3D Scene Reconstruction and Virtual Tour

CS436: Computer Vision Fundamentals

Overview

This project challenges you to build a complete Structure from Motion (SfM) and visualization pipeline, inspired by applications like Matterport or Microsoft's original Photosynth. Your goal is to take a collection of 2D photographs of a static scene, captured from various viewpoints, and transform them into an interactive 3D experience. The final application will consist of two primary outputs: a sparse 3D point cloud representing the geometry of the scene, and a "virtual tour" viewer that allows a user to smoothly navigate between the original camera viewpoints. This project will serve as a practical, hands-on implementation of the core theoretical concepts covered in this course, culminating in a visually compelling and interactive final product.

The core of this project is to create a system that can deduce 3D information from 2D images. This is the fundamental promise of multiview geometry. You will implement a full, albeit simplified, SfM pipeline. This involves detecting and matching invariant features across multiple images, using epipolar geometry to recover the initial camera poses and 3D structure from two views, and then incrementally adding more views to expand the reconstruction. A key insight for the final visualization is that you will not be creating a dense, photorealistic 3D model. Instead, you will use the recovered sparse 3D point cloud as a geometric "scaffold" that enables navigation between the high-resolution 2D source images, creating the illusion of moving through a 3D space.

For reference and inspiration, explore some example tours here: MPskin Showcase.

Data Collection

The success of your entire project depends on the quality of your input images. Follow these instructions carefully to capture a good dataset.

1. **Choose a suitable scene.** The best scenes are static (nothing is moving), well-lit, and rich in texture. An object on a cluttered desk, a bookshelf, or a small room with furniture are excellent choices. Avoid large, blank, textureless surfaces like white walls, and avoid reflective or transparent surfaces like windows and mirrors.

2. **Move the camera, do not just rotate.** This is the most important rule. To perceive depth, the algorithm needs to see the scene from different positions. This is called parallax. Do not stand in one spot and simply turn. Instead, take a few steps to the side between each shot. For an object on a table, move the camera in a wide arc around it.

3. **Ensure significant overlap.** Each image should overlap with the previous one by about 60-80%. This means a large part of the scene in one photo should still be visible in the next. This is essential for the feature matcher to find enough corresponding points to link the images together.

4. **Maintain consistent lighting.** Try to capture all your photos under the same lighting conditions. Avoid harsh shadows or bright glare, as these can confuse the feature detector. Diffuse, even lighting is ideal.

5. **Capture sharp images.** Make sure your photos are in focus and free of motion blur. If your phone camera allows, use a "pro" mode to lock the focus and exposure settings so they don't change between shots. A sequence of 15-30 images is a good target for this project.

Tools and Libraries

For this project, it is highly recommended to use Python. The following libraries will be essential:

- **OpenCV:** For all core computer vision tasks, including feature detection/matching, epipolar geometry, triangulation, and PnP.
- **NumPy:** For all numerical and matrix operations.
- **Open3D:** An excellent library for visualizing and debugging your 3D point clouds at each stage.
- **Three.js:** The recommended library for building the final interactive web-based virtual tour.

Implementation Phases

Phase 1: The Two-View Foundation

Your first task is to build the foundational block of any SfM system by implementing a two-view reconstruction pipeline. To begin, you must capture a suitable image pair that provides parallax; do not simply rotate the camera in one spot, but instead physically move it between shots, such as taking one large step to the side while keeping the target object in frame. You will also need to construct the camera's 3x3 intrinsic matrix, K, which is required for all geometric calculations. You will approximate it: assume the principal point (c_x, c_y) is the image center and the focal lengths (f_x, f_y) are equal to the image width. With this data, you will detect and match features between the two images using SIFT or ORB, filtering for high-quality matches with Lowe's ratio test. From these matches, you must estimate the **Essential Matrix**, E, by passing your matched points and the intrinsic matrix K to the 'cv2.findEssentialMat' function with RANSAC. Next, decompose E to find the four possible relative camera poses (rotation R and translation t) using 'cv2.recoverPose'. Your implementation must then triangulate the matched 2D points into 3D space for each of the four possible poses and perform a **cheirality check** to disambiguate the pose, selecting the single '[R—t]' solution that places the majority of triangulated 3D points in front of both cameras. The final output for this phase is a notebook that takes two images and produces a valid sparse 3D point cloud, which should be saved to a '.ply' file for viewing.

