CS5330 Project Report: 3D Reconstruction Pipeline for Object Modeling

Ramez Mubarak
Northeastern University
NUID: 001865560
mubarak.r@northeastern.edu

Abhijit Kaluri
Northeastern University
NUID: 002853560
kaluri.a@northeastern.edu

Abstract

This project will focus on 3D reconstructions of physical objects from pictures, an important task in robotics applications such as navigation and object manipulation. The project will involve implementing a 3D reconstruction pipeline, using techniques from lectures and journals, to recover the 3D structure of scenes captured from multiple 2D images. We will explore key challenges such as feature detection, camera pose estimation, and point cloud generation.

1 Project Overview

1.1 Introduction

3D reconstruction from 2D images is a key technology in computer vision, with applications in robotics, AR/VR, structural analysis, and urban planning. However, it presents challenges such as integrating edge and feature detection, depth and pose estimation, and addressing variability in lighting, texture, and perspective. While state-of-the-art solutions exist, they are often costly.

This project uses open-source libraries to create a 3D reconstruction program that is accessible to anyone, especially those using smartphones. By focusing on open-source development, it lowers barriers to entry, enabling broad adoption across industries. The program generates accurate point clouds from 2D images captured with a standard iPhone camera, handling a variety of object scales—from small items to large structures. The point clouds are formatted for visualization in open-source Python packages, providing an intuitive platform for further analysis and integration.

1.2 Related works

Several journals were investigated before this project started. The following journals proved useful for understanding and building our own 3D pipeline from scratch.

- O. Faugeras, L. Robert, S. Laveau, G. Csurka, C. Zeller, C. Gauclin, and I. Zoghlami, "3-D reconstruction of urban scenes from image sequences" [1]
- J. Wu, Z. Cui, V. S. Sheng, P. Zhao, D. Su, and S. Gong, "A comparative study of SIFT and its variants" [2]

Submitted to 34th Conference on Neural Information Processing Systems (NeurIPS 2020). Do not distribute.

- K. G. Derpanis, "Overview of the RANSAC algorithm" [3]
- W. Guilluy, L. Oudre, and A. Beghdadi, "Video stabilization: Overview, challenges and perspectives" [4]
- Z. Zhang, "Camera calibration" [5]
- B. Rosenhahn, "Pose estimation revisited" [6]

The paper by Faugeras et al. (2004) discusses 3D reconstruction of urban scenes from image sequences, providing significant insights into vision-based modeling techniques for urban environments [1]. Wu et al. (2013) present a comparative study of the SIFT algorithm and its variants, offering a comprehensive analysis of feature detection and matching methods [2]. Derpanis (2010) provides an overview of the RANSAC algorithm, emphasizing its application in robust parameter estimation despite noisy data [3]. Guilluy et al. (2021) examine the challenges and future directions of video stabilization techniques, addressing the complexities involved in stabilizing dynamic video footage [4]. Zhang (2021) explores camera calibration techniques in computer vision, highlighting their role in improving the accuracy of 3D reconstruction models [5]. Finally, Rosenhahn (2003) revisits pose estimation, offering a detailed examination of methods to improve the precision of 3D object localization [6].

1.3 Open-source libraries used

- OpenCV
- VidStab
- NumPy
- Open3D
- Matplotlib

2 Methodology

2.1 Overview

Our 3D pipeline starts by extracting and stabilizing frames from a video for reconstruction. If a calibration pattern is available, camera calibration is performed using open-source libraries to estimate camera intrinsics and distortion coefficients.

Keypoints and feature descriptors (e.g., SIFT) are detected and matched across frames using an open-source feature matching algorithm. This allows the estimation of the essential camera matrix and relative camera pose using RANSAC. Once camera poses and correspondences are established, 3D coordinates are triangulated to create a point cloud. This point cloud is then visualized using tools like OpenCV, PCL, or Open3D for further analysis.

The pipeline is modularized for ease of testing, troubleshooting, and enhancement, resulting in a 3D model for use in robotics and computer vision applications.

2.1.1 Frame extraction and stabilization

The first step in the 3D reconstruction pipeline involves extracting and stabilizing frames from a video using the function `stabilize_video_and_save_images_with_vidstab`. This function reads the video file, stabilizes each frame using VidStab's smoothing window, and saves the stabilized frames as images in the specified output folder. The stabilization reduces motion artifacts, improving feature detection and matching in subsequent steps. The function returns the total number of frames processed, ensuring better alignment for the 3D reconstruction pipeline, especially in videos with jitter or shaking.

