CS436 - Computer Vision - Project Report - Group 2

Muhammad Saad Haroon Zammad Bin Ziyad Khan
GitHub Repo Link

Abstract

The digitization of physical landmarks through 3D reconstruction has become a cornerstone of modern Computer Vision (CV) applications, offering significant potential in fields such as cultural heritage preservation, Augmented Reality (AR) development, and virtual tourism. This project aimed to reconstruct a 3D model of the Brandenburg Gate using CV techniques and deploy it within an AR environment via an Android application. The pipeline utilized feature detection and matching, Structure from Motion (SfM), and Multi-View Stereo (MVS) for 3D reconstruction, followed by deployment through FlutterCube. Despite limitations such as the absence of AR compatibility on the chosen device, the app successfully rendered an interactive 3D model, demonstrating the feasibility of integrating 3D reconstruction and AR for practical applications. The outcomes underline the potential of these technologies to enhance user engagement and accessibility, contributing to advancements in CV and AR domains.

1. Introduction

The digitization of physical landmarks through 3D reconstruction has become a cornerstone of modern Computer Vision (CV) applications. This process holds significant potential in fields such as cultural heritage preservation, Augmented Reality (AR) development, and virtual tourism. The primary goal of this project was to reconstruct a 3D model of **Brandenburg Gate** using CV techniques and subsequently deploy our model in an AR environment on an Android application. The application can then be used to capture photos of any landmark from different angles and reconstruct its 3D version just by using those images. By integrating advanced methodologies, such as Structure from Motion (SfM) and leveraging ARCore for deployment, this project aims to bridge the gap between digital reconstruction and interactive real-world visualization.

The 3D reconstruction pipeline begins with feature detection and matching to identify the corresponding points across multiple images of a landmark. These correspondences are used to estimate camera poses and compute a sparse 3D point cloud, which is further refined into a dense model using multi-view stereo (MVS). Alternatively, photogrammetry-based techniques involve depth estimation and depth map fusion to generate a detailed 3D representation. The final model is then deployed in an Android application using ARCore, allowing users to visualize and interact with the landmark in augmented reality.

This project is significant for several reasons. Firstly, it offers a cost-effective and efficient approach to creating 3D representations of cultural and historical landmarks, which can aid in their digital preservation and accessibility. Secondly, the integration of AR extends the functionality of the 3D model by providing an immersive user experience. For example, tourists can explore landmarks in virtual environments before physically visiting them, and educators can use interactive AR tools to teach history and architecture more effectively. Lastly, the project provides a platform for applying and evaluating state-of-the-art CV techniques in a real-world scenario, fostering innovation and technical proficiency.

Through this work, our aim is to demonstrate the feasibility of combining 3D reconstruction with augmented reality for practical applications. The results of this project will highlight the potential of these technologies in improving user engagement and accessibility, contributing to ongoing advancements in CV and AR.

2. Methodology

The methodology was divided into several key stages to ensure a systematic approach to 3D reconstruction and mobile visualization.

2.1. Set-Up

2.1.1. Environment Configuration

The initial phase of the project required setting up the development environment for the Android app. The app's purpose was to provide a platform for displaying the reconstructed 3D model. We used the skeleton app provided as part of the project resources and made the necessary configurations to ensure compatibility with our devices and project requirements.

The following dependencies were installed:

- **Java 17:** The latest supported version of Java for Gradle compatibility.
- **Gradle 7.5-all.zip:** For building and running the Flutter app.
- **Flutter SDK:** The official Flutter framework was used to develop and run the Android app. The installation followed the official Flutter documentation, ensuring all dependencies, including the Android SDK, were configured.

After completing the installation, we verified the environment setup by running the `flutter doctor` command. This command confirmed that all necessary components were installed and ready for use.

2.1.2. App Instructions

Once the environment was ready, we tested the provided skeleton app by performing the following steps:

1. **Download and Extract:** The `app.zip` file containing the skeleton app was downloaded and extracted into a project folder.
2. **Open Project:** The extracted folder was opened in Visual Studio Code (VS Code).
3. **Connect Device:** The mobile device (POCO X3 GT) was connected to the computer using a USB cable, with USB Debugging enabled in Developer Options.
4. **Run App:** Using the terminal in VS Code, the `flutter run` command was executed. The app was built and installed on the mobile device.
5. **Camera Testing:** Once installed, the app was launched on the device. On pressing the *Open Camera* button, the camera interface appeared, confirming the app's basic functionality.

These steps ensured the proper configuration of the skeleton app and verified that the camera functionality worked as intended. Any issues encountered during this phase were resolved by revisiting the Flutter installation and the Android device debugging setup.

