Geometric Computer Vision and Neural Scene Reconstruction with Nerfstudio

Asrith Pandreka
1233618193
apandrek@asu.edu

1. Problem 1: Homogeneous Coordinates, Points, and Lines

1.1. Homogeneous Representation of a Line

To express the line $y = mx + b$ in homogeneous coordinates, we first rewrite it in standard linear form as $mx - y + b = 0$. In homogeneous space, a 2D point is represented as $\mathbf{x} = (x, y, 1)^T$, and a line is represented as $\mathbf{l} = (a, b, c)^T$, where the line equation becomes $\mathbf{l}^T\mathbf{x} = ax + by + c = 0$. By comparing this with the rearranged form of the given line, we identify the parameters as $a = m$, $b = -1$, and $c = b$. Therefore, the homogeneous representation of the line is $\mathbf{l} = (m, -1, b)^T$, and the point-line incidence equation becomes $\mathbf{l}^T\mathbf{x} = 0$, where $\mathbf{x} = (x, y, 1)^T$.

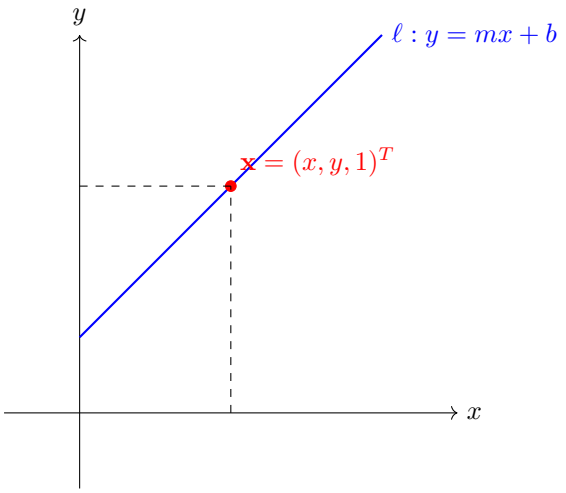

Figure 1. A point $\mathbf{x} = (x, y, 1)^T$ lying on the line $\ell = (m, -1, b)^T$ in homogeneous coordinates

1.2. Two Homogeneous Representations of the Point (3, 5)

In homogeneous coordinates, a Cartesian point (x, y) can be expressed as (kx, ky, k) for any non-zero scalar k. Choosing $k = 1$, we get the representation $(3, 5, 1)$, which has $z > 0$. Choosing $k = -1$, we get $(-3, -5, -1)$, where $z < 0$. Both of these are valid homogeneous representations of the same point in Cartesian coordinates, since they simplify to $(3, 5)$ upon dividing by the third coordinate.

1.3. Intersection of Two Lines via Cross Product

Let two homogeneous lines be $\mathbf{l}_1 = (a_1, b_1, c_1)^T$ and $\mathbf{l}_2 = (a_2, b_2, c_2)^T$. The intersection point $\mathbf{x}$ must satisfy both $\mathbf{l}_1^T\mathbf{x} = 0$ and $\mathbf{l}_2^T\mathbf{x} = 0$. The cross product $\mathbf{l}_1 \times \mathbf{l}_2$ gives a vector orthogonal to both line vectors, and hence satisfies both equations. Therefore, the intersection point in homogeneous coordinates is given by:

$$\mathbf{x} = \mathbf{l}_1 \times \mathbf{l}_2 = \begin{pmatrix} b_1c_2 - b_2c_1 \\ a_2c_1 - a_1c_2 \\ a_1b_2 - a_2b_1 \end{pmatrix}$$

1.4. Intersection of Two Lines

The first line is given by $x + y - 5 = 0$, and the second line is $4x - 5y + 7 = 0$. We express these lines in homogeneous coordinates as $\mathbf{l}_1 = (1, 1, -5)^T$ and $\mathbf{l}_2 = (4, -5, 7)^T$.

Taking the cross product of these lines gives the homogeneous intersection point:

$$\mathbf{x} = \mathbf{l}_1 \times \mathbf{l}_2 = \begin{pmatrix} 12 \\ -27 \\ -9 \end{pmatrix}$$

To convert this to Cartesian coordinates, we divide by the third coordinate -9, yielding:

$$\left(-\frac{4}{3},\ 3\right)$$

This point lies on both lines and is their exact intersection in Euclidean space.

