AIM 836 - Assignment 2 Report - 3D Rendering using NeRF and Gaussian Splatting

Siddharth Kothari (IMT2021019)

December 17, 2024

Contents

1 Objective

To create a 3D rendering of an object using the following steps:

1. Capture a set of 2D images using a mobile camera.
2. Generate a point cloud using Structure-from-Motion (SfM).
3. Render the 3D object using Neural Radiance Fields (NeRF).
4. Render the 3D object using Gaussian Splatting.

This report documents the methodology, tools, and experimental outcomes, including comparisons between NeRF and Gaussian Splatting and their outputs.

2 Problem Statement Description, Motivation and Implementation

2.1 Problem Statement

This project investigates advanced computational methods for 3D reconstruction and rendering. It focuses on transforming a physical object into a realistic 3D model using a combination of image acquisition, reconstruction, and rendering approaches. The process begins by capturing multiple images of the object, which are processed to create a 3D point cloud using Structure-from-Motion (SfM). The generated point cloud is then transformed into a 3D model using two cutting-edge techniques: Neural Radiance Fields (NeRF) and Gaussian Splatting. The project aims not only to implement these methods but also to assess their comparative advantages. By analyzing NeRF and Gaussian Splatting, the study seeks to highlight their distinctions in computational demands, visual fidelity, and practical applications.

2.2 Implementation

The implementation followed the workflow provided by the repository https://github.com/mapillary/OpenSfM

2.2.1 Step 1: Capture 2D Images

The initial and most critical step in 3D reconstruction is to acquire a series of images that capture the object from various angles. The accuracy of the reconstruction process heavily depends on the quality and variety of these images.

Image Capture Process: The object (in this case, a pair of coffee mugs) was positioned on a stable surface under consistent lighting, and a mobile phone camera was used to record a video of the object. To ensure all angles were captured, the object remained stationary while the camera was moved around it.

Frames were extracted from the video at regular intervals using FFmpeg, providing a dataset with diverse perspectives while minimizing redundancy. The following command was used to extract the frames:

```
ffmpeg -i ./input_video.mp4 -qscale:v 1 -qmin 1 -vf fps=2 %04d.jpg
```

This command extracts two frames per second from the video, offering sufficient image overlap for effective feature matching while keeping the total number of frames manageable for processing.

Why This Approach Was Taken: Using a video instead of capturing individual photos manually ensured smoother transitions between images and reduced the chance of missing angles. While a turntable would have provided ideal rotation, manual rotation proved effective with careful execution.

Two of the images obtained from the above process are shown in Figures 1a and 1b -

(a) (b)

Figure 1: Some sample images captured from the video

Challenges and Solutions:

- **Lighting Inconsistencies**: Uneven illumination or shadows could disrupt the reconstruction process. This was addressed by positioning the object in a consistently lit environment.
- **Motion Blur**: Sudden movements during the object rotation could cause blurring, complicating feature matching. To prevent this, the video was recorded slowly and with steady motion.
- **Full Coverage**: Achieving a complete 360-degree view of the object required careful planning. Additional attention was given to capturing extra images of intricate or detailed regions.

2.2.2 Step 2: Point Cloud Generation Using SfM

After preparing the 2D images, the next step was to create a sparse 3D point cloud using Structure-from-Motion (SfM). SfM is a widely used computer vision technique that reconstructs the 3D structure of a scene and estimates camera positions based solely on overlapping 2D images.

How SfM Works: Structure-from-Motion involves multiple stages, including feature detection, camera pose estimation, and 3D triangulation. Below is an outline of the key steps:

1. **Feature Detection and Matching**: This step identifies distinct, repeatable features such as corners, edges, or textured patterns within the images. Algorithms like SIFT (Scale-Invariant Feature Transform) or AKAZE are typically used for this purpose. These features are then matched across overlapping image pairs using their descriptors, ensuring consistent recognition of the same real-world point across multiple images.

   For instance, if a corner or edge of the object appears in three images from different perspectives, the algorithm identifies and matches that point across these images.

