# 3D Point-Cloud and Surface Mesh Rendering using COLMAP on Brutus Dataset

Deepratna Awale

dawale@mun.ca

## 1. Introduction

The primary goal of this project is to explore the implementation of COLMAP on the BRUTUS Dataset. The data presented is 24 HD video files from which we extract a frame (say frame 0) for every video and attempt to make a 3D scene from the acquired images. The frames we acquire should be taken from the same timestamp to avoid "pixel motion". The final goal is obtaining a dense representation (using MVS) of images from the BRUTUS Dataset.

### 1.1. COLMAP Pipeline

COLMAP is a graphical and command-line interfaced general-purpose Structure-from-Motion (SFM) and Multi-View Stereo (MVS) pipeline. It has a variety of features for reassembling ordered and unordered image collections. Using Structure-from-Motion, image-based 3D reconstruction from images traditionally recovers a sparse representation of the scene and the camera poses of the input images first. This output is fed into Multi-View Stereo to create a dense scene representation. [1]

### 1.2. Brutus Dataset

Brutus is a multi-camera mid-range system for recording panoramic light field video content. The proposed system captures light fields with a wide baseline (0.9 meters), high resolution (>20 pixels per degree), and a large field of view (>220) at 60 frames per second. The array consists of 24 time-synchronized cameras distributed across the surface of a hemispherical plastic dome 60cm in diameter. We use Z CAM E2 professional cinema cameras mounted to the dome's exterior with 3D printed brackets. The dome, mounts, triggering hardware, and cameras are all reasonably priced, and the array is simple to build. Using the recently developed Deep-View Video pipeline, we can record and render subjects as close to the cameras as 50cm.[2]

## 2. Utility Scripts

There's a directory called COLMAP_Utility_Scripts in "../scripts/python/" which contains some Python scripts that I created to extract image data, obtain various intrinsic parameters for the camera, and some un-distortion scripts. Description and usage of scripts:

### 2.1. Extract frames:

Extract_frames.py will extract frames from all video files in the directory passed onto it on the command line. For example, if you run the following command in the command line (in the Utility Scripts directory)

```
python extract_frames.py "absolute\directory\with\video_files"
```

Note: I'm using forward slash to escape all special characters. The output will be generated in the Directory passed in a directory called Frames.

## 2.2. Undistort frames:

Undistort_frames.py will undistort frames using FOV and POV parameters (which I have approximated and taken from BRUTUS[2] ). For the Brutus dataset, it is recommended that you undistort the frames right after you run the extract frames script as the Brutus Data set has very high distortion. We do not have the camera intrinsic to support proper un-distortion in COLMAP. So instead of trying to undistort images in COLMAP which crops them to an unusable extent, we will use this script to first undistort the images to a usable extent and then use a SIMPLE_RADIAL_FISHEYE camera model in COLMAP to attempt further to undistort images before dense reconstruction. Usage:

```
python undistort_frames.py "path\to\frames\directory"
```

## 2.3. Export Intrinsic to text:

You won't need this in most cases. If you have the intrinsic data available in JSON format this script will allow you to load it into Python, then export it in txt format; this text file is styled according to COLMAP Database parameters. Which is "Camera_ID Camera_Type Width Height Focal_length Principal_Point_x Principal_Point_y Distortion_coefficient_1 Distortion_coefficient_2". Then we can either manually use this to update the database in COLMAP or use a Python script to do so. The structure of the JSON file should be of the following format:

```
view = {'name': 'camera_0001',
 'position': [
       0.003311238794086777,
       0.000126384684934784,
       0.42421246053630285
   ],
 'orientation': [
       -0.03295944563072383,
       0.02964606619481096,
       -0.0333775100719531 6
   ],
 'focal_length': 1111.3638715594666,
 'pixel_aspect_ratio': 1.0,
 'principal_point': [
       1284.7296468363463,
       925.4651609728676
   ],
 'height': 1920.0,
 'width': 2560.0,
 'radial_distortion': [
       0.0970525284520715,
       -0.01708587111009977,
       0.0
   ],
 'projection_type': 'fisheye'
}
```

If you have intrinsic data for a rig at hand, then it is recommended to use it to create a better sparse model from the images. Further information can be found in the COLMAP Documentation.[3]

## 2.4. Undistort Using Intrinsic Information

The 'undistort_using_intrinsic_information.py' Python script will undistort images using the aforementioned JSON file's intrinsic information. It is recommended to use this if you have the intrinsic at

hand (at the time of making this project I did not). This will help you undistort images before you feed them into the COLMAP pipeline. Alternatively, you can also try feeding the intrinsic parameters directly into COLMAP and observe the difference.

# 3. COLMAP Terminology

COLMAP gives a comprehensive guide to using their pipeline.[3] In the following section, I will first describe the terminology and the steps I used to acquire the output. The following descriptions and figures are taken from [4]. Please note that I have only given a summary of the necessary topics that should be understood for this project; this report neither covers the maths behind most concepts nor all concepts.

## 3.1. Image Description

On one hand, approaches can be classified according to whether they describe the content globally (for example, "this image depicts a mountain") or locally (for example, "this region/pixel in the image depicts a tree"). On the other hand, approaches differ in the description's specificity, whether the description is on the category level (e.g., tree, mountain, etc.) or on the instance level (e.g., General Sherman Tree, Mount Everest, etc.). We use image description approaches from both ends of the spectrum in image-based 3D modeling. On a high level, local approaches allow for the precise geometric relationship between two images to be reconstructed at the instance level. In contrast, global approaches are less specific and less accurate but allow for the efficient association and categorization of images.