2.1.2 Camera calibration

Camera calibration corrects lens distortion and estimates intrinsic parameters like focal length and principal point. When a calibration pattern (e.g., checkerboard) is available, the `cv2.calibrateCamera` function in OpenCV is used to estimate these parameters. In this pipeline, a generic calibration matrix is estimated from iPhone image data, ensuring accurate 3D reconstruction by compensating for lens distortion and camera miscalibration.

2.1.3 Feature extraction

Feature detection is the process of identifying distinctive points or regions in each image that can be used to match features across multiple frames. This is a critical step in establishing correspondences between images and is necessary for reconstructing the 3D geometry of the object. The `detect_features` function is used to detect keypoints and compute feature descriptors in each frame. Popular feature detection algorithms, such as SIFT, ORB, or SURF, are utilized to identify and describe the unique characteristics of these keypoints. SIFT (Scale-Invariant Feature Transform) is known for its robustness to scaling, rotation, and changes in illumination, while ORB (Oriented FAST and Rotated BRIEF) is a fast and efficient alternative. These keypoints provide the foundation for matching features across frames, which is essential for understanding the spatial relationship between different viewpoints of the object. In this paper's pipeline, SIFT is the main method used for feature extraction.

2.1.4 Feature matching

After detecting features in each frame, the next step involves matching these features across frames to establish correspondences. In this pipeline, The match_features function uses FLANN-based Matching (cv2.FlannBasedMatcher) to perform feature matching between two sets of feature descriptors (desc1 and desc2). It initializes the FLANN matcher with a KD-tree algorithm for efficient nearest-neighbor search and performs k-Nearest Neighbors (k-NN) matching with k=2. For each pair of matches, it applies Lowe's ratio test (comparing the distance of the two nearest neighbors) to retain only the good matches, defined as those where the distance of the closest match is less than 65% of the distance to the second closest match. This helps in rejecting false matches and improving the accuracy of feature matching.

By identifying high-quality matches, the pipeline can robustly track motion and determine feature correspondences between frames. This is critical for accurate camera pose estimation and 3D reconstruction, as it ensures that only meaningful and consistent matches contribute to the generation of the 3D point cloud.

2.1.5 Pose estimation

Camera pose estimation determines the position and orientation of the camera relative to the object in the 3D reconstruction pipeline. In this approach, we use a custom RANSAC-based fundamental matrix estimation followed by OpenCV's essential matrix computation.

First, matched keypoints between frames are passed into the `ransac_fundamental_matrix` function, which uses RANSAC to estimate a reliable fundamental matrix by identifying inlier matches. This process iteratively refines the matrix by evaluating epipolar distances and maximizing inliers.

Next, the inlier matches are used to compute the essential matrix using `cv2.findEssentialMat`, which incorporates a second RANSAC layer to handle geometric inconsistencies. The essential matrix provides the camera's relative rotation and translation.

The `cv2.recoverPose` function is then used to extract the rotation (`R`) and translation (`t`) matrices, ensuring consistency with the camera's intrinsic parameters. Finally, triangulation using `cv2.triangulatePoints` generates 3D points corresponding to the matched features, forming the basis for the 3D point cloud used in reconstruction.

2.1.6 Point cloud generation

With the camera poses and feature matches established, the next step is to generate the 3D point cloud. The `generate_point_cloud` function triangulates the matched points from multiple frames to compute their 3D coordinates. Triangulation involves finding the intersection of multiple camera rays corresponding to the same 2D feature points from different frames. The resulting 3D coordinates are then aggregated into a point cloud, which represents the 3D structure of the object. The accuracy of the point cloud depends heavily on the quality of the feature matching and camera pose estimation steps. For scenes with sparse or low-quality feature matches, additional techniques such as bundle adjustment may be applied to optimize the 3D points and camera poses for better reconstruction quality.

2.1.7 Point cloud visualization

The final step in the pipeline is the visualization of the 3D point cloud. Once the 3D coordinates have been generated, they are rendered using 3D visualization tools like OpenCV, PCL (Point Cloud Library), or Open3D. The `visualize_point_cloud` function is responsible for rendering the point cloud, allowing the user to inspect the 3D structure and identify areas that may need further refinement or improvement. Visualization provides an intuitive way to analyze the accuracy of the reconstruction and is useful for debugging and optimizing the pipeline. In addition to visual inspection, the point cloud can be further processed for applications such as object recognition, scene understanding, or integration into a larger system, such as a robotic navigation platform.

