# CMPUT 428 Project Proposal: Photorealistic 3D Scenes using SfM with Gaussian Splatting

University of Alberta

John Tyler
jetyler@ualberta.ca
1704463

Shijie Bu
sbu1@ualberta.ca
1720680

## Abstract

Obtaining 3D models from uncalibrated image sequences is a topic that we have honed in on in CMPUT 428. Generating reliable 3D models from images could revolutionize industries such as computer video game development, or animated movie production. For example, computer video games or computer simulation programs for workplace training [5] often desire to emulate immersive graphics for the user especially with the introduction of virtual reality. While realistic graphics are only one component of creating a feeling of immersion for users [6], they act as an important stepping stone towards creating an experience that feels real. However, to create photorealistic scenes that attempt to capture an environment in the real world is not an easy task, as demonstrated in Lab 3 of CMPUT 428. Attempting to reconstruct 3D models from projective video sequences using the affine matrix factorization methods produced inaccurate results. These poor results were further exacerbated by the challenge of initializing tracked points that would produce numerically stable results, and would not fail during the video sequence. In order to capture an accurate digital representation of a 3D object, the typical approach also involves using a meshing algorithm to find a suitable representation of the object using polygons or triangles, and then applying traditional rendering techniques to visualize the model with simulated lighting. The latest technique for visualizing such photorealistic environments is Gaussian Splatting [1], where images of the scene alongside a 3D point cloud are used to train Gaussian distributions for color and intensity of each point in the scene, and are blended seamlessly using rasterization. Gaussian Splatting must be trained offline, but provides an alternative to the traditional approach of meshing and texturing a 3D point cloud, and then applying traditional rendering techniques. 3D scenes created from Gaussian Splats can be viewed from novel camera positions, and the trained Gaussian radiance fields already encode the lighting of the real world scene. In our project, we will apply the fundamental Structure-from-Motion (SfM) techniques seen in class [4] to produce the 3D point clouds that will create photorealistic 3D environments composed of Gaussian radiance fields. We build our own SfM algorithm, taking inspiration from COLMAP [2, 3], a state-of-the-art SfM pipeline capable of creating accurate 3D point clouds from general image sequences. We visualize our 3D point clouds and render the results with Gaussian Splatting, and compare them to results from COLMAP.

# 1. Introduction

Recovering 3D information from 2D images is the heart of computer vision. Structure-from-Motion (SfM) refers to capturing 3D geometry from a series of images or video frames, and is a topic of great importance in CMPUT 428, as well as Part IV of Hartley and Zisserman [4]. SfM is related to photogrammetry, but poses a more general challenge, as the camera positions are not known in advance. In class we saw the derivation of the affine factorization method to simultaneously recover the camera positions and 3D point coordinates from a matrix of perfect point correspondences between images capturing the same object. Later, in Lab 3, we put this into practice, and saw that the affine factorization method can produce striking results for images captured from an orthographic camera, but was not sufficient to provide accurate results for images captured from a projective camera.

However, existing software such as COLMAP [2,3], have found success in the general SfM task. Using an incremental reconstruction algorithm, rather than a one-shot factorization, COLMAP is able to produce accurate 3D point clouds from general, unordered images, captured from different cameras. The emphasis of our work is to capture the underlying framework that enables COLMAP to succeed, and show that elementary techniques presented in Hartley and Zisserman [4] and the CMPUT 428 lectures provide a strong basis to tackle the SfM problem.

To demonstrate the impact of our SfM algorithm, we use Gaussian Splatting [1], a novel rendering technique, to visualize the reconstructed scene to highlight one potential application of an accessible SfM algorithm. Gaussian Splatting uses a 3D point cloud, a set of images, and correspondences of those 3D points in the provided images, to create a densified point cloud. Each point in the densified cloud is instead a 3D gaussian, with volume, orientation, color, and opacity, with each of these attributes trained by an iterative algorithm to produce a photorealistic rendering that matches the input images of the scene. The rendering of the scene can be visualized in real time, from novel views, after training. Gaussian Splatting works entirely as a cloud of points or Gaussians, rather than as a meshed object typically used in 3D modelling and computer video games. We compare the quality of rendered scenes that use our SfM algorithm for the initial 3D point cloud with those that use COLMAP [2,3].