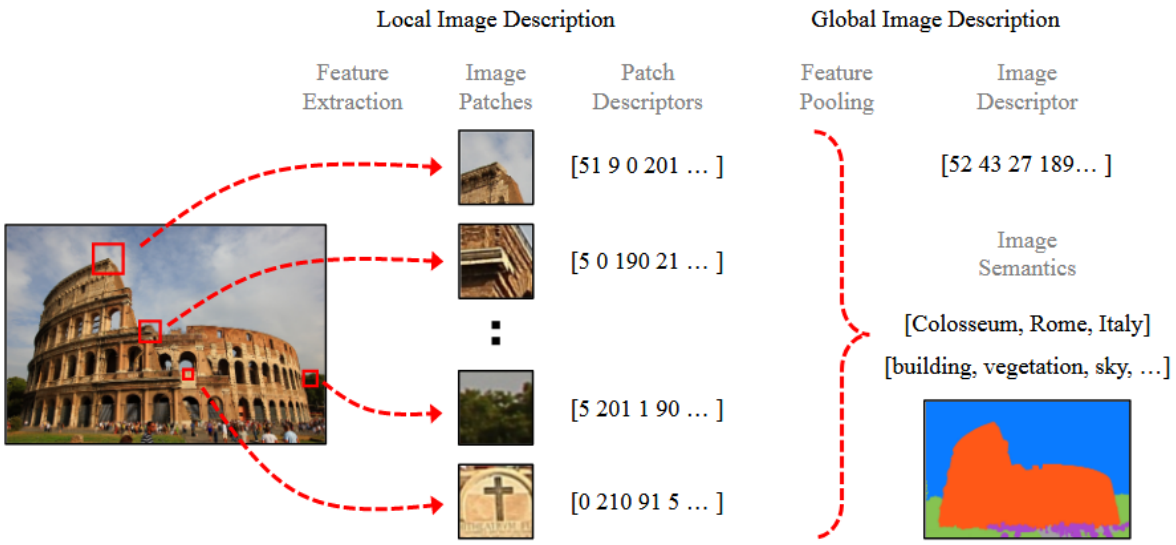


*Figure 1 Local vs Global Image Description*

## 3.2. Correspondence Search

The first stage in an image-based 3D reconstruction pipeline, starting with an unordered collection of images *I*, determines the two-view geometry for image pairs with scene overlap (see Figure 2). The result is a scene graph with images and scene points as nodes. Bidirectional edges between images indicate scene overlap and are annotated with a model that describes the two-view geometry. Undirected edges indicate

visibility between images and scene points.

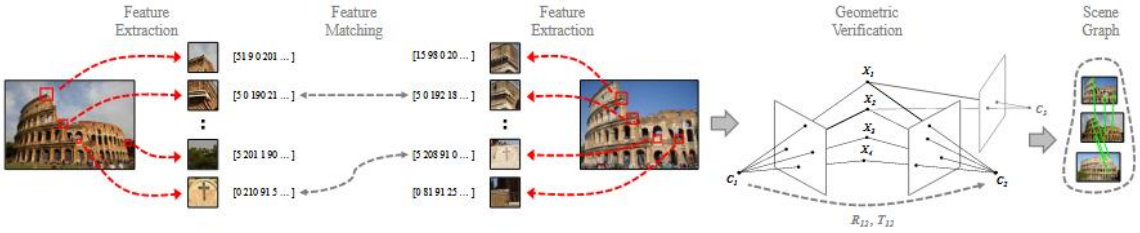


*Figure 2 Correspondence search begins with feature extraction and continues with feature matching and geometric verification. The scene graph is created by connecting the verified matches and two-view geometries.*

### 3.2.1. Feature Extraction

The pipeline first detects sets of local image features for each image $I_i$ in the input collection I.

$$F_i = \{g_j, d_j\} \mid j = 1..N_{F_i}$$

The local feature geometry $g_j$ (e.g., location, orientation, scale) and an appearance descriptor $d_j$ are represented. (e.g., a histogram of gradients). These features should be invariant under radiometric and geometric changes so that features of the same object can be uniquely recognized in other images with different appearances in the next stage.

### 3.2.2. Feature Matching

The second stage uses the features $F_i$ as an appearance description of the images to determine which pairs of input images see the same part of the captured scene. The naive approach compares all pairs of local features in each image pair to test for scene overlap in every possible combination. If a sufficient number of local features are similar with respect to their appearance description $d_j$, an image pair is considered to have scene overlap. This method has O ($N_I^2 N_{F_i}^2$) computational complexity and is prohibitively expensive for large image collections. To obtain a global description of an image, a popular approach with an effective computational complexity of O($N_i N_{Fi}$) employs hierarchical indexing of local features in a vocabulary tree. The compact global image description is then used in linear time with respect to the number of images to efficiently find the visually most similar-looking images for each image in the collection. The hierarchical index can then be used to find correspondence between local features in linear time with respect to the number of features for each visually similar image pair. Aside from the traditional approach, there are several other approaches to the problem of scalable and efficient matching. This stage produces a set of potentially overlapping image pairs:

$$\mathrm{C} = (\mathrm{I}_a, \mathrm{I}_b, \mathrm{M}_{ab}) | \mathrm{I}_a, \mathrm{I}_b \in \mathrm{I}, \mathrm{a} < \mathrm{b}$$

and the putative feature correspondences $M_{ab} \subset F_a \, x \, F_b$ that are associated with each potentially overlapping image pair.

### 3.2.3. Geometric Verification

The third stage verifies the potentially overlapping image pairs C geometrically. Because feature matching is solely based on appearance, it is not guaranteed that corresponding features will map to the same scene point due to incorrect feature-matching decisions. As a result, the putative feature matches are validated in this stage by attempting to robustly estimate a transformation that maps feature points between images based on their two-view geometry. The set of geometrically verified images C can be transformed into a scene graph, which will be used as input for the subsequent reconstruction stage. Overlapping images and

scene points are nodes in the scene graph. Directed edges between images indicate scene overlap and are annotated with a model that describes the two-view geometry. Undirected edges indicate visibility between ages and scene points. It should be noted that using transitive correspondence information can make scene points visible in more than two views.

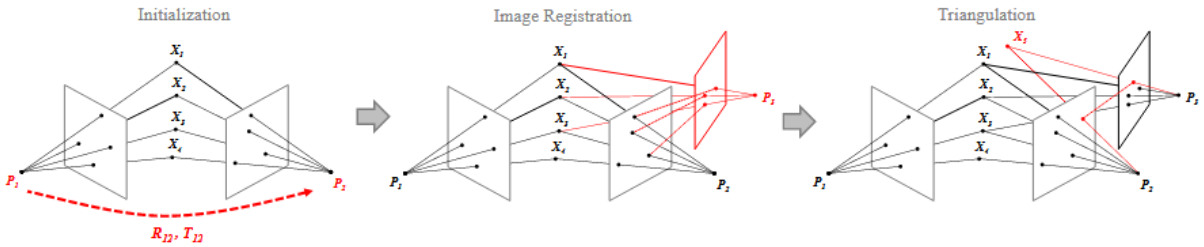


*Figure 3 The initialization, image registration, and triangulation stages of an incremental sparse reconstruction pipeline.*

