Interactive Visual Debugger for 3D Reconstructions

Keita AZUMA
ETH Zürich
D-INFK
kazuma@ethz.ch

Aron Tse Rong CHOO
ETH Zürich
D-MAVT
archoo@ethz.ch

Yang LI
ETH Zürich
D-INFK
yangli1@student.ethz.ch

Jixian TANG
UZH
IfI
jixian.tang@uzh.ch

Zador PATAKI
ETH Zürich
Computer Vision and Geometry Group
zador.pataki@inf.ethz.ch

Jiaqi CHEN
ETH Zürich
Computer Vision and Geometry Group
jiaqi.chen@inf.ethz.ch

1. Introduction

Understanding complex 3D reconstruction pipelines, such as COLMAP's Structure-from-Motion (SfM), often requires advanced expertise. Despite its widespread use in computer vision research, the intricacies of COLMAP's incremental process, particularly how image registration and reconstruction stages affect outcomes, remain challenging to understand intuitively. To bridge this gap, we developed an interactive Mixed Reality (MR) application for Meta Quest 3. By enabling real-time visualization and user-controlled interaction with COLMAP's pipeline, this project provides researchers with a "visual debugger," fostering deeper insights into SfM processes. Our code can be found at https://github.com/keitaiazuma33/MixedReality23, while our demo video can be found here.

2. Related Works

2.1. COLMAP Overview

COLMAP [2] is a state-of-the-art Structure-from-Motion (SfM) pipeline widely used in computer vision research to generate ground truth 3D reconstructions, including sparse point clouds and camera poses, from large-scale unordered image sets. Figure 1 shows the COLMAP pipeline, including the preprocessing stage, which can be broken down into five primary components:

1. **Feature Extraction and Matching:** Keypoints are extracted from images using algorithms such as Scale-Invariant Feature Transform (SIFT). These image features are stored in the COLMAP database to find feature correspondences between images based on descriptor similarity.
2. **Image Registration:** Camera positions and orientations are estimated by solving the geometric relationships between images using the matched feature correspondences.
3. **Triangulation:** 3D points are incrementally added to the reconstruction by triangulating using newly registered images and their corresponding feature matches.
4. **Bundle Adjustment:** Nonlinear optimization is applied to jointly refine camera parameters and 3D points by minimizing the overall reprojection error.
5. **Outlier Filtering:** Points or matches that deviate significantly are iteratively identified and removed to improve the reconstruction's overall accuracy. This process is done iteratively until convergence.

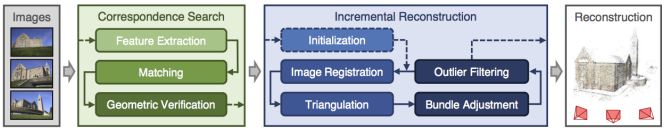

Figure 1. **COLMAP's Incremental Structure-from-Motion pipeline.**

3. Methods

COLMAP is a widely used method for generating high-quality 3D reconstructions from a pre-defined set of images. However, the resulting point cloud and camera poses are the outcomes of the complex interaction between many pipeline components, making it challenging for computer vision researchers to fully understand the specific role and impact of individual components. This challenge becomes particularly significant when modifying the COLMAP pipeline or

attempting to intuitively diagnose failure cases within the current pipeline.

Moreover, COLMAP is inherently designed as a closed-loop system that processes an entire image set in a single run. Consequently, it is not well-suited for fine-grained adjustments or detailed observation of intermediate stages within the pipeline. For researchers, this poses difficulties in exploring the system's behavior, such as how incremental image registration impacts the reconstruction quality or how adjustments to specific stages influence the final results.

Our Mixed Reality (MR) visualizer addresses these challenges by enabling an interactive exploration of each pipeline step within a 3D virtual environment. This tool provides users with the ability to dynamically adjust and observe the effects of modifications at various stages, thus fostering a more intuitive understanding of the complex reconstruction process of COLMAP.

3.1. Backend Implementation

Our approach involved extending COLMAP's Python API (`pycolmap` [1] [2]) to facilitate incremental and interactive operations, overcoming limitations inherent in its default pipeline. The primary contributions of our implementation are as follows.