To motivate our project, we consider the importance of realistic computer graphics in computer video games, and simulations. Using SfM on real world scenes is a direct way to capture a real environment for a computer simulation. An advantage of using Gaussian Splatting to render scenes captured from real world images rather than meshing and texturing a 3D model is the reduced amount of human modifications to the result. In Lab 3 we saw that even with an accurate 3D point cloud it is not easy to recover a meshed 3D model, and would require manual post-processing in software such as MeshLab [7]. We visualize our Gaussian Splatting results in the 3D modelling software Unreal Engine [8] to demonstrate the readiness of Gaussian Splatting as a medium for photorealism in currently popular software for 3D game and simulator development.

# 2. Our Work

## 2.1 Bob's Contributions: The Core Algorithm

I was responsible for implementing the core SfM reconstruction pipeline, which started with just getting the first two images to work. Once those two were selected, I computed the essential matrix and recovered the relative camera pose using OpenCV's **findEssentialMat** and **recoverPose**. I manually set the first camera's pose to identity, and the second one's pose came directly from the decomposition.

Right after recovering the pose, I triangulated the initial set of 3D points using OpenCV's **triangulatePoints** but only kept those with acceptable reprojection error and positive depth. This gave us our first sparse reconstruction, just a few points, but it is where we started to build the entire scene.

To expand the reconstruction, I wrote the logic to register new views. For each new image, if enough 2D-3D correspondences exist (based on previously triangulated points), I estimate the camera pose using **solvePnPRansac**. This step turned out to be a little fragile at times: it requires good enough 3D structure to lock onto, or else PnP just fails. But once it works, the camera pose is added, and we immediately triangulate new points by pairing this new image with already registered ones.

I also implemented both global and local bundle adjustment using **Ceres Solver**. The global BA adjusts all registered camera poses and all 3D points jointly, while the local BA focuses on a sliding window of recent views to refine the solution without slowing everything down. I used an angle-axis representation for camera rotations and applied a robust loss (**Huber**) to guard against outliers.

One thing that caused more trouble than expected was getting camera intrinsics. I wrote a utility that reads focal length and equivalent 35mm focal length from **EXIF** metadata. From that, I calculate the physical sensor width and derive the focal length in pixels. If the **EXIF** data is missing or malformed, I fall back to hardcoded intrinsics.

Lastly, I built a basic GUI using **GLFW** and **ImGui**. It lets users select an image folder, trigger the SfM pipeline, and quit the application. It's simple but very effective for testing and demoing the system.

## 2.2 John's Contributions: Feature Detection, Correspondences, and Algorithm Output

To identify the key features in each image that can be matched between views from different camera positions, I utilized the GPU accelerated **Scale Invariant Feature Tracking (SIFT)** [9] algorithm provided by OpenCV. SIFT locates features such as edges and corners of 3D objects that can be detected in images of different rotations, or scale (distance from the point) with respect to the view that initially captured the feature. Features of this kind allow for robust matching of keypoints captured in views that are nearby one another. SIFT performed particularly well on the dataset used for the majority of the evaluation of our SfM algorithm [10],

as it detected many points on the highly textured brick building from University of North Carolina. I used **Lowe's Ratio Test** to ensure the point correspondences were accurate. This allows for the measurement of confidence for a feature match between a pair of images by comparing the similarity between the first best correspondence, and the second best correspondence, to ensure there is limited ambiguity in the point correspondences we use.

The SIFT function is the entry point to OpenCV's feature matching API, returning key points present in each image. I was next tasked with finding descriptors that describe the relation and correspondences of these key points between other views provided in the set of images. I decided that a brute force search, provided by OpenCV's **BFMatcher**, would be best suited to determine the presence of any feature correspondences between two images.