## 3.3. Sparse Reconstruction

The scene graph generated by the correspondence search stage is fed into the incremental reconstruction stage. The outputs are intrinsic and extrinsic calibration:

$$P = \{P_c = K_c[R_c T_c] \mid c = 1 \ldots N_p\}$$

or reconstructed images in a common reference frame and the triangulated scene structure represented by a set of points

$$X = \{X_k \in R^3 \mid k = 1 \ldots N_x\}$$

Without prior knowledge of the scene, the datum of the reconstruction's reference frame has an arbitrary scale, orientation, and location. One might think that chaining the relative two-view transformations of overlapping images in the scene graph would yield the absolute extrinsic camera calibrations. The scale ambiguity of each two-view reconstruction and the accumulation of errors due to imperfect two-view reconstructions are inherent problems with this approach. There are two approaches to solving this problem that are fundamentally different: incremental and global reconstruction methods. On the one hand, incremental algorithms begin the reconstruction with a two-view reconstruction and then repeatedly register new images into the existing reconstruction. This is then triangulated, filtered, and refined using bundle adjustment to reduce accumulated errors. Global reconstruction methods, on the other hand, simulate the recovery of the absolute camera motion as a joint optimization over all the relative two-view geometries in the scene graph. Before estimating the global, absolute camera locations, this optimization is decomposed into the independent recovery of global, absolute camera orientations. Finally, global methods refine the reconstruction by performing a full global bundle adjustment. Global reconstruction methods are typically more efficient but generally less robust than incremental methods. Repeated bundle adjustments are the primary cause of incremental methods' slower runtime, especially for large reconstructions. However, repeated bundle adjustments and continuous scene refinement during incremental reconstruction typically lead to greater robustness in the two-view reconstruction with respect to uncalibrated input images and outliers.

## 3.4. Bundle Adjustment

Even though their products are highly correlated, image registration and triangulation are separate procedures. Uncertainties in the camera pose propagate to triangulated points and vice versa, and additional triangulations may improve the initial camera pose through increased redundancy. Incremental reconstruction results usually drift quickly to a non-recoverable state without further refinement. Bundle adjustment reduces accumulated errors as a joint non-linear refinement of camera parameters P and point parameters X. In the incremental reconstruction pipeline, bundle adjustment is a computationally demanding step. The Schur complement trick [43], which solves the reduced camera system first and then updates the points via back-substitution, is motivated by the special structure of parameters in bundle adjustment problems. Because the number of cameras is usually less than the number of points, this scheme is usually more efficient. There are two approaches to solving the system: exact and inexact step algorithms.

Exact methods solve the bundle adjustment system by storing and factoring it as a dense or sparse matrix [59, 199] with an $O(N_P^2)$ space complexity and an $O(N_P^3)$ time complexity. Inexact methods approximate the solution, typically using an iterative solver, such as preconditioned conjugate gradients with $O(N_P)$ time and space complexity. Direct algorithms are the preferred method for up to a few hundred cameras, but they are prohibitively expensive in large-scale settings. While sparse direct methods significantly reduce complexity for sparse problems, they are prohibitively expensive for large unordered image collections due to much denser connectivity graphs. In this case, indirect algorithms are the preferred method.

## 3.5. Dense Reconstruction

The sparse reconstruction stage typically uses only a few distinct image features to recover the structure and motion for efficiency. As a result, the reconstructed set of scene points is quite sparse, representing only a rough approximation of the captured real world. The final dense reconstruction stage will densify this sparse approximation to a richer representation, such as a dense point cloud or textured surface mesh.

### 3.5.1. Multi-View Stereo

Multi-view stereo is a computer vision technique that estimates the depth and normal of pixels in every camera calibrated during the sparse reconstruction stage. The technique compares pixels in one image with all pixels along the corresponding epipolar line in the other image using epipolar geometry and appearance similarity. However, because this process frequently produces noisy depth maps, researchers have developed methods to enforce surface smoothness priors and accumulate evidence across multiple views in order to obtain more accurate results. Surface normal information can be derived from gradient data or by incorporating normal inference directly. View selection is an important step in multi-view stereo, and it is typically performed using shared visibility information from the sparse scene reconstruction.

### 3.5.2. Multi-View Fusion

The second step in dense modeling is Multi-View Fusion to combine depth and normal estimates from image space to generate a dense point cloud in scene space. This step assists in obtaining a globally consistent and compact representation of the scene by utilizing the redundancy of multiple views to filter out outliers and increase the representation's accuracy. By requiring consistency in-depth and normal estimates, a more compact and accurate scene representation can be obtained, which can be used for visualization, rendering, and image-based localization.

### 3.5.3. Surface Reconstruction

A point cloud is insufficient for many practical applications, necessitating a surface mesh model. Examples include collision detection in robotics or games, efficient and high-fidelity rendering, and so on. Because

of the structure of the real world, mesh parameterization results in a much more compact representation for most scenes. While a point cloud requires many instances to densely model regular surfaces, a suitable surface mesh parameterization can potentially model an entire object with a single entity. In most cases, a simple triangular mesh can very well approximate the geometry of a scene with locally planar surfaces. Creating surface texture maps from images typically results in superior rendering quality compared to point cloud renderings.

# 4. Methodology

## 4.1. Extracting Data from the BRUTUS Dataset

The data set contains 24 video files, but we need images. We must extract the first frame from each video to get those images, using 'extract_frames.py'. The frames extracted should be from the same timestamp. By default, the script is configured to obtain the 1st frame from each video. Since the BRUTUS is synchronous, all cameras will have the same time space at any given time frame. Executing the script will give you a verbose output of current events and finally output a directory named "Frames" in the path passed onto it.

*Figure 4 Distorted Frame acquired from the BRUTUS video (Bagpipes).*

These frames now must be undistorted to be fed into COLMAP. Without the intrinsic parameters, COLMAP can't undistort these images significantly to help form a dense model. We can undistort the images using 'undistort_frames.py' and pass it to the path of the Frames directory in the Command Line. This will roughly

undistort the images and remove the fisheye and radial distortion to create a directory ‘Undistorted’ inside the Frames directory. These undistorted images will be our input to COLMAP.