1. **Pipeline Breakpoints:** By introducing thread-based synchronous control, our system allows users to pause COLMAP's execution in between components. This enables interactive inspection of intermediate reconstruction results before proceeding to the next step in the pipeline.
2. **Dynamic Image Registration:** Functionality was developed to dynamically add or remove images from the reconstruction process and reregister them as needed. This deviates from COLMAP's default design, which assumes a fixed and static set of input images.

These modifications enabled the dynamic generation and logging of key outputs, including point clouds, camera poses, and reconstruction metadata. These outputs were seamlessly integrated into the front end for visualization and user interaction. In the following, we detail the implementation of each major functionality.

3.1.1 Pipeline Breakpoints

Pipeline breakpoints were realized through a multi-threaded architecture that synchronizes interactions between COLMAP and the client application. Specifically, the system leverages condition variables to temporarily suspend COLMAP's operations while waiting for user input from the client. This mechanism ensures that reconstruction progress is dynamically adjustable without rerunning COLMAP.

The server maintains two primary threads: one for handling client requests and another for executing COLMAP processes. These two threads share a single condition variable and a shared data structure for information exchange between them.

We designed three primary tasks: the first is registering new images ("New Image", task name **`n`**), the second is deregistering previously registered images ("De-register Image", task name **`r`**), and the third is re-registering deregistered images ("Re-register Image", task name **`a`**).

Tasks **`n`** and **`a`** additionally allow users to specify whether to execute the full COLMAP pipeline (indicated in `full_pipeline`).

- If `full_pipeline` is specified, the user-specified images (one image for task **`n`** or multiple images for task **`a`**) are added to the reconstruction in the order determined by COLMAP's `find_next_image` function.
- If `full_pipeline` is not specified, the COLMAP server hands over the condition variable to the response server after completing each pipeline component (breakpoint). This allows the user to decide whether to skip the next pipeline component or proceed with it. This approach gives the user flexible control over the execution of:
  - (1) triangulation,
  - (2) local bundle adjustment
  - (3) global bundle adjustment

  while also enabling them to observe the impact on the point cloud and camera poses as each component is executed. Once the user provides input—either to skip the next component or to process the remaining pipeline at once—the server resumes the COLMAP process from its paused state.

  (Note: To enhance system flexibility, if COLMAP decides to skip global bundle adjustment due to insufficient model growth from local BA, the user will still be given the option to execute Global BA, along with an explanation. Whether global BA is required by COLMAP will be shown as a `user_message` in the GUI.)

3.1.2 Dynamic Image Registration

Dynamic image registration was implemented by directly interacting with COLMAP's low-level functions via `pycolmap`. Our approach allows us to manipulate the image database and reconstruction state dynamically during runtime. The implementation involved the following steps:

- **Image Addition and Removal:** We modified the pipeline to accommodate user requests for adding new images or removing previously registered ones. Newly added images are immediately processed through feature extraction and matching, followed by registration into the reconstruction. Conversely, removed images are deregistered using the `deregister_image` function provided by `pycolmap`, and their associated 3D points are removed from the point cloud.

- **Re-registration Mechanism:** For images that are re-registered after deregistration, they are added to the current reconstruction. This essentially modifies the order in which COLMAP adds images, allowing the user to reorder the images as desired. Additionally, users can choose to re-register multiple images simultaneously. In this case, they can either select a preferred order for re-registration or let COLMAP determine the optimal order based on the selected images (`find_next_image`).

- **Logging and Metadata Updates:** To maintain synchronization with the frontend, all changes to the reconstruction state (`cameras.txt`, `images.txt`, `points3D.txt`, and `reconstruction.ply`) are logged and transmitted to the client in real-time.

These enhancements allow users to iteratively refine and experiment with the reconstruction process, offering a level of control and flexibility that was previously unavailable in the original COLMAP implementation. By connecting this with the frontend, we enable researchers to interact with complex 3D reconstruction pipelines.

3.2. Frontend Development

The frontend provides real-time visualization of COLMAP's outputs, including sparse point clouds and estimated camera poses. Utilizing Unity's `MeshTopology.Points`, point clouds are rendered directly from COLMAP's data. Additionally, camera poses are visualized as interactive cubes, allowing users to observe their spatial orientations and positions within the 3D environment.

By implementing real-time visualization, interactive controls, and precise coordinate transformations, the frontend empowers researchers to intuitively explore and debug COLMAP's incremental pipeline stages, thereby enhancing their understanding and facilitating advanced experimentation.