2. **Camera Pose Estimation**: Once the features are matched, the relative positions and orientations of the cameras that captured the images are determined. This is achieved through geometric constraints, such as epipolar geometry, and refined using optimization techniques like Bundle Adjustment. This ensures that the 3D points align accurately with the viewpoints.

3. **Point Cloud Reconstruction**: Using the matched features and camera poses, the algorithm triangulates the 3D coordinates of points in the scene. This produces a sparse 3D point cloud, which consists of spatial points representing the object's surface. Although sparse, this point cloud serves as a critical intermediate step toward generating a denser model.

Using OpenSfM for Point Cloud Generation: OpenSfM, an open-source library tailored for SfM, was utilized to streamline the process. It offers a complete pipeline encompassing feature detection, feature matching, camera pose estimation, and point cloud generation.

Key Features of OpenSfM: OpenSfM simplifies the SfM workflow while providing advanced configurability for expert users. Key features include:

- Compatibility with various feature detectors such as SIFT, SURF, and AKAZE, ensuring versatility across different image types.
- Automatic extraction of intrinsic camera parameters from EXIF metadata, minimizing the need for manual calibration.
- Capability to generate sparse or dense point clouds based on project requirements.

Output of SfM: The primary result of this stage is a sparse 3D point cloud that represents the object's geometry. It captures the object's key structural details, while the estimated camera positions indicate the viewpoints during reconstruction. This sparse representation becomes the foundation for subsequent rendering steps.

Outlined below are the specific steps performed:

1. **Initialize the Project**: Maintain the directory structure as ./data/<project folder name>/images, with all extracted images from FFmpeg placed inside the images directory.
2. **Execute the OpenSfM Pipeline**: OpenSfM provides an intuitive command-line interface to run the SfM pipeline. The following command was used to execute all steps sequentially:

```
bin/opensfm_run_all data/<project folder name>
```

   To view the outputs, initialize the viewer by running:

```
./viewer/node_modules.sh
```

3. **Visualize Outputs**: Start the server with the command:

```
python3 viewer/server.py
```

   Then navigate to *http://localhost:8080* and select the reconstruction file to view.

Additionally, dense point clouds can be found at: *./data/<project folder name>/undistorted/depthmaps/merged.ply*

Challenges and Mitigations Although the SfM process is robust, several challenges arose during implementation, which were mitigated as follows:

- **Featureless Surfaces**: Surfaces with minimal texture or reflectivity posed difficulties for feature detection and matching. To overcome this, textured objects were prioritized, and artificial markers were applied to featureless areas when required.
- **Insufficient Overlap**: Reconstruction accuracy depends heavily on significant overlap between consecutive images. To ensure adequate overlap (60–70%), the image capture process was meticulously planned, and densely sampled frames were used.
- **Scaling Issues**: SfM generates a relative reconstruction without an absolute scale. To resolve this, known dimensions or reference objects were incorporated during post-processing for accurate scaling of the reconstruction.

2.2.3 Step 3: 3D Rendering Using NeRF

Neural Radiance Fields (NeRF) is a technique that renders 3D scenes from 2D images by learning a volumetric representation of the scene. This method encodes the scene as a continuous neural network that outputs color and density values for points in space, enabling the generation of novel views of the scene.

NeRF Overview: NeRF reconstructs a 3D scene by training a neural network on a series of 2D images, accompanied by their respective camera parameters. The core idea is to represent the 3D space as a neural radiance field that maps 3D coordinates and viewing directions to RGB colors and volumetric densities.

- **Input:** A set of 2D images captured from known viewpoints, along with camera intrinsic and extrinsic parameters.
- **Output:** A neural representation capable of synthesizing photorealistic images from arbitrary viewpoints.

Implementation Steps: The following steps were taken during the implementation:

1. **Data Preparation:** The images and the reconstruction.json file obtained from OpenSfM were directly used as input by placing them in the appropriate directory structure.
2. **Training:** The 'nerf-reconstruction.ipynb' file in the nerf directory of the repository was executed to train the model and save it, ensuring the dataset was correctly set up.
3. **Evaluation:** The 'testingAndMeshExtraction.ipynb' file was run to generate the final rendered 3D output.