2.1.3. Device-Specific Details

For this deliverable, we used a POCO X3 GT device, which is not ARCore-compatible. Despite this limitation, the skeleton app provided a robust framework for testing the basic functionality of the camera, ensuring that subsequent tasks involving 3D model integration would proceed smoothly.

2.2. Dataset Selection

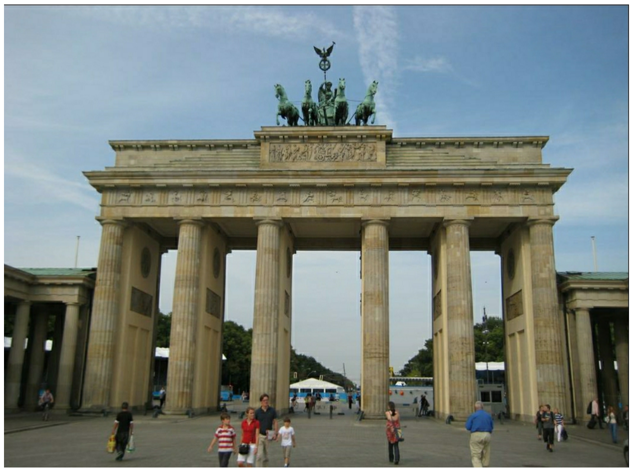

Figure 1. An example image that was used

For our project, we used the Brandenburg Gate dataset, a comprehensive collection of **1,400** images that capture the architectural splendor of this iconic landmark from diverse perspectives. To ensure the quality and relevance of our 3D reconstruction process, we performed a meticulous image selection process.

2.3. Dataset Selection

For our project, we used the Brandenburg Gate dataset, a comprehensive collection of **1,400** images that capture the architectural splendor of this iconic landmark from diverse perspectives. To ensure the quality and relevance of our 3D reconstruction process, we performed a meticulous image selection process.

Initially, all 1,400 images were manually reviewed to identify and eliminate those that were unsuitable for reconstruction. This subset refinement process focused on removing images that were blurred, distorted, or contained excessive visual noise, such as crowds of people or obstructions that obscured key architectural features. From this initial dataset, we curated a final subset of **500** high-quality images.

Particular attention was paid to ensuring diversity within the selected subset. By including a representative sample of photos captured from varying angles and positions, our goal was to achieve comprehensive coverage of the intricate details and structural elements of the Brandenburg Gate. This diverse dataset plays a crucial role in the reconstruction process, as it provides the necessary perspectives to accurately model all facets of the structure, including its façade, columns, and relief sculptures.

Figure 2. An example image that was discarded

2.4. Image Pre-processing

2.4.1. Resizing and Grayscale Conversion

The selected images were resized to **800x600** pixels to reduce computational load while preserving important features. Grayscale conversion simplified feature detection by focusing on structural elements.

2.4.2. Histogram Equalization

We also applied histogram equalization for brightness normalization, ensuring consistent lighting across images and improving contrast. This enhanced the visibility of important features, crucial for accurate 3D reconstruction.

These preprocessing steps optimized the dataset for feature detection and matching, improving both the accuracy and efficiency of the reconstruction pipeline.

2.5. Feature Detection and Matching

2.5.1. Initial Approach with ORB

We initially began the feature detection and matching process using ORB (Oriented FAST and Rotated BRIEF), a fast and efficient algorithm often preferred for real-time applications. However, ORB did not provide satisfactory results for our 3D reconstruction task, as it struggled to reliably detect keypoints in images with significant scale and rotation variations.

2.5.2. Shift to SIFT

As a result, we shifted to the Scale-Invariant Feature Transform (SIFT) algorithm, which is specifically designed to handle scale and rotation invariance. SIFT is more robust in detecting distinctive keypoints under various transformations, making it better suited for tasks that require high accuracy in feature matching, such as 3D reconstruction.

Using SIFT, we detected more reliable keypoints and computed corresponding descriptors that remained consistent across different images, even in the presence of lighting changes or partial occlusions. For feature matching, we employed the brute-force matcher with Lowe's ratio test, which compared descriptors from consecutive images, selecting the best matches based on their distance and filtering out ambiguous correspondences. This feature matching was performed for every successive image pair in the dataset, ensuring that each image was matched with its neighbor.

After the initial matching, we refined the results using RANSAC, which eliminated outliers by estimating a homography matrix and retaining only geometrically consistent matches. Finally, we visualized the best matches between image pairs to assess the quality of the feature detection and matching process. The shift to SIFT, combined with Lowe's ratio test and RANSAC, resulted in a robust set of feature correspondences that significantly improved the accuracy of our 3D reconstruction pipeline.