The transformation matrices of the cameras are created using quaternions and translations, whose signs are flipped as described above. At the same time, the script reads the transformation matrix of a reference object (the point cloud) in Unity. It then sets cameras as children of the reference object in Unity. The child-setting process occurs immediately after loading the point cloud and setting its position and rotation parameters to zero in Unity. As shown in Fig. **??**, this ensures the visualized cameras move and rotate along with the point cloud.

Figure 2. **A simple computer recontruction**

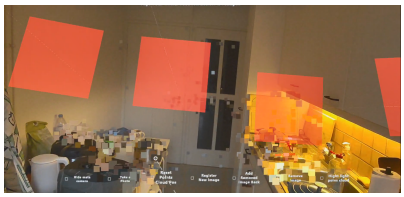

Figure 3. **Visualization of colmap's camera poses**

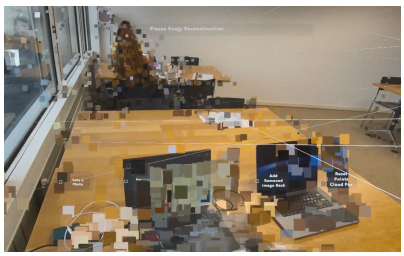

Figure 4. **Visualization of a reconstructed room**

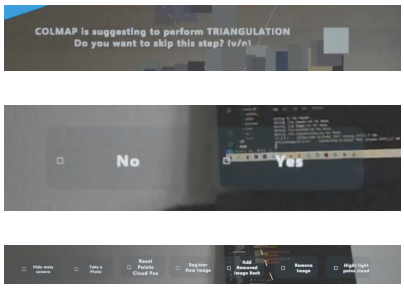

Figure 5. **Pipeline visualization of 3D reconstruction and Unity integration GUI**

3.2.1 Implemented Functionalities

Users can interact with the SfM pipeline through a suite of interactive controls:

- **Image Management**: Users can start the reconstruction process by importing images, adding or removing images in real-time, and observing the immediate effects on the reconstruction (see Figure 2).
- **Pipeline Navigation**: The application allows users to step through different stages of the SfM pipeline, including image registration, triangulation, local bundle adjustment, and global bundle adjustment. Each stage's output can be individually examined, and the user maintains the control over the pipeline execution by deciding which stages to proceed or not As shown in Fig. 5.
- **Selective Visualization Controls**: Users can independently toggle the visibility of different data elements: Meta Quest 3's recorded camera poses can be hidden to clearly showcase COLMAP's estimated poses, while the point cloud reconstruction can be selectively highlighted. These visualization controls enable effective visual comparison between the estimated geometry and the real-world object.

3.2.2 Coordinate Transformation and Spatial Alignment

Accurate spatial alignment between COLMAP's reconstruction and Unity's MR coordinate system presents two key challenges in mixed reality visualization. The first challenge arises from the inherent coordinate system mismatch: COLMAP reconstructs the scene in an arbitrary coordinate frame, while Quest 3 operates in Unity's MR world space. Direct transformation between these spaces results in inconsistent spatial relationships, manifesting as arbitrary rotations and translations in the visualization. We address this by computing a robust initial transformation hypothesis that not only aligns the coordinate frames but also maintains spatial consistency during subsequent point cloud updates, effectively preventing spatial drift and discontinuities. The second challenge stems from COLMAP's inherent scale ambiguity [2] in structure-from-motion reconstruction. We resolve this by establishing correspondences between COLMAP's estimated camera poses and Quest 3's recorded MR world poses, enabling us to compute a reliable scale factor. This dual-solution approach ensures stable and geometrically consistent mixed reality visualization, bridging the gap between computer vision reconstruction and mixed reality rendering.

3.2.3 Transformation Matrix

The transformation from COLMAP's to Unity's coordinate system is represented by:

$$\mathbf{T} = \mathbf{S} \cdot \mathbf{R} \cdot \mathbf{C} + \mathbf{t}$$

Where: $\mathbf{T}$ is the transformed position in Unity's system, $\mathbf{S}$ is the scaling matrix, $\mathbf{R}$ is the rotation matrix, $\mathbf{C}$ represents the original COLMAP coordinates, and $\mathbf{t}$ is the translation vector.