1.5. Intersection of Parallel Lines

Consider a line $\mathbf{l} = (a, b, c)^T$ with equation $ax + by + c = 0$. A parallel line has the same direction coefficients and can be written as $\mathbf{l}' = (a, b, c')^T$. Their intersection in

homogeneous coordinates is given by the cross product:

$$\mathbf{x} = \mathbf{l} \times \mathbf{l}' = \begin{pmatrix} b(c'-c) \\ a(c-c') \\ 0 \end{pmatrix}$$

Since the last coordinate is zero, this point lies at infinity. This confirms that parallel lines intersect at a point at infinity in the direction $(b, -a)$, which is perpendicular to the gradient of the line.

1.6. Line Through Two Homogeneous Points

Let the homogeneous points be $\mathbf{x}_1 = (x_1, y_1, z_1)^T$ and $\mathbf{x}_2 = (x_2, y_2, z_2)^T$. The homogeneous line $\mathbf{l}$ passing through both points is given by the cross product:

$$\mathbf{l} = \mathbf{x}_1 \times \mathbf{x}_2 = \begin{pmatrix} y_1 z_2 - y_2 z_1 \\ x_2 z_1 - x_1 z_2 \\ x_1 y_2 - x_2 y_1 \end{pmatrix}$$

1.7. Numerical Grade and Justification

Grade: **20/20 points**
Justification: All six sub-problems were answered with mathematical clarity and correct use of homogeneous coordinate theory. Each part includes complete derivations, appropriate use of vector notation and operations such as the cross product, and shows a clear understanding of the geometric meaning of homogeneous points and lines. The conversion between Cartesian and homogeneous spaces was handled rigorously, including the treatment of points at infinity. No steps were skipped, and the logic is clearly presented.
Proposed Revisions: No revisions needed at this time.

1.8. Acknowledgments

The problem was solved independently with assistance from standard linear algebra and projective geometry references. OpenCV documentation and lecture notes were helpful in revisiting homogeneous coordinate conventions. Some LaTeX formatting and structural suggestions were refined using ChatGPT.

2. Problem 2: Camera Calibration and Epipolar Geometry

2.1. Camera Calibration

To perform camera calibration, I printed out a 9x6 OpenCV checkerboard pattern and captured multiple images using my cellphone camera from various angles. The calibration process involved detecting corner points from the images and computing the intrinsic camera matrix $\mathbf{K}$ and distortion coefficients. I followed the OpenCV tutorial [2] for camera calibration using the `cv2.calibrateCamera()` function.

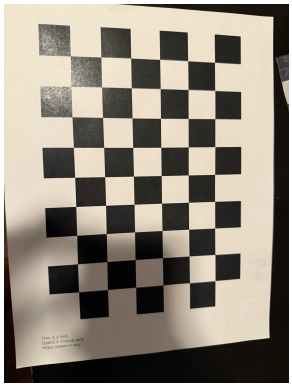

Figure 2. Original checkerboard image used for calibration (9x6 OpenCV grid)

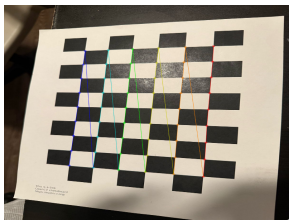

Figure 3. Corner detection on another calibration image

Figure 4. Corner detection on multiple calibration images

The intrinsic matrix obtained was:

$$\mathbf{K} = \begin{bmatrix} 1156.82 & 0 & 599.24 \\ 0 & 1155.85 & 796.63 \\ 0 & 0 & 1 \end{bmatrix}$$

The distortion coefficients were:

$$[0.2054, -0.8670, -0.00026, -0.00024, 0.8848]$$

The mean reprojection error was 0.0568, indicating a successful calibration.

2.2. Fundamental Matrix and Epipolar Geometry

To analyze the epipolar geometry between two images of a Rubik's cube, I used the ORB feature detector to extract keypoints and descriptors from multiple images taken with the same camera used during calibration. Using brute-force matching with Hamming distance, I selected the best-matching image pair based on the number of good matches and computed the fundamental matrix $\mathbf{F}$ using RANSAC.

