SFM-MVS to Mesh

Roderick Lan
1706751

Abdullah Khadelli
1722102

***Abstract*—Structure From Motion (SFM) and Multiview Stereo (MVS) are computer vision algorithms commonly found in a wide range of domains and use cases, one of the most prevalent being photogrammetry. In this project, we investigate the ideal methodology and object properties for photogrammetry of specific objects using SFM, and explore an altered approach to the feature extraction and feature matching steps in the algorithm. Additionally, we evaluate the effectiveness of two algorithms for surface reconstruction on our SFM-MVS (point cloud) outputs.**

1. Introduction

One of the most common applications of Structure From Motion (SFM) and Multiview Stereo (MVS) is in photogrammetry. These two techniques are complementary to each other, and result in a dense point cloud reconstruction of an object (or environment) from image data. However, in many fields where photogrammetry is used, a 3D mesh is likely more useful than a point cloud is. As such, time is often spent cleaning resultant point clouds and converting them to meshes, when the brunt of this process could be automated and simplified. Furthermore, SFM can be relatively slow for many datasets. The general outline of SFM algorithms involves feature extraction, feature matching, triangulation and bundle adjustment. Each of these steps can take considerable time, however there is not much to change with regards to triangulation and bundle adjustment. On the other hand, feature extraction (SIFT [1] based) and feature matching can be altered to be computationally faster at a minimal cost to accuracy. In a nutshell, this project report will explore proposed alterations to feature extraction and feature matching in SFM, investigate the ideal methodologies and object types for this altered approach, and evaluate algorithms relating to point cloud processing and mesh reconstruction. The source code for the project can be found on GitHub.

2. Theory and Technical Details

This project largely focuses on alterations to SFM and post processing SFM-MVS output for mesh construction. The general workflow and steps taken in this project can be seen in Figure 1 and are the following:

1. We implement a custom BRISK-SIFT feature extraction stack for SFM
2. We implement feature matching using approximate nearest neighbors for speed, with outlier removal from Lowe's Ratio Test [1] and MAGSAC++ [2]
3. We implement a feature processing method to further simplify key point matches
4. Our features and matches are used to build sparse and dense reconstructions via COLMAP [3]
5. Resultant point clouds are processed and used to generate meshes

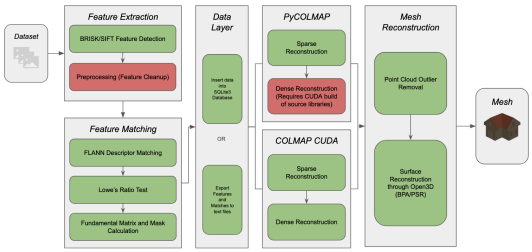


Figure 1. Pipeline/Project Workflow

2.1. SFM-MVS

SFM algorithms generally consist of the following steps: feature extraction, feature matching, triangulation, and bundle adjustment.
Traditionally, feature extraction in SFM algorithms use Lowe's SIFT approach for both feature detection and description [1]. Detection in SIFT involves a Gaussian pyramid-like structure of octaves for scale space representation. Each octave of scale space contains a set of Gaussian convolved images separated by a scale factor, and each octave uses a down-sampled image relative to the previous octave. The extrema (within a given window) of a difference-of-Gaussian (DoG) (Figure 2) function applied to adjacent Gaussian images within octaves is used to detect stable key points in scale space. Mathematically, the scale space is defined as a function $L(x, y, \sigma)$, produced from a variable-scale Gaussian, $G(x, y, \sigma)$ and the input image $I(x, y)$ [1]:

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y)$$

where $*$ is the convolution operation. The DoG function takes the form:

$$D(x, y, \sigma) = (G(x, y, k\sigma) - G(x, y, \sigma)) * I(x, y)$$
$$= L(x, y, k\sigma) - L(x, y, \sigma)$$

with some constant multiplicative scale factor k. DoG approximates the normalized Laplacian of Gaussian [4] , whose local extrema was shown to correspond to stable key points/blobs. SIFT descriptors [1] take the gradient directions over a spatial region around feature points. A histogram of these gradient directions (orientations) is normalized and invariant against rotation, scale, and brightness. The descriptors are stored as a 128-dimensional vector containing these histogram values corresponding to the gradient magnitudes and are known for their distinctiveness. BRISK feature detection [5] builds on a computationally optimized version of FAST 9-16 adapted for scale invariance via a simplified representation of scale space (compared to SIFT [1]). Due to the simplified scale space representation and use of a computationally efficient FAST 9-16 detection algorithm, BRISK is considerably faster than SIFT while still detecting high quality key points. The set of detected points with BRISK is similar to many SIFT detected points. Additionally, since both approaches revolve around scale, rotation, and illuminance invariance, and since description is independent of detection, SIFT descriptors can be easily applied to BRISK key points. For the purposes of this project, we rely on OpenCV's implementation of both SIFT [6] and BRISK [7]. Feature matching in SFM (with SIFT descriptors) is done by identifying a nearest neighbor (based on L2 distance) between descriptors in 2 views [1]. This is efficiently approximated through [8].

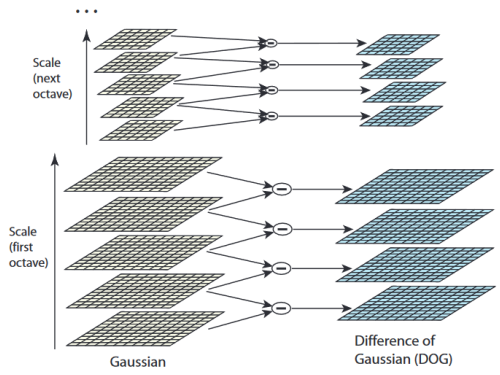
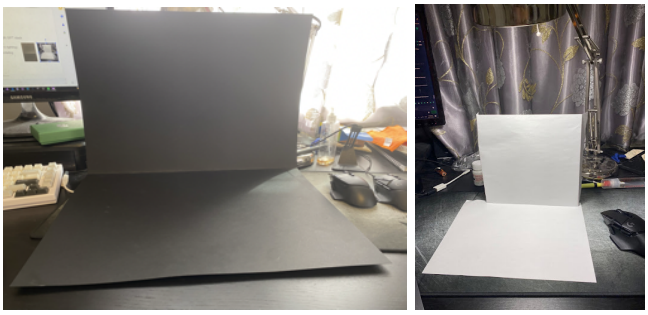


Figure 2. Difference of Gaussians and Sift Scale Space
Source: Adapted from [1]

Lowe's ratio test [1] compares the distance measures of the 2 nearest neighbors and rejects matches where the ratio between the distances from these neighbors is greater than or equal to some value l (ie. $d_1 \geq l \cdot d_2$), eliminating false matches. [9]'s MAGSAC++ [2], an improved version of RANSAC, further refines these matches by finding inliers during a fundamental matrix estimation between randomly sampled matches. Two view geometry estimation, triangulation and bundle adjustment, as well as multiview stereo, are handled through COLMAP [3] [10] [11], resulting in sparse and dense point cloud reconstructions (with estimated surface normals).

2.1.1. Image Capturing. Image capturing for SFM was primarily focused on small objects with three things in mind:

1. Brightness constancy (via controlled lighting)
2. Angle variety and image overlap
3. Limited background noise (via. lightboxes/backdrops as shown in Figure 3) (*note: this is not present for some earlier datasets*)



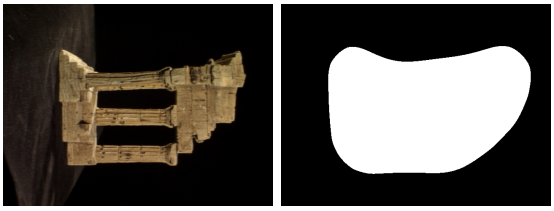
(a) Black backdrop (b) White backdrop

Figure 3. Lightbox-like backdrops

Angle variety and image overlap was achieved by taking images in short intervals, between either camera motion or object motion (ie. physical rotation). Overlap for large objects (ie. buildings) was achieved through capturing every 3-5 steps.
The overall methodology for capturing smaller objects throughout the project was based on [12] and [13].

2.2. Feature Preprocessing

During feature extraction, further processing can be done to isolate key points on objects. We apply a combination of image filtering, blurring, and canny edge detection to build a binary mask (show in Figure 4) covering the object of interest. This mask is used to discard key points which

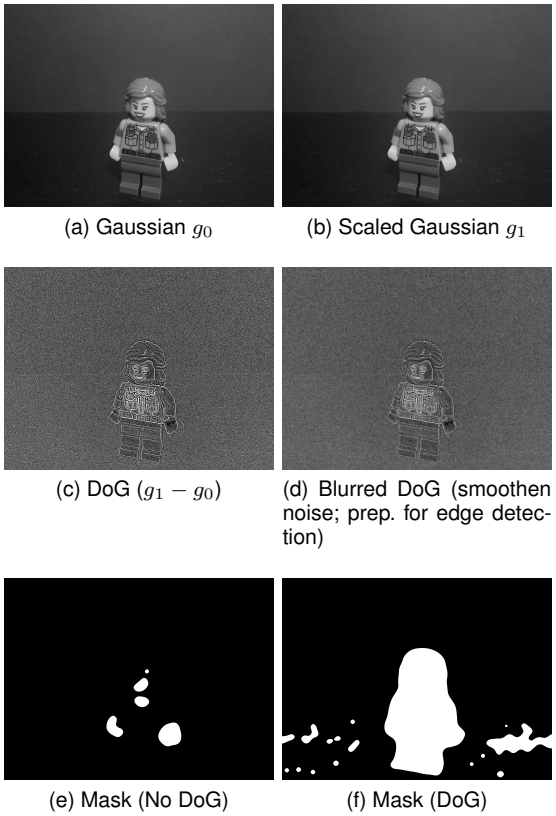


(a) Initial Image (b) Binary mask

Figure 4. Binary Mask. Dataset from [12]

are determined to be background noise. Preparation for canny edge detection involves noise reduction through a gaussian blur for effective detection [14]. Though OpenCV's

implementation [15] applies a 5×5 Gaussian kernel, further reduction is often necessary. Additionally, we apply a DoG (shown in Figure 5) based approach to accentuate edges and blobs for lower texture objects and noisier datasets. Sobel



(a) Gaussian g_0 (b) Scaled Gaussian g_1

(c) DoG ($g_1 - g_0$) (d) Blurred DoG (smoothen noise; prep. for edge detection)

(e) Mask (No DoG) (f) Mask (DoG)

Figure 5. Difference of Gaussian based preprocessing approach. (σ-scale factor $k = 2$) Inspired by [1].

filters applied in the horizontal and vertical directions allow us to obtain edge gradients and directions [15] as follows:

$$G = \sqrt{G_x^2 + G_y^2}, \quad \theta = \arctan\left(\frac{G_y}{G_x}\right)$$

where G is the edge gradient and θ is the edge direction. Non-maximum suppression removes unwanted pixels that may not be part of an edge [15]. Hystersis thresholding determines edge criterion via a user defined lower-upper bound (l, u):

i $G > u \rightarrow$ edge
ii $G < l \rightarrow$ not an edge
iii $G \in [l, u] \rightarrow$ classified based on connectivity to edge pixels

We use the edge image, $E(x, y)$, to build a mask by:

1. Convolving a Gaussian kernel on E
2. Binary thresholding is applied to the convolved image (forming the initial mask M)
3. Apply two median blur filters to M

We can apply our mask to the source frame and exclude key points that are irrelevant to the image. Though this does not directly affect the dense point cloud reconstruction from MVS, it can decrease matches from background noise, thus improving match performance and the accuracy of two view geometry. A general outline of our feature preprocessing approach is shown in Figure 6 and an example is how in Figure 7

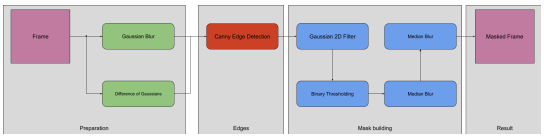
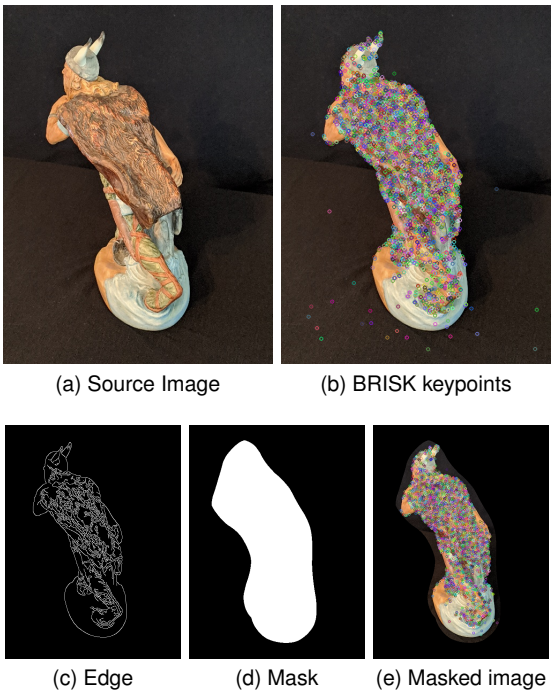


Figure 6. Feature preprocessing pipeline. Dataset from [13]



(a) Source Image (b) BRISK keypoints

(c) Edge (d) Mask (e) Masked image

Figure 7. Feature preprocessing example

2.3. Point Cloud Processing

This project uses Open3D's implementation of the Ball-Pivot Algorithm (BPA) [16] [17] and Poisson Surface Reconstruction (PSR) [18] [17], two widely used mesh reconstruction techniques. BPA involves a "ball" with a user-defined radius, r, which pivots along a manifold surface of the 3D object. Triangles are formed when the ball touches 3 points without "falling through", eventually leading to a triangle mesh once all points are considered [16]. This process can be repeated with various different radii to improve accuracy. An implementation trick we discovered with radii tuning was defining them as an array of scaled average neighbor distances. This approach often gave the best results with respect to BPA. PSR [18] formalizes surface reconstruction as a poisson problem and creates very

smooth surfaces that robustly approximate noisy data using a point cloud, its normals, and an octree of a user specified depth. However, PSR assumes the point cloud represents an object rather than the environment surrounding it, as such its robustness to noise is limited when background points are introduced. Additionally, BPA [16] is generally not as robust as PSR for noise in an object point cloud, however it handles background noise well (given enough separation between points). To alleviate the effect of noise, we use Open3D's radial and statistical point cloud outlier removal algorithms to automate background noise removal [19], improving overall mesh quality for both approaches.

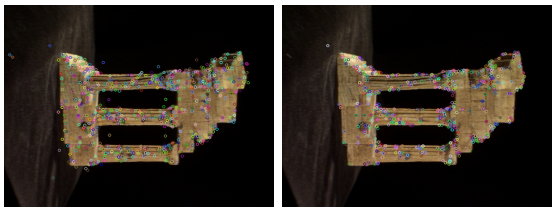
3. Results

3.1. Initial Evaluation

Initial evaluation involved a direct benchmark of BRISK and SIFT feature detectors for various datasets. (Benchmarks were collected on the same Ryzen 9 5900HS 16GB RAM system on six datasets) Both Figures 8 and 9, as well

(a) SIFT Features (b) BRISK Features

Figure 8. SIFT vs BRISK detected features (ssd box)



(a) SIFT Features (b) BRISK Features

Figure 9. SIFT vs BRISK detected features (templeRing [12])

as Table 1, demonstrate a substantial difference in overall ability. Referring to Figure 8, we see that SIFT detection is able to find nearly all the points BRISK detection finds, as well as more subtle features like the entire edge of the box (which BRISK largely fails to detect). In cases where these types of low-texture, subtle features are important, our BRISK-SIFT SFM may not be viable compared to traditional SIFT-SIFT. However, figure 9 suggests that the more "nuanced" feature points may also be more akin to noise (detections on black background) and unneeded. Furthermore, BRISK is considerably faster than SIFT (∼2.3x faster) as seen in Table 2 and still produces a large number of high quality points in textured regions (ie. areas with text in Figure 8) (surprisingly, BRISK yields 29% more points than SIFT does on average).

TABLE 1. KEYPOINT EXTRACTION (NO. OF KEYPOINTS EXTRACTED)

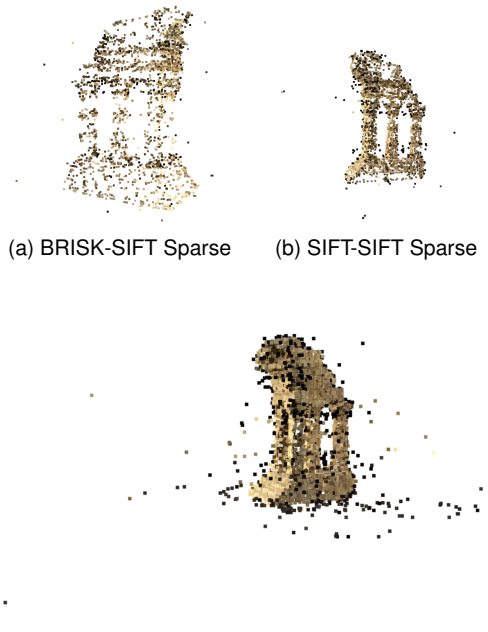
Dataset	SIFT	BRISK
Temple Ring [12]	39945	42789
Viking [13]	58604	145037
Lego5	60767	53192
Keyboard	219924	204750
Ace Coffee Building	164260	212897

TABLE 2. KEYPOINT EXTRACTION (TIME (S))

Dataset	SIFT	BRISK
Temple Ring [12]	3.8003	1.2922
Viking [13]	5.3268	3.2936
Lego5	87.4832	16.2658
Keyboard	44.5370	12.0966
Ace Coffee Building	18.0179	7.2542

3.2. COLMAP Comparison

We evaluate the results of BRISK-SIFT feature extraction against both traditional SIFT-SIFT (on our SFM stack) and results from COLMAP on the "templeRing" [12] dataset. In Figure 10, we see that our SFM stack (for



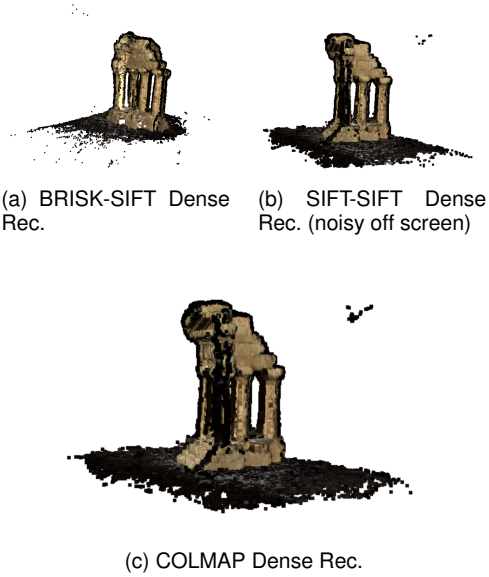
(a) BRISK-SIFT Sparse (b) SIFT-SIFT Sparse

(c) COLMAP Sparse

Figure 10. Sparse Reconstruction of our SFM stack (with BRISK-SIFT and SIFT-SIFT) and full COLMAP

both BRISK-SIFT and SIFT-SIFT) results in a sparser point

cloud compared to pure COLMAP, however considerably less visible noise as well. With BRISK-SIFT in particular, our SFM stack results in a sparser point cloud compared to other methods (likely due to less detections), however it also contains the least amount of noise in the reconstructed points. In Figure 11, we see that the dense reconstruction from COLMAP's MVS [11] with our BRISK-SIFT SFM stack is nearly identical to both our SIFT-SIFT and COLMAP with the exception of a small gap. This is likely due to the difference in matched features, causing slightly different camera and pose estimation. Nonetheless, all these point clouds provide a reasonably accurate 3D represenation of the object.



(a) BRISK-SIFT Dense Rec. (b) SIFT-SIFT Dense Rec. (noisy off screen)

(c) COLMAP Dense Rec.

Figure 11. Dense Reconstruction of our SFM stack (with BRISK-SIFT and SIFT-SIFT) and full COLMAP

3.3. Feature Preprocessing Evaluation

We evaluate our feature preprocessing (a method of key point cleanup during feature extraction) with a benchmark test (same as Table 1 and Table 2) and a visual test on the "lego5" dataset.

TABLE 3. FEATURE PREPROCESSING (FT. EXTRACTION TIME (S))

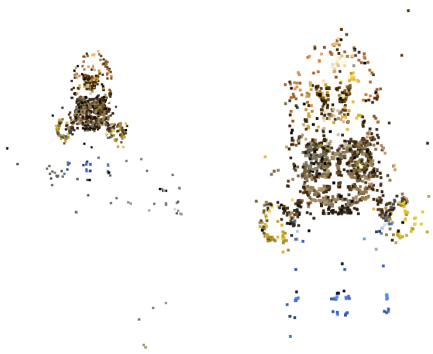
Dataset	No Img Preproc.	Image Preproc.
Temple Ring [12]	1.2922	2.3747
Viking [13]	3.2936	4.6455
SSD Box	5.8452	9.6800
Lego5	16.2658	30.7978
Keyboard	12.0966	22.5780
Ace Coffee Building	7.2542	12.3442

TABLE 4. FEATURE PREPROCESSING (FT. MATCHING TIME (S))

Dataset	No Img Preproc.	Image Preproc.
Temple Ring [12]	23.600	22.989
Viking [13]	88.046	87.223
SSD Box	38.348	36.611
Lego5	104.346	101.194
Keyboard	178.749	178.099
Ace Coffee Building	77.786	72.757

3.3.1. Benchmarks. Table 4 shows that feature preprocessing provides a marginal benefit for matching ($\sim$2% reduction in time on average). However, Table 3 shows that it also causes a $\sim$70% increase in feature extraction time (on average), significantly lengthening the overall search process for a slight refinement ($\sim$5% less matches on average). Still, it is worth noting that the increased feature extraction times (Table 3) are still faster than using SIFT (Table 2)

3.3.2. Visual Test. To further evaluate feature preprocessing, we apply our DoG implementation to the "lego5" dataset. The resulting reconstructions are shown in Figures 12 and 13. From Figure 12, we see that there is much more



(a) No Ft. Preproc. (b) Ft. Preproc.

Figure 12. Sparse Reconstruction without vs with ft. preprocessing

noise in the non-feature preprocessed sparse reconstruction.

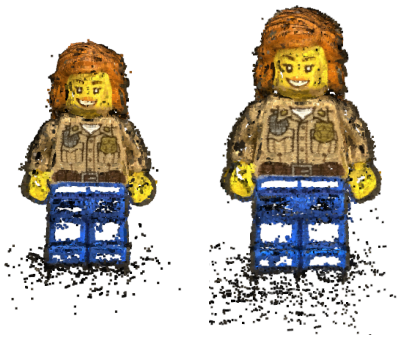
The dense reconstruction of the feature preprocessed dataset (shown in Figure 13) also appears to be denser than the non-preprocessed dataset (note the)

3.4. Noisy Data

Acknowledgments

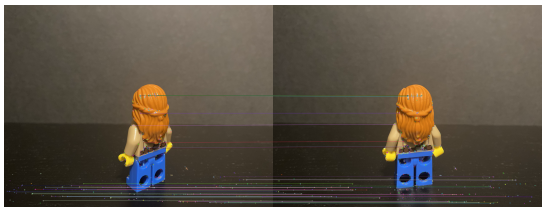
References

[1] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International journal of computer vision*, vol. 60, pp. 91–110, 2004.

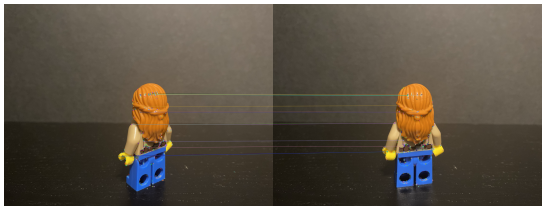


(a) No Ft. Preproc. (b) Ft. Preproc.

Figure 13. Dense Reconstruction without vs with ft. preprocessing



(a) No Ft. Preproc.



(b) With Ft. Preproc.

Figure 14. Feature Matches without vs with Feature Preproccessing on lego5 sample image

[2] D. Barath, J. Noskova, M. Ivashechkin, and J. Matas, "Magsac++, a fast, reliable and accurate robust estimator," 2019.

[3] "Colmap," https://github.com/colmap/colmap, 2024.

[4] T. Lindeberg, "Feature detection with automatic scale selection," *International journal of computer vision*, vol. 30, pp. 79–116, 1998.

[5] S. Leutenegger, M. Chli, and R. Y. Siegwart, "Brisk: Binary robust invariant scalable keypoints," in *2011 International Conference on Computer Vision*, 2011, pp. 2548–2555.

[6] "Opencv: cv::sift class reference," docs.opencv.org. [Online]. Available: https://docs.opencv.org/4.x/d7/d60/classcv_1_1SIFT.html

[7] "Opencv: cv::brisk class reference," docs.opencv.org. [Online]. Available: https://docs.opencv.org/4.x/de/dbf/classcv_1_1BRISK.html

[8] "Opencv: cv::flannbasedmatcher class reference," docs.opencv.org. [Online]. Available: https://docs.opencv.org/3.4/dc/de2/classcv_1_1FlannBasedMatcher.html

[9] "Opencv: Usac: Improvement of random sample consensus in opencv," docs.opencv.org. [Online]. Available: https://docs.opencv.org/4.x/de/d3e/tutorial_usac.html

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

[11] J. L. Schönberger, E. Zheng, M. Pollefeys, and J.-M. Frahm, "Pixelwise view selection for unstructured multi-view stereo," in *European Conference on Computer Vision (ECCV)*, 2016.

[12] "vision.middlebury.edu/mview/data," vision.middlebury.edu. [Online]. Available: https://vision.middlebury.edu/mview/data/

[13] R. Shilliday, "sfm/datasets/viking at master · rshilliday/sfm," GitHub, 03 2020. [Online]. Available: https://github.com/rshilliday/sfm/tree/master/datasets/Viking

[14] J. Canny, "A computational approach to edge detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-8, no. 6, pp. 679–698, 1986.

[15] "Canny edge detection," docs.opencv.org. [Online]. Available: https://docs.opencv.org/4.x/da/d22/tutorial_py_canny.html

[16] F. Bernardini, J. Mittleman, H. Rushmeier, C. Silva, and G. Taubin, "The ball-pivoting algorithm for surface reconstruction," *IEEE Transactions on Visualization and Computer Graphics*, vol. 5, no. 4, pp. 349–359, 1999.

[17] "Surface reconstruction - open3d 0.18.0 documentation," www.open3d.org. [Online]. Available: https://www.open3d.org/docs/release/tutorial/geometry/surface_reconstruction.html

[18] M. Kazhdan, M. Bolitho, and H. Hoppe, "Poisson surface reconstruction," in *Proceedings of the fourth Eurographics symposium on Geometry processing*, vol. 7, no. 4, 2006.

[19] "Point cloud outlier removal — open3d 0.9.0 documentation," www.open3d.org. [Online]. Available: https://www.open3d.org/docs/0.9.0/tutorial/Advanced/pointcloud_outlier_removal.html