Phase 2: Incremental SfM and Refinement

With the two-view system working, you will extend it to handle a sequence of images. The goal of this phase is to incrementally add new views to the existing reconstruction. For each new image, you will first match its 2D features against the features from the previous image that have already been triangulated into 3D map points. This establishes a set of 2D-3D correspondences. Using these correspondences, you will compute the pose of the new camera using a **Perspective-n-Point (PnP)** algorithm, using 'cv2.solvePnPRansac' for a good solution. Once the new camera is localized within the existing map, you will identify new feature matches between it and previous views and triangulate these to create new 3D points, thus growing the map. As you add more views, you will notice that small errors from each PnP step accumulate, causing the reconstruction to "drift" or bend. To correct this, a global optimization technique called **Bundle Adjustment** is typically used. This process simultaneously refines all camera poses and 3D point locations to minimize the overall reprojection error, resulting in a much more accurate and geometrically consistent model.

Phase 3: The Interactive 'Photosynth' Visualization

The final phase is to create the virtual tour application. You will build a viewer that leverages the refined camera poses and the sparse point cloud. The visualization starts by displaying the first source image, as if the user is looking through that camera. To enable navigation, you will construct a **view graph**,

where camera poses are nodes and edges connect cameras that share many visible 3D points. When a user interacts with the view (e.g., by clicking), your application will determine the best nearby camera to navigate to based on the view graph. The transition between the current camera's image and the next camera's image should be a smooth animation. This is achieved by interpolating the camera's 3D position (using linear interpolation, **lerp**) and orientation (using spherical linear interpolation, **slerp**) from the start pose to the end pose, while simultaneously cross-fading between the two images. During this animated transition, you can render the sparse point cloud to provide a sense of 3D motion and structure.

Coding Practices and Project Structure

Professional projects require well-structured and maintainable code. While Jupyter notebooks are excellent for experimentation and visualization, they are not suitable for implementing a multi-stage pipeline. **You are required to structure your main logic in Python (`.py`) files.** A notebook may be used for your weekly submission reports and for showing your results for whole working pipeline, but it should import and call functions from your structured Python codebase.

Your code should be modular, with clear and well-documented functions for each major step of the pipeline (e.g., `find_matches()`, `estimate_essential_matrix()`, etc.). Proper function naming, comments, and logical organization will be part of the evaluation.

Your GitHub repository should reflect professional engineering practices. It must contain a detailed `README.md` file with clear, step-by-step instructions on how to:

- Set up the environment and install dependencies,
- Run your code and reproduce your results,
- Visualize the outputs of each stage (feature matching, reconstruction, and visualization).

Weekly Milestones

- **Week 1: Setup & Feature Matching.** Set up your project environment, collect your full image dataset, and implement a script to find and visualize filtered feature matches between two images.
- **Week 2: Two-View Reconstruction.** Implement the full pipeline to compute the Essential matrix, recover camera pose, and triangulate 3D points from an image pair. Your submission is a script that outputs a valid '.ply' point cloud.
- **Week 3: Multi-View SfM & Refinement.** Extend your pipeline to incrementally add your full image sequence using PnP. After observing the drift, apply a refinement step. Submit a refined multi-view point cloud.
- **Week 4: Interactive Visualization.** Build the 'Photosynth-style' virtual tour using a graphics library like Three.js. The viewer must allow navigation between camera poses with smooth, animated transitions. Your submission is a functional prototype of this viewer.
- **Week 5: Finalize and Submit.** Complete and format all your working. Submit a report of your work along with complete code as per instructions. You may be asked to explain your workings to a TA or instructor

Final Submission

Your final submission will be a single PDF report of 4-5 pages. The report must detail your methodology for each phase, present your results (including visualizations of feature matches, point clouds

before/after refinement, and your final tour), and include a discussion of the challenges and outcomes. **The abstract of your report must contain a single, clickable link to the public GitHub repository containing your project.** This repository represents your full submission and must contain all source code, a short demonstration video (2-3 minutes) of your interactive tour, the image dataset you used, and a comprehensive ‘README.md‘ file with clear instructions for running your entire pipeline.