3D Reconstruction: Structure from Motion and Multi-view Stereo

Team Tangram: **Lu Li, Yuren Pang**

May 12, 2020

Abstract

Building 3D models has a wide range of applications: computer vision, robot navigation, 3D mapping, 3D printing, geomorphology, and cultural heritage archival and preservation. Generating a 3D model from a sequence of images is a very useful approach to build a 3D model because it is much cheaper and accessible than techniques such as 3D scanners. [7]

In this paper, we explore tools to recover 3D model from 2D images using a sequence of Structure from Motion (SfM), Multi View Stereo (MVS), Meshing, and Texturing techniques. Given a set of images, SfM estimates the camera parameters and positions and then generates a set of sparse point clouds. [17] The MVS algorithm then takes in the sparse point cloud, and builds a dense reconstruction iteratively from the sparse points. [5] Then, we apply the meshing and texturing techniques to recover a complete 3D model.

The goal of this paper is 1) to understand the ideas behind state-of-the-art SfM and MVS algorithms; and 2) to examine different open source SfM and MVS pipelines that implements these algorithms. We would like to test how these pipelines work, given input image datasets of various quality.

1 Introduction

1.1 Previous Work

The basic 3D reconstruction algorithms can be described as "given a set of photographs of an object or a scene, estimate the most likely 3D shape that explains those photographs, under the assumptions of known materials, viewpoints, and lighting conditions". [6] A large part of the recent success of MVS is due to the success of the underlying Structure from Motion

(SfM) algorithms that compute the camera parameters from the input images. This enables users to reconstruct 3D models without having prior knowledge of the camera calibration and thus is a highly convenient and cost effective.

There are several open source SfM pipelines and MVS pipelines and there has been studies evaluating some of the SfM pipelines. For instance, Bianco et al. conducted research to test a few SfM pipelines using data sets with ground truth 3D models and thus evaluate each pipeline using metric scores. [2] However, given the limited amount of data sets with ground truth models, the researchers only experimented with object-level reconstructions but not scene-level or crowded scene level.

We extend the previous evaluation by performing state-of-the-art 3D reconstruction pipelines on a variety of pictures and scenes. These pipelines include VisualSfM, Theia, COLMAP, OpenMVG, OpenMVS, MVE, and combinations of the six pipelines. In this project, we experiment with the pipelines using datasets of objects, scenes, and crowded scenes and provide **visual** comparisons in terms of the sparse point cloud, dense point cloud, and textured surface reconstruction results.

We start with basic introductions of SfM and MVS algorithms. Then, we describe our datasets, along with a brief introduction for each pipeline we tested. Next, we present our experiment results visually, followed by a discussion of our findings. Lastly, we conclude our paper and propose some further directions.

2 Structrue from Motion (SfM)

Structure from motion (SfM) is a photogrammetric technique for obtaining topographic data from digital imagery. In short, this process takes a set of 2D images as input and outputs the camera calibration and sparse point cloud in 3D space. The math behind this involves a linear algebra proof. [16]

First of all, we define the image points as $\{(x_{fp}, y_{fp}|f = 1, ..., F; p = 1, ..., P\}$, where F is frame (picture) and P is points in each frame. (x, y) are the locations of the feature points on image.

The mean of these feature points:

$$a_f = \frac{1}{P}\sum_{p=1}^{P} x_p, \quad b_f = \frac{1}{P}\sum_{p=1}^{P} y_p$$

After that, we can get the normalized points $\tilde{x_{fP}}, \tilde{y_{fP}}$ from above

$$\tilde{x_{fP}} = u_{fP} - a_{fP}, \quad \tilde{y_{fP}} = u_{fP} - b_{fP}$$

Our goal is to translate the 2D points to a 3D world, but we can reversely take the orthographic projection from $S_p = (X'_p, Y'_p, Z_p)$ to (X_p, Y_p). We note that i, j, k are unit vectors along X, Y, Z.

$$x_{fP} = i_f^T(S_p - t_f), \quad y_{fP} = j_f^T(S_p - t_f), \quad k_f = i_f \times j_f$$

Therefore,

$$\tilde{x_{fP}} = x_{fP} - a_f = i_f^T(S_p - t_f) - \frac{1}{P}\sum_{p=1}^{P} x_p = i_f^T(S_p - t_f) - \frac{1}{P}\sum_{q=1}^{P} i_f^T(S_q - t_f)$$

$$= i_f^T(s_p - \frac{1}{P}\sum_{q=1}^{P} S_q)$$

If the origin of world is at the centroid of object points, the second term is 0. Therefore,

$$\tilde{x_{fP}} = i_f^T S_P, \quad \tilde{y_{fP}} = j_f^T S_P$$

We can formulate a matrix W, the registered measurement matrix.

$$\tilde{W} = \begin{bmatrix} \tilde{U} \\ \tilde{V} \end{bmatrix} = \begin{bmatrix} \tilde{u_{11}} & \dots & \tilde{u_{1p}} \\ \vdots & & \\ \tilde{u_{F1}} & \dots & \tilde{u_{FP}} \\ \tilde{v_{11}} & \dots & \tilde{v_{1p}} \\ \vdots & & \\ \tilde{v_{F1}} & \dots & \tilde{v_{FP}} \end{bmatrix} = \begin{bmatrix} i_1^T \\ \vdots \\ i_F^T \\ j_1^T \\ \vdots \\ j_F^T \end{bmatrix} \begin{bmatrix} s_1 & \dots & s_P \end{bmatrix} = RS$$

Now the only task left is to find a translation matrix.

$$x_{fp} = \tilde{x_{fp}} + a_f = i_f^T(S_p - t_f)$$

$$a_f = -t_f i_f^T$$

$$W = \tilde{W} + \mathbf{T} = RS + (a_1, ..., a_f, b_1, ..., b_f)^T$$

where a_f is a projection of camera translation along x-axis. Ideally $\tilde{W}$ should have at most rank 3 because the object is in a 3D world. However, the noise of an image dictates us

to approximate the rank of the ideal registered measurement matrix. We can use singular value decomposition (SVD) to do it.