Scaling Matrix To match units between COLMAP and Unity:

$$\mathbf{S} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & s_z \end{bmatrix}$$

where s_x, s_y, and s_z are scaling factors along the respective axes.

Rotation Matrix Addressing orientation differences via Euler angles:

$$\mathbf{R} = \mathbf{R}_z(\theta_z) \cdot \mathbf{R}_y(\theta_y) \cdot \mathbf{R}_x(\theta_x)$$

Translation Vector Positioning the model within Unity's scene:

$$\mathbf{t} = \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}$$

3.2.4 Alignment Procedure

To align the coordinate systems, we took these steps:

1. **Calculate Scale Factor**:

$$s = \frac{\sqrt{\sum_{i=1}^{N} \|\mathbf{p}_i^{\text{Unity}}\|^2}}{\sqrt{\sum_{i=1}^{N} \|\mathbf{p}_i^{\text{COLMAP}}\|^2}}$$

2. **Determine Rotation Matrix** via Singular Value Decomposition (SVD) following Umeyama [3]:

$$H = P_{\text{COLMAP}}^T P_{\text{Unity}}, \quad H = U\Sigma V^T$$

$$R = \begin{cases} VU^T & \text{if } \det(VU^T) > 0 \\ V \cdot \text{diag}(1, 1, -1) \cdot U^T & \text{otherwise} \end{cases}$$

3. **Compute Translation Vector**:

$$\mathbf{T} = \mathbf{c}_{\text{Unity}} - sR\mathbf{c}_{\text{COLMAP}}$$

where $\mathbf{c}_{\text{Unity}}$ and $\mathbf{c}_{\text{COLMAP}}$ are the centroids of Unity and COLMAP point sets, respectively.

4. **Apply Transformation** to each COLMAP point:

$$\mathbf{p}_i^{\text{Transformed}} = sR\mathbf{p}_i^{\text{COLMAP}} + \mathbf{T}$$

These steps ensure precise alignment of the point cloud and camera poses within Unity, maintaining accurate spatial relationships and orientations for effective MR visualization, as illustrated in Fig. 4.

3.2.5 Integration with Backend

The frontend and backend communicate via a client-server architecture. The frontend (Meta Quest 3) sends user commands and receives incremental reconstruction data, enabling real-time updates.

3.2.6 Visualize Camera Poses

The `ShowCamerasAsChildren` script transforms camera poses from COLMAP into Unity's coordinate system. This involves:

Reading Backend Outputs. The script reads quaternions (QW, QX, QY, QZ) and translations (TX, TY, TZ) for cameras from backend outputs.

Coordinate System Transformation. To align cameras from the backend outputs with Unity's coordinate system, we flip the signs of QX, QY, TX, and TY.

Visualizations. The transformation matrices of the cameras are created using quaternions and translations, whose signs are flipped as described above. At the same time, the script reads the transformation matrix of a reference object (the point cloud) in Unity. It then sets cameras as children of the reference object in Unity. The child-setting process occurs immediately after loading the point cloud and setting its position and rotation parameters to zero in Unity. As shown in Fig. 3, this ensures the visualized cameras move and rotate along with the point cloud.

3.2.7 Load Point Cloud Without Button Implementation

The `LoadPointCloud` script manages point cloud visualization and interaction: Real-time point cloud loading and updating. Dynamic transformation and positioning. Backend synchronization for consistent visualization

3.3. System Architecture

The application adopts a client-server architecture as seen in Figure 6, where:

Client (Meta Quest 3): Captures and sends image data, receives visualization data, and renders 3D reconstructions.

Server: Processes requests, executes COLMAP operations, and sends incremental results back to the client.

Figure 6. **Data communication flow between COLMAP and the Unity application.** The server employs thread-based synchronous processing to implement breakpoints within the pipeline. Reconstruction results up to the breakpoints are visualized interactively in the Client (Unity app).

4. Evaluation

We opted to use a number of qualitative and quantitative methods to gauge the effectiveness of our app, as seen below:

Qualitative Methods

1. Think-Aloud Protocol: Users were asked to verbalize their thoughts while navigating through the app;

2. Observational Study: Users were observed while they interacted with the app, and moments of confusion and hesitation were noted;

3. Post-test Survey: Users were asked to rate the ease of use and perceived usefulness of the app on a Likert-scale of 1 to 5, with 1 being the lowest and 5 being the highest.