To efficiently determine the set of all feature correspondences, and in turn determine all of the unique key points present on the 3D structure of the building, I was left with the task of selecting which pairs of images to invoke the brute force search upon. We propose four selection algorithms that determine which pairs of images we invoked the brute force search upon during the matching phase of the SfM algorithm.

Figure 1. Matching methods. From top to bottom, the brute force matching method (demonstrated on one image), the sequential matching method, the window matching method, and the anchor matching method.

The initial, and most robust matching method is the **brute force matching method**. This matching method searches for feature correspondences in every pair of input images. This allows for arbitrary image ordering, but in order to demonstrate the advantages over the following matching methods, we compare its performance in the setting where images are provided in sequential order captured from movement circling an object. The brute force matching method allows for feature correspondences between views captured many images apart from one another, allowing for long range correspondences between distant views, and loops around the object to be detected and indicated to correspond to the same 3D point.

To reduce the number of pairs of images we match, and explore a setting with realistic constraints, assuming sequential images are provided as input, further matching methods were explored. The **sequential matching method** only matches a view to the previous and subsequent view. This matching method misses out on long range correspondences, and leads to many inaccuracies upon views looping around an object to a previously seen portion of the 3D structure. To improve the accuracy of the sequential matching method and prevent failures in the new view registration phase of the SfM algorithm, the **window matching method** uses a sliding window to match each view with a small group of surrounding views. To account for long range feature correspondences, the window matching algorithm was modified to periodically **"anchor"** a view. This selected view to anchor will be matched with a broad window of surrounding views, to ensure a robust view to register can always be found periodically.

Finally, to prepare our reconstruction for the Gaussian Splatting rendering and to standardize our program with other SfM programs, I formatted the output of the program to match that of COLMAP. This involved packaging information regarding the cameras used to capture each image, the image file names corresponding to each view and the 3D points visible in each view, and the list of 3D points with their coordinates, RGB color value, and the views they are present in. After creating three text files matching the output format of COLMAP, they are converted into binary files for fast processing by the Gaussian Splatting training algorithm.

# 3. Experiments

To evaluate the performance of our Structure-from-Motion (SfM) algorithm, we compare our results using three metrics:

1. SfM computation time
2. 3D point cloud accuracy
3. Rendering quality/compatibility with Gaussian Splatting (section 3.2 and 3.3 only)

The run time of the SfM algorithm is indicative of the resource requirements of the algorithm, and reveals how accessible it is to obtain a 3D reconstruction from uncalibrated images for those that cannot access the latest hardware or cloud computing resources. All experiments were run on a gaming PC with a NVIDIA GeForce RTX 2070 SUPER GPU, and an AMD Ryzen 7 3700X CPU.

The 3D point cloud accuracy will be assessed visually. The SfM experiments were mostly conducted on a set of images provided by the authors of COLMAP, that capture a building on the campus of University of North Carolina [10]. Robustly evaluating the accuracy of a 3D point cloud poses a challenge, as creating a ground truth 3D point cloud would involve collecting real world measurements for each point in the sparse point cloud produced by an execution of the SfM algorithm with given parameters, or input images.

However, we saw an alternative evaluation method in Lab 3, where measurements between the critical, geometry defining points of the hotel building were measured as a ground truth. These points often correspond to edges or corners on a building, which are also the key points that are likely to be included in an SfM algorithm's reconstruction. The distance between two corresponding points in the reconstruction were used to scale the entire 3D point cloud to provide an absolute scale calibration that should match the ground truth measurements. We could then find the rest of the corresponding pairs of points, and measure the difference between the real world distance between them, and the distance between them in the absolute scale calibrated point cloud. This would allow us to evaluate the accuracy of our 3D point cloud with only a limited number of real world measurements provided in advance. We forgo this evaluation method, as collecting real world measurements is prohibited by time constraints, and measurements would not be provided for the images collected from online resources.

The rendering quality of the Gaussian Splatting result is also assessed visually. Although the authors of the Gaussian Splatting paper provide methods for evaluating the accuracy of the rendering by using a hold-out set of test images [1], and computing renderings from the same view as those held-out images, we only consider the visually apparent artifacts present in the renderings. This is to place an emphasis on the most substantial impacts on the rendering results caused by varying degrees of accuracy in the 3D point cloud used to initialize the Gaussian Splatting procedure.