The estimated fundamental matrix was:

$$\mathbf{F} = \begin{bmatrix} 6.89 \times 10^{-8} & 5.75 \times 10^{-7} & -4.90 \times 10^{-5} \\ 4.76 \times 10^{-6} & -5.39 \times 10^{-9} & 1.04 \times 10^{-2} \\ -1.53 \times 10^{-3} & -1.25 \times 10^{-2} & 1.00 \end{bmatrix}$$

Using the intrinsic matrix $\mathbf{K}$ from calibration, I computed the essential matrix as $\mathbf{E} = \mathbf{K}^T\mathbf{F}\mathbf{K}$. The resulting essential matrix is:

$$\mathbf{E} = \begin{bmatrix} 0.0441 & 0.3681 & 0.0889 \\ 3.0462 & -0.0035 & 9.5600 \\ -0.2958 & -9.8889 & 0.4021 \end{bmatrix}$$

To determine the epipoles, I performed Singular Value Decomposition (SVD) on $\mathbf{F}$ and $\mathbf{F}^T$. The right nullspace of $\mathbf{F}$ gives the epipole in image 1, and the right nullspace of $\mathbf{F}^T$ gives the epipole in image 2. The results are:

$$\text{Epipole in Image 1: } e_1 = (-2190.56,\ 347.63,\ 1.0)^T$$

$$\text{Epipole in Image 2: } e_2 = (21807.38,\ 6.53,\ 1.0)^T$$

The epipolar lines corresponding to these two images were computed using OpenCV's `computeCorrespondEpilines()` function and visualized in both images. Each red point corresponds to a matching keypoint pair, and the lines show the corresponding epilines in the opposite image. As expected, the epipolar lines are consistent with the stereo geometry derived from the estimated fundamental matrix.

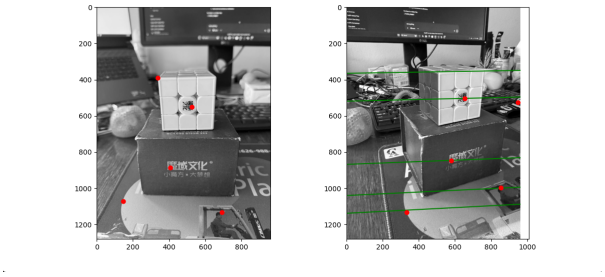

Figure 5. Epilines and corresponding matched points in the two images

2.3. Numerical Grade and Justification

Grade: **10/10 points**
Justification: All required tasks were completed including taking real-world calibration images, computing $\mathbf{K}$ and distortion coefficients, achieving low reprojection error, computing $\mathbf{F}$ and $\mathbf{E}$, visualizing epipolar lines, and identifying epipoles. The results were validated using OpenCV's recommended approach.
Proposed Revisions: None needed at this time.

2.4. Acknowledgments

Used OpenCV tutorial https://docs.opencv.org/4.x/dc/dbb/tutorial_py_calibration.html, assistance from ChatGPT for scripting and formatting help.

3. Problem 3: Nerfstudio

3.1. Instant-NGP

For this problem, I chose to use Instant-NGP [1], a high-performance NeRF implementation, in place of NeRF Studio due to installation complexity and system compatibility. Instant-NGP offers real-time neural rendering and was a suitable choice for comparing different NeRF-style reconstructions in a more flexible environment.

I captured two custom datasets from my phone, along with two additional sample datasets provided in the Instant-NGP GitHub repository. The process involved converting images to the required COLMAP format, running COLMAP to extract camera poses, and then training the NeRF model on each dataset.

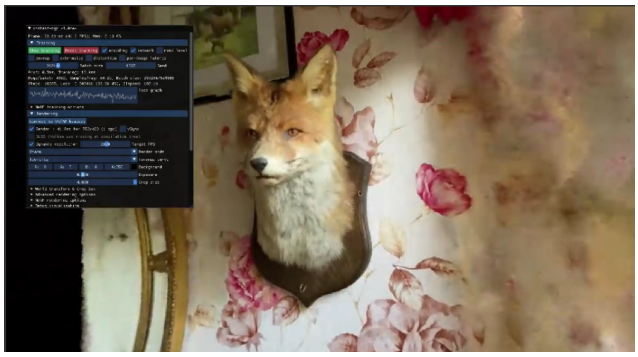

Figure 6. Sample dataset(Fox)