2.6. Camera Pose Estimation

2.6.1. COLMAP Integration

For this part, we utilized the **COLMAP** repository to extract camera intrinsic parameters and estimate relative camera poses between image pairs. COLMAP is a powerful SfM tool that provides methods to read camera and image data from its binary files, which we leveraged for this task.

We first read the camera parameters from the COLMAP binary file, `cameras.bin`, using the `read_cameras_binary` function provided by the COLMAP Python API. This file contains the intrinsic parameters for each camera used during the image capture process, including the focal length (f_x, f_y) and the principal point coordinates (c_x, c_y). These parameters are essential for constructing the camera matrix, denoted as K, which is used in the camera pose estimation process.

2.6.2. Fundamental and Essential Matrices

For each image in our subset, we matched its camera ID from the `images.bin` file with the corresponding camera parameters in the `cameras.bin` file. Using the camera parameters, we built the camera intrinsic matrix K for each image, which has the following form:

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

This matrix represents the intrinsic properties of the camera, such as the focal length and the location of the optical center.

Once we had the intrinsic matrices K for each image, we proceeded with camera pose estimation between pairs of images. For this, we used the keypoints and descriptors from feature matching to compute the fundamental matrix using the `cv2.findFundamentalMat` function. This matrix relates corresponding points between two images, and from it, we computed the essential matrix E, which encapsulates the camera motion between the two views. The essential matrix was used in the `cv2.recoverPose` function to recover the rotation matrix R and translation vector T, which define the relative camera pose between the two images.

By applying this procedure to each image pair, we were able to estimate the relative camera poses dynamically. These transformations were used to build the global camera poses incrementally, providing the necessary data for the 3D reconstruction of the landmark.

2.7. Linear Triangulation and 3D Point Cloud

In this stage, we built a sparse 3D point cloud by triangulating matched keypoints from consecutive image pairs. For each pair, we used the camera poses and intrinsic matrices to normalize the 2D keypoints and construct projection matrices. Using these, we triangulated the corresponding points into 3D space using the `cv2.triangulatePoints` function and converted the results into Euclidean coordinates. The triangulation process was repeated for each image pair, accumulating the 3D points to form the sparse point cloud. This process is crucial for generating the 3D structure of the scene, laying the groundwork for further detailed reconstructions.

2.8. Mesh Generation and Visualization

2.8.1. Point Cloud to Mesh Conversion

In this stage, we generated a 3D mesh from the sparse point cloud. First, we created a point cloud object and estimated the normals for each point. Then, we applied Poisson Surface Reconstruction to create a smooth triangle mesh from the points. To clean up the mesh, we removed low-density triangles, retaining only the higher-density ones for better quality. Finally, we visualized the mesh by recomputing the vertex normals for improved shading and rendering the mesh in Open3D, ensuring a clear and accurate 3D surface model. This process converts the sparse point cloud into a detailed 3D model for further analysis.

2.9. App Deployment

In the final stage of the project, the reconstructed 3D model was integrated into a mobile application for visualization. Since the chosen device (POCO X3 GT) lacked AR compatibility, we utilized FlutterCube, a lightweight plugin for 3D model rendering in Flutter applications. This allowed us to display the 3D model in a mobile app environment, even without AR capabilities.

2.9.1. FlutterCube Integration

FlutterCube (*$\hat{0}$.1.1*)

was added as a dependency in the Flutter project. This required modifying the `pubspec.yaml` file to include the plugin and creating an `assets` folder for storing the 3D model. The following lines were added to the `pubspec.yaml`:

```
dependencies:
  fluttercube: ^0.1.1

flutter:
  assets:
    - assets/landmark_model.obj
```

Listing 1. Pubspec.yaml Modifications

The 3D model, converted into the `.obj` format, was stored in the `assets` folder as `landmark_model.obj`. FlutterCube was used to render the model within the app, enabling users to interact with it through basic transformations like rotation and zoom.

2.9.2. Implementation Details

The main application logic was updated in the `main.dart` file to integrate FlutterCube. The key code snippet for loading and rendering the model is shown below:

```
import 'package:flutter/material.
    dart';
import 'package:flutter_cube/
    flutter_cube.dart';

void main() {
  runApp(MyApp());
}

class MyApp extends StatelessWidget
    {
  @override
  Widget build(BuildContext context)
       {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(title: Text('
            Landmark Viewer')),
        body: Cube(
          onSceneCreated: (Scene
              scene) {
            scene.world.add(Object(
              fileName: 'assets/
```

```
                            landmark_model.obj
                            ',
                    ));
                },
              ),
            ),
          );
        }
      }
```

Listing 2. FlutterCube Integration Code

The app interface was designed with simplicity, featuring a 3D viewer occupying the main screen. Users could rotate, zoom, and explore the reconstructed 3D model interactively.