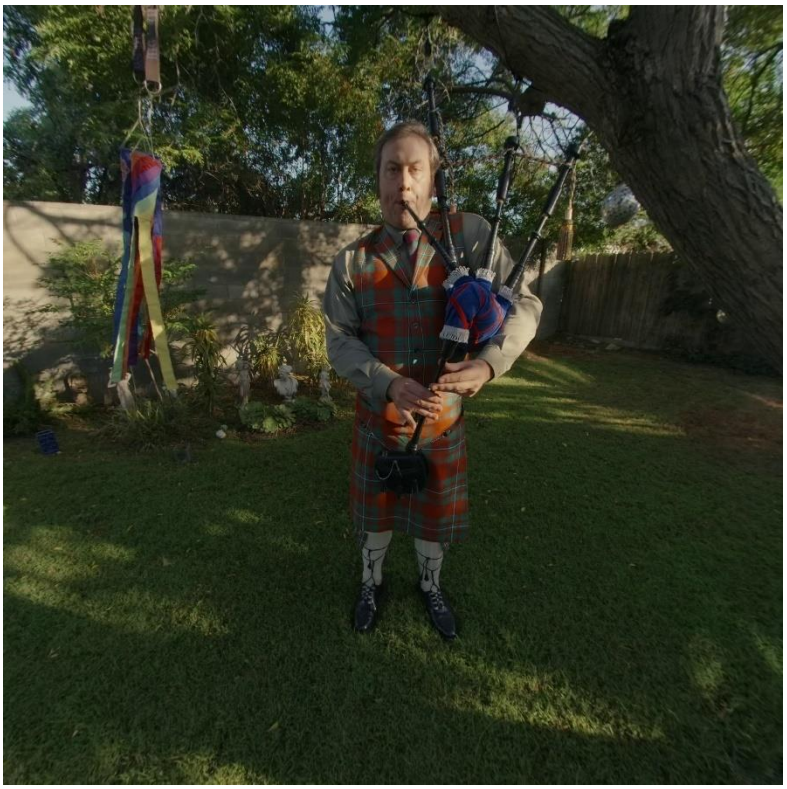

*Figure 5 Undistorted output of Figure 1.*

**Note:** *If the intrinsic information would have been available, we would have been able to better undistort the images. Camera intrinsic information also plays an important role in sparse reconstruction covered later.*

## 4.2. Using COLMAP on BRUTUS Dataset

I will here forth describe the procedure I used to create point clouds and reconstruct surfaces; along with it I’ll also comment on what different routes you can take. For an in-depth tutorial/ guide on using COLMAP, you should refer to [3]. If you use a Windows system [3] recommends pre-built binaries. The pre-built binaries I am using (and supplying in the technical content) have been compiled for CUDA (if possible I recommend using CUDA for better performance).

### 4.2.1. Create a new project using GUI Tool

1. Open the COLMAP software using the batch file ‘COLMAP.bat’ in the parent directory of pre-compiled binaries i.e., “/colmap-version-system-[CUDA]/COLMAP.bat”.
2. Go to File > New project.

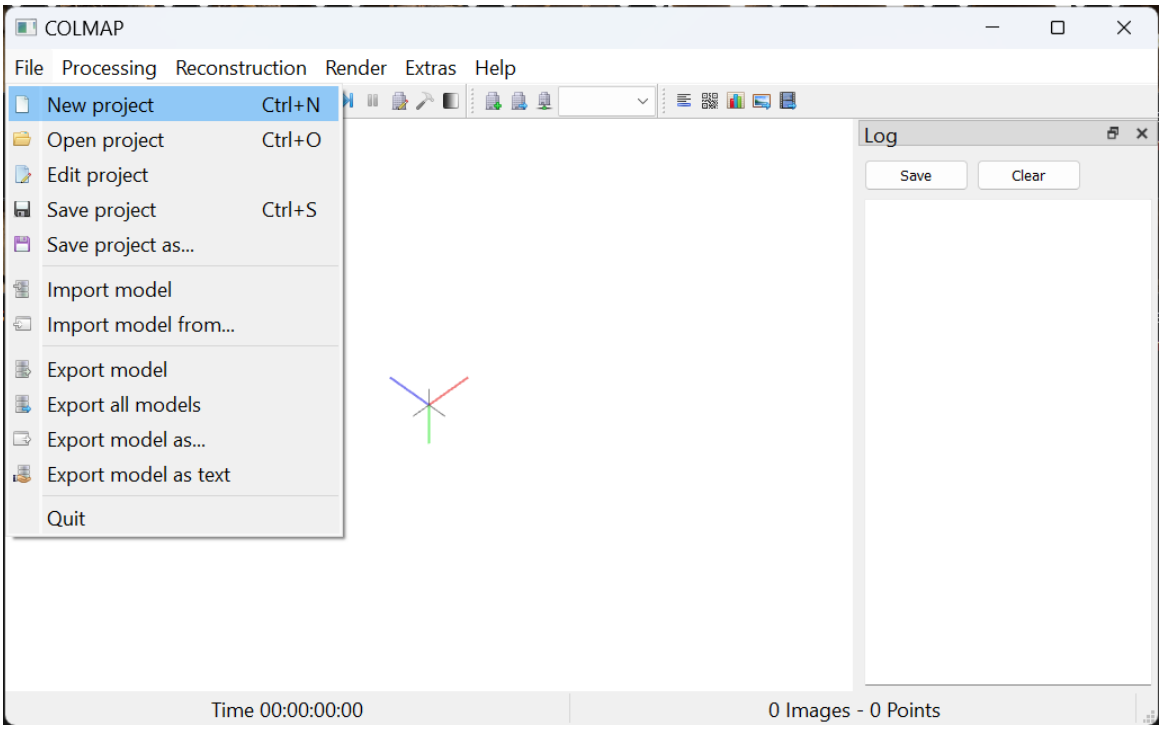


*Figure 6 COLMAP Window Navigating to New Project*

3. Click on New (Database), Choose working directory (I create a new directory named COLMAP in the directory containing data)
4. Select the directory of 'Undistorted images' as mentioned in Section 4.1. Hit Save.

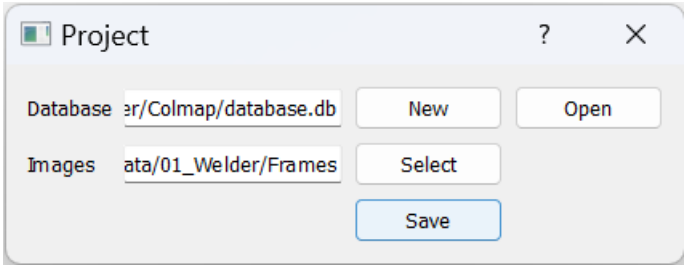


*Figure 7 Creating New database and configuring image directory*

### 4.2.2 Feature Extraction and Matching

1. Go to Processing > Feature extraction. In Camera Model: Select SIMPLE_RADIAL_FISHEYE. Make sure Shared for all images is unticked. If not already selected, select Parameters from EXIF. Leave the other setting to default and hit Extract.