2.2.4 Step 4: 3D Rendering Using Gaussian Splatting

Gaussian Splatting is an alternative approach to 3D rendering that uses a collection of 3D Gaussians to represent the scene. Unlike NeRF, it directly reconstructs and renders the object using spatially distributed Gaussian primitives, offering faster and more efficient performance in some cases.

Gaussian Splatting Overview: Gaussian Splatting works by representing the 3D scene as a set of overlapping Gaussian primitives in space. Each Gaussian captures local color and density information, enabling efficient and smooth rendering. In contrast to NeRF, which learns a neural representation of the scene, Gaussian Splatting utilizes geometric primitives for rendering.

- **Input:** A 3D point cloud (generated by SfM) and associated color information.
- **Output:** A rendered 3D object or scene generated from the Gaussian primitives.

Implementation Steps: The implementation followed the workflow provided by the repository https://github.com/inuex35/360-gaussian-splatting:

1. **Input Preparation:** The images and the reconstruction.json file from OpenSfM were used as inputs for Gaussian Splatting.
2. **Training:** The model was trained using the prepared inputs with the following command:

```
python train.py -s data/<directory name> --panorama
```

The rendered outputs were saved in the outputs directory. However, the point clouds generated initially did not include color, so the following command was used to convert them into colored point clouds:

```
python3 convert_ply.py
```

Challenges and Observations Gaussian Splatting performed particularly well with objects that had rich geometric details but struggled with finer textures. Key observations include:

- The method was faster than NeRF, especially during the rendering phase, due to its lightweight representation.
- The quality of the rendered output was heavily influenced by the density and accuracy of the input point cloud, with sparse or noisy point clouds yielding suboptimal results.

3 Results and Outputs

3.1 Results from OpenSfM

The Dense Point Cloud generated by OpenSfM can be seen in Figure 2.

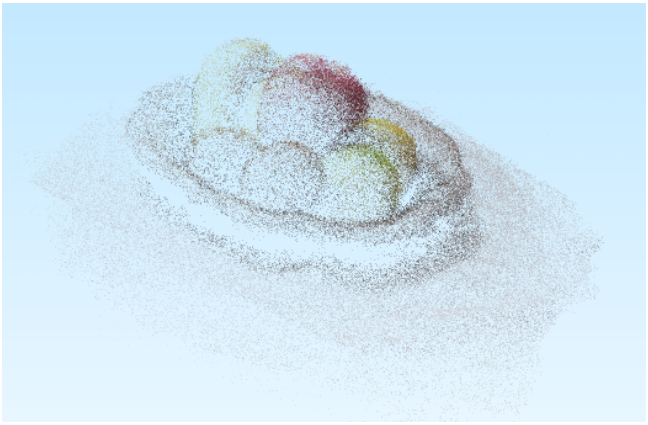
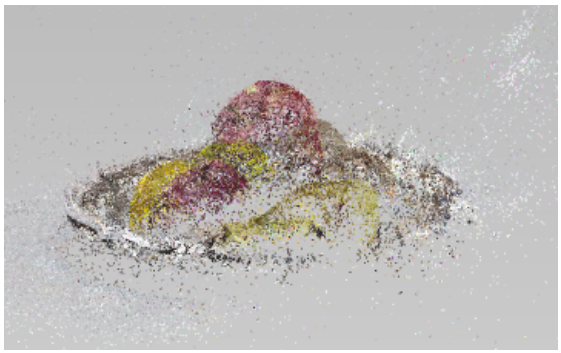


Figure 2: Output of OpenSfM

3.2 Results from Gaussian Splatting

The outputs from Gaussian Splatting can be observed in Figure 3.