2.1 Incremental SfM Pipeline

Given the overview of the math above, a typical SfM pipeline implementation consists of two phases: the **correspondence search phase** and the **iterative reconstruction phase**. [3] The first phase of correspondence search phase generates an output called *Scene Graph* that finds the relations between the input 2D images. This step first extract a collection of local features to describe the interesting key points from the images. The **scale-invariant feature transform (SIFT)**[11] feature detection algorithm is used in all pipeline described in the remainder of this paper. During the second step of feature matching, the extracted features are used to determine which pairs of images portray common parts of the scene. In other words, it finds overlapping image pairs. The third step is geometric verification which ensures that the matches we previously found are accurate and output the scene graph.

During the second phase of iterative reconstruction, the pipeline initializes a pair of geometrically verified images from a dense region of the scene graph. The points in common to the two images are used as the first points of the reconstructed cloud and to establish the first two cameras. The reconstruction initiation is the start of the iteration. Then, new images are added to the reconstruction. We need to calculate the position of the camera that acquired the newly registered image. This is achieved by computing the key points in the newly added image and those associated with the previous image. After estimating the camera pose, triangulation defines the 3D coordinates of the new points that can be added to the construction and thus generate a more "dense" point cloud (though we still exist in the realm of sparse point clouds). The final step of this phase is bundle adjustment. It aims to prevent inaccuracies in the estimation of the camera pose which leads to wrong triangulation of point clouds.

All pipelines in this paper implements incremental SfM pipeline to generate sparse points, though a global SfM pipeline option is available for COLMAP and Theia. Global SfM considers the entire scene graph at the same time instead of incrementally adding more and more images during the reconstruction phase.[9] It takes a set of pairwise relative poses between cameras as input, and computes the orientation of all cameras in a global reference through motion averaging algorithms. [18] Because it only takes one bundle adjustment, some argues that the global SfM pipeline is faster and more readily parallelized. [14]

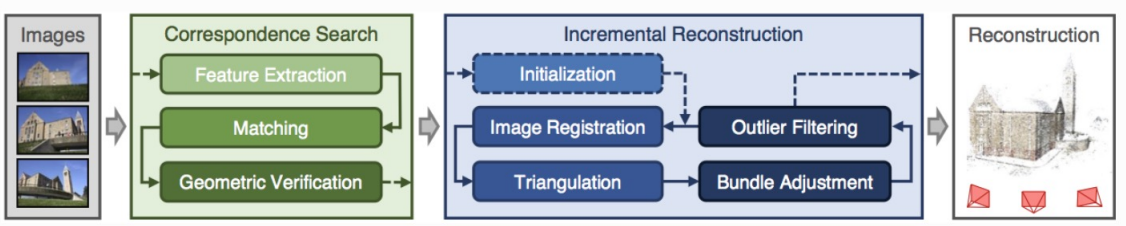


Figure 1: COLMAP's incremental Structure-from-Motion pipeline. [?]

3 Multi-View Stereo

After getting a sparse model that contains 3D feature points from SfM, we can use MVS algorithms to recover a dense point cloud. The techniques are based on a consistency function which measures whether a 3D model is consistent with the input images.

3.1 Types of MVS algorithm

MVS algorithms can be roughly classfied into four classes based on their output scene representations: *Voxel-based* approaches require a bounding box that contains the scene and thus works better for object level reconstruction. *Deformable polygonal mesh* demand a starting point such as a visual hull model to initialize the corresponding optimization process and thus is limited in their applicability. It is especially useful for free-viewpoint rendering, an example of such application is Google Earth. *Multiple depth maps* are particularly effective for the view-dependent texture mapping, an example of which is the Google Streetview, but it require fusing individual depth maps into a single 3D model. *Patch-based* methods approximate the surface by a set of small patches and are in general simpler and more effective.

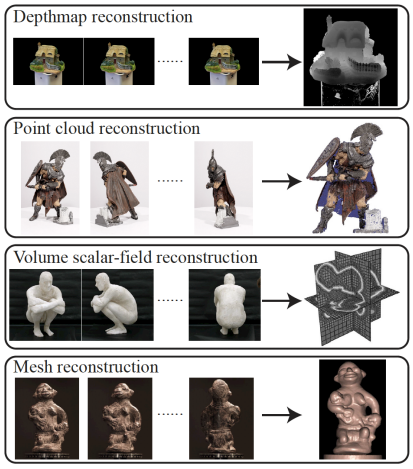


Figure 2: Four types of output scene representations.[6]

These four types of MVS algorithms each has its own advantage in different kinds of graphics applications. Since the patch-based algorithm is the underlying algorithm behind VisualSFM and OpenMVS, in the following discussion, we will focus on patch-based approaches.

3.2 Patch Model and Photometric Discrepancy Function.

We can think of a patch as a local tangent plane which approximates a surface. The geometry of the patch is determined by its center $c(p)$ and a unit normal vector $n(p)$ that points to towards the cameras observing it, and a reference image $R(p)$ in which p is visible.

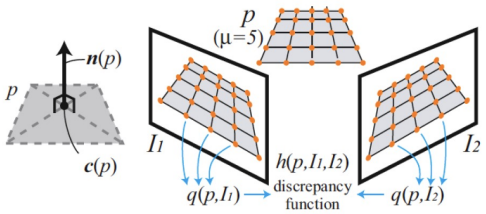


Figure 3: Left: a patch is a (3D) rectangle p with its center denoted as $c(p)$ and its normal denoted as $n(p)$. Right: the photometric discrepancy $h(p, I_1, I_2)$ of a patch is given by one minus the normalized cross correlation score between sets $\mathbf{q}(p, I_i)$ of sampled pixel colors.

Let $V(p)$ denote the set of images in which p is visible. Then, the photometric discrepancy function $g(p)$ for p is defined as

$$g(p) = \frac{1}{|V(p) \setminus R(p)|} \sum_{I \in V(p) \setminus R(p)} h(p, I, R(p)),$$

where $h(p, I_1, I_2)$ is the **normalized cross correlation score**, one of the most common and successful photo-consistency measures used in MVS algorithms. Basically, it is a pairwise photometric discrepancy function which returns a score that measures how dissimilar I_1 and I_2 are, the more dissimilar two images are, the higher their pairwise photometric discrepancy score will be.