2.9.3. Device Testing

The application was tested on a POCO X3 GT. Despite the device's lack of AR compatibility, the app successfully rendered the model, allowing users to visualize the reconstruction. The absence of AR functionality was mitigated by providing interactive controls for 3D exploration.

3. Results

3.1. Feature Matching

We were able to get pretty good results for our feature matching with approximately **80-100** good matches on every image pair. The best feature matching image result is displayed below.

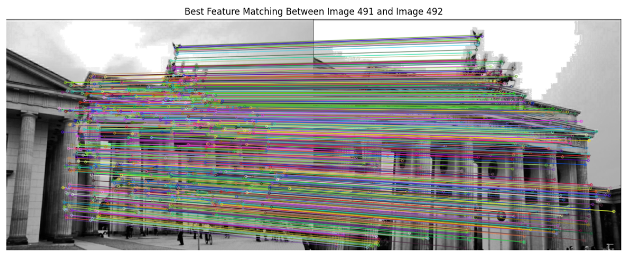

Figure 3. Best Image Pair Feature Matching Result

The app deployment stage culminated in a functional mobile application capable of rendering the reconstructed 3D model. Figure 6 illustrates the app interface, showing the 3D model displayed on the POCO X3 GT.

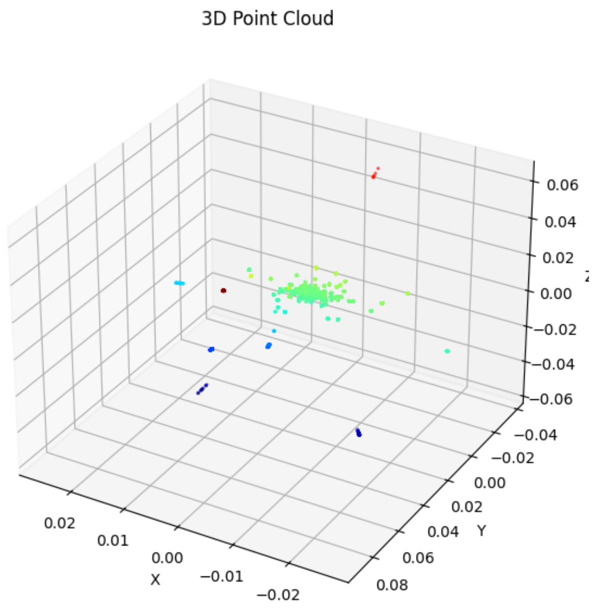

Figure 4. 3D Point Cloud

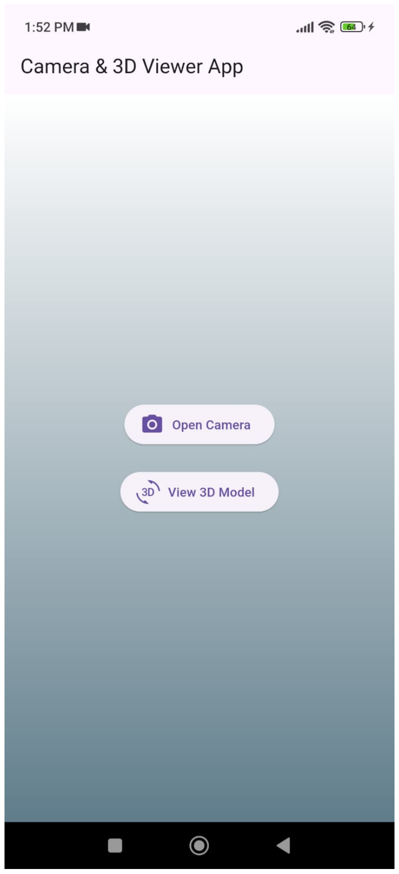

Figure 6. App interface displaying the reconstructed 3D model.

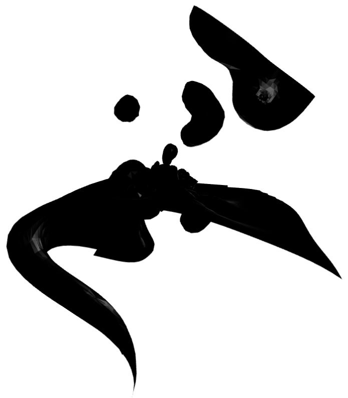

Figure 5. 3D Model

The model was displayed accurately, preserving its geometry and texture, as seen during the desktop visualization stages. Users could interact with the model through touch gestures, including rotation, scaling, and panning.

