Report

Author: 謝承翰, R14922068

Demo video

Q1: https://youtu.be/hF7ydz3GS2Q
Q2: https://youtu.be/avW175tTSyA

Environment

The machine's operating system is Ubuntu 22.04.2 and the kernel version is 6.5.0-41-generic.
I use **miniconda** to manage the environment. The version of miniconda is 25.7.0.
The installation steps can be found in https://www.anaconda.com/docs/getting-started/miniconda/install#linux-terminal-installer
After miniconda is installed, you can build the environment by the command below:

```
conda env create -f env.yml
```

How to run

You can run the following commands to verify the program (for problem 2):

```
conda activate 3DCV

python 2d3dmathcing.py
```

Note that the `data/` and `transform_cube.py` must be at the same directory as the `2d3dmathcing.py` file.

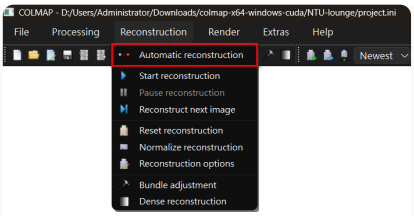
Problem 1

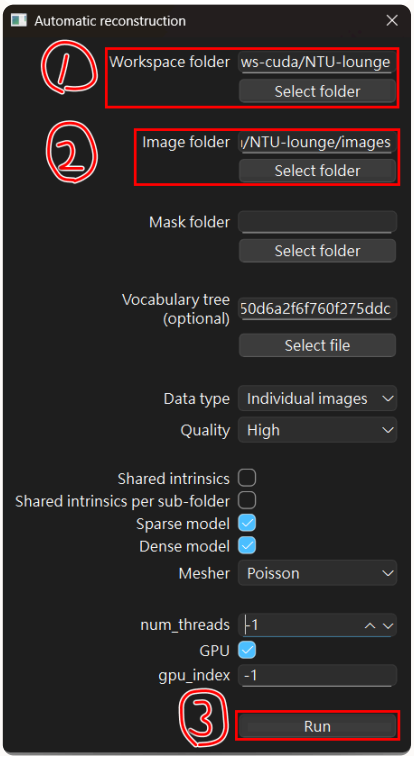
The procedure can be summarized roughly as following:

1. Use COLMAP to generate 3D point cloud, camera pose and trajectory.
2. Use the information calculated by COLMAP to build dense 3D point cloud, 3D triangle mesh model and textured model in *Agisoft Metashape*.
   The reason why I use Agisoft Metashape to build the dense 3D point cloud and triangle mesh model is because it provide high-quality 3D models than other tools but still easy to use.
   The scene I used is the lounge in International Youth Center (國青大樓) 7F.

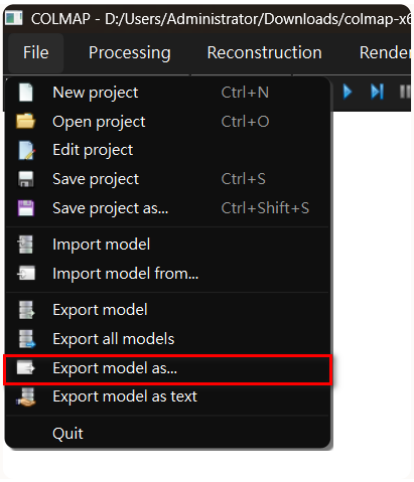
Detailed procedure

1. Create the project.
2. Reconstruction -> Automatic reconstruction. Choose the workspace and image folder. And wait for a while (~90 min).