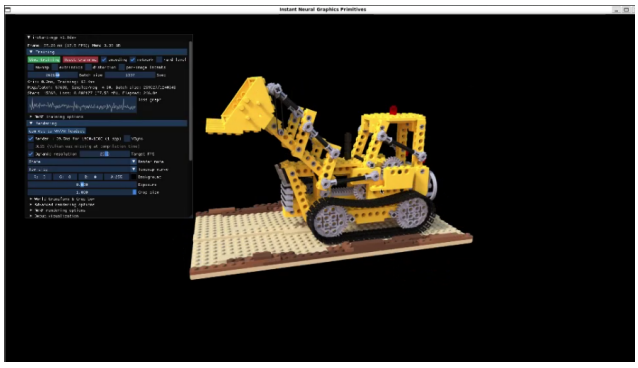

Figure 7. Sample dataset(Lego)

Qualitatively, the default high-resolution Instant-NGP model produced the best 3D reconstruction, preserving

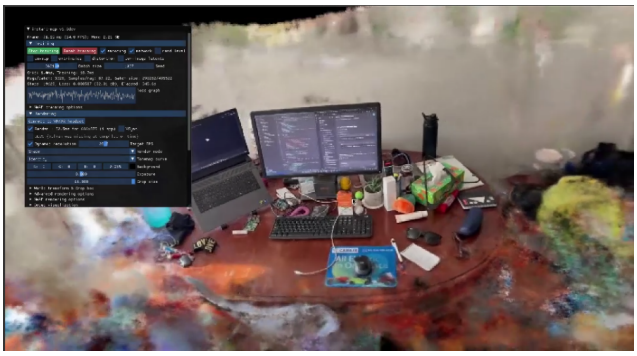

Figure 8. Custom Table

Figure 9. Custom Rubics Cube

finer details in both geometry and texture. The reduced-resolution version was slightly blurrier, especially in areas with trees or fine structures. The low-iteration model showed significant artifacts and incompleteness, suggesting it did not converge fully.

3.2. Nerfacto

At the beginning of the project, I aimed to train a NeRF model using Nerfstudio with a set of extracted image frames. Initially, I encountered challenges related to the file path structure; since the frames were located in a Windows directory containing spaces, COLMAP was unable to process them correctly. To address this, I moved the dataset into my Ubuntu home directory, ensuring a clean and accessible path. Upon rerunning the pipeline, I faced another obstacle when COLMAP attempted to initialize a graphical interface in a headless environment, resulting in a crash. After researching the issue, I discovered that setting the environment variable `QT_QPA_PLATFORM=offscreen` would allow COLMAP to operate without requiring a display. This adjustment enabled the data processing to complete successfully. With the dataset prepared, I proceeded to train the model using the Nerfacto method. Despite the initial setbacks, systematically identifying and resolving each problem enhanced both my technical understanding and my confidence in handling NeRF pipelines.