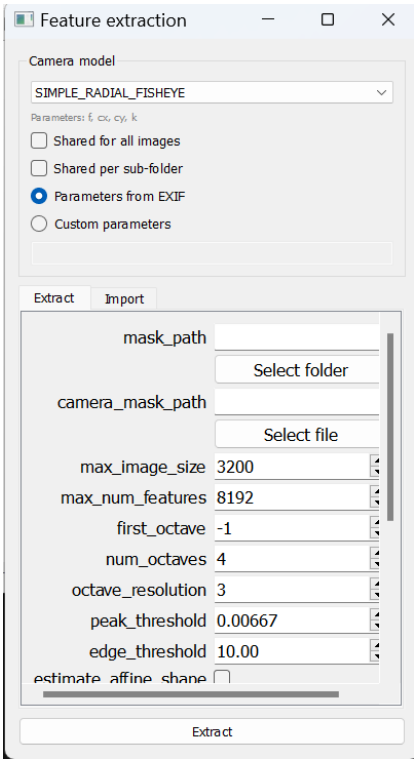


*Figure 8 Feature Extraction*

2. Select Feature Matching from the Processing Menu. You need not configure anything here, just hit Run. (Use Exhaustive Search since we only have 24 images).

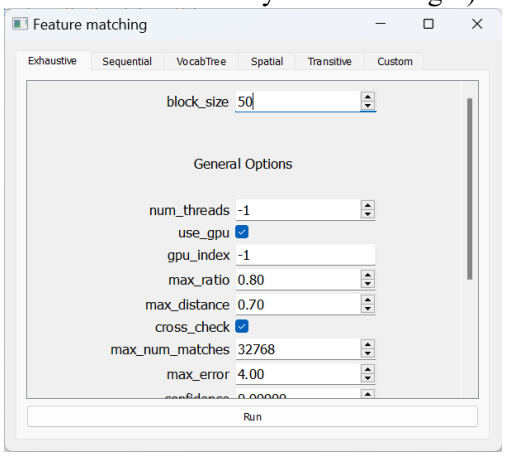


*Figure 9 Feature Matching*

3. You may choose to go to database management to peek at Features extracted and two view geometries.

*Figure 11 The red dots represent key points/features.*

4. If you choose overlapping images in the database viewer, you can choose to see the two-view-geometries.

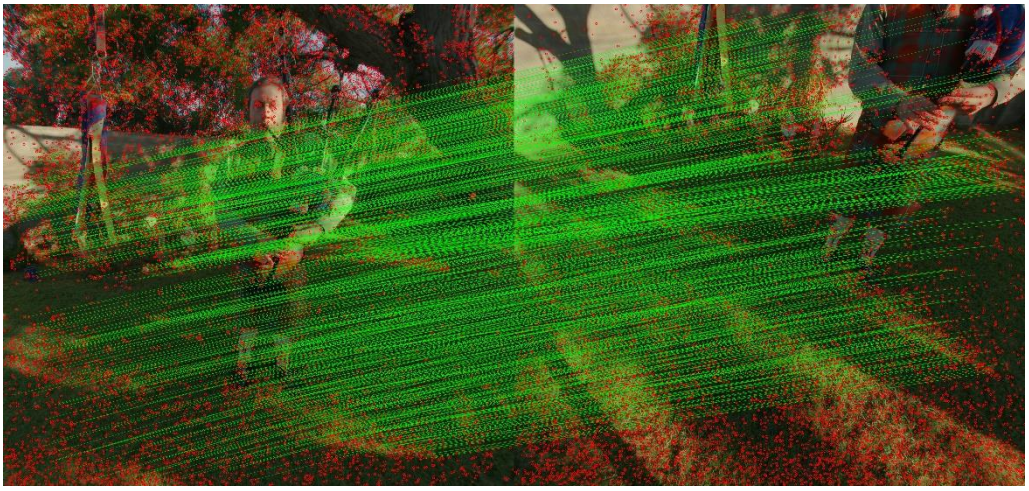

*Figure 10 Two-view Geometry*

### 4.2.3 Sparse Reconstruction

1. You can start Sparse Reconstruction: Reconstruction Menu > Start reconstruction.
2. You may choose to run Bundle adjustment a few times (it refines the sparse reconstruction).
3. After the reconstruction is finished, you will have a model similar to this:

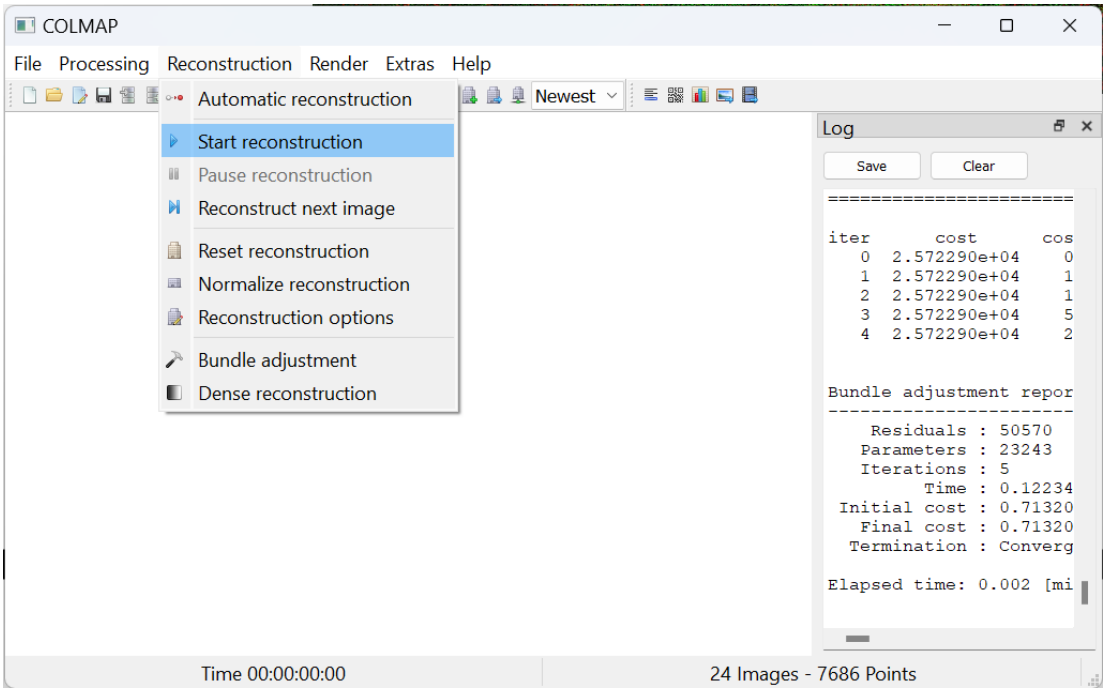


*Figure 12 Sparse Reconstruction in COLMAP*

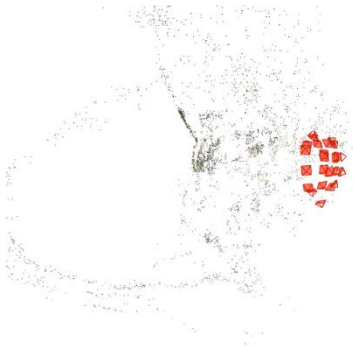

*Figure 13 Sparse reconstruction in COLMAP*

### 4.2.4 Dense Reconstruction

1. Go to Reconstruction > Dense Reconstruction> Click on Select.
2. Make a new directory called dense in your COLMAP Working directory.
3. Select the new directory. Hit select folder.
4. Click on Undistortion.
5. After images are undistorted, we will get option to run Stereo on these undistorted images. Click on Stereo.

6. Stereo will take a substantial amount of time. After the process is complete, you can view the photometric Depth and Normal Maps.
7. Fusion will use the Depth and Normal Maps and make a point cloud according to the sparse model.
8. Poisson will make a mesh (surface reconstruction) from the point cloud.

## 5. Outputs and Discussion

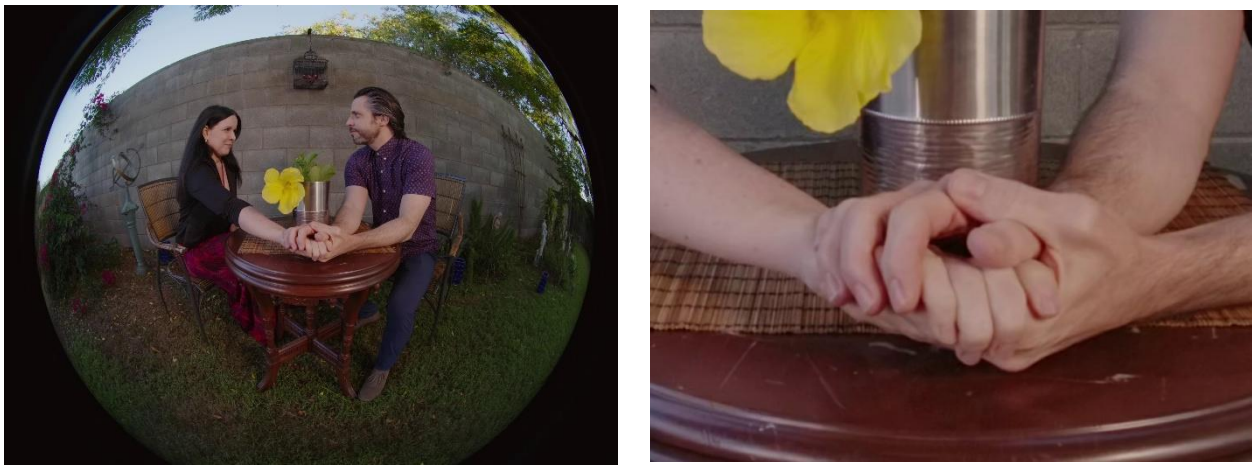

*Figure 14 Undistortion done by COLMAP if images are not preprocessed.*

*Figure 14 Inhouse Undistortion (left) and Undistortion by COLMAP on our output(right)*

While trying to undistort images in COLMAP for the dense reconstruction process, it is observed that COLMAP fails to undistort the images by itself and significantly crops the images to a point where it loses most of its data (Figure 15). Then even if this image is used for reconstruction, we will have very few key

points to make a point cloud. Thus, I have proposed to preprocess the images by undistorting before feeding them into COLMAP. If you observe the images in Figure 14, there is very little difference, if any at all.

*Figure 16 Convex formation of Camera Rig (Expected)*

In figure 13, the camera models are shown to be in a concave formation, but the BRUTUS rig is of the convex formation, this is a major issue since we do not have the intrinsic data for the cameras. If we did have intrinsic data provided like in [5] then we would be able to form a correct rig formation in COLMAP (figure 14). I manually fed the intrinsic parameters using the script in section 2.4 and was able to get a proper formation of the camera rig. This helped me get a better dense model later.

The underlying principle used to estimate depth information differs between a photometric depth map and

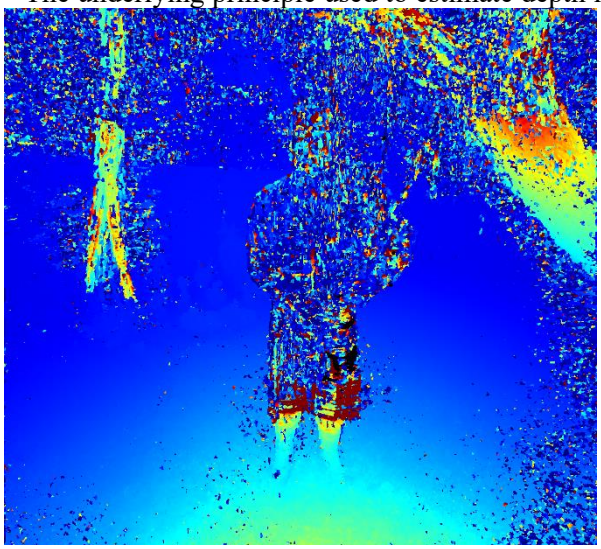

*Figure 17 Photometric Depth Map*

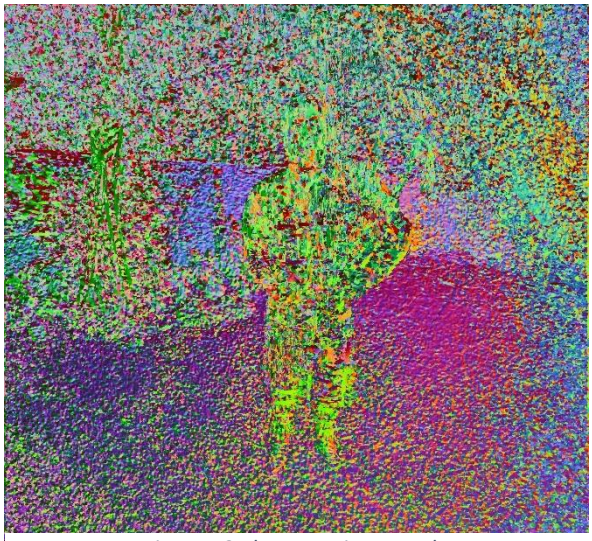

*Figure 18 Photometric Normal Map*

a geometric depth map. A photometric depth map is created by analysing the brightness or colour changes in an image or series of images, whereas a geometric depth map is created by triangulating known 3D points or analysing stereo images. Photometric depth maps are faster to compute than geometric depth maps, but they may contain errors due to lighting variations, texture patterns, or occlusions. Geometric depth maps, on the other hand, are more accurate but require more computational resources and may be limited in certain

scenarios, such as transparent or reflective surfaces. Both types of depth maps have advantages and disadvantages and can be useful.