3 Experiments

3.1 Proof of concept

3.1.1 Overview

To evaluate the performance and robustness of our 3D object reconstruction pipeline, we relied on established datasets that have been widely used and tested in the computer vision and robotics communities. Leveraging such datasets ensures that our methods are validated against benchmark scenarios, enabling fair comparisons and reproducible results.

For this purpose, we selected the **Temple Dataset**[7] as the primary dataset for experimentation. This dataset provides over 300 well-processed images of the Dioskouroi Temple in Agrigento, Sicily. As a simple and extensively tested dataset for 3D reconstruction, it is an ideal choice for a proof of concept.

By utilizing this dataset, we were able to focus on validating key aspects of the pipeline, such as feature detection, feature matching, camera pose estimation, and 3D reconstruction, without being hindered by challenges such as noisy or unstructured data. The high-quality, well-curated images allowed us to thoroughly evaluate the foundational components of our approach and ensure robust performance in controlled conditions.

3.1.2 Desired results

This experiment is designed to answer the following questions:

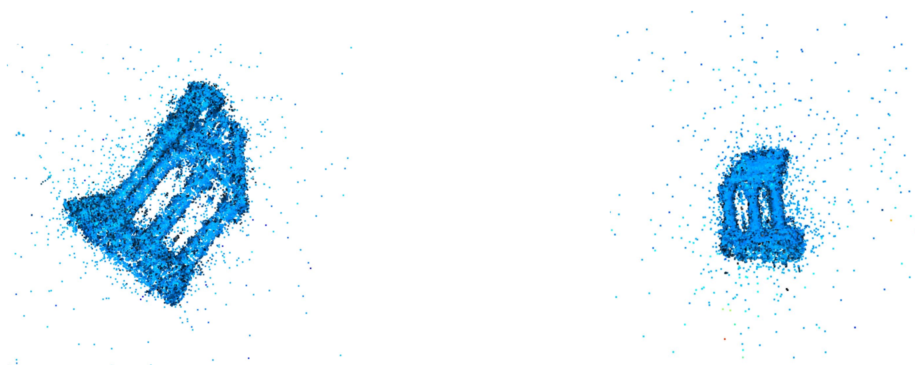
- How effectively does the pipeline perform on the Temple Dataset, a well-established benchmark in the 3D reconstruction community?
- Can the pipeline reliably detect and match features in high-quality images with clear textures, structured depth variations, and consistent lighting conditions?
- How accurately can the camera pose estimation and 3D reconstruction steps perform when using a simple and extensively tested dataset like the Temple Dataset?

- Is the pipeline compatible with benchmark datasets, enabling fair comparisons and reproducible results within the field of computer vision?

3.1.3 Results

(a) Image 1 (b) Image 2

Figure 1: Dioskouroi Temple 3D Model



(a) Image 1 (b) Image 2

Figure 2: 3D reconstruction of the Dioskouroi Temple

Using the Temple Dataset, we successfully created a 3D reconstruction of the Dioskouroi Temple model, demonstrating the effectiveness and reliability of our pipeline. The pipeline performed well on this benchmark dataset, validating its capability to handle high-quality, well-structured inputs. Features were detected and matched consistently across images with clear textures and structured depth variations, ensuring robust feature correspondence. Camera pose estimation were accurate, enabling precise 3D reconstruction of the temple's structure. The successful reconstruction confirms the pipeline's potential applicability to more complex datasets and real-world scenarios in computer vision and robotics.

3.2 Custom dataset

Following preliminary experiments with the Temple Dataset, our next step is to transition from static images to dynamic video input captured with a standard iPhone camera. This aligns with our project's goal of creating an accessible 3D reconstruction pipeline that processes real-world video data from everyday devices.

We will capture iPhone videos, convert them into custom datasets using a stabilization algorithm, and process them through the pipeline. This will validate key aspects of our algorithm, such as frame extraction, feature detection, and matching under real-world conditions like motion blur, varying lighting, and non-ideal trajectories. It will also allow us to integrate and evaluate frame stabilization, which wasn't applied in the preliminary experiments.

By testing on real-world video data, we aim to demonstrate the practical applicability of the pipeline, refine it for better robustness to camera motion, improve computational efficiency, and enhance the quality of generated 3D point clouds.