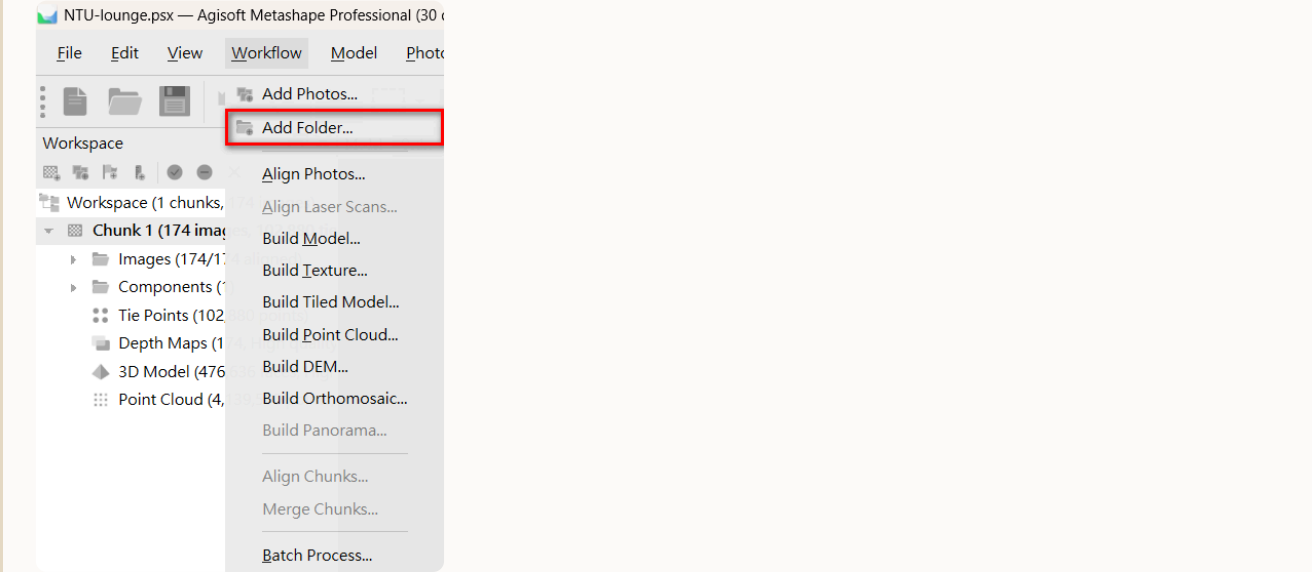




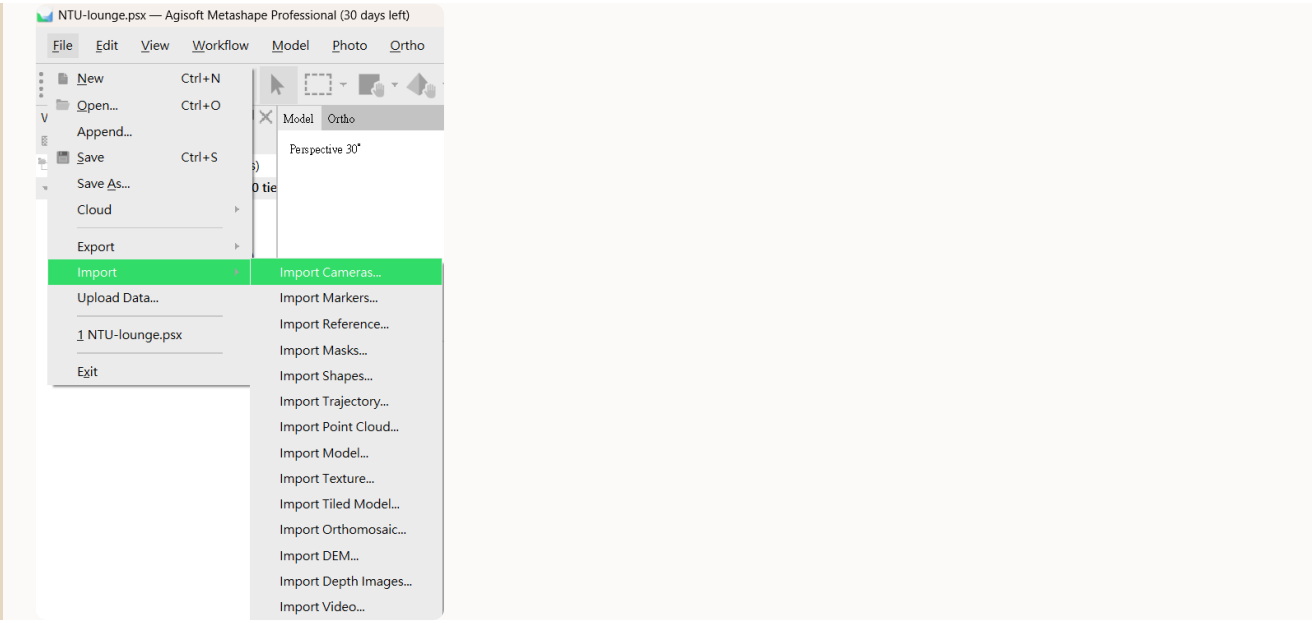
3. Export model as XXX.nvm



4. Open Agisoft Metashape