Surface normals can be estimated using two methods: photometric normal maps and geometric normal maps. Geometric normal maps are generated directly from the scene's 3D geometry, typically from a depth map or a point cloud. They represent the scene's true surface normals and are thus accurate and consistent. Geometric normal maps, on the other hand, are sensitive to noise and errors in the depth map or point cloud, which can lead to unreliable normal estimates. Photometric normal maps, on the other hand, are estimated from the image's brightness gradients. They are based on the assumption that different surfaces reflect light differently and thus produce different brightness gradients. While photometric normal maps are less susceptible to noise and errors in the depth map, they can be influenced by lighting changes and materials with complex reflectance properties.

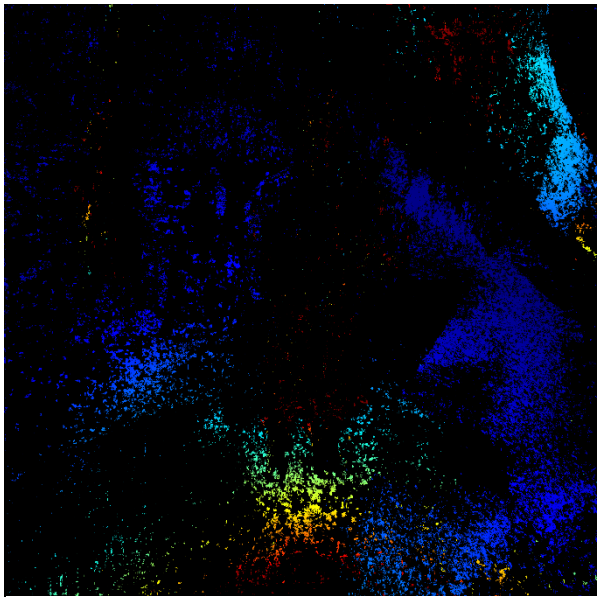

*Figure 19 Geometric Depth Map*

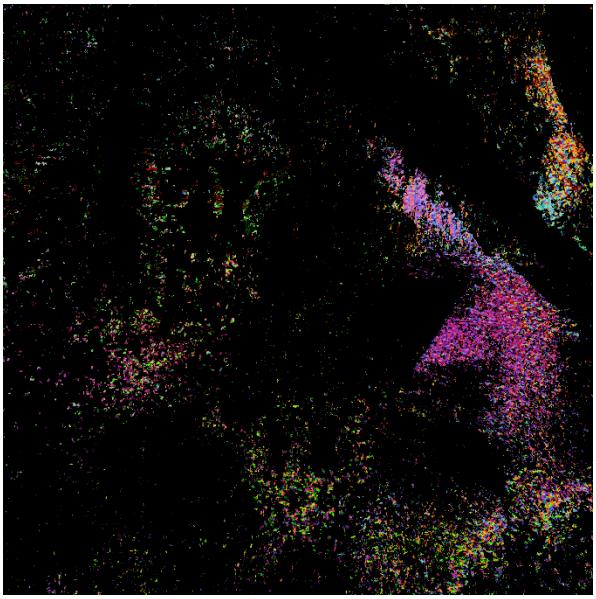

*Figure 20 Geometric Normal Map*

Figures 17 and 19 are the photometric depth map and geometric depth map of the image. The Photometric Depth map is relatively accurate in our case. While the geometric depth map is missing the person! From these maps we can actually predict that the 3d model will have a substantial portion of (the branch on the right, and a little bit of background). The photometric normal map again is near to what is represented in the image. While the geometric normal map is missing most of the data. These maps are expected to be unsmooth due to the nature of Normals.

In Figure 21 we can observe a rough outline of the tool rack behind and a welder with red helmet. This mesh was made from 01 Welder from Deep View Video Dataset [5]. This is the best point cloud I have been able to produce with quality setting set to Extra. This was possible only when I introduced the intrinsic parameters in the database. Before entering the intrinsic parameters, we could only produce point clouds that were highly distorted (Figure 22).

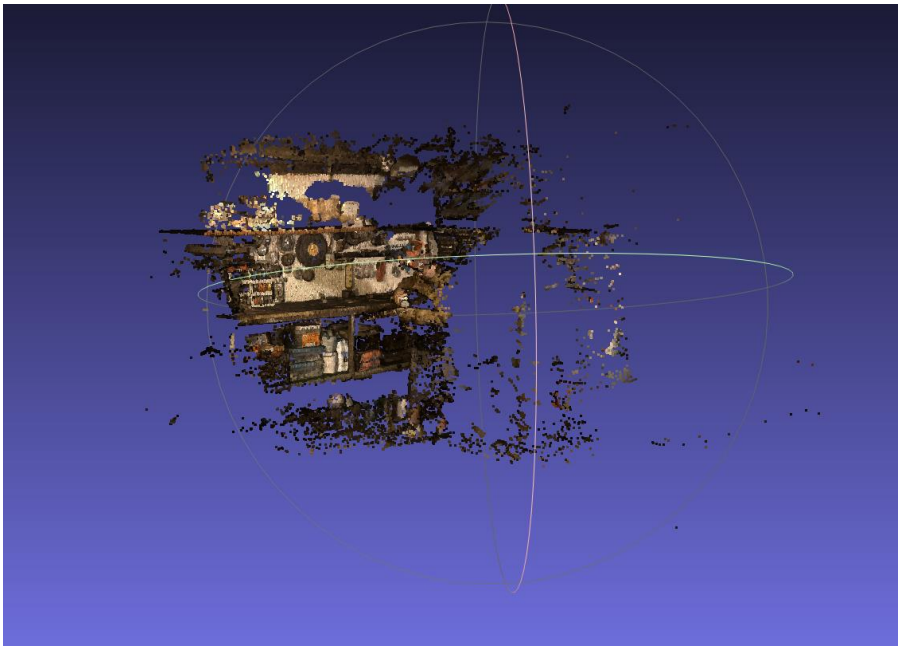

*Figure 21 COLMAP Cloud formation with intrinsic information.*

*Figure 22 COLMAP Cloud formation without intrinsic information.*

The following Poisson mesh was obtained from the cloud in Figure 21. Both the fused and Poisson meshes are included in the dataset accompanying the technical report.