## 3.1 Matching Algorithm Results

One of the largest components of the run time for our SfM algorithm is the matching stage. Before performing any steps to compute 3D structure, we must compute the complete set

(to the best of our capabilities) of feature correspondences between one image, and the other images in the input. By searching for a key point identified by the feature detection algorithm (in our case, SIFT [9]) across the other images, we can determine there is a given 3D point that is visible in multiple views. These correspondences can later be used for triangulation to compute 3D coordinates for that point during the two view initialization, or new view registration stage.

To match features between any selected pair of images and obtain feature correspondences, we must perform a brute force search over one image, for each keypoint from the other image. This computation is extremely expensive, especially with high resolution images, even with the excellent speedup provided by OpenCV's GPU accelerated matching function. Our variable for optimizing performance is not the matching itself, but instead the selection of which pairs of views to match with.

We compare each of the proposed feature matching algorithms from Section 2.2, comparing the runtime and accuracy of the 3D point cloud when using each method.

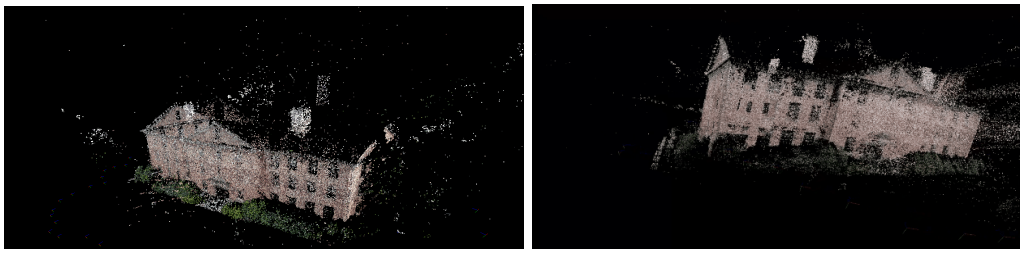

Figure 2. 3D point clouds computed with the brute force matching method (left), and with the sequential matching method (right).

In Figure 2, the 3D point cloud result of the brute force matching method is compared with the sequential matching method. First note that due to the run time (to be discussed in more detail soon) of the brute force matching method, we could not generate the point cloud on the full set of images, as we did for the sequential matching method. Instead, the set of images was limited to 37 images of the front of the building, compared to 85 images.

Both the matching techniques successfully capture the rigid 3D structure of the building, indicated by the precise corners wrapping around the left and right side of the building, visualized in both point clouds. Both point clouds also capture the 3D structure of the front face of the building, where the stairs and center of the building juts out toward the viewer.

However, the sequential matching method produces a point cloud with a fatal error. The back right corner of the building fails to be accurately reconstructed, caused by many failed view registration attempts. The right side of the building appears warped, and is sent seemingly to infinity, as the algorithm fails to estimate the position of the camera with a registered view, and does not have enough point correspondences to any successfully registered view to triangulate points accurately.

Recall that the sequential matching method only matches a view with the view preceding it, and the successive view. This means that even a single failure in the next view registration

phase of the algorithm can cause catastrophic failure. If one view fails to register, which may be due to a small number of feature correspondences with previously registered views, it causes a domino effect, guaranteeing the next views to be likely to fail, as there will be no view that is already registered for them to match with. There will not be enough points to use to estimate the camera pose accurately. The next view that attempts to register will fail as well, as the sequential matching method will not have matched this image with the image that is two views before it. This will be a key component of the matching methods that will be brought up when comparing the remaining matching methods. As identified by the authors who created COLMAP, "Choosing the next best view is critical, as every decision impacts the remaining reconstruction. A single bad decision may lead to a cascade of camera mis-registrations and faulty triangulations" [1]. When we reduce the maximum number of already registered views that a given view could have matched with, we reduce the possibility of an accurate registration.