(a)

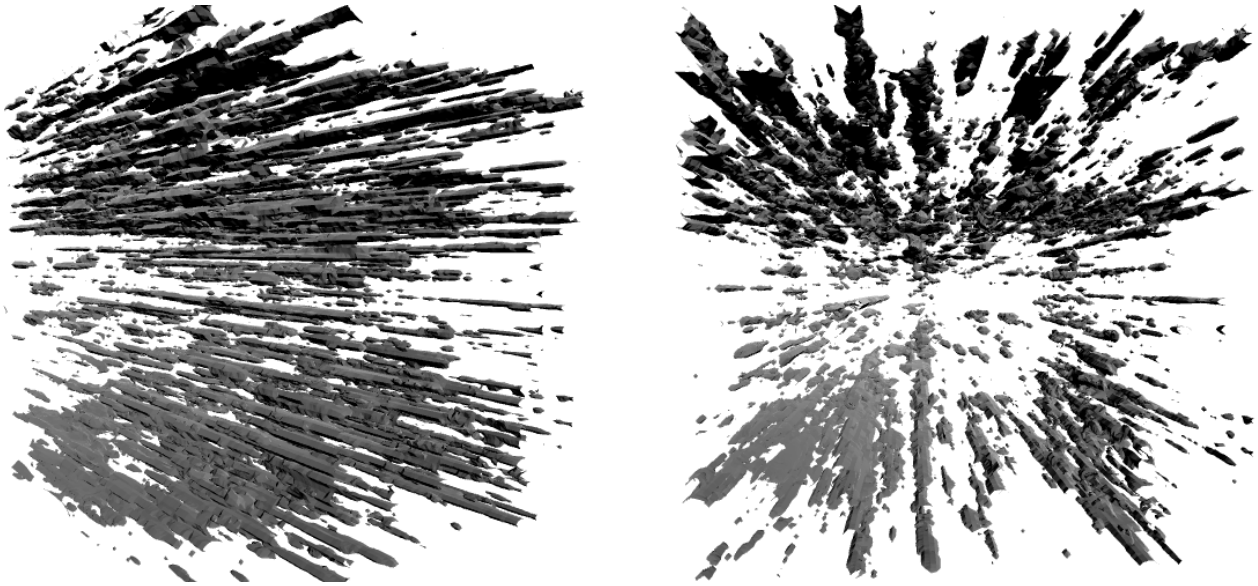


(b)

Figure 3: Outputs from Gaussian Splatting

3.3 Results from NeRF

The outputs from NeRF can be observed in Figure 4.



(a) (b)

Figure 4: Outputs from Gaussian Splatting

4 Comparison Between NeRF and Gaussian Splatting

Key Differences

- **Representation:** NeRF uses a continuous neural network that maps 3D coordinates to color and density, while Gaussian Splatting employs discrete 3D Gaussians as geometric primitives.
- **Computational Efficiency:** NeRF is computationally intensive and requires substantial training time. In contrast, Gaussian Splatting is faster during both setup and rendering.
- **Quality of Output:** NeRF excels at rendering photorealistic images, particularly for complex lighting and detailed textures. Gaussian Splatting produces high-quality results but may struggle with areas having sparse input data.
- **Input Requirements:** NeRF requires well-prepared datasets with accurate camera parameters, whereas Gaussian Splatting depends more heavily on the quality of the input point cloud.

Project Observations

- NeRF delivered superior results for scenes with intricate textures and complex lighting, but its long training time made it less practical for rapid prototyping.
- Gaussian Splatting was much faster and required fewer computational resources, making it a better option for applications where speed is critical.
- The choice between NeRF and Gaussian Splatting depends on the specific requirements — NeRF is best for photorealistic rendering, while Gaussian Splatting offers a good balance between speed and quality.

5 Relevant Links

The output ply files, along with the zip files containing the codes can be found at the following link - https://iiitbac-my.sharepoint.com/:f:/g/personal/siddharth_kothari_iiitb_ac_in/EgSHXonnCIhDvo7EOcXc0VQBTCvbSgsLLZv0qqO07BfUMA?e=NJY2Wi
The GitHub repository for the project can be accessed at: https://github.com/siddharth-kothari9403/Point_Cloud_Rendering.