Quantitative Metrics

Task: Capture 5 images on the headset and visualize first 3D reconstruction.

- Completion time (in seconds);
- Number of mis-clicks.

4.1. User Study

Cognizant of the fact that app development is an iterative process, we sought to produce a minimum viable product (MVP) as soon as possible so that user feedback can be collected to improve future iterations. The efforts outlined in the previous sections culminated in an MVP app that was ready for a user study on Demo Day. A summary of the data collected is outlined below:

4.1.1 Qualitative Results

Thanks to the Think-Aloud Protocol, crucial information regarding the usability and ease of navigation through our app was gathered. The first user of our app, Professor Marc Pollefeys, remarked that the buttons in the app were placed too far from the center of the user's field of view which caused difficulties in reading the text in the button. Furthermore, another user pointed out interaction challenges,

such as difficulty in moving and re-sizing the visualized point cloud. Lastly, one user commented that there were excessive and redundant buttons, such as the button for loading the next reconstruction after taking new images (which could have just been automatically loaded), and also the button for reconstruction options that were not possible in the current breakpoint.

Through the observational study, it was noted that users were momentarily perplexed by the unchanged reconstruction after taking an action that should have changed the reconstruction (because they were unaware that they had to click the "Load Reconstruction" button for the point cloud to update). Furthermore, slight displeasure was observed when the new point cloud was loaded at a location different than the previous point cloud, causing the user to have to readjust their focus onto the new point cloud location. On the flip side, users were visibly delighted upon visualizing a point cloud that truly represented the scene in front of them.

The post-test survey responses were collected for each of our 8 users. Using a Likert-scale from 1 to 5, ease of use received an average score of 2.1 and perceived usefulness received an average score of 3.8.

4.1.2 Quantitative Results

The time that it took for users to complete the task of capturing 5 images on the app and visualizing the resultant 3D reconstruction was collected and compared to that of an expert user (ourselves). The number of mis-clicks were also recorded. Across the 8 users in the study, users took an average of 47.8 seconds with a standard deviation of 9.2 seconds to complete the task, along with an average number of mis-clicks of 1.8. By contrast, the expert user took an average of 25.4 seconds with a standard deviation of 3.5 seconds to complete the task. The expert user had no mis-clicks.

From this data, we see a 1.9x slowdown of a first-time user compared to the expert user, which suggests that the app in its MVP version can still be made more user-friendly.

5. Discussion

The aforementioned user feedback proved to be instrumental in guiding the improvements that can be seen in the latest version of our app (demo video) to make it more user-friendly. Most notably, the buttons are now placed in a more central location of the user's field of view, redundant buttons were removed, and point clouds are now initialized in metric scale and overlaid onto the actual scene without unnecessary movements.

5.1. Limitations and Future Work

Firstly, the Meta Quest 3's onboard processing capacity limited real-time SfM execution, necessitating reliance on server-based computation. Exploring GPU-accelerated computation or optimizing server-client communication could reduce latency and enhance real-time capabilities.

Secondly, our implementation focused on sparse point clouds, omitting dense reconstruction stages due to computational complexity. Extending the visualizer to support dense point clouds and texture mapping would provide a more comprehensive understanding of SfM outcomes.

Lastly, there is currently limited support for advanced user inputs, such as custom feature detection and matching algorithms or triangulation thresholds. Allowing users to specify conditions such as minimum triangulation angles or feature extraction strategies would enable more detailed experimentation.

5.2. Conclusion

This project has demonstrated the feasibility of using MR to aid researchers in understanding the intricacies of SfM processes. By enabling researchers to interact with and explore reconstruction pipelines in a step-by-step manner, the visualizer promotes a more intuitive understanding of complex 3D modeling workflows. However, its reliance on external computation highlights the importance of optimizing future implementations for scalability and portability.

References

[1] P. E. Sarlin, C. Cadena, R. Siegwart, and M. Dymczyk. From coarse to fine: Robust hierarchical localization at large scale. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 12716–12725. IEEE, 2019. 2

[2] J. L. Schonberger and J. M. Frahm. Structure-from-motion revisited. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4104–4113. IEEE, 2016. 1, 2, 4

[3] S. Umeyama. Least-squares estimation of transformation parameters between two point patterns. *IEEE TPAMI*, 13(4):376–380, 1991. 4