The reduced number of matches comes with a significant improvement in runtime. The brute force matching algorithm matches every pair of images, meaning the time complexity with respect to the number of images, n, is O(n^2). On the other hand, the sequential matching method performs only O(n) many matches. The sequential matching result with 85 images took ~20 minutes to complete. To contrast, the brute force matching algorithm with 37 images took ~8 hours to complete. This made it infeasible to attempt the brute force matching method on the full set of images, to capture the full geometry of the building - limiting the effectiveness of the brute force matching method.

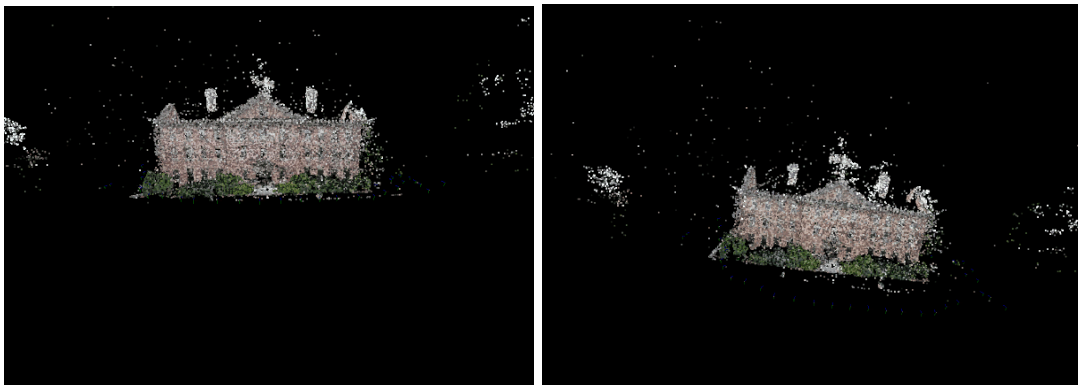

Figure 3: 3D point clouds computed with the window with anchor matching method (left), and with the window matching method (right).

In Figure 3 we compare the two improved O(n) matching methods. On the left we see the result computed using the window with anchor matching method, and on the right the result using the simple window matching method, which took ~2 hours and 30 minutes and ~2 hours, respectively, to complete on the same set of 37 images used by the brute force matching method in Figure 1. These methods are able to provide a degree of accuracy comparable to the brute force matching method, while using only a fraction of the computational time.

While saving computation time, these sliding window matching methods produce slightly less accurate results. We see that both of the improved methods successfully capture the rigid 3D structure of the building without any erroneous warping. However, these methods are susceptible to large numbers of outliers, as seen by the sparse clusters of points that don't correspond to any objects from the series of images. Furthermore, these methods produced especially dense point clouds, as shown in Figure 4. This is likely due to a single keypoint on the building's surface being matched between multiple pairs of images within a window, but not all of them - leading to multiple 3D points being assigned to the same feature correspondence.

Figure 4. Density of the 3D point cloud from the window matching methods.

One apparent advantage of the SfM accuracy of the window methods compared to the brute force matching method appears in the accurate depiction of the planarity of the surface of the building. In Figure 2, we observed there to be what appeared like layers to the front of the building, where there were two planes of points that both appeared like the front face of the building, but one was hidden behind the first plane. This inaccuracy is no longer present when we use the window matching method.

## 3.2 COLMAP vs. Ours

COLMAP was able to produce the SfM results from the set of 37 images in less than 1 hour.