The successful deployment validated the feasibility of using FlutterCube for rendering 3D models on devices without AR capabilities. The simple integration and smooth performance of the app demonstrated the effectiveness of the chosen approach.

4. Discussions

Our reconstruction is not optimal because the order of images in our sequential pairing did not ensure smooth transitions between views, leading to gaps and inaccuracies in some parts of the model. However, the top of the landmark was reconstructed quite well since it appeared prominently in almost all image pairs, ensuring robust feature matching and triangulation. The lower sections suffered from incomplete geometry due to fewer overlapping keypoints between adjacent pairs. Additionally, texture detail is limited because the sparse reconstruction focuses more on geometry than dense texture mapping. Future improvements could involve reordering images or using global feature matching strategies.

The deployment of the 3D model as a mobile application using FlutterCube allowed us to achieve a key milestone in our project: delivering a functional and interactive platform to visualize reconstructed landmarks. This implementation, while simplified compared to AR-based solutions, provides a practical method for showcasing reconstructed 3D models on devices without AR capabilities.

4.1. Accomplishments

The app successfully rendered the reconstructed 3D model of the Brandenburg Gate, offering users an interactive platform to visualize the structure. By leveraging FlutterCube, a lightweight 3D rendering plugin, we avoided the complexity associated with AR frameworks, enabling deployment on non-AR devices. This approach demonstrated the feasibility of delivering functional 3D reconstruction solutions for education, tourism, and cultural preservation.

4.2. Challenges Faced

1. **AR Plugin Limitations:** The POCO X3 GT's lack of ARCore support restricted the interactive features to basic 3D transformations, such as rotation, zoom, and panning, instead of immersive AR experiences.

2. **Rendering on Non-AR Devices:** FlutterCube provided a functional alternative for visualization but lacked advanced AR features like occlusion and spatial awareness, impacting the quality of interaction.

3. **Feature Matching Accuracy:** While SIFT provided reliable matches, the sequential pairing of images led to gaps in reconstruction, particularly in areas with fewer overlapping keypoints.

4.3. Future Directions

To improve the current implementation, the following directions are proposed:

- **Enhanced Reconstruction Pipelines:** Future experimentation could explore the use of global feature matching strategies instead of sequential matching to ensure smoother transitions and more complete reconstructions. Techniques like graph-based matching or deep learning-based feature detection could further enhance accuracy.

- **Incorporating Dense Texture Mapping:** Sparse reconstructions provide structural accuracy but lack detailed textures. Incorporating dense texture mapping techniques would enhance the realism of the 3D models.

- **ARCore Compatibility:** Deploying the application on ARCore-compatible devices would enable advanced features like real-world interactions, spatial tracking, and environmental awareness, significantly improving user experience.

- **Alternative Frameworks:** Exploring platforms like Unity or Unreal Engine could allow integration of more sophisticated AR functionalities and higher-quality visualizations.

4.4. Proposed Additional Experimentation

- **Dataset Augmentation:** Augmenting the dataset with synthetic images generated through GANs could improve feature detection in underrepresented areas.
- **Evaluation of Reconstruction Metrics:** Future experiments could quantitatively evaluate reconstruction quality using metrics like Structural Similarity Index (SSIM) and point cloud completeness.
- **Hardware-Accelerated Deployment:** Investigating the performance of hardware-accelerated devices, such as those with GPUs or dedicated AR hardware, could provide insights into scalability and optimization.
- **Multiple Landmark Reconstruction:** Extending the pipeline to support multiple landmarks in a single application would demonstrate its generalizability and scalability.

4.5. Lessons Learned

This project highlighted the importance of adaptability in addressing hardware and software limitations. Despite constraints, the combination of 3D reconstruction and mobile visualization showed great potential for applications in tourism, education, and cultural heritage. With incremental improvements and additional experimentation, this work serves as a stepping stone for more advanced AR-integrated applications.

Conclusion

The integration of 3D reconstruction techniques with mobile visualization provided a functional and interactive platform for showcasing reconstructed landmarks. While the project faced challenges such as limited AR compatibility and texture details, the use of FlutterCube enabled the deployment of a lightweight application capable of rendering 3D models on non-AR devices. This approach demonstrated the feasibility of delivering practical solutions for education, tourism, and cultural preservation. Future work could focus on reordering images for better reconstruction accuracy, adopting global feature matching strategies, and deploying the application on ARCore-compatible devices to incorporate immersive AR functionalities. The project serves as a foundation for exploring the intersection of 3D reconstruction and AR in accessible, cost-effective ways.