# RBE549 Project3 SfM and NeRF

Haoying Zhou
Department of Robotics Engineering
Worcester Polytechnic Institute
Worcester, MA, 01609
Email: hzhou6@wpi.edu

Zhentian Qian
Department of Robotics Engineering
Worcester Polytechnic Institute
Worcester, MA, 01609
Email: zqian@wpi.edu

## I. Phase 1: Traditional Approach

TABLE I: Average reprojection Error in pixels. Since the structure from the motion pipeline takes one image at an image. For frame ID 2, the average reprojection is calculated for frames 1 and 2. For frame ID 3, the average reprojection is calculated for frames 1, 2 and 3, so on and so forth.

| Stage | Frame ID | | | |
|---|---|---|---|---|
| | 2 | 3 | 4 | 5 |
| Epipolar | 0.48 | 0.80 | 1.05 | 1.73 |
| Linear Triangulation | 4.30 | 0.54 | 0.71 | 1.17 |
| Non-linear Triangulation | 4.30 | 0.54 | 0.70 | 1.15 |
| Linear PnP | | 4.80 | 7.32 | 2.95 |
| Non-linear PnP | | 1.07 | 1.67 | 1.21 |
| Before BA | 4.30 | 0.30 | 0.39 | 0.28 |
| After BA | 0.22 | 0.23 | 0.26 | 0.27 |

### A. Estimating Fundamental Matrix

For the error term in Algorithm 1: Get Inliers RANSAC [1], instead of using the residual:

$$x_{2j}^T F x_{1j} \tag{1}$$

which do not take the scale of the fundamental matrix into consideration, we use the distance of the feature point to the epiline as the error metric. The distance of feature point 1 to the epiline is:

$$\frac{x_{2j}^T F x_{1j}}{\|x_{2j}^T F\|} \tag{2}$$

The distance of feature point 2 to the epiline is:

$$\frac{x_{2j}^T F x_{1j}}{\|F x_{1j}\|} \tag{3}$$

The largest of the two distances is considered the error. For the consensus set returned by the RANSAC algorithm, we would estimate the fundamental matrix using all correspondences in the consensus set for improvement. This fundamental matrix is further refined by running a non-linear optimization minimizing the error term discussed above.

The estimated fundamental matrix are visualized by its epilnes in Fig. 1. We can see that corresponding feature points lie on the computed epilines, indicating an accurate result. The average error for the first two frames is 0.48 pixels, as summarized in Table I. However, when looking at the epipoles, we can see that they both reside on the right side of the image. Their positions are contradictory since if camera 1 is on the left side of camera 2, camera 2 must be on the right side of camera 1. This behavior could be caused by the points being largely planar and the translation between camera frames with respect to feature points' depth being small, causing the fundamental matrix estimation to be inherently non-robust. Nevertheless, we do not observe any detrimental performance caused by this behavior.

We would also like to point out that the estimation of the fundamental matrix is only necessary for the first two frames. For the rest of the frames, the fundamental matrix can be directly computed so long as we know the frames' pose, which is determined by PnP. Suppose $[R_i \mid t_i]$ would transform world points into frame $i$, $[R_j \mid t_j]$ would transform world points into frame $j$. The transformation that would transform points from frame i into frame $j$ is then:

$$R_i^j = R_j R_i^T \tag{4}$$

$$t_i^j = t_j - R_i^j t_i \tag{5}$$

The essential matrix is then:

$$E = [t_i^j]_\times R_i^j \tag{6}$$

Finally, the fundamental matrix can be calculated as:

$$F = K^{-T} E K^{-1} \tag{7}$$

We can see from Table I that the fundamental matrix calculated in this fashion has a slightly larger error (for frame IDs 3-5). However, it is still good enough to establish correspondence and reject outliers. On the other hand, it is more effective than running the RANSAC algorithm for every possible image pair.

### B. Estimate Essential Matrix from Fundamental Matrix

Given the fundamental matrix, the essential matrix $E$ can be estimated as:

$$E = K^T F K \tag{8}$$

### C. Estimate Camera Pose from Essential Matrix

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

1) $C_1 = U(:, 3)$ and $R_1 = UWV^T$
2) $C_2 = -U(:, 3)$ and $R_2 = UWV^T$

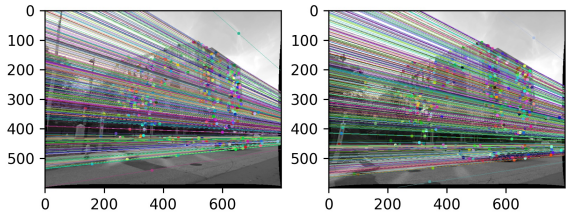



Fig. 1: Fundamental matrix represented by epilines in frame 1 and 2.)

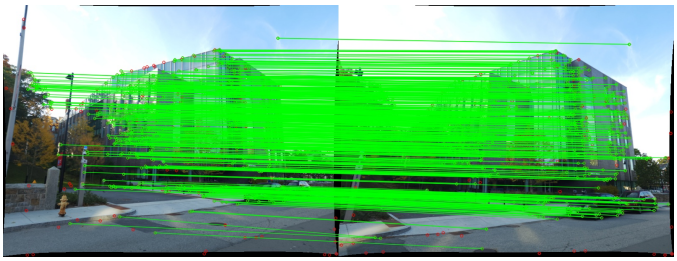


Fig. 2: Feature matching after RANSAC. (Green: Selected correspondences; Red: Rejected correspondences)

3) $C_3 = U(:,3)$ and $R_3 = UW^TV^T$
4) $C_4 = -U(:,3)$ and $R_4 = UW^TV^T$

If for any configuration, $det(R_i) = -1$, the camera pose must be corrected, i.e., $C_i = -C_i$ and $R_i = -R_i$.

### *D. Check for Cheirality Condition using Triangulation*

*1) Linear triangulation [2]:* In each image we have a measurement $\mathbf{x} = P\mathbf{X}$, $\mathbf{x}' = P'\mathbf{X}$, and these equations can be combined into a form $A\mathbf{X} = \mathbf{0}$, which is an equation linear in X.

First, the homogeneous scale factor is eliminated by a cross-product to give three equations for each image point, of which two are linearly independent. For example for the first image, $\mathbf{x} \times (P\mathbf{X}) = 0$ and writing this out gives:

$$\mathbf{x}(p^{3T}\mathbf{X}) - (p^{1T}\mathbf{X}) = 0 \tag{9}$$
$$\mathbf{y}(p^{3T}\mathbf{X}) - (p^{2T}\mathbf{X}) = 0 \tag{10}$$
$$\mathbf{x}(p^{2T}\mathbf{X}) - y(p^{1T}\mathbf{X}) = 0 \tag{11}$$

where $\mathbf{p}^{iT}$ are the rows of $P$. These equations are linear in the components of $\mathbf{X}$. An equation of the form $A\mathbf{X} = \mathbf{0}$ can then be composed, with

$$\mathbf{A} = \begin{bmatrix} \mathbf{x}p^{3T} - p^{1T} \\ \mathbf{y}p^{3T} - p^{2T} \\ \mathbf{x}'p'^{3T} - p'^{1T} \\ \mathbf{y}'p'^{3T} - p'^{2T} \end{bmatrix} \tag{12}$$

where two equations have been included from each image, giving a total of four equations in four homogeneous unknowns. This is a redundant set of equations since the solution is determined only up to scale. Similar to how we estimate the fundamental and homography matrix, the solution is the unit singular vector corresponding to the smallest singular value of A.

The initial triangulation plot showing all four possible camera poses is visualized in Fig. 3. The camera configuration produces the maximum number of points satisfying the cheirality condition and with a depth smaller than 50 is chosen as the best camera pose. In this case, solution 1 is selected.

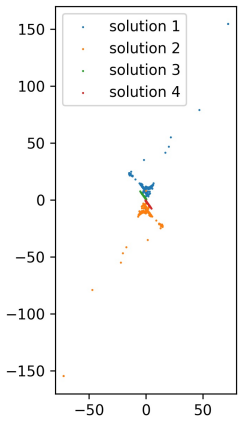



Fig. 3: Initial triangulation plot with disambiguity, showing all four possible camera poses.)

*2) Non-Linear Triangulation:* The linear and non-linear triangulated points are visualized in Fig. 4. Their re-projections are visualized in Fig. 5. The results suggest that the two results are very close, the accuracy improved by non-linear optimization is marginal. Because of accurate camera pose estimation and correct point correspondences, even with linear triangulation, the results are already satisfactory. This claim is backed by the reprojection error in Table I, the error reduced by non-linear triangulation is smaller than 0.02 pixels.

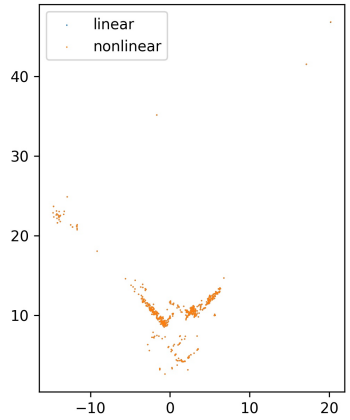
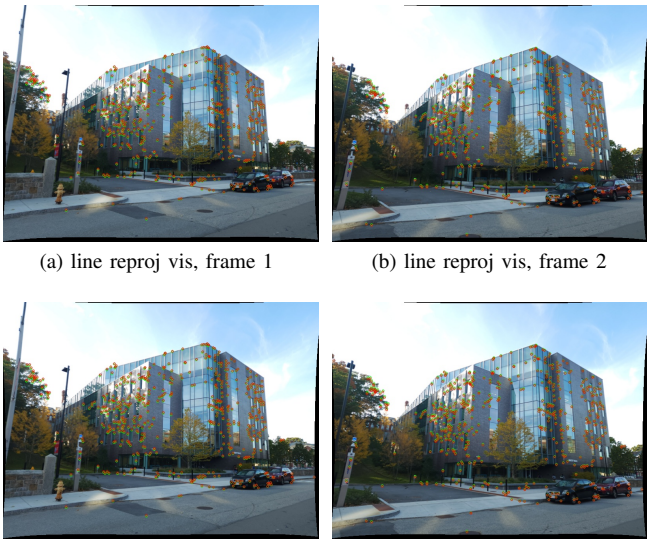



Fig. 4: Comparison between non-linear vs linear triangulation.



(a) line reproj vis, frame 1 (b) line reproj vis, frame 2

(c) nonline reproj vis, frame 1 (d) nonlin reproj vis, frame 2

Fig. 5: Comparison of projections between non-linear vs linear triangulation. First column corresponds to first image and second column corresponds to second image. Top row shows output of linear triangulation and bottom row shows output of non-linear triangulation. Green dots show the feature detections and red dots show reprojections.

### *E. Perspective-n-points*

*1) Linear Camera Pose Estimation:* We use a simple linear algorithm for determining $P$ given a set of four $2D$ to $3D$ point correspondences, $\mathbf{x}_i \leftrightarrow \mathbf{X}_i$. The transformation is given by the equation $\mathbf{x}_i = P\mathbf{X}_i$. Note that this is an equation involving homogeneous vectors; thus the 3-vectors $\mathbf{x}_i$ and $P\mathbf{X}_i$ are not equal, they have the same direction but may differ in magnitude by a nonzero scale factor. The equation may be expressed in terms of the vector cross product as:

$$\mathbf{x}_i \times P\mathbf{X}_i = 0 \tag{13}$$

. This form will enable a simple linear solution for $P$ to be derived.

If the j-th row of the matrix $P$ is denoted by $\mathbf{p}^{jT}$, then we may write:

$$P\mathbf{X}_i = \begin{bmatrix} \mathbf{p}^{1T}\mathbf{X}_i \\ \mathbf{p}^{2T}\mathbf{X}_i \\ \mathbf{p}^{3T}\mathbf{X}_i \end{bmatrix} \tag{14}$$

Writing $\mathbf{x}_i = (x_i, y_i, \omega_i)^T$, the cross product may then be given explicitly as:

$$\mathbf{x}_i \times P\mathbf{X}_i = \begin{bmatrix} y_i\mathbf{p}^{3T}\mathbf{X}_i - \omega_i\mathbf{p}^{2T}\mathbf{X}_i \\ \omega_i\mathbf{p}^{1T}\mathbf{X}_i - x_i\mathbf{p}^{3T}\mathbf{X}_i \\ x_i\mathbf{p}^{2T}\mathbf{X}_i - y_i\mathbf{p}^{1T}\mathbf{X}_i \end{bmatrix} \tag{15}$$

Since $\mathbf{p}^{jT}\mathbf{X}_i = \mathbf{X}_i^T\mathbf{p}^j$ for $j = 1, \ldots, 3$, this gives a set of three equations in the entries of $P$, which may be written in the form:

$$\begin{bmatrix} \mathbf{0}^T & -\omega_i\mathbf{X}_i^T & y_i\mathbf{X}_i^T \\ \omega_i\mathbf{X}_i^T & \mathbf{0}^T & -x_i\mathbf{X}_i^T \\ -y_i\mathbf{X}_i^T & x_i\mathbf{X}_i^T & \mathbf{0}^T \end{bmatrix} \begin{bmatrix} \mathbf{p}^1 \\ \mathbf{p}^2 \\ \mathbf{p}^3 \end{bmatrix} = \mathbf{0} \tag{16}$$

These equations have the form $A_i\mathbf{p} = 0$, where $A_i$ is a $3 \times 12$ matrix, and $\mathbf{p}$ is a 12-vector made up of the entries of the matrix $P$,

$$\mathbf{p} = \begin{bmatrix} \mathbf{p}^1 \\ \mathbf{p}^2 \\ \mathbf{p}^3 \end{bmatrix}, P = \begin{bmatrix} p_1 & p_2 & p_3 & p_4 \\ p_5 & p_6 & p_7 & p_8 \\ p_9 & p_{10} & p_{11} & p_{12} \end{bmatrix} \tag{17}$$

Each point correspondence gives rise to two independent equations in the entries of $P$. Given a set of $n$ such point correspondences, we obtain a set of equations $A\mathbf{p} = 0$, where $A$ is the matrix of equation coefficients built from the matrix rows $A_i$ contributed from each correspondence, and $\mathbf{p}$ is the vector of unknown entries of $P$. The non-zero solution $\mathbf{p}$ is the unit singular vector corresponding to the smallest singular value of $A$. The procedure for estimating projection matrix $P$ is summarized in algorithm 1.

Since $K$ is known, we can estimate $\{R, T\}$:

$$k[R, T] = \gamma \begin{bmatrix} p_1 & p_2 & p_3 & p_4 \\ p_5 & p_6 & p_7 & p_8 \\ p_9 & p_{10} & p_{11} & p_{12} \end{bmatrix} \tag{18}$$

$$\Rightarrow \gamma R = K^{-1} \begin{bmatrix} p_1 & p_2 & p_3 \\ p_5 & p_6 & p_7 \\ p_9 & p_{10} & p_{11} \end{bmatrix} \tag{19}$$

$$K^{-1} \begin{bmatrix} p_1 & p_2 & p_3 \\ p_5 & p_6 & p_7 \\ p_9 & p_{10} & p_{11} \end{bmatrix} = U \begin{bmatrix} d_1 & 0 & 0 \\ 0 & d_2 & 0 \\ 0 & 0 & d_3 \end{bmatrix} \tag{20}$$

$$\Rightarrow \gamma \approx \frac{d_1 + d_2 + d_3}{3} \Rightarrow R = UV^T \tag{21}$$

$$\Rightarrow T = K^{-1}[p_4 \quad p_8 \quad p_{12}]^T/\gamma \tag{22}$$

The reproject based on the output of linear PnP is visualized in Fig. 6. We can see that linear PnP still has large reprojection errors. According to Table I, error at this stage is the largest of all. Since the pose of frame 2 is extracted from the Essential frame matrix, PnP is not required for this frame, hence the missing entry in Table I.

**Algorithm 1:** linear camera pose estimation for $P$

**Objective:** Given $n \geq 6$ 2D to 3D point correspondences $\mathbf{x}_i \leftrightarrow \mathbf{X}_i$, determine the projection matrix $P$ such that $\mathbf{x}'_i = P\mathbf{X}_i$.

Writing $\mathbf{x}_i = (x_i, y_i, \omega_i)$. For each correspondence $\mathbf{x}_i \leftrightarrow \mathbf{X}_i$ compute the matrix

$$A_i = \begin{bmatrix} \mathbf{0}^T & -\omega_i \mathbf{X}_i^T & y_i \mathbf{X}_i^T \\ \omega_i \mathbf{X}_i^T & \mathbf{0}^T & -x_i \mathbf{X}_i^T \end{bmatrix};$$

Assemble the $n$ $2 \times 12$ matrices $A_i$ into a single $2n \times 12$ matrix $A$;

Obtain the SVD of $A$. The unit singular vector corresponding to the smallest singular value is the solution $\mathbf{p}$. Specifically, if $A = UDV^T$ with $D$ diagonal with positive diagonal entries, arranged in descending order down the diagonal, then $\mathbf{p}$ is the last column of $V$;

The matrix $P$ is determined from $\mathbf{p}$ as

$$P = \begin{bmatrix} p_1 & p_2 & p_3 & p_4 \\ p_5 & p_6 & p_7 & p_8 \\ p_9 & p_{10} & p_{11} & p_{12} \end{bmatrix};$$

*2) PnP RANSAC:* To remove incorrect matches, The perspective transformation and the corresponding camera pose are computed using Random Sample Concensus algorithm described in algorithm 2.

**Algorithm 2:** RANSAC

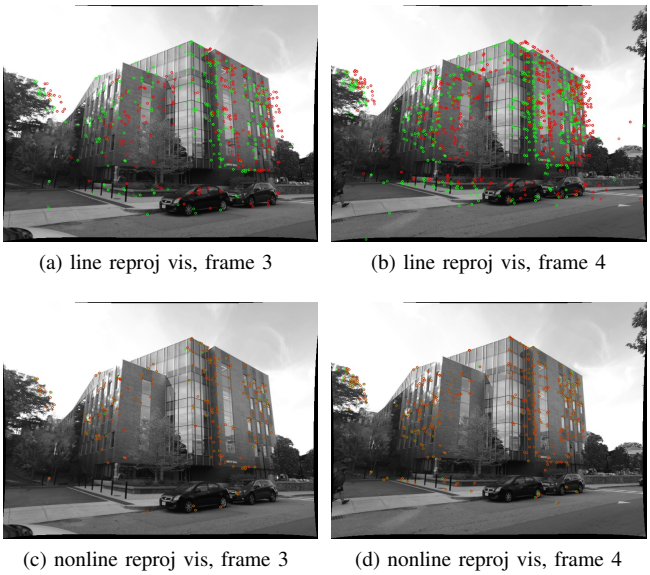
**while** $iterations < N_{max}$ **do**
- Select six images pairs (at random), $\mathbf{x}_i$ from image , $\mathbf{X}'_i$ from 3D points;
- Compute perspective transformation $P$ between the previously picked point pairs;
- Extract camera pose $[R \mid T]$ according to (21)-(22);
- Reconstruct perspective transformation as $P = K[R \mid T]$;
- Compute inliers where $SSD(\mathbf{x}_i, P\mathbf{X}_i) < \tau$, where $\tau$ is some user chosen threshold and $SSD$ is sum of square difference function;
- increment $iterations$;

**end**

Keep largest set of inliers;

Re-compute least-squares $\hat{P}$ estimate on all of the inliers;

Extract camera pose $[R \mid T]$ according to (21)-(22).



(a) line reproj vis, frame 3 (b) line reproj vis, frame 4

(c) nonline reproj vis, frame 3 (d) nonline reproj vis, frame 4

Fig. 6: Comparison of projections between non-linear vs linear PnP. The first column corresponds to the third frame and the second column corresponds to the fourth frame. The top row shows the output of linear PnP and the bottom row shows the output of non-linear PnP. Green dots show the feature detections and red dots show reprojections.

*3) Nonlinear PnP:* To obtain more accurate results by minimizing the projection error, we run the nonlinear PnP algorithm. The reprojection error is largely reduced by this step, as shown in the bottom row of Figure 6. Same conclusion may be drawn by looking at the reprojection errors of linear and nonlinear PnP in Table I.

### *F. Bundle Adjustment*

*1) Visibility Matrix:* Since we are using the large-scale BA in scipy, the visibility matrix is used to construct the sparse structure of the jacobian matrix for speed up, as visualized in Figure 7. For visualization, the jacobian matrix is transposed. Each column indicates a reprojection error term, each row indicates a parameter. If one parameter has an effect on a reprojection error, the corresponding jacobian matrix entry would be one, otherwise zero. The first few rows correspond to the camera poses, if the reprojection error is taken on that camera frame, the entry would be one. The remaining rows correspond to the coordinates of the 3D points, if the reprojection error is computed for a particular 3D point, the corresponding entry would be 1.

*2) Bundle Adjustment:* The reprojection error before and after BA for the first two frames are visualized in Figure 8. The reprojection of the 3D points after BA is visualized in Figure 9. We can see that after BA, the reprojection error is largely reduced. The qualitative analysis summarized in Table I also endorses this claim.

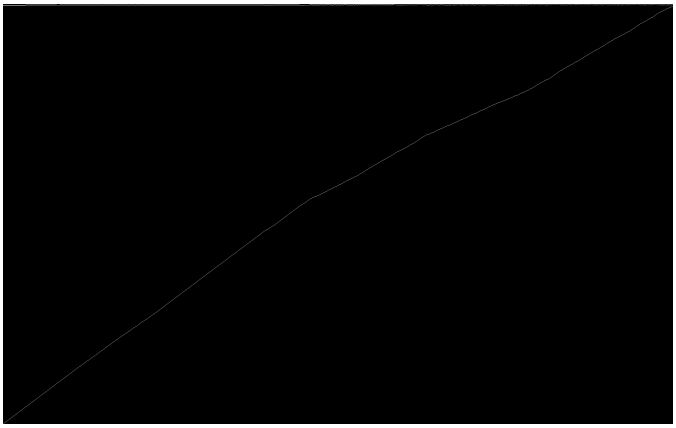
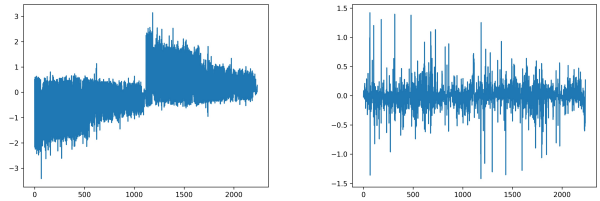


Fig. 7: The transposition of the constructed sparse jacobian matrix.



(a) Reprojection errors before BA (b) Reprojection errors after BA

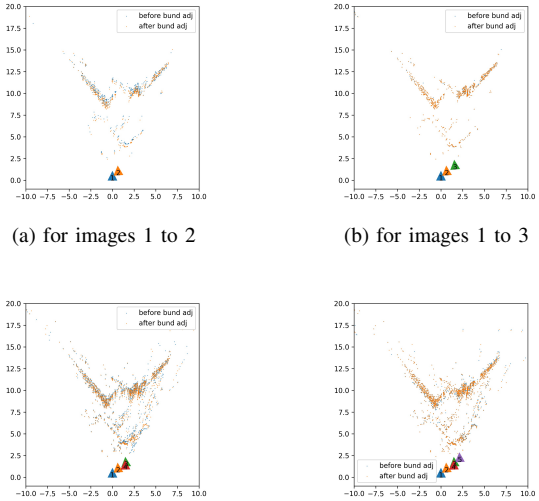
Fig. 8: Reprojection errors on frames 1 and 2 before and after BA.

(a) frame 1 (b) frame 2

Fig. 9: Projections after BA. The first column corresponds to the first frame and the second column corresponds to the second frame. Green dots show the feature detections and red dots show reprojections.

The final reconstructed scene after Sparse Bundle Adjustment (SBA) is visualized Figure 10. We can see that as a new image frame comes in, new 3D map points are spawned and the average reprojection error increases. However, after performing bundle adjustments, the reprojection error is again minimized. Same observations can be made by looking at reprojection errors before and after BA in Table I. As we collect more and more observations, we can also see that the shape of the building and the street begins to emerge in Figure 10.

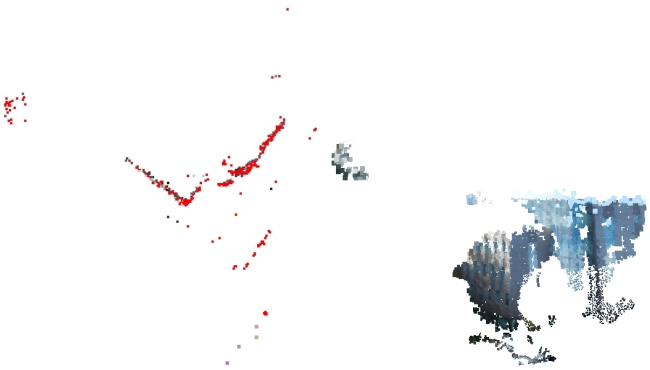
Compare our results to against VSfM [3, 4] output in Figure 11, we can see that the results are similar.




(a) for images 1 to 2 (b) for images 1 to 3

(c) for images 1 to 4 (d) for images 1 to 5.

Fig. 10: The final reconstructed scene after Sparse Bundle Adjustment (SBA).



(a) (b)

Fig. 11: Outputs from Visual SfM. Left to right: top view, and dense reconstruction.

## II. Phase 2: Deep Learning Approach

For this section, the network archetecture and the overall algorithm is following the paper [5, 6]. And the corresponding code take some GitHub repositories [6, 7, 8] as reference. The data is obtained from the course subject [1].

### A. Network Parameters

The network architecture is shown in Figure 12.

Corresponding parameter is shown in Table II:

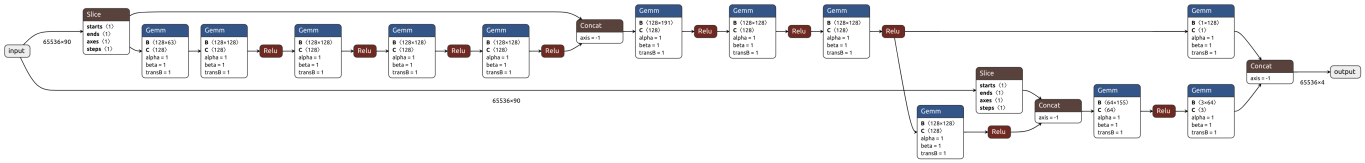



Fig. 12: NeRF Network Architecture

| Parameter Name | Value |
|---|---|
| Training Iterations | 200,000 |
| Neural Network Layer | 8 |
| Neural Network Neuron | 128 |
| position encoding function number | 10 |
| direction encoding function number | 4 |

TABLE II: NeRF Parameters

For the loss function, we use mean square error(MSE)[9] as the loss function. Also, we introduce Peak signal-to-noise ratio(PSNR)[10] as the accuracy metric for the evaluation.

For the optimizer, we use Adam [11] optimizer with learning rate 0.005.

As stated in the given paper and GitHub repository [5, 6], we will use two identical models for coarse image and fine image training.

### *B. Result*

The training loss is shown in Figure 13:

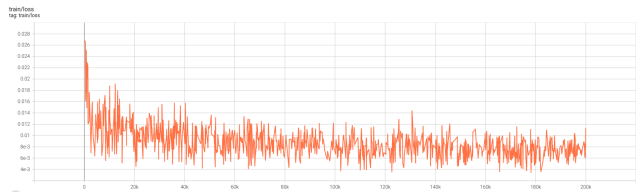



Fig. 13: Train Loss

The training PSNR[10] is shown in Figure 14:

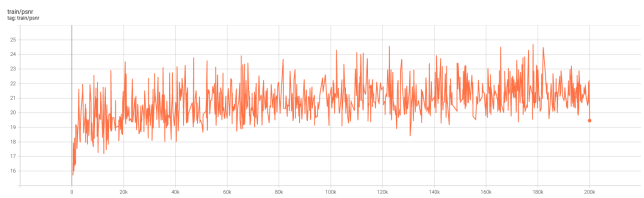



Fig. 14: Train Peak Signal-to-Noise Ratio

The validation PSNR[10] is shown in Figure 15:

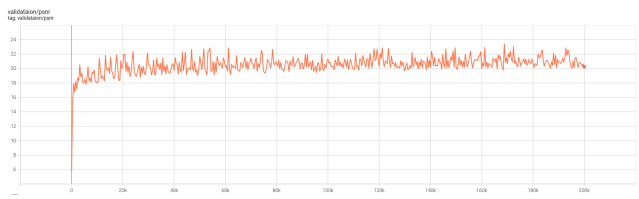
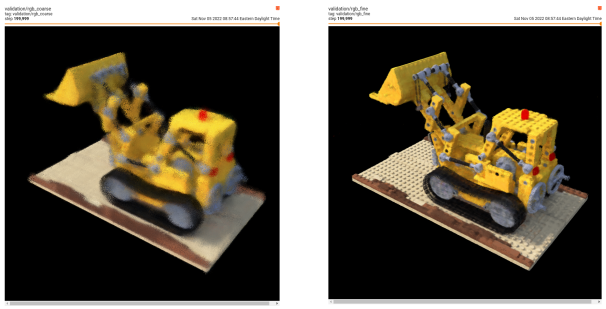



Fig. 15: Validation Peak Signal-to-Noise Ratio

Furthermore, we select one random image shown in Figure 16 from the validation set as the ground truth:


Fig. 16: Example: Ground Truth

And the coarse and fine output from the neural network is shown in Figure 17:



(a) coarse output (b) fine output

Fig. 17: Example: Output for NeRF

As shown in Figure 18, it is one of the frames which can construct a gif:

Fig. 18: Novel View of the GIF

### C. Discussion

The NeRF network training and evaluation take the computational power heavily. It takes at least 2 hours to finish the training and 20 min for evaluation.

Except for the computation time consuming, another major challenge is the differences of functions between Pytorch and Numpy or Math.

For example, `torch.meshgrid(···)` and `numpy.meshgrid(···, index='xy')` has some difference on the generated meshgrid sequence order. Therefore, the solution is that we have to reconstruct a meshgrid function by ourselves. There are multiple functions with similar problems stated above. In addition, since `torch.searchsorted()` has difference performance compared to `numpy.searchsorted()`, we have to use a custom package [12] for searching pytorch tensors in sequence.

Moreover, another potential issue is about `model.train()` and `model.eval()` in Pytorch. We should enable those two functions for coarse model only when training. Otherwise, the fine model cannot have the expected results. But for the evaluation, we should enable the functions for both coarse and find models.

Furthermore, if we don't use the camera poses as input, then coarse model will have blank output and fine model will have the coarse output.

## References


[1] Nitin J. Sanket and Lening Li. *P3 Guidence*. URL: https://rbe549.github.io/fall2022/proj/p3/.

[2] Richard Hartley and Andrew Zisserman. *Multiple view geometry in computer vision*. Cambridge university press, 2003.

[3] Changchang Wu et al. "Multicore bundle adjustment". In: *CVPR 2011*. IEEE. 2011, pp. 3057–3064.

[4] Changchang Wu. "Towards linear-time incremental structure from motion". In: *2013 International Conference on 3D Vision-3DV 2013*. IEEE. 2013, pp. 127–134.

[5] Ben Mildenhall et al. "Nerf: Representing scenes as neural radiance fields for view synthesis". In: *Communications of the ACM* 65.1 (2021), pp. 99–106.

[6] Ben Mildenhall et al. *NeRF Repository*. URL: https://github.com/bmild/nerf.

[7] Lin Yenchen et al. *NeRF Repository*. URL: https://github.com/yenchenlin/nerf-pytorch.

[8] Krishna Murthy and Mikhail Grankin. *NeRF Repository*. URL: https://github.com/krrish94/nerf-pytorch.

[9] Mark D Schluchter. "Mean square error". In: *Encyclopedia of Biostatistics* 5 (2005).

[10] Alain Hore and Djemel Ziou. "Image quality metrics: PSNR vs. SSIM". In: *2010 20th international conference on pattern recognition*. IEEE. 2010, pp. 2366–2369.

[11] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[12] Antonie Liutkus and Krishna Murthy. *Custom Pytorch Function Package*. URL: https://github.com/aliutkus/torchsearchsorted.