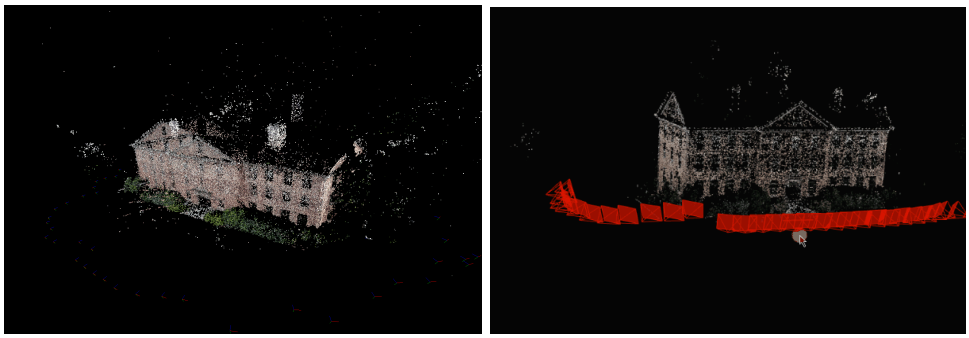

Figure 5. 3D point clouds computed by our SfM algorithm (left) and COLMAP (right).

One particular improvement in COLMAP's 3D point cloud over our own is the planarity of the front of the building. The point cloud produced by our SfM algorithm creates multiple erroneous layers of the front face of the building. Additionally, COLMAP produces a much sparser point cloud, likely due to fewer erroneous or incomplete feature correspondences.

Figure 6. 3D Gaussian Splatting trained using 3D point cloud from our algorithm (left) and COLMAP (right).

Figure 6 depicts the rendered results after 7,000 iterations of the Gaussian Splatting training after using the SfM point cloud produced by our algorithm, and COLMAP. Both renderings capture the texture of the building and trees, as well as the lighting of the scene. The reflectance in the windows of the building appears accurate in both renderings. The details of the doorway and the railing on the steps appear fairly accurately in both renderings.

Both of the renderings suffer from significant artifacts, likely partially due to the reduced training time of only 7,000 iterations rather than 30,000. Both renderings poorly capture the surrounding scene, such as the sidewalk in front of the building, or the sky. Additionally, the roof of the building - which was mostly occluded in the images used to produce the SfM and Gaussian Splatting results - appears entirely see through from the novel views.

Our result falls short of the rendering produced with SfM points from COLMAP in a few places. The stairs of the building were never accurately reconstructed by our SfM algorithm,

leading to a large blurry result upon rendering. Additionally, the shrubbery in front of the building was captured more accurately by COLMAP's point cloud, visible by the large spherical shapes of the bushes in COLMAP's point cloud. This led to a more accurate depiction of the bushes meeting the building. The unseen views of the bushes from above were filled in with a blur when rendered with COLMAP's point cloud, where our SfM results led to a fragmented rendering, where the Gaussians were not able to fill in the unseen geometry of the bushes.

## 3.3 Other 3D Scenes

Figure 7. 3D Gaussian Splatting results (left) and the 3D point cloud (right) from 23 images of a clinic.

Figure 7 reveals the SfM results and Gaussian Splatting rendering from a set of 23 images of an Insight medical clinic. The SfM procedure took ~6 hours using the brute force matching method. We see that the SfM found a large number of points of the textured parking lot visible in most of the images. However, the point cloud representing the building itself appears very sparse, due to the low texture material of the walls, except for the red brick tower in the center. The point cloud accurately captures the edges of the building, converting the geometry and rectangular structure of the building.

The rendering reveals that even with a dense point cloud representing the ground in front of the building, the densification process in the Gaussian Splatting training was not enough to avoid fragmentation in the rendering of the parking lot. Additionally, the logo on the building's tower appears illegible, with not enough 3D points in the SfM geometry to capture the geometry of the sign. The Gaussians could not accurately fit the shape of the sign, leading to the fine details of the text being poorly visible.

Figure 8. 3D Gaussian Splatting results (left) and the 3D point cloud (right) from 25 images of John's messy desk.

In Figure 8 we see the performance of the SfM algorithm and the Gaussian Splatting results on a set of images capturing a messy desk. The SfM reconstruction took ~6 hours using the brute force matching method. We first acknowledge the incompleteness of the 3D point cloud. Although there was no lack of objects on the desk, the point cloud only appears to capture the computer monitor, keyboard, and tower. This is likely due to the matte texture of various books, sticky notes, etc., present in the scene, and their incompatibility with SIFT. This reveals the importance of the presence of highly textured surfaces that the feature detection algorithm can use for matching. Furthermore, the SfM results were very inaccurate, as the planar monitor screen was detected to have a gaping pit in the center where the rest of the screen appears to be.