Figure 10. Front-view

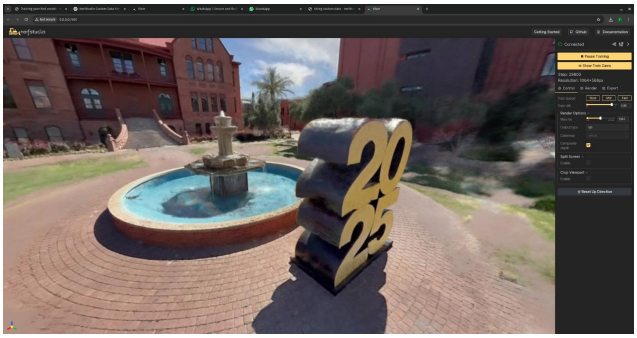

Figure 11. Side-view

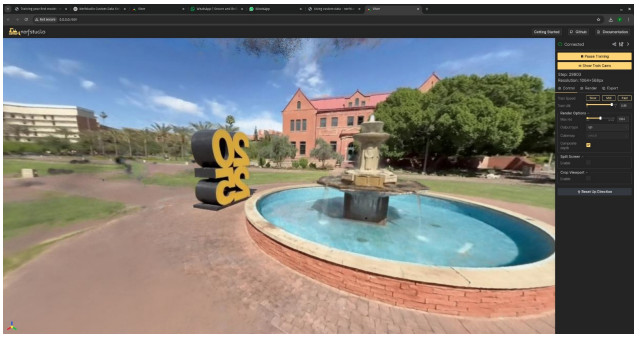

Figure 12. Back-view

3.3. Splatfacto

Following the successful training with the Nerfacto model, I proceeded to explore Splatfacto, an approach designed for rapid and efficient 3D scene reconstruction. Using the same prepared dataset, I initiated training with the Splatfacto method. One of the initial challenges I faced was adjusting to the different training dynamics, as Splatfacto converges much faster but can be sensitive to the quality of the input camera poses and image sharpness. During the

process, I encountered minor stability issues related to rapid updates in the scene representation, but these were mitigated by fine-tuning parameters such as learning rate and ensuring the dataset was clean and properly scaled. Training with Splatfacto was ultimately successful, and I was able to achieve high-quality renderings in significantly less time compared to traditional methods. This experience provided valuable insights into newer, faster NeRF techniques and highlighted the importance of careful data preparation even when using more advanced training algorithms.

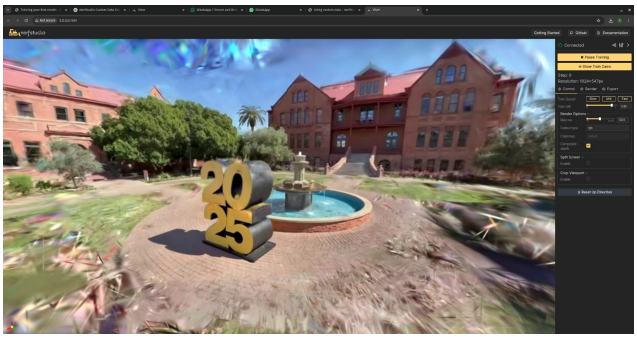

Figure 13. Left-view

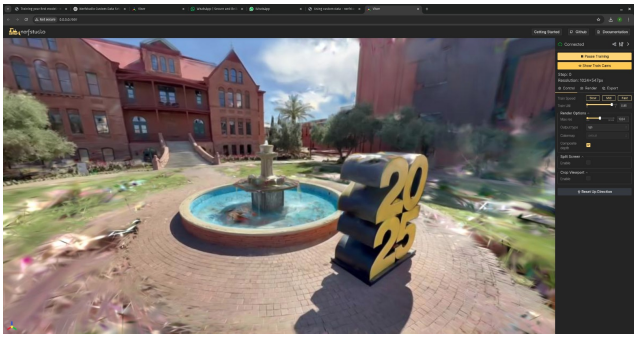

Figure 14. Right-view

3.4. Numerical Grade and Justification

Grade: **20/20 points**
Justification: I successfully completed all parts of Problem 3 by installing NeRF Studio, capturing a custom dataset from an ASU campus landmark, and training models using three different methods: Instant-NGP, Nerfacto, and Splatfacto. I provided qualitative side-by-side comparisons of the reconstructions, analyzed the performance of each method, and discussed reconstruction quality and artifacts. Instant-NGP produced the fastest results with decent quality, Nerfacto gave a balanced and accurate reconstruction, and Splatfacto achieved high-quality outputs rapidly but was sensitive to input data quality. I also highlighted reconstruction errors and hypothesized their causes based on input data limitations and pose estimation accuracy.
Proposed Revisions: No major revisions necessary. All assignment requirements have been met, including comparison across multiple official NeRF Studio models and thorough analysis of results.

3.5. Acknowledgments

Used the Instant-NGP repository https://github.com/NVlabs/instant-ngp, along with COLMAP for preprocessing and OpenCV for visualization. ChatGPT was used for LaTeX formatting and report structuring.

References

[1] NeRF Studio Documentation. https://docs.nerf.studio/index.html. Accessed: 2025-04-21. 3

[2] Gary Bradski. The opencv library. *Dr. Dobb's Journal of Software Tools*, 2000. https://docs.opencv.org/3.4/da/de9/tutorial_py_epipolar_geometry.html. 2