3.2.1 Desired results

This experiment is designed to answer the following questions:

- Can the pipeline process video input captured using a standard iPhone camera as effectively as it processes static images?
- How well does the pipeline perform on custom datasets created from iPhone-captured videos, including the frame stabilization algorithm?
- What challenges arise in frame extraction, feature detection, and feature matching when dealing with real-world video conditions, such as motion blur, varying lighting, and non-ideal camera trajectories?
- How does the integration of frame stabilization affect the overall performance and reliability of the pipeline in processing dynamic video input?
- How well does the system align with the project's goal of enabling accessible 3D reconstruction for robotics and computer vision applications?
- What insights can be gained regarding the robustness of the pipeline to camera motion, including handheld movement during video capture?
- What refinements are necessary to enhance the quality and accuracy of the 3D reconstructions when transitioning to dynamic video input?

3.2.2 Results

Our pipeline faced significant challenges when processing iPhone videos, with some experiments resulting in the point cloud failing to generate or even causing the pipeline to crash. These issues were largely due to factors such as motion blur, camera shake, varying lighting conditions, and non-ideal camera trajectories, which led to inaccurate feature detection, unreliable feature matching, and poor camera pose estimation. To overcome these limitations, we plan to explore more advanced techniques like Neural Radiance Fields (NeRF), which have demonstrated the ability to generate high-quality 3D reconstructions from unstructured image data. NeRF's advanced modeling capabilities could help address the issues we encountered and improve the robustness of our pipeline in real-world video processing.

3.3 Further Development: NeRF Pipeline Implementation

Neural Radiance Fields (NeRF) have become a prominent method for generating realistic 3D reconstructions from a set of 2D images. In this project, we integrated NeRF into our existing point cloud pipeline to reconstruct the geometry and appearance of the *Temple* dataset from the Middlebury Vision dataset. This section outlines our implementation process, the design decisions made, the challenges encountered, and potential directions for future improvements.

3.3.1 Pipeline Integration

Our objective was to enhance the robustness of our point cloud generation pipeline by incorporating NeRF. The Middlebury Vision *Temple* dataset provided a comprehensive collection of images with accurate camera parameters, making it suitable for training and evaluating our NeRF model.

Data Preparation We started by organizing the dataset, ensuring that each of the 312 high-resolution images was correctly paired with its corresponding intrinsic and extrinsic camera parameters. This alignment was essential for accurate ray casting and depth estimation during the NeRF training process. The images captured the temple from various angles, providing diverse perspectives to the model.

Model Training Using PyTorch, we developed and trained the NeRF model from scratch on our system. The model architecture consisted of a multilayer perceptron (MLP) with positional encoding to capture spatial details. Training was conducted over 100 epochs using the Adam optimizer with a learning rate of 5×10^{-4}. We employed a validation split of 10% to monitor the model's performance and prevent overfitting.

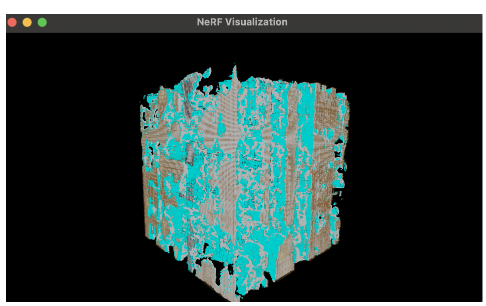
Point Cloud Generation After training, the NeRF model was utilized to generate a dense point cloud representing the temple's geometry. We sampled eight million points within a $200{\times}200{\times}200$ grid bounded by `bbox_min=(-2, -2, -2)` and `bbox_max=(2, 2, 2)`. Each point was evaluated for density and color, retaining those with a density above a threshold of 0.1. This resulted in a point cloud containing approximately three million points, intended to accurately reflect the temple's structure and appearance.

Mesh Generation Converting the point cloud into a coherent mesh involved two surface reconstruction techniques:

- **Poisson Surface Reconstruction:** This method was used to create a smooth and continuous mesh from the point cloud. The initial reconstruction produced a mesh with over 15 million vertices. To manage this, we removed low-density vertices, reducing the mesh to approximately 15.7 million vertices. Further cropping to the bounding box ensured spatial alignment, and re-estimating normals improved the mesh's visual quality.
- **Marching Cubes Algorithm:** To capture finer geometric details, we applied the Marching Cubes algorithm, which generated a mesh with around 3.4 million triangles. This method provided a more detailed representation compared to Poisson reconstruction but with a lower vertex count, making it more computationally efficient.

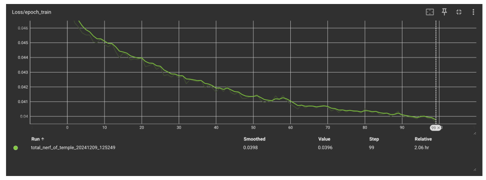
3.3.2 Results and Observations

Upon completing the mesh generation processes, we visualized the results using Open3D's interactive tools. The visualization included both the generated point cloud and the reconstructed meshes. However, the resulting mesh did not accurately depict the temple's intricate structure and instead resembled an irregular cube. This deviation indicated that the current NeRF implementation did not fully capture the scene's complexities.