Therefore, roughly speaking, we can interpret the photometric discrepancy score of $p, g(p)$, as the **average pairwise photometric discrepancy score** between an image and the reference image $R(p)$, over all images that contain p.

If the surface is not lambertian, we can handle undesirable effects by simply ignoring images with bad photometric discrepancy scores. In other words, only images whose pairwise photometric discrepancy score with the reference image $P(p)$ is below a certain threshold α are used.

3.3 Patch Optimization and Image Model

Having defined the photometric discrepancy function for a patch p, our goal is to reconstruct patches whose discrepancy scores are small. Each p is reconstructed in two steps: 1. initialization of the corresponding parameters, namely, its center $c(p)$, normal $n(p)$, visible images $V^*(p)$, and the reference image $R(p)$; and 2. optimization of its geometric components $c(p)$ and $n(p)$, which is optimized by simply minimizing the photometric discrepancy score $g^*(p)$. This is turned into an optimization problem and is solved by a conjugate gradient method.

The biggest advantage of the patch based surface representation is its flexibility. However, this also means that it lack the connectivity information and makes accessing neighboring patches more difficult. Thus, we need to keep track of the image projections of reconstructed patches in their visible images. Basically, we divide each image up into cells and project the patch onto each images in $V^*(p)$. Then, each cell remembers the set of patches that porject into it.

Above are the key elements of patch based MVS. It is explained in greater detail in [7].

4 Experiment Description

4.1 Data Sets

In this report, we test each pipeline against 7 datasets, 4 of which are available online and 3 of which are created by us. In general, data sets can be roughly divided into three categories based on its images: object, scene, and crowded scene.

Images of **objects** are those where a single, compact object is usually fully visible in a set of uncluttered images taken from all around it, and it is relatively straightforward to extract the apparent contours of the object. In our dataset, *dog, temple,* and *dino* are images of object. Images of **scenes** are those where the target object(s) may be partially occluded and/or embedded in clutter, and the range of viewpoints may be severely limited. Typical examples are outdoor scenes with buildings, vegetation, etc. In our case, images in *castle, house,* and *obstacle* data sets are images of scene. Lastly, in images of **crowded scenes**, moving obstacles appear in different places in multiple images of a static structure of interest (e.g., people passing in front of a building). The dataset of *store* is crowded scene as there are passing cars in different images.

Figure 4 shows part of the input images of all seven data sets used in our experiments.

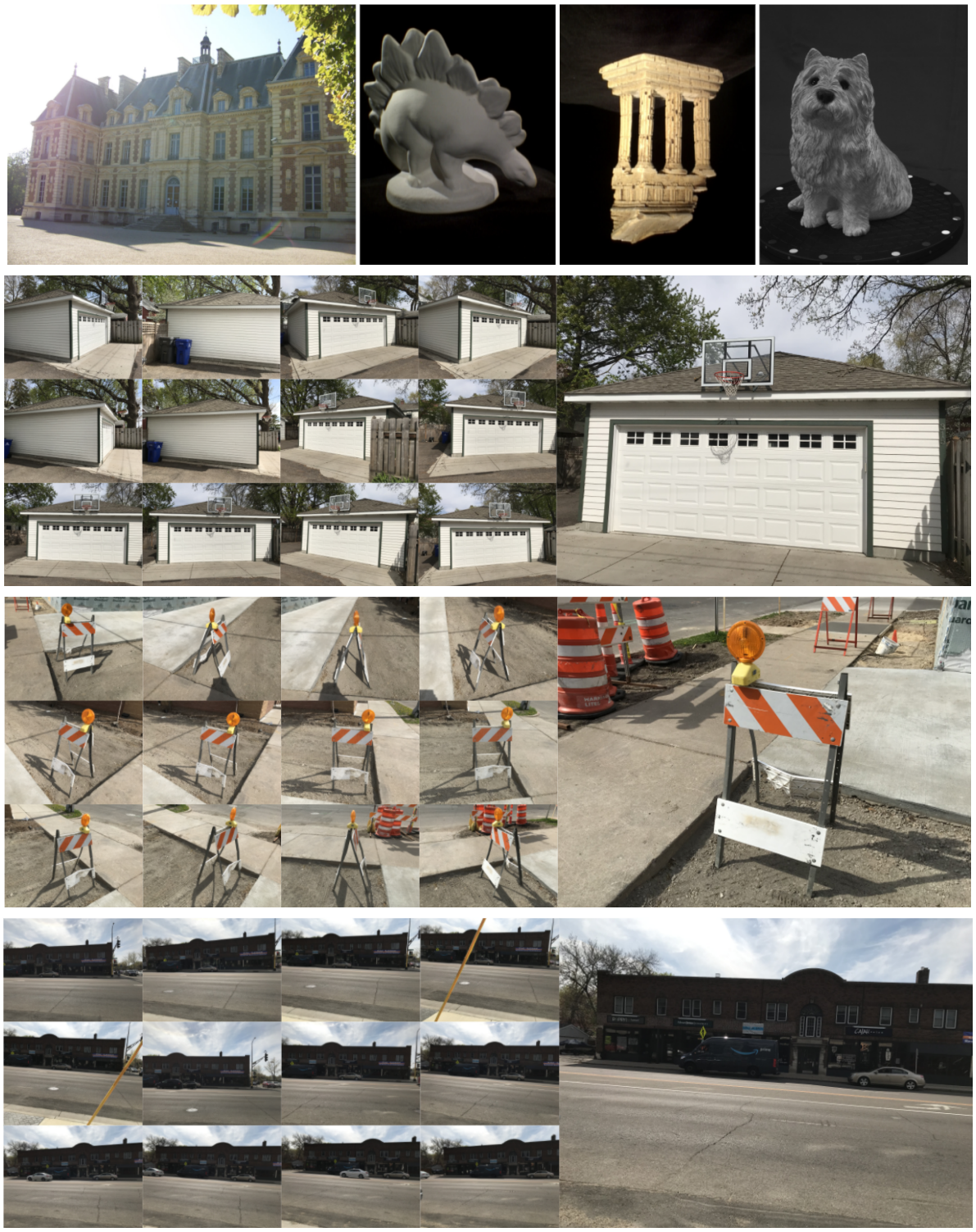


Figure 4: Datasets: castle, dino, temple, dog, house, obstacle, store, from left to right, top to bottom.

