POLITECNICO DI TORINO

Master's Degree in Mathematical Engineering

Master's Degree Thesis

Segmenting dynamic points in 3D scenarios

Supervisors

Prof. Tatiana TOMMASI

Prof. Chiara PLIZZARI

Candidate

Francesco BORGNA

March 2024

Summary

Ma che dici Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Acknowledgements

ACKNOWLEDGMENTS

“HI”
Goofy, Google by Google

Table of Contents

List of Tables

List of Figures

Acronyms

SfM

Structure from Motion

pcd

Pointcloud

Part I

Related Works

1. Epic Kitchens
2. Epic Fields
3. Photogrammetry
4. COLMAP
5. NeRF
6. NeuralDiff
7. Monocular Depth Estimation
8. motion estimation
9. (N3F)
10. (Gaussian Splatting)

Chapter 1

Datasets

Chapter 2

Neural Rendering

Chapter 3

Photogrammetry

Photogrammetry is the science and technology of obtaining reliable information about physical objects and the environment through the process of recording, measuring and interpreting photographic images and patterns of electromagnetic radiant imagery and other phenomena[1].

It comprises all techniques concerned with making measurements of real-world objects features from images. Its utility range from the measuring of coordinates, quantification of distances, heights, areas and volumes, preparation of topographic maps, to generation of digital elevation models and orthophotographs. The functioning rely mostly on optics and projective geometry rules.

As first assumption we have the modellization of the camera as a simplified version of itself: the *Pinhole Camera*. As in the first designed cameras(*camera obscura*), in the Pinhole Camera world's light is captured through a pinhole and then projected into the *focal plane*.

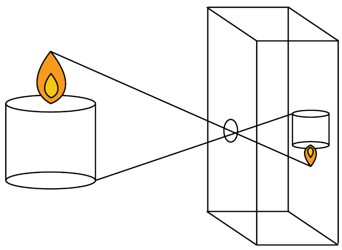

Figure 3.1: Pinhole Camera

A Pinhole camera is characterized by two collection of parameters:

- **Extrinsic** parameters: gives us information on location and rotation in the world.

- **Intrinscic** parameters: gives us internal property such as: focal length, field of view, resolution...

These parameters can be rewritten in their corresponding matrices:

$$Intrinsic = K = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

where:

- f_x, f_y are are the *focal lengths* of the camera in the x and y directions, they are needed to keep the image aspect ratio.
- c_x, c_y are the coordinates of the *principal point* (the point where the optical axis intersects the image plane).

$$Extrinsic = \begin{bmatrix} \mathbf{R}_{3x3} & \mathbf{t}_{3x1} \\ 0_{1x3} & \mathbf{1}_{1x1} \end{bmatrix}$$

where:

- $\mathbf{R}_{3x3}$ is a rotation matrix (Aggiungi le tre combinazioni...)
- $\mathbf{t}_{3x1}$ is a translation vector

Extrinsic matrix is also known as the 4x4 transformation matrix that converts points from the world coordinate system to the camera coordinate system.

Exploting homogeneous coordinates we can rewrite the image capturing process of a specific camera as the combination of its characteristic matrices:

$$\begin{bmatrix} u \\ v \\ z \end{bmatrix} = \begin{bmatrix} f_x & s & c_x & 0 \\ 0 & f_y & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{R}_{3x3} & \mathbf{t}_{3x1} \\ 0_{1x3} & \mathbf{1}_{1x1} \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}$$

3.1 Structure from Motion: SfM

Structure from motion (SfM) is the process of estimating the 3-D structure of a scene from a set of 2-D images. SfM is used in many applications, such as 3-D scanning, augmented reality, and visual simultaneous localization and mapping [2].

We can compute SfM in different ways depending on the data and tools at our disposal. Factors such as the **type** and the **number** of cameras can imply using