(a) NeRF Visualization

However, the training loss decreased over time, proving that the pipeline is trainable, motivating future development.



(a) Loss per Epoch for Training Set

4 Discussion & Conclusions

4.1 Challenges and Limitations

Several factors contributed to the unsatisfactory mesh outcome:

- **Grid Resolution:** The chosen grid resolution of 200×200×200 may have been insufficient to capture the temple's finer details. Higher resolutions could potentially provide a more accurate reconstruction but would require more computational resources.
- **Density Thresholding:** Setting the density threshold at 0.1 might have excluded essential points necessary for an accurate mesh. Adjusting this threshold could help retain more critical points.
- **Training Duration:** Training the NeRF model for 100 epochs may not have been adequate for the model to fully learn the scene's intricate geometry. Extending the training period could enhance the model's performance.
- **Surface Reconstruction Parameters:** The parameters used in Poisson Surface Reconstruction and Marching Cubes played a significant role in the mesh quality. Fine-tuning these parameters could improve the accuracy of the reconstructed mesh.

4.2 Future Work

To address the challenges encountered and improve the NeRF pipeline's performance, the following steps are going to be taken:

- **Increase Grid Resolution:** Elevating the grid resolution to 300×300×300 or higher could help capture more detailed structures of the temple, albeit with increased computational demands.
- **Adjust Density Thresholds:** Experimenting with lower density thresholds may allow the retention of more points that are crucial for accurate mesh generation.
- **Extend Model Training:** Continuing the training process for an additional 50 epochs could enable the NeRF model to better learn and represent the temple's complex geometry.
- **Optimize Surface Reconstruction Parameters:** Refining the parameters used in Poisson Surface Reconstruction and Marching Cubes could lead to more accurate and detailed meshes.
- **Explore Advanced Techniques:** Incorporating hierarchical sampling or other advanced NeRF variants might enhance the model's ability to capture intricate details.

4.3 Conclusion

Integrating NeRF into our point cloud pipeline provided a foundational framework for 3D reconstruction of the *Temple* dataset. While the initial results did not fully meet our expectations, the process highlighted critical areas for improvement and set the stage for future enhancements. This implementation serves as an educational demonstration of NeRF's capabilities and the complexities involved in achieving accurate 3D reconstructions.

References

[1] O. Faugeras, L. Robert, S. Laveau, G. Csurka, C. Zeller, C. Gauclin, and I. Zoghlami, "3-D reconstruction of urban scenes from image sequences," *Computer Vision and Image Understanding*, vol. 70, no. 2, pp. 156-173, 2004. [Online]. Available: https://doi.org/10.1006/cviu.1998.0665.

[2] J. Wu, Z. Cui, V. S. Sheng, P. Zhao, D. Su, and S. Gong, "A comparative study of SIFT and its variants," *Measurement Science Review*, vol. 13, no. 3, pp. 122-131, 2013. Available: https://intapi.sciendo.com/pdf/10.2478/msr-2013-0021.

[3] K. G. Derpanis, "Overview of the RANSAC algorithm," vol. 4, no. 1, pp. 2-3, 2010. Available: http://www.cs.yorku.ca/ kosta/CompVis_Notes/ransac.pdf.

[4] W. Guilluy, L. Oudre, and A. Beghdadi, "Video stabilization: Overview, challenges and perspectives," *Signal Processing: Image Communication*, vol. 90, p. 116015, 2021. Available: https://www.sciencedirect.com/science/article/abs/pii/S0923596520301697

[5] Z. Zhang, "Camera calibration," in *Computer Vision: A Reference Guide*, Cham: Springer International Publishing, 2021, pp. 130-131. [Online]. Available: https://link.springer.com/referenceworkentry/10.1007/978-3-030-63416-2_164.

[6] B. Rosenhahn, *Pose estimation revisited*, Ph.D. dissertation, 2003. [Online]. Available: https://macau.unikiel.de/servlets/MCRFileNodeServlet/dissertation_derivate_00000842/d842.pdf..

[7] D. Scharstein, "Multi-View Stereo Dataset: Temple of the Dioskouroi," Middlebury College, Mar. 28, 2012. [Online]. Available: https://vision.middlebury.edu/mview/data/