Table 4.1 lists the number of input images, their approximate size, whether we have their ground truth or not, their type, and their source.

Name	Images	Image Size	Ground Truth	Type	Source
castle	12	2832 × 2128	No	scene	OpenMVG dataset
dino	48	480 × 640	No	object	Middlebury dataset
temple	47	480 × 640	No	object	Middlebury dataset
dog	60	1024 × 1280	Yes	object	ToHoku University
obstacle	28	4032 × 3024	No	scene	pictures took with iPad
store	14	4032 × 3024	No	crowded scene	pictures took with iPad
house	14	4032 × 3024	No	scene	pictures took with iPad

4.2 Pipelines

VisualSFM is a GUI application for 3D reconstruction using structure from motion. [19] This system produces sparse point cloud in $O(n)$ time on its major steps while maintaining the reconstruction accuracy. One differentiator is the preemptive feature matching algorithm that reduces the running time of typical full-pairwise matching from $O(n^2)$ to $O(n)$. The system exploits parallelism and sorts features of each image into decreasing scale order. For each image pairs during image registration, the system matches the first few features of the two registered images. If the number of matches is smaller than a threshold, the *for-loop* continues to the next image. Instead of comparing all features of all pictures, this method allows putting most efforts in the ones that are more likely to be matched. This system can also produce dense point cloud reconstruction by integrating the execution of PMVS/CMVS tool chain outlined in the last section.

Multi-View Environment (MVE) is an implementation of image reconstruction pipeline which features SfM, MVS, and surface reconstruction. Given a set of images, it can generate a sparse point clouds by incremental SfM. Instead of PMVS/CMVS in VisualSfM, MVE uses Multi-View Stereo for Community Photo Collections approach which reconstructs a depth map for every view [8]. Although this approach might produce redundancy because many views are overlapping, it scales to large scenes as only a small set of neighboring views is required for geometry reconstruction. As a result, this approach creates very dense point clouds, thus preserving more details for mesh reconstruction later. MVE implements Floating Scale Surface Reconstruction (FSSR) that converts dense points to mesh geometry. Compared to that is the Screened Poisson Surface Reconstruction [10] implemented in MeshLab.

Theia is aimed at providing efficient and reliable algorithms for SfM as a C++ library. [18] Theia can only perform SfM, meaning that the output of this algorithm is sparse point cloud. Unlike the rest of the SfM pipelines in this report, it is recommended to use global SfM algorithm.

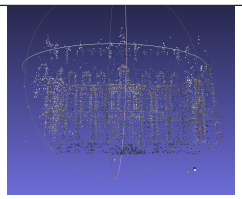
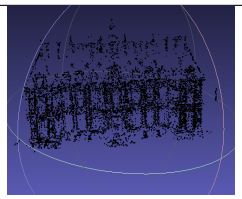
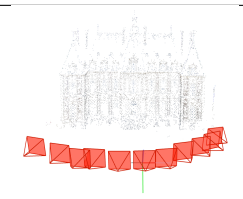
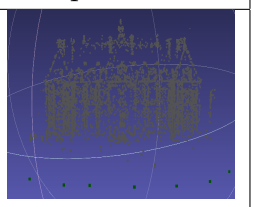
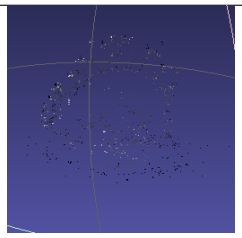
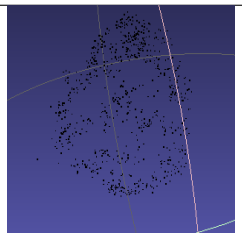
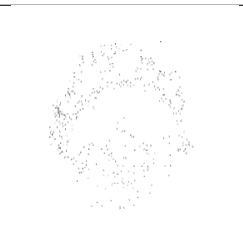
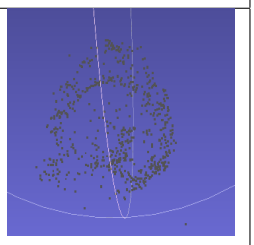
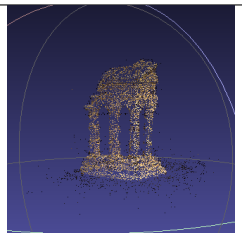
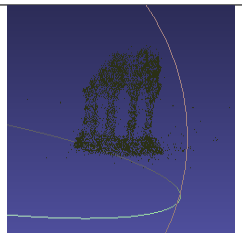
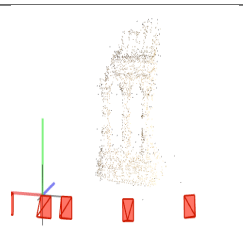
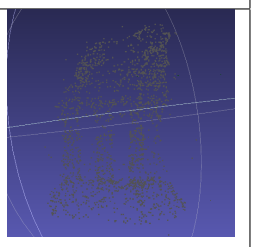
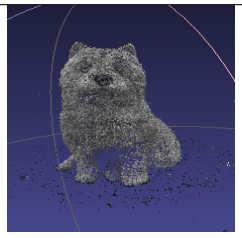
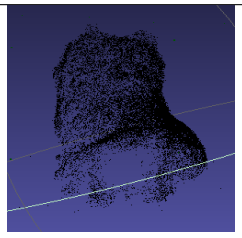
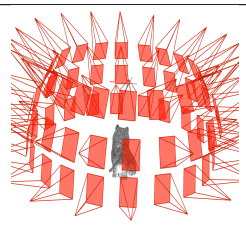
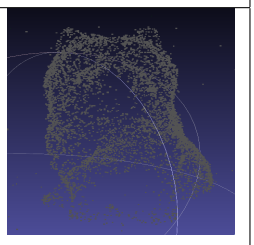
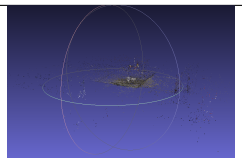
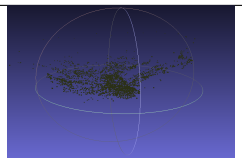
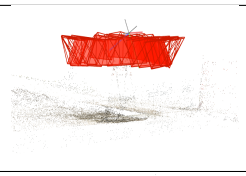
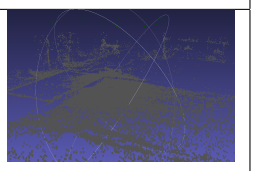
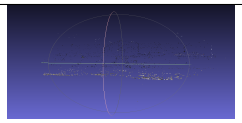
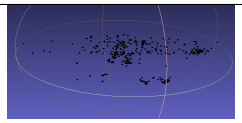
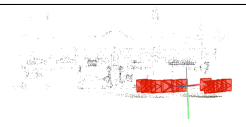
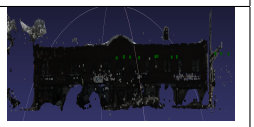
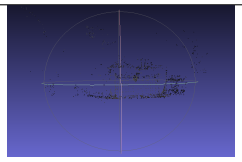
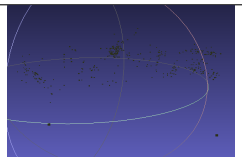
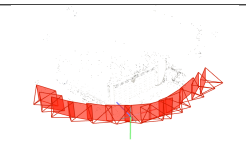
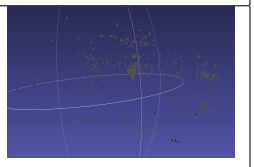
COLMAP is an open-source implementation of the incremental SfM and MVS pipeline. It can reconstruct any scene while also provides enhancements in robustness, accuracy and scalability. Among all the pipelines we've run, COLMAP is the most user-friendly one as it comes with an intuitive **graphical user interface (GUI)** that is very easy to run. However, it requires CUDA to perform dense reconstruction. If the user does not have CUDA, they have to export the sparse reconstruction for different MVS pipelines. There is not much documentation as to how to connect it with such MVS pipelines such as OpenMVS.