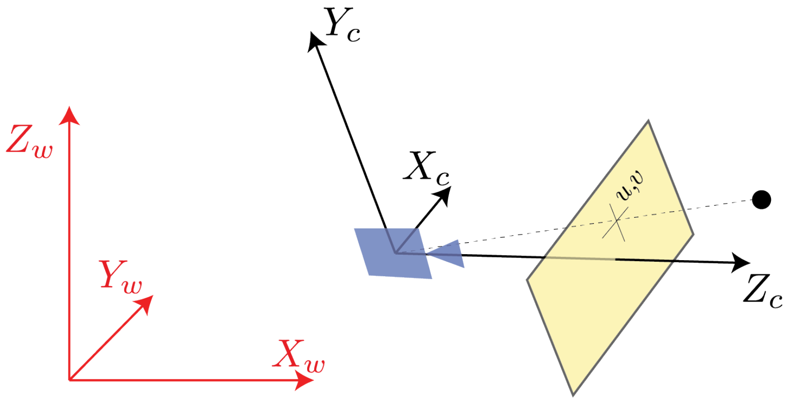

Figure 3.2: Reference systems

different methods. Also the **ordering** of the frames may be relevant for a succesful reconstruction.

The most basic scenario consists in two images captured from the same camera from two different point of view:

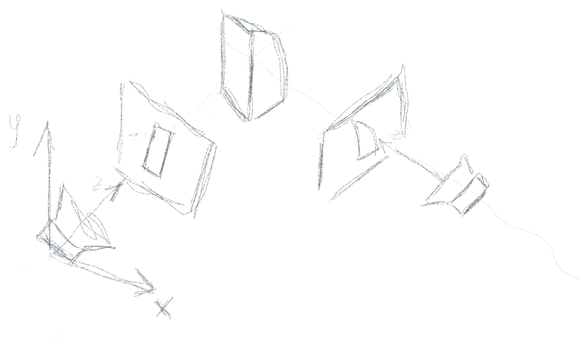

Figure 3.3: Basic SfM Scenario

In this case the 3-D structure can be recovered *up to scale*, meaning that the relations between the obtained points is faithful to the reality, but it can differ from the real size by a *scale* factor. The scale factor could be retrieved if we know the size of an object in the scene or by having informations from other sensors.

In the basic scenario the method consists in the following main steps:

1. **Feature Detection** and **Matching**

2. Estimating **Fundamental matrix**

3. Estimating **Essential Matrix** from Fundamental matrix

4. Estimating **Camera Pose** from Essential Matrix

3.1.1 Feature Detection and Matching

In order to find corresponding points in two images we need to detect some points of interest in each image. Instead of trying to match each pixel, which would be computational inefficient and possibly misleading due to color, it has been shown succesful to use *feature points*. Feature points are relavant points which encapsulate the local appeareance of its surrounding pixels which is invariant under changes in illumination, translation, scale and in-plane rotation. Good features are *unique*, *can be easily tracked* and *can be easily compared*.

A practical example could be to imagine how us humans find correspondances in pictures, looking at image 3.4. If we would be asked to find the exact position of the six patches in the underlying image, the job would be easier for patches E and F, being corners they have less possible misleading correspondances, as happens instead for patches A or B.

Figure 3.4: Feature Detection Example [3]

In the same way feature extractor works for computers. Two of the first algorithms are infact based on corners detection: **Harris Corner Detector** and **Shi-Tomasi Corner Detector**. Depending on the differences between the two images to be compared, different algorithms have been developed. Here I report some of them as reported in [3]:

- **SIFT**: Harris corner detector is not good enough when scale of image changes. Lowe developed a breakthrough method to find scale-invariant features and it is called SIFT
- **SURF** (Speeded-Up Robust Features): faster SIFT version
- **FAST Algorithm for corner detection** All the above feature detection methods are good in some way. But they are not fast enough to work in real-time applications like SLAM. There comes the FAST algorithm, which is really "FAST".
- **BRIEF**(Binary Robust Independent Elementary Features)SIFT uses a feature descriptor with 128 floating point numbers. Consider thousands of such

features. It takes lots of memory and more time for matching. We can compress it to make it faster. But still we have to calculate it first. There comes BRIEF which gives the shortcut to find binary descriptors with less memory, faster matching, still higher recognition rate.

- **ORB(Oriented FAST and Rotated BRIEF)** Open source alternative to SIFT and SURF released by OpenCV.

Figure 3.5: SIFT features extracted

After having found these points of interest, we need to match them from the different pictures. This step is also known as *Feature Matching*. The most basics algorithms are:

- **Brute-Force Matcher**: It takes the descriptor of one feature in first set and is matched with all other features in second set using some distance calculation. And the closest one is returned.

- **FLANN**: which stands for *Fast Library for Approximate Nearest Neighbors*. It contains a collection of algorithms optimized for fast nearest neighbor search in large datasets and for high dimensional features. It works faster than BFMatcher for large datasets.