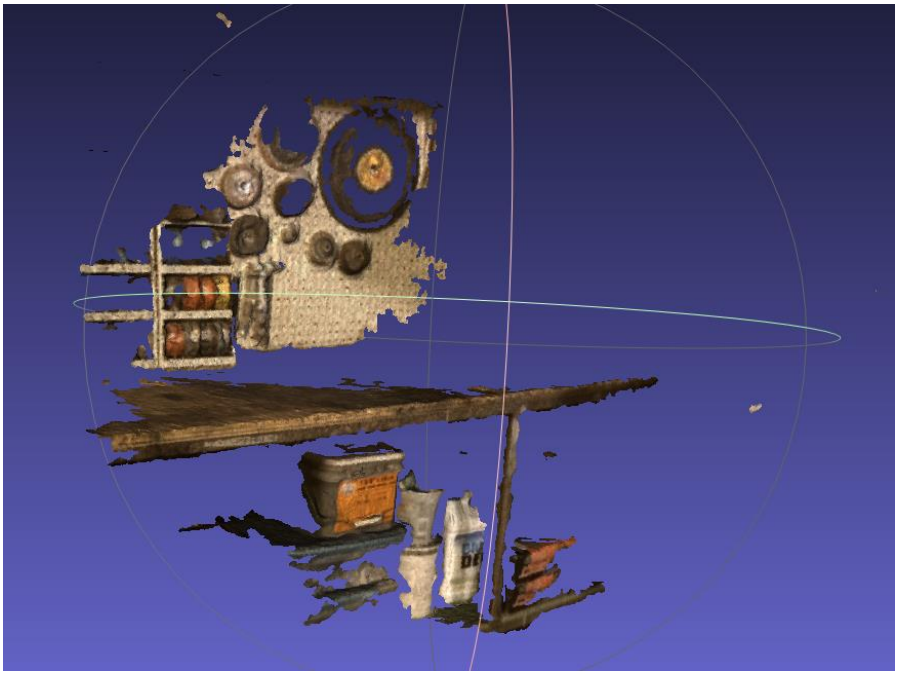

*Figure 23 Poisson Mesh for Figure 21*

Mesh Results for the BRUTUS Bagpipe data are very incoherent. We can barely make out the tree and the man. This is after trying various camera configurations (decreasing the number of cameras to focus on one object), undistorting the images to get a better map. Using the distorted images as is either crashes COLMAP while undistorting/ dense reconstruction or otherwise creates incoherent 3D-point-clouds.

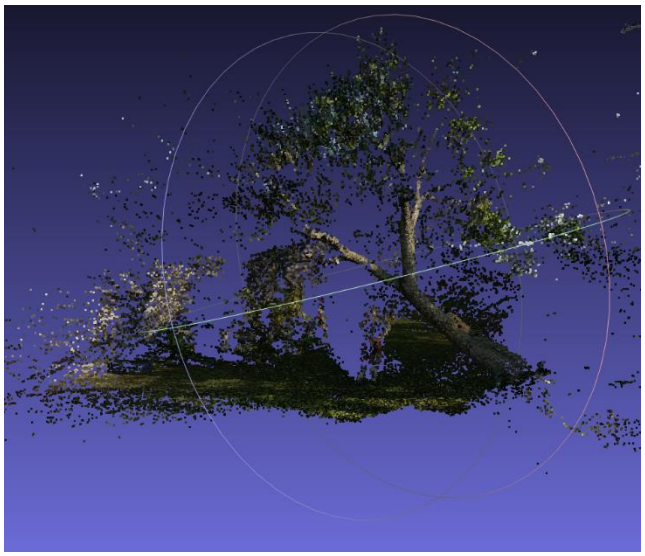

*Figure 15 Stereo of Bagpipes from BRUTUS Dataset*[2]

## 6. Conclusion

COLMAP uses images to make a 3D-point-clouds and meshes using SFM and MVS pipeline. This requires a substantial number of images of the same object to creates its geometric depth and normal maps. The Brutus Dataset contains a set of 24 video files capturing a subject and its surroundings. If we wish to capture a 3D-model of an object we need cameras that are focusing on that object, which is not true in the case of Brutus dataset which aims to give us a 6DOF recording of an event. Due to the scarcity of images that have correlation with the main subject, it is hard to construct a 3D-point-cloud from the given dataset. For reference, the Gerrard Hall dataset that comes with COLMAP contains 100 images and it still can't fully capture the geometry like trees around the hall.

Apart from having a very few images to work with, we also require the intrinsic data for each camera, which will help us make a better representation of the rig using triangulation. Having intrinsic data will help us with triangulation and un-distortion (hypothetically). However, getting a better triangulation doesn't necessarily help as it just may get us a proper rig shape, but doesn't help us conserve most of the geometry during sparse reconstruction (there are many points scattered around, and not many points that make up the subject). Additionally, the Brutus dataset exclusively has humans as subjects, which is a hard subject compared to rectangular buildings.

Since I do not have camera intrinsic parameters for Brutus, I will draw an analogy from Deep View Dataset [5] which has a similar rig with 46 cameras whose intrinsic values are provided. Even when the intrinsic parameters are available, and a correct convex formation (expected) is observed in triangulation (Figure 16) we still can't preserve enough detail to even mesh the Welder (the subject of the scene).

I conclude that the number of images (which cannot be changed) and the configuration of the cameras (in a convex hull which point in different directions) is the main reason we cannot formulate a proper mesh, say if we were to have a concave hull pointing at the subject, we would have a substantially higher chance of making a coherent 3D-point-cloud and mesh.

## 7. References


[1] Johannes L. Schönberger, "COLMAP Source Code," *https://github.com/colmap/colmap Github*.

[2] M. Broxton *et al.*, "A Low Cost Multi-Camera Array for Panoramic Light Field Video Capture," in *SIGGRAPH Asia 2019 Posters*, New York, NY, USA: ACM, Nov. 2019, pp. 1–2. doi: 10.1145/3355056.3364593.

[3] "COLMAP Documentation," *https://colmap.github.io/index.html*. GithubIO (accessed Apr. 08, 2023).

[4] J. L. Schönberger and J.-M. Frahm, "Structure-from-Motion Revisited," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[5] Michael Broxton *et al.*, "Deep View Video Dataset," *Github*, 2020.