OpenMVG is an open-source library to solve Multiple View Geometry problems.[15][13][12] It provides an implementation of the SfM pipeline for both the incremental and global case. Sparse reconstruction can be exported in different file formats for different MVS pipelines such as OpenMVS.

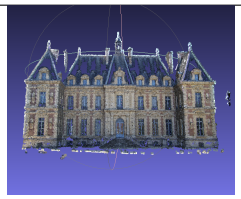
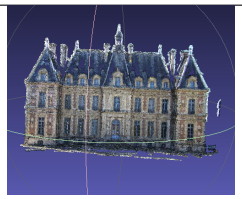
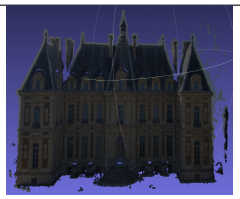
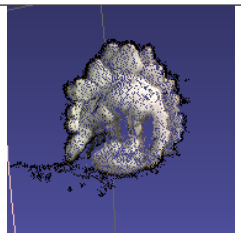
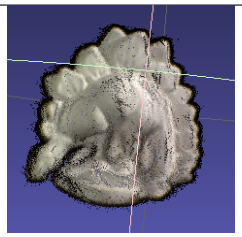
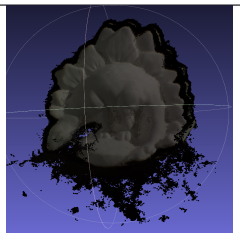
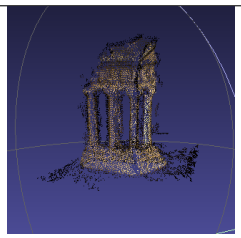
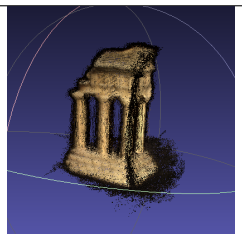
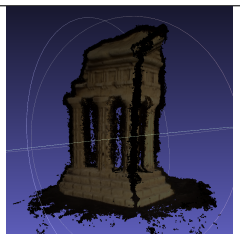
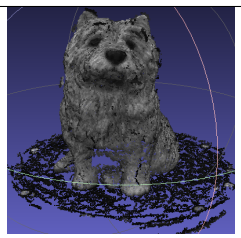
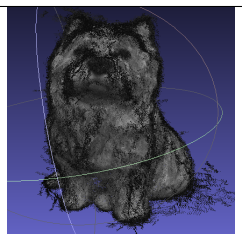
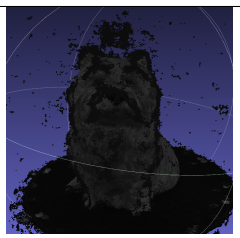
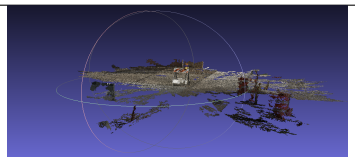
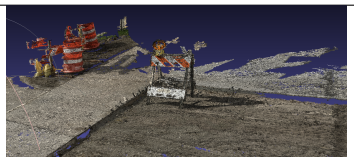
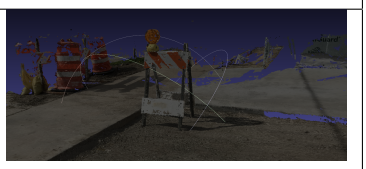
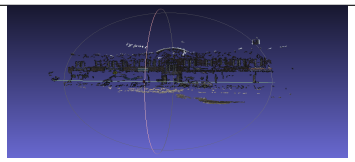
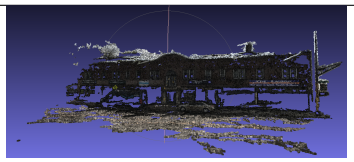
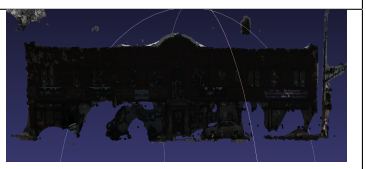
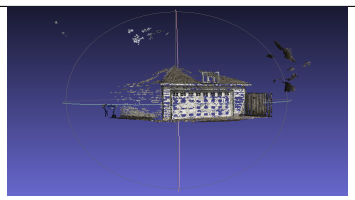
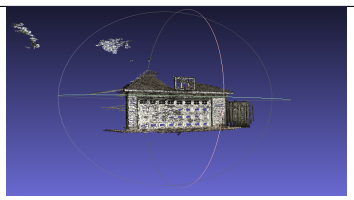
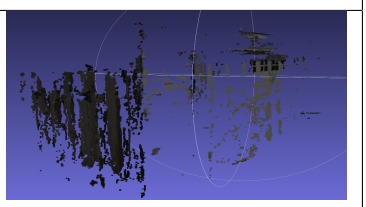
OpenMVS is an MVS pipeline, which takes in output from a SfM pipeline and provide a complete set of algorithms to recover the **full surface** of the scene as a textured mesh. An OpenMVS pipeline first creates a complete and accurate as possible point-cloud. Then, it performs mesh reconstruction and refinement to estimates a mesh surface that explains the best the input point-cloud while recovering all fine details. Lastly, it completes mesh texturing by computing a sharp and accurate texture to color the mesh. It also comes with a script which incorporates OpenMVG and OpenMVS, which provides very nice results.
For the MVS step, OpenMVS uses a patched-based algorithm that is slightly different from PMVS. [1] [4]