[1]

3.1.2 Estimating Fundamental and Essential Matrices

The Fundamental (F) and the Essential (E) matrices allow to relate the projection of a point located in space from one image to the other. These matrices are based

[1]This is a footnote with additional information.

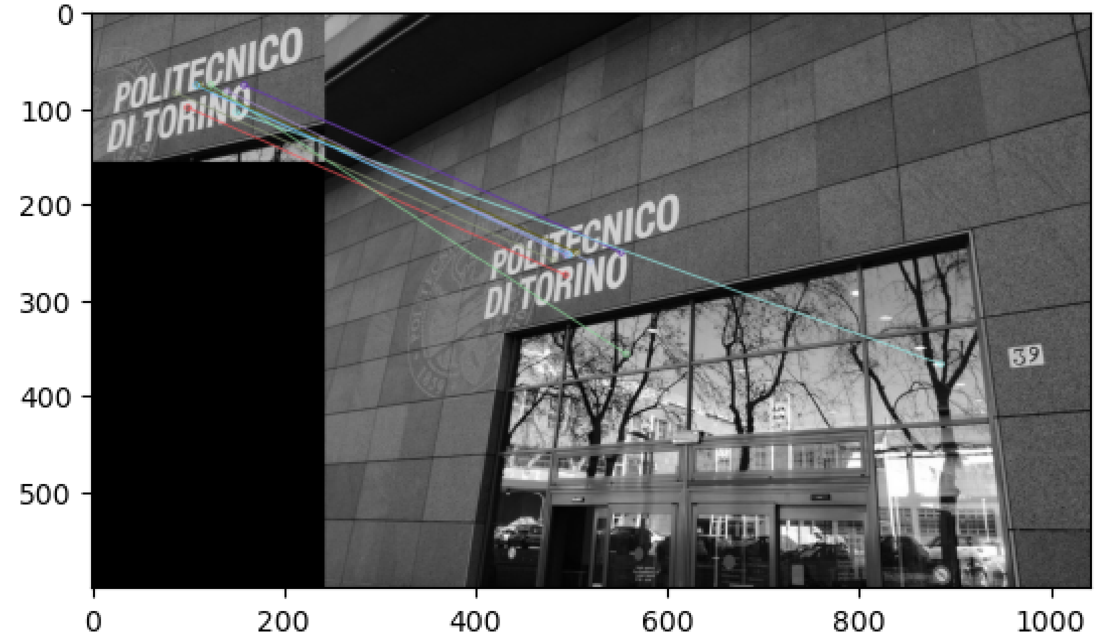

Figure 3.6: Features correspondence computed using BruteForce Matcher.

on the so called *Epipolar Geometry*, which describes the relationship between two images. As we can see from Figure(3.7), given two cameras C_l and C_r the following

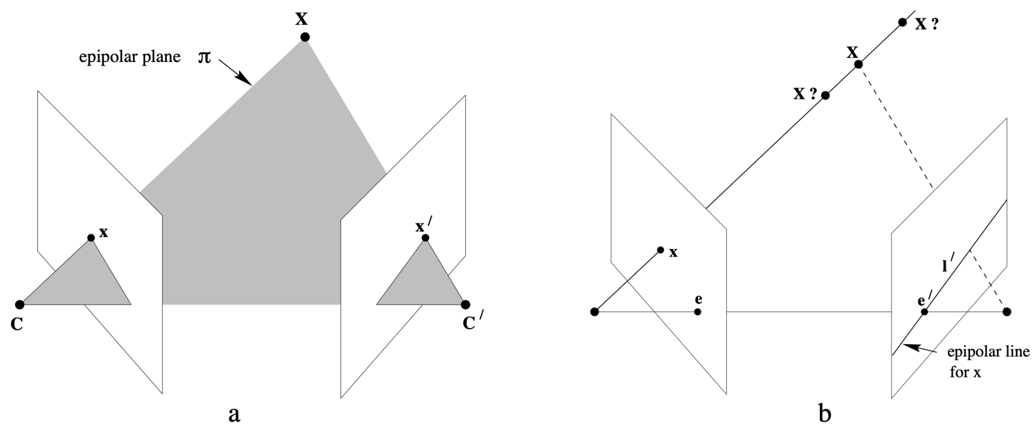

Figure 3.7: Point correspondence geometry.

definitions can be given:

- The **Epipole**: which is the point of intersection of the **baseline**(the line that connects the two camera centers C_l and C_r) with the image plane. In

Figure(3.7) denoted by e_i.