and create project.
5. Add the images folder.

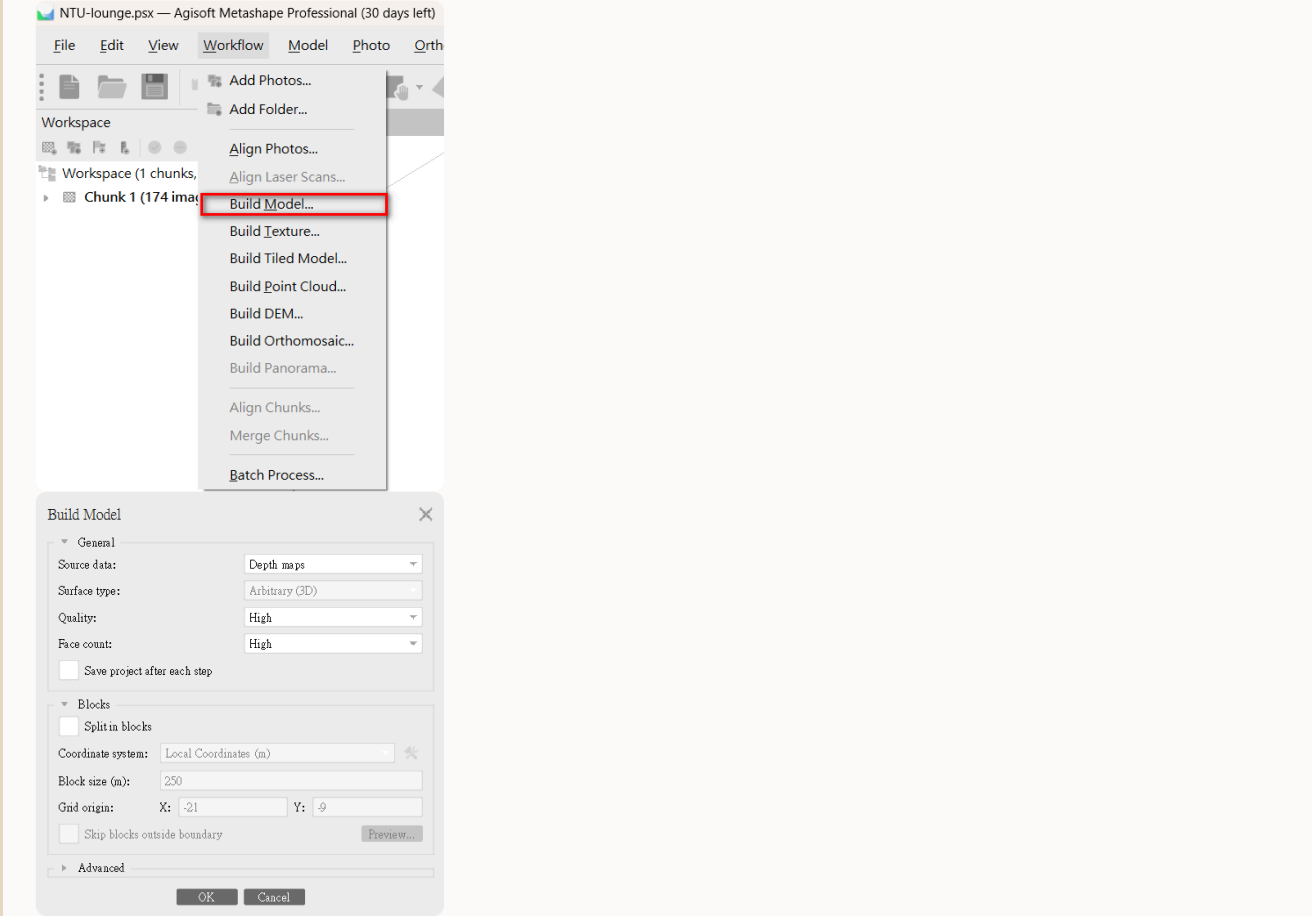


6. Import the XXX.nvm file saved in COLMAP before.