5 Experiment Results

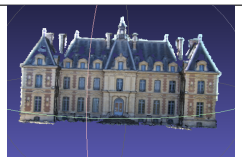
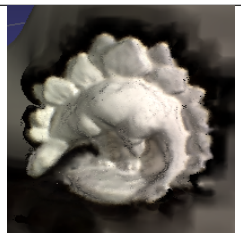
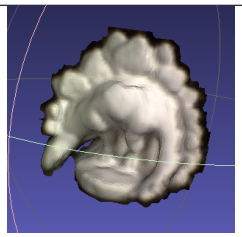
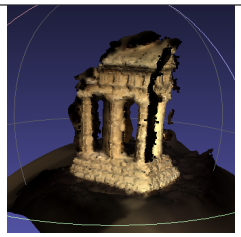
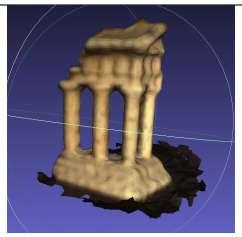
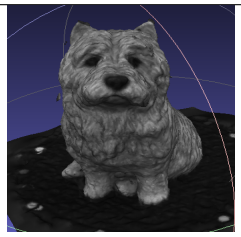
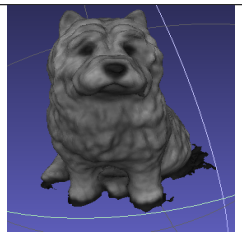
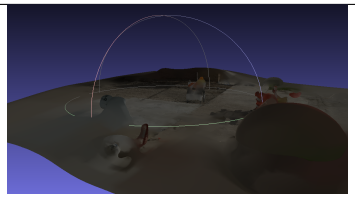
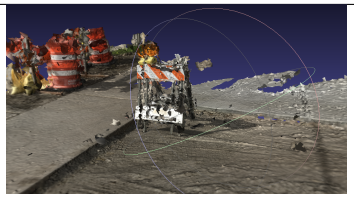
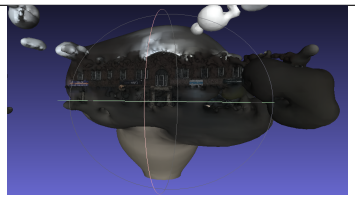
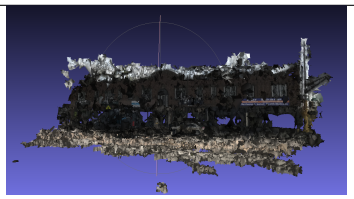
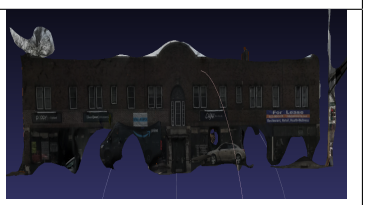
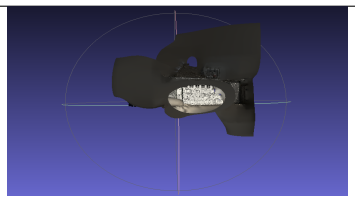
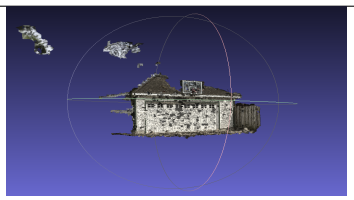
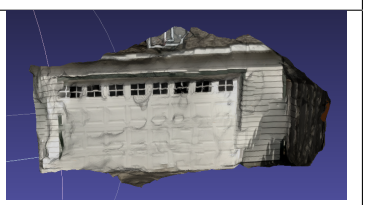
5.1 Sparse Point Cloud

alg	VisualSfM	Theia	COLMAP	OpenMVG
castle				
dino				
temple				
dog				
obstacle				
store				
house				

5.2 Dense Point Cloud

alg	VisualSfM	MVE	OpenMVG
castle			
dino			
temple			
dog			
obstacle			
store			
house			

5.3 Surface and Mesh

alg	VisualSfM	MVE	OpenMVG/OpenMVE
castle			
dino			
temple			
dog			
obstacle			
store			
house			

6 Findings

In terms of the sparse point clouds constructed, VisualSfM, Theia, OpenMVG and COLMAP are almost equally good for most datasets. As we can see from Figure 6, COLMAP was able to recover the left wall of the house, while the OpenMVG missed it. This in turn leads to the less desirable surface by OpenMVS, which also completely left out the left wall. We can also see that Theia constructs the most sparse point cloud among all. For the obstacle dataset, we can't tell the outline of the point by Theia, whereas you can view a rough shape of roadblock. Among all pipeline, Theia took the least amount of time to run due to only one bundle adjustment. This is perhaps the tradeoff between speed and output quality.

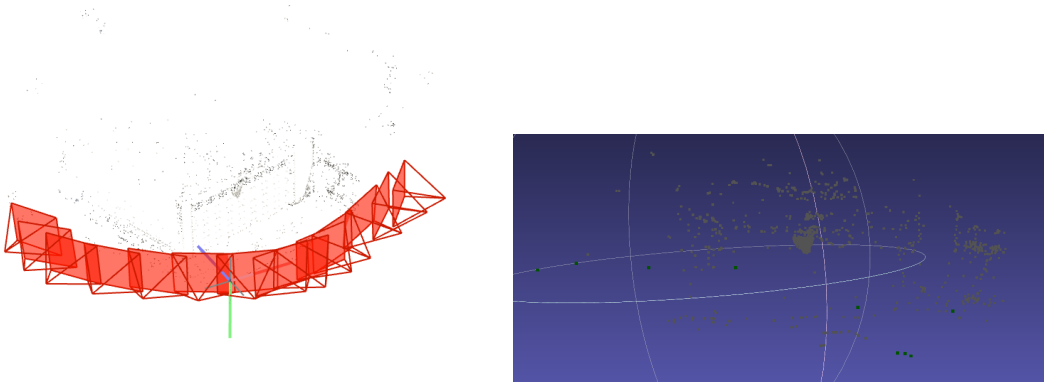


Figure 5: Sparse point cloud by COLMAP and OpenMVG

Given the better sparse point cloud by COLMAP, we were optimistic that the OpenMVS surface reconstruction result using COLMAP's sparse cloud could potentially outperform the one when combined with OpenMVG. However, there was no official documentation on how to combine these two pipelines from either COLMAP or OpenMVS. We managed to build a refined mesh surface using OpenMVS with the sparse point cloud from COLMAP, but not the fully textured one. However, even from the refined mesh model, we can see that the combination of COLMAP and OpenMVS is indeed better in terms of completion than the surface by OpenMVG and OpenMVS, as shown in Figure 6.

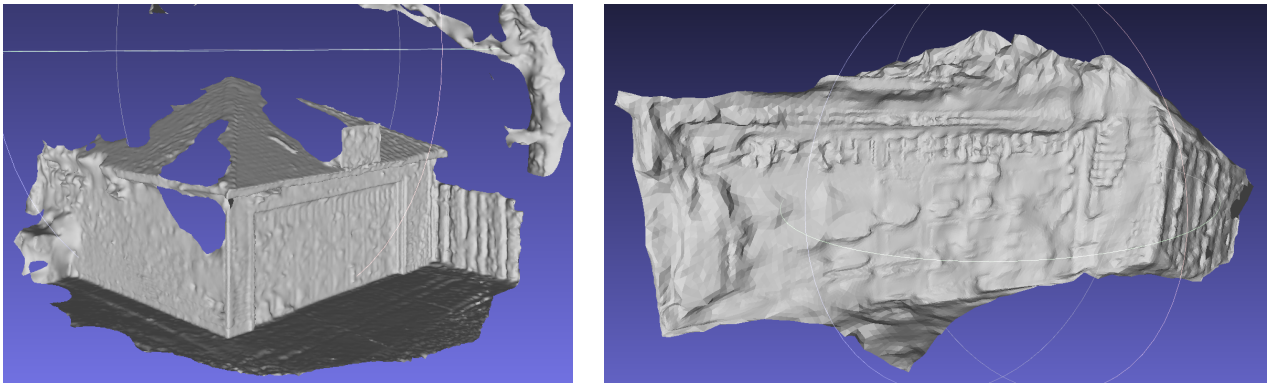


Figure 6: Sparse point cloud by COLMAP and OpenMVG

We also found that the combination of OpenMVG and OpenMVS pipelines is robust in terms of both object and scene level reconstruction. This is reflected from the reconstructed surface of the construction barrier data set, as shown in Figure 7. Despite the limited number of input pictures in the data set (28), we can tell that the final obstacle is very distinguishable, even the orange traffic cone has a reasonable shape. The road surface is very smooth. For the *store*, the final mesh does not have a lot of noise and weird shape.

Note that the object graph is obtained by zooming in on the reconstructed surface in the mesh visualization software MeshLab. We also recorded a short video demonstrating the exploration of the reconstructed scene using Meshlab. The quality of completion and level of details was quite good given the fact that we only took 28 pictures, with an original intention only to capture the construction barrier. **Here** is a link to the video, we encourage curious readers to check it out. You can also check out all the pictures on our **Google Drive**.

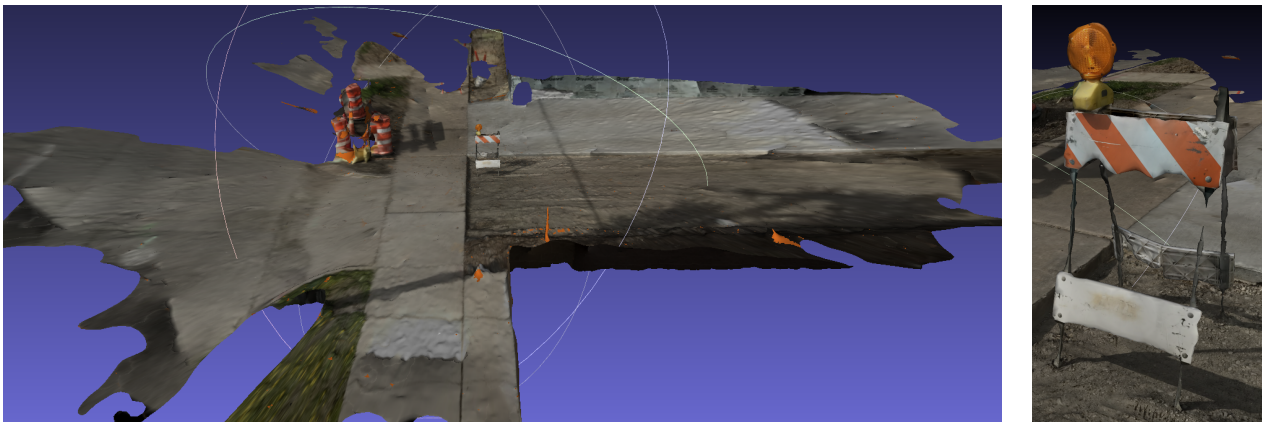


Figure 7: Construction barrier scene level and object level surface

As for reconstructing a crowded scene *store*, the OpenMVG and OpenMVS combination performed well and was able to filter out the passing vehicles. The VisualSfM and Theia did

poorly in terms of both sparse and dense point clouds. We can observe that there are multiple big gaps and holes in the result. This is a disadvantage for mesh generation later because the algorithm simply does not have enough point to create detailed surface. VisualSfM and Theia implements the Screened Poisson Surface Reconstruction, which leads to very weird protruding bubbles. This also happened to the *house*, perhaps because the lack of enough pictures for the two pipelines.

An interesting observation is that all the pipelines did better on **object** better than **scene** and especially the **crowded scene**. On the one hand, we might have more quality input pictures for the four **object**. The 3D reconstruction of *dog* did a particularly good job, perhaps due to the large input dataset (60). We could reasonably speculate that if we have more dataset extracted from a video, the reconstruction might work significantly better.

7 Conclusion

In this project, we explored six SfM and MVS pipelines, VisualSFM, MVE, Theia, COLMAP, OpenMVG, OpenMVS, along with a combination of OpenMVG and OpenMVS. We studied the basic ideas and principles underlying SfM and MVS algorithms. We then applied these pipelines to seven data sets, four of which were obtained online and three of which were created by us. These seven data sets were relatively comprehensive as they contain images of objects, scenes, and crowded scenes. We contribute to the literature with visual comparisons of results from different pipelines at multiple stages of the reconstruction pipeline using data sets covering three types of images: object, scene, and crowded scene.

8 Further Directions

Our work could be further extended in the following aspects. First, we could write a script for extending the serendipitous findin of COLMAP and OpenMVS pipeline. We could compare it against the combination of OpenMVG and OpenMVS. Second, we can test on datasets with more complicated details, such as a tree. Lastly, we could integrate the existing repositories, mostly in C++, to Python. The pipelines require a complex set of dependencies which cost us a lot of time to install. It would be awesome to have an better user experience by having a simpler Python implementation. Lastly, we could reconstruct scenes from larger datasets, such as an entire room. We could look into the appropriate number of pictures to put into a pipeline so we do not sacrifice too much time for a slightly more detailed result.

References

[1] C. Barnes, E. Shechtman, A. Finkelstein, and D. B. Goldman. PatchMatch: A randomized correspondence algorithm for structural image editing. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 28(3), Aug. 2009.

[2] S. Bianco, G. Ciocca, and D. Marelli. Evaluating the performance of structure from motion pipelines. *Journal of Imaging*, 4:98, 08 2018.

[3] S. Bianco, G. Ciocca, and D. Marelli. Evaluating the performance of structure from motion pipelines. *Journal of Imaging*, 4(8):98, 2018.

[4] M. Bleyer, C. Rhemann, and C. Rother. Patchmatch stereo - stereo matching with slanted support windows. In *BMVC*, January 2011.

[5] Y. Furukawa, B. Curless, S. M. Seitz, and R. Szeliski. Towards internet-scale multi-view stereo. In *2010 IEEE computer society conference on computer vision and pattern recognition*, pages 1434–1441. IEEE, 2010.

[6] Y. Furukawa and C. Hernandez. Multi-view stereo: A tutorial. *Foundations and Trends in Computer Graphics and Vision*, 9:1–148, 06 2015.

[7] Y. Furukawa and J. Ponce. Accurate, dense, and robust multiview stereopsis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(8):1362–1376, 2010.

[8] M. Goesele, N. Snavely, B. Curless, H. Hoppe, and S. M. Seitz. Multi-view stereo for community photo collections. In *2007 IEEE 11th International Conference on Computer Vision*, pages 1–8. IEEE, 2007.

[9] N. Jiang, Z. Cui, and P. Tan. A global linear method for camera pose registration. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 481–488, 2013.

[10] M. Kazhdan and H. Hoppe. Screened poisson surface reconstruction. *ACM Transactions on Graphics (ToG)*, 32(3):1–13, 2013.

[11] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[12] L. Moisan, P. Moulon, and P. Monasse. Automatic homographic registration of a pair of images, with a contrario elimination of outliers. *Image Processing On Line*, 2:56–73, 2012.

[13] P. Moulon and P. Monasse. Unordered feature tracking made fast and easy. In *CVMP 2012*, page 1, 2012.

[14] P. Moulon, P. Monasse, and R. Marlet. Global fusion of relative motions for robust, accurate and scalable structure from motion. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3248–3255, 2013.

[15] P. Moulon, P. Monasse, R. Perrot, and R. Marlet. Openmvg: Open multiple view geometry. In *International Workshop on Reproducible Research in Pattern Recognition*, pages 60–74. Springer, 2016.

[16] M. Shah. Lecture 15: Structure from motion.

[17] N. Snavely, S. M. Seitz, and R. Szeliski. Photo tourism: exploring photo collections in 3d. In *ACM Siggraph 2006 Papers*, pages 835–846. 2006.

[18] C. Sweeney, T. Hollerer, and M. Turk. Theia: A fast and scalable structure-from-motion library. In *Proceedings of the 23rd ACM international conference on Multimedia*, pages 693–696, 2015.

[19] C. Wu. Towards linear-time incremental structure from motion. In *2013 International Conference on 3D Vision-3DV 2013*, pages 127–134. IEEE, 2013.