- An **Epipolar plane**, which is any plane containing the baseline.
- An **Epipolar line** which is the intersection of any epipolar plane with the image plane.

Now that we have a clear understanding of the underlying geometry, let's proceed to see the actual derivation of the two matrices.

Let's consider the following scenario,reported in Figure(3.8): we have a point X and two cameras C_l and C_r. The relative positions are respectively x_l and x_r, while the pixel projections are p_l and p_r. If we consider as reference the left camera, the position of the right one is shifted of a vector $\mathbf{t}$.

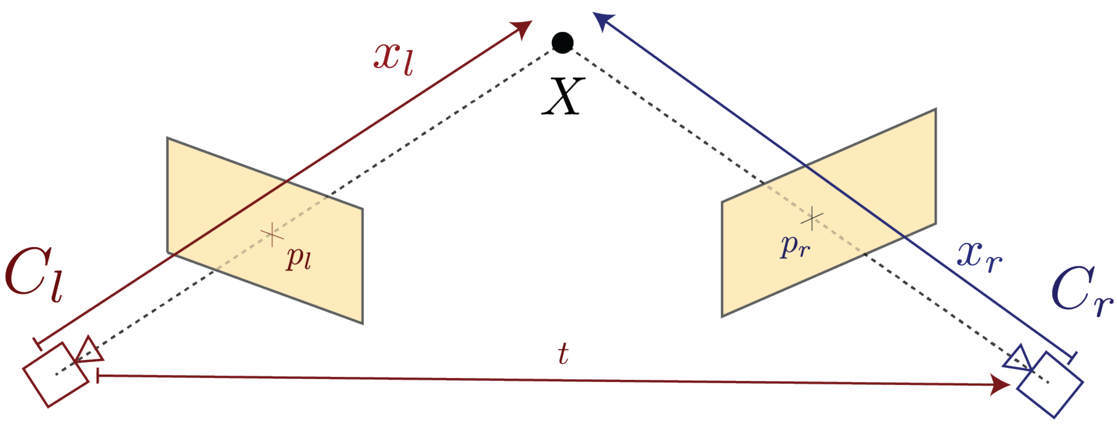

Figure 3.8

We can extract the Essential Matrix by making the following considerations:

$$x_l \cdot (t \times x_l) = 0 \tag{3.1}$$

but x_l can be written as:

$$x_l = Rx_r + t \tag{3.2}$$

So Equation(3.1) becomes:

$$x_l \cdot (t \times (Rx_r + t)) = x_l \cdot (t \times Rx_r) \stackrel{a}{=} x_l^T \, [t]_\times \, Rx_r = 0 \tag{3.3}$$

where equality (a) is due to the cross-product matrix notation[2]. We call Essential Matrix the term $[t]_\times R$, such that:

$$x_l^T [t]_\times R x_r = x_l^T E x_r = 0 \tag{3.4}$$

In a similar way we can get the Fundamental Matrix, by replacing the relative positions with the pixel positions. Recalling that the pixel positions are linked to the relative positions by:

$$p_l = \frac{1}{z_l} K_l x_l \quad p_r = \frac{1}{z_r} K_r x_r \tag{3.5}$$

where z_i are the focal distances.We can substitute in Eq.(3.4) and obtain:

$$p_l^T z_l K_l^{-1^T} E K_r^{-1} z_r p_r = 0 \quad z_l, z_r \neq 0 \tag{3.6}$$

Since z_i are constants can be simplified, obtaining:

$$p_l^T K_l^{-1^T} E K_r^{-1} p_r = p_l^T F p_r = 0 \quad z_l, z_r \neq 0 \tag{3.7}$$

where F is the Fundamental Matrix. We can thus write the relation between the two matrices:

$$E = K_l^T F K_r \tag{3.8}$$

3.1.3 Estimating Camera Pose from Essential Matrix

Since $[t]_\times$ is skew symmetric and R is orthonormal(since it is a rotation matrix), if we know the Essential Matrix we can decompose it in its components[3]using singular value decomposition.