The Gaussian Splatting rendering of the scene reveals the impact of a poor SfM point cloud initialization on the final result. Sticky notes appear floating in space, oddly stretched Gaussians in front of the monitor blur and obscure the view, and objects such as the chair in front of the desk appear totally incomprehensible.

# 4. Conclusion

Throughout this project, we implemented a complete incremental Structure-from-Motion (SfM) pipeline from scratch, inspired by COLMAP, which enabled us to deeply understand the mechanics behind multi-view geometry. Each stage, from camera initialization to view registration and bundle adjustment, was hand-coded and fine-tuned, allowing for full control and customization of the reconstruction process.

**Comparison with COLMAP**

When benchmarked against COLMAP, our implementation shares core structural stages such as essential matrix recovery, triangulation, and bundle adjustment. But unlike COLMAP's fully automated black-box approach, our system offers transparency at every step, in a simpler way, making it especially valuable for debugging, experimentation, and educational use. While COLMAP achieves higher reconstruction density and robustness on large datasets, our system achieves comparable results on controlled sequences and provides clearer insight into intermediate states such as match maps, triangulated points, and camera poses.

**Potential Extensions**

If given more time, we would extend the system to support global SfM, loop closure detection, and robust methods on outlier filtering. Integrating advanced feature descriptors, GPU-accelerated matching with FLANN or Faiss, and dynamic scene filtering would significantly boost the robustness and scalability of the pipeline. Additional GUI functionality, such as real-time 3D visualization and pose graph editing, could further improve usability (in our early Python implementation, this was completed, but as the work switched to C++, this feature was deprecated due to additional complexity).

**Significance and Real-World Applications**

This project matters because 3D reconstruction is the backbone of many real-world technologies, like AR/VR, robotics, autonomous driving, and virtual exhibitions. When machines or users need to understand the shape of the world around them, SfM is one of the key tools that makes it possible.

We didn't just use existing tools—we built the entire pipeline ourselves. That gave us full control and a much deeper understanding of how the system works. This is especially important in real applications, where things don't always go as planned and you need flexibility and accuracy.

What makes this even more meaningful is how it connects to the idea of immersion. Studies like the one by Cummings and Bailenson (2015) have shown that what really makes an experience feel "real" isn't just flashy graphics—it's how well the system responds to your movements and reflects the real environment. Our SfM system aims to support that kind of

presence by creating accurate, dynamic models of real spaces. Whether it's guiding a robot, preserving a heritage site, or building a VR training space, this kind of work helps bring digital systems closer to how humans perceive and move through the world.

## 5. References


[1] 3D Gaussian Splatting for Real-Time Radiance Field Rendering
https://repo-sam.inria.fr/fungraph/3d-gaussian-splatting/

[2] Structure-from-Motion Revisited
https://openaccess.thecvf.com/content_cvpr_2016/papers/Schonberger_Structure-From-Motion_Revisited_CVPR_2016_paper.pdf

[3] COLMAP
https://github.com/colmap

[4] Multiple View Geometry in Computer Vision
https://www.cambridge.org/core/books/multiple-view-geometry-in-computer-vision/0B6F289C78B2B23F596CAA76D3D43F7A

[5] UPS Virtual Reality Training
https://about.ups.com/ca/en/our-stories/innovation-driven/virtual-reality-helping-to-create-safety-for-ups-drivers.html

[6] How Immersive Is Enough? A Meta-Analysis of the Effect of Immersive Technology on User Presence
https://vhil.stanford.edu/sites/g/files/sbiybj29011/files/media/file/cummings-mp-how-immersive.pdf

[7] MeshLab
https://www.meshlab.net/

[8] Unreal Engine
https://www.unrealengine.com/en-US

[9] Distinctive Image Features from Scale-Invariant Keypoints
https://www.cs.ubc.ca/~lowe/papers/ijcv04.pdf

[10] South Building at UNC Chapel Hill
https://colmap.github.io/datasets.html