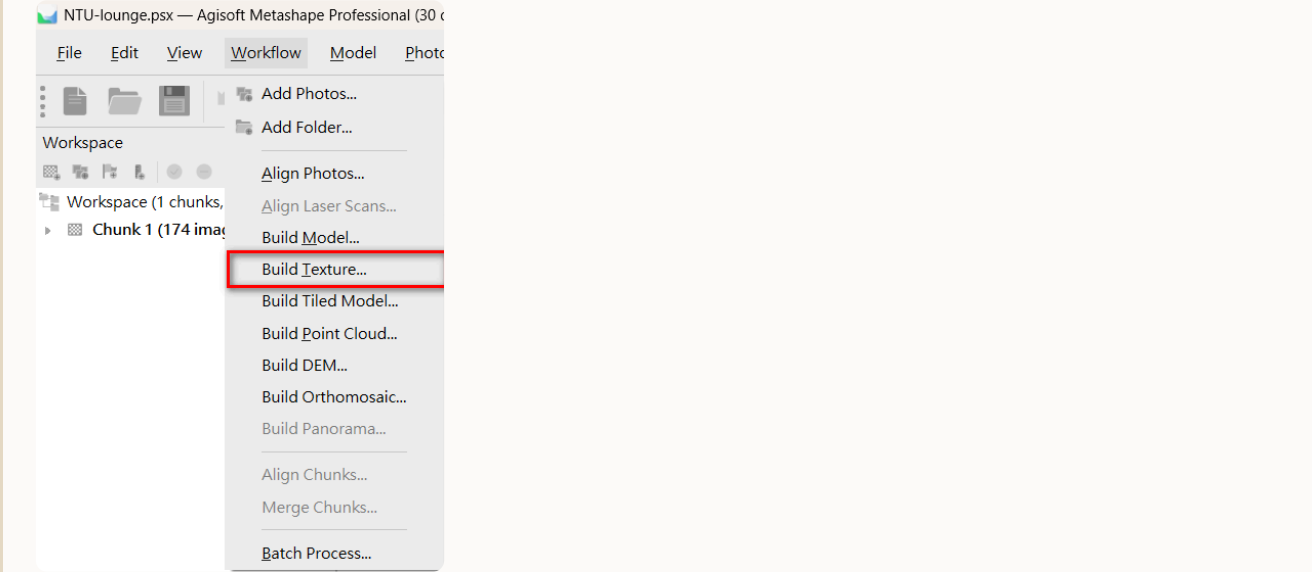


7. Build the dense point cloud.

8. Build the 3D triangle mesh model. Note the source data should be *depth maps*.

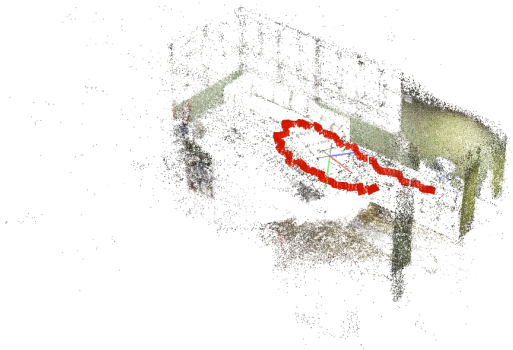


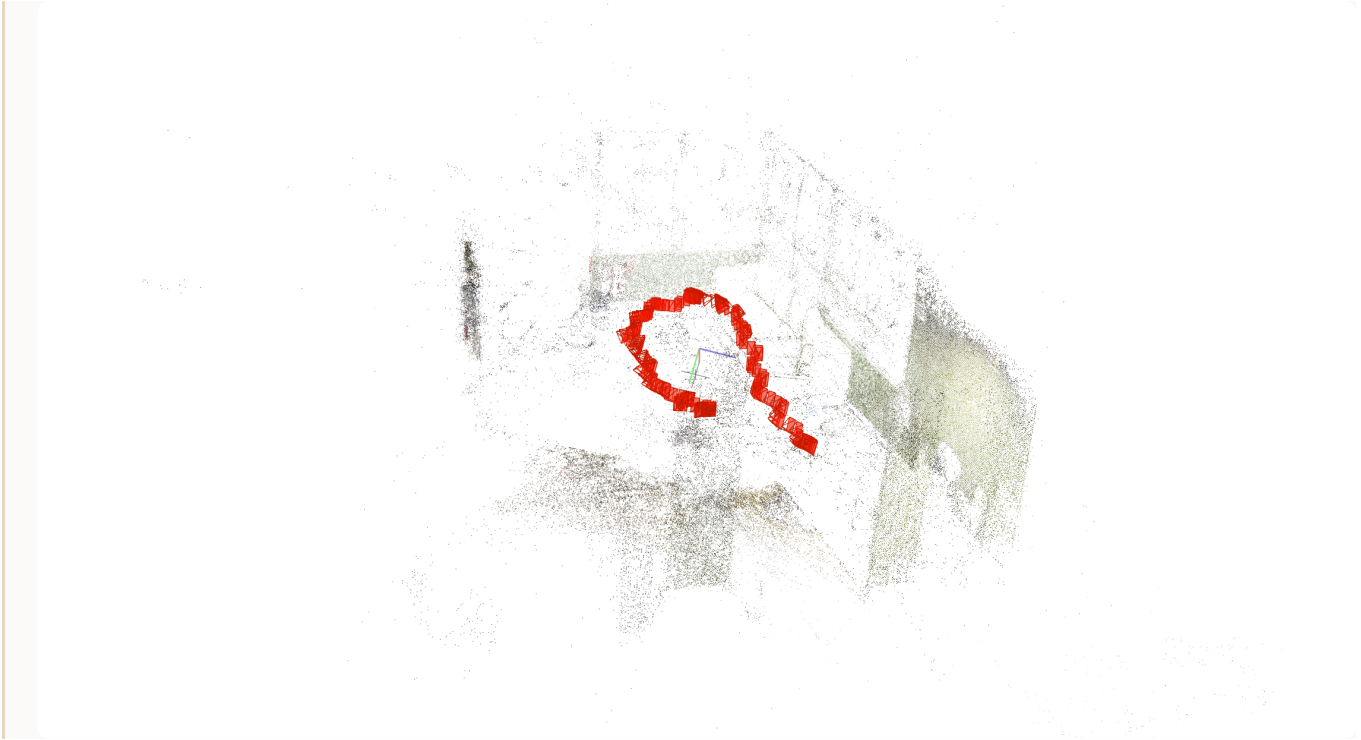
9. Build the texture model.



