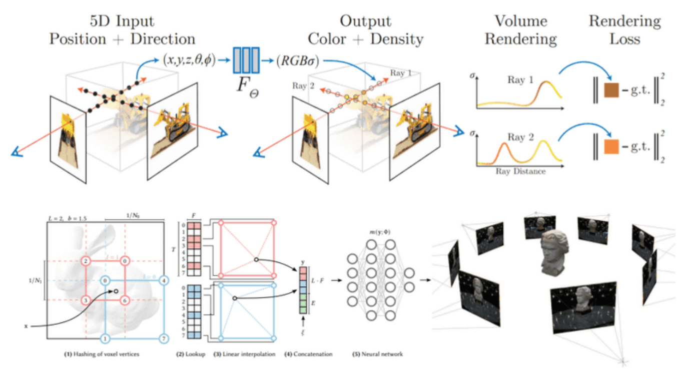

Fig. 8. NERF -workflow

intrinsics. Each input image is first passed through a small encoder network, which produces a set of feature vectors.

- Feature Aggregation: The feature vectors from all the input images are then aggregated using a simple max-pooling operation to produce a single, compact feature vector.
- Network Body: The aggregated feature vector is then passed through a multi-layer perceptron (MLP) network, which consists of several fully connected layers with ReLU activations. This network is responsible for learning the underlying 3D geometry and appearance of the scene.
- Radiance Estimation: The output of the MLP network is then passed through a final layer that produces the RGB color and opacity values at each 3D point in the scene. This is done using a softplus activation function to ensure that the radiance values are always positive.

Overall, TinyNerf is a compact and efficient architecture that can render high-quality 3D images with a small number of input views.

B. Dataset and Workflow

A dataset of pictures of a Lego structure is available. We also receive camera poses of the photographs that go with them. NeRF involved the following steps:

1) To generate rays from each pixel of the image, we start with the pixel position as the origin of the ray and a unit vector along the vector connecting the camera center and the pixel position as the direction. However, since all values are in the 2D image plane and in pixel coordinates, the following steps need to be followed to obtain the rays:

- Convert the image pixel coordinates to normalized co-ordinates with respect to the camera center, taking into account that the COLMAP frame is (X,-Y,-Z) and assuming Z = -1.
- Multiply this vector by the camera to world transforma- tion matrix (rotation part only) to convert the ray vector to the world frame.
- Obtain the ray direction unit vector by dividing the vector by its magnitude.

$$C \approx \underbrace{\sum_{i=1}^{N} T_i \alpha_i}_{\text{weights}} \overset{\text{colors}}{c_i}$$

$$T_i = \prod_{j=1}^{i-1}(1-\alpha_j)$$

$$\alpha_i = 1 - e^{-\sigma_i \delta t_i}$$

- The origin of the ray will be the translation part of the camera to world transformation matrix.

2) After obtaining the direction and origin of the ray, the only remaining task is to decide on the sampling parameter for generating the ray. We perform uniform sampling along the ray with some added noise to expose the model to new data, which can result in better outcomes.

3) To enhance the quality of the results and render high-
frequency features, we will be utilizing positional encoding. The encoding function utilized in NeRF involves six terms for encoding. Before inputting the values to the training network, all input values are encoded individually.

4) The input data is fed into our network, which comprises a series of fully connected layers. The network then provides us with output in the form of volume density and RGB value at the specific sample point.

5) The network's output is comprised solely of the RGB
color value and volume density at a specific location. These predicted values are used to render the 3D scene. To determine the color of a particular point, we insert the predictions from the network into the classical volume rendering equation. The equation takes the following form:

We begin by calculating the transmittance until the specific sampling position using the volume density. We then multiply this with the predicted color at that position to obtain the final color in the image, known as the radiance field. This process is repeated for all pixels in the image.

6) After obtaining all the RGB color values through 3D volume rendering, we can compute the photometric loss by comparing these predicted color values to the actual image values.

C. Network Training Parameters

Following are the parameters used to train Train NeRF model:

$$\mathcal{L} = \sum_{i}^{N} ||\mathcal{I}_i - \hat{\mathcal{I}}_i||_2^2$$

$$\boldsymbol{\theta} = \arg\min_{\boldsymbol{\theta}} \mathcal{L}$$

Fig. 9. Loss Function

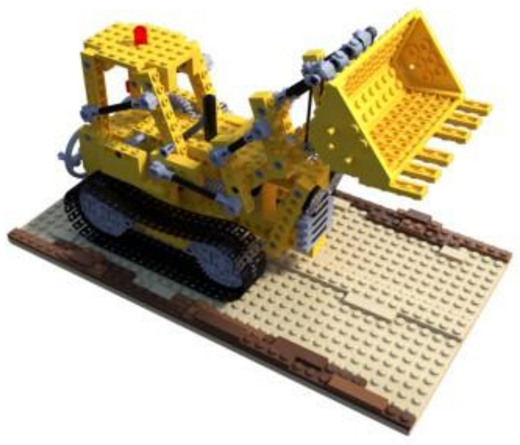

Fig. 10. Original Image

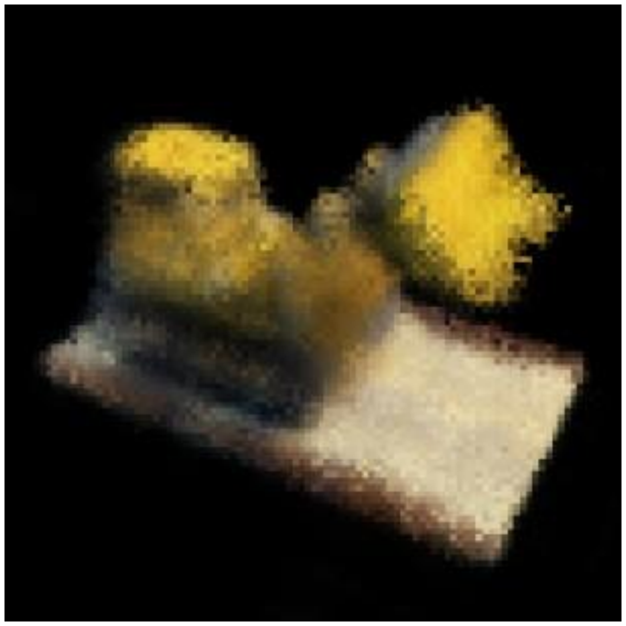

Fig. 11. Rendered Image

- Epochs = 1000
- Mini Batch size = 4096
- Near point = 2
- Far point = 6
- Number of samples on 1 ray = 64
- Learning rate = 5e-3
- Number of terms in encoding function = 6
- Image input size = 100 x 100

D. Conclusion

In conclusion, the initial results are good. The model was trained for more time and epochs the results were same on this image size. If we used full sized images (800 x800), had used more number of samples along the ray, then only we could obtain sharp results.

REFERENCES

[1] Mildenhall, Ben, et al. ”Nerf: Representing scenes as neural radiance fields for view synthesis.” Communications of the ACM 65.1 (2021):
99-106.

[2] https://github.com/yenchenlin/nerf-pytorch.

[3] https://github.com/krrish94/nerf-pytorch.

[4] https://medium.com/swlh/nerf-neural-radiance-fields-79531da37734

[5] https://theaisummer.com/nerf/

[6] Large-scale bundle adjustment in scipy, ”https://scipy-cookbook.readthedocs.io/items/bundle$_a$djustment.html”