---

[2]

$$\mathbf{a} \times \mathbf{b} = [\mathbf{a}]_\times \mathbf{b} = \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

[3]

$$[t]_\times = UW\Sigma U^T \tag{3.9}$$

$$R = UW^T V^T \tag{3.10}$$

$$W = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.11}$$

3.1.4 Multi-view Structure from Motion

Now that we have grasped the basics to find images correspondances, let's see how we can relate multiple image to recover the structure of a scene. The last stage is called *bundel adjustment* and it is a iterative algorithm used to adjust structure and motion parameters by minimising a cost function. The possible methods for bundle adjustment are:

- **Sequential**: which work by considering an additional images at each time, extending in this way the initial reconstruction.
- **Factorization**: work by computing camera poses and scene geometry using every image measurement at the same time.

Bundle Adjustment is needed since the image measurements are usually noisy. Minimising an appropriate cost function we can obtain a clean model as in a Linear Regression.

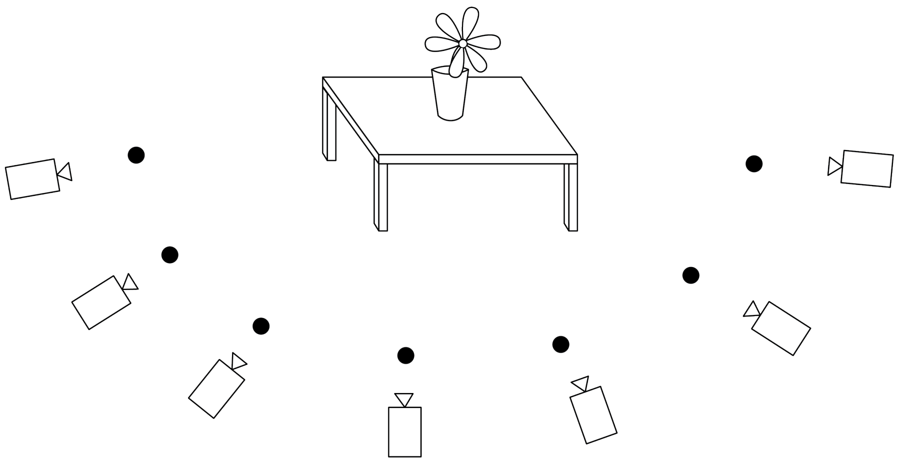

Figure 3.9: Multiple View Scenario.

3.1.5 COLMAP

Abandoning theory and moving on to practice, let's see COLMAP.

"COLMAP is a general-purpose, end-to-end image-based 3D reconstruction pipeline (i.e., Structure-from-Motion (SfM) and Multi-View Stereo (MVS)[4]) with

[4] *Even if they seem similar, these two pipelines have different goals. In fact, SfM The primary goal of SfM is to estimate the 3D structure of a scene and camera poses (positions and orientations) simultaneously from a set of 2D images taken from different viewpoints. MVS,on the other hand, is*

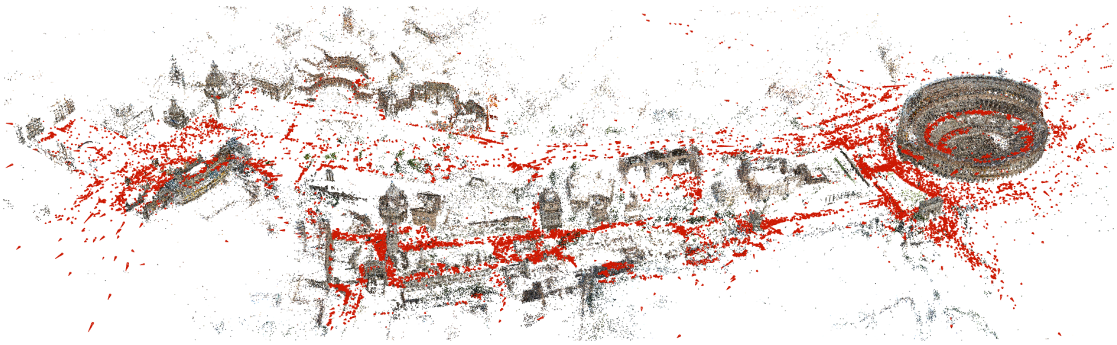

Figure 3.10: Building Rome in one day

a graphical and command-line interface. It offers a wide range of features for reconstruction of ordered and unordered image collections. The software runs under Windows, Linux and Mac on regular desktop computers or compute servers/clusters. COLMAP is licensed under the BSD License"[4].

The concepts we explained in the theoretical part are not always applicable In real world, or their results are not quite satisfying. In literature a vastity of authors tried and succeded in obtaining refined algorithms for specific scenarios. Anyway they still lacked a general-purpose method. With COLMAP they managed to compensate for the lack of generalization. The actual implementation and characteristics are explained from the authors in [5, 6].

Usage

The usage of COLMAP is pretty straight forward. After the creation of a project folder with a *images* directory containing the images we want to process we can simply launch the script reported in Listing(3.1).

Listing 3.1: Automatic COLMAP Reconstruction

```
#!/bin/bash
#The project folder contains a folder "images" with all images.

DATASET_PATH=/path/to/project
colmap automatic_reconstructor \
    --workspace_path $DATASET_PATH \
    --image_path $DATASET_PATH/images \
    --single_camera 1
```

specifically focused on dense 3D reconstruction. It involves estimating the depth or 3D coordinates for every pixel in the images to create a detailed 3D model with meshes and textures.

Anyway the program offers ample freedom to the single usage of the various steps need for *SfM* or *MVS*. Here I report the actual pipeline that I used for the extraction of the pointclouds for our experiments.

Listing 3.2: COLMAP Single Commands

```
#!/bin/bash
DATASET_PATH="/scratch/fborgna/EPIC_Diff/"

colmap feature_extractor \
    --database_path $DATASET_PATH/database.db \
    --image_path $DATASET_PATH/images \
    --ImageReader.single_camera 1 \

colmap exhaustive_matcher \
    --database_path $DATASET_PATH/database.db

mkdir $DATASET_PATH/sparse

colmap mapper \
    --database_path $DATASET_PATH/database.db \
    --image_path $DATASET_PATH/images \
    --output_path $DATASET_PATH/sparse \
    --camera_model SIMPLE_PINHOLE \

mkdir $DATASET_PATH/dense

colmap image_undistorter \
    --image_path $DATASET_PATH/images \
    --input_path $DATASET_PATH/sparse/0 \
    --output_path $DATASET_PATH/dense \
    --output_type COLMAP \
    --max_image_size 2000

colmap patch_match_stereo \
    --workspace_path $DATASET_PATH/dense \
    --workspace_format COLMAP \
    --PatchMatchStereo.geom_consistency true

colmap stereo_fusion \
    --workspace_path $DATASET_PATH/dense \
```

```
        --workspace_format COLMAP \
        --input_type geometric \
        --output_path $DATASET_PATH/dense/fused.ply

    colmap poisson_mesher \
        --input_path $DATASET_PATH/dense/fused.ply \
        --output_path $DATASET_PATH/dense/meshed-poisson.ply


```

3.1.6 Monocular Depth Estimation

Until now we have always considered scenarios in which multiple images were available from different points of view. What if we have just one image?

Appendix A

Galileo

```python
import os
os.system("echo 1")
```

$\mathcal{O}(n \log n)$

`numpy`

Appendix B

Math Notation

$$\mathbf{a} \times \mathbf{b} = [\mathbf{a}]_\times \mathbf{b} = \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

$$\mathbf{a} \times \mathbf{b} = [\mathbf{b}]_\times^T \mathbf{a} = \begin{bmatrix} 0 & b_3 & -b_2 \\ -b_3 & 0 & b_1 \\ b_2 & -b_1 & 0 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

Bibliography

[1] Wolf and Dewitt. 2000; McGlone, 2004 (cit. on p. 5).

[2] Mathworks 2023. URL: https://www.mathworks.com/help/vision/ug/structure-from-motion.html#:~:text=Structure%20from%20motion%20(SfM)%20is,localization%20and%20mapping%20(vSLAM). (cit. on p. 6).

[3] OpenCV 2024. URL: https://docs.opencv.org/4.x/df/d54/tutorial_py_features_meaning.html (cit. on p. 9).

[4] Colmap 2024. URL: https://colmap.github.io/index.html (cit. on p. 15).

[5] Johannes Lutz Schönberger and Jan-Michael Frahm. «Structure-from-Motion Revisited». In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016 (cit. on p. 15).

[6] Johannes Lutz Schönberger, Enliang Zheng, Marc Pollefeys, and Jan-Michael Frahm. «Pixelwise View Selection for Unstructured Multi-View Stereo». In: *European Conference on Computer Vision (ECCV)*. 2016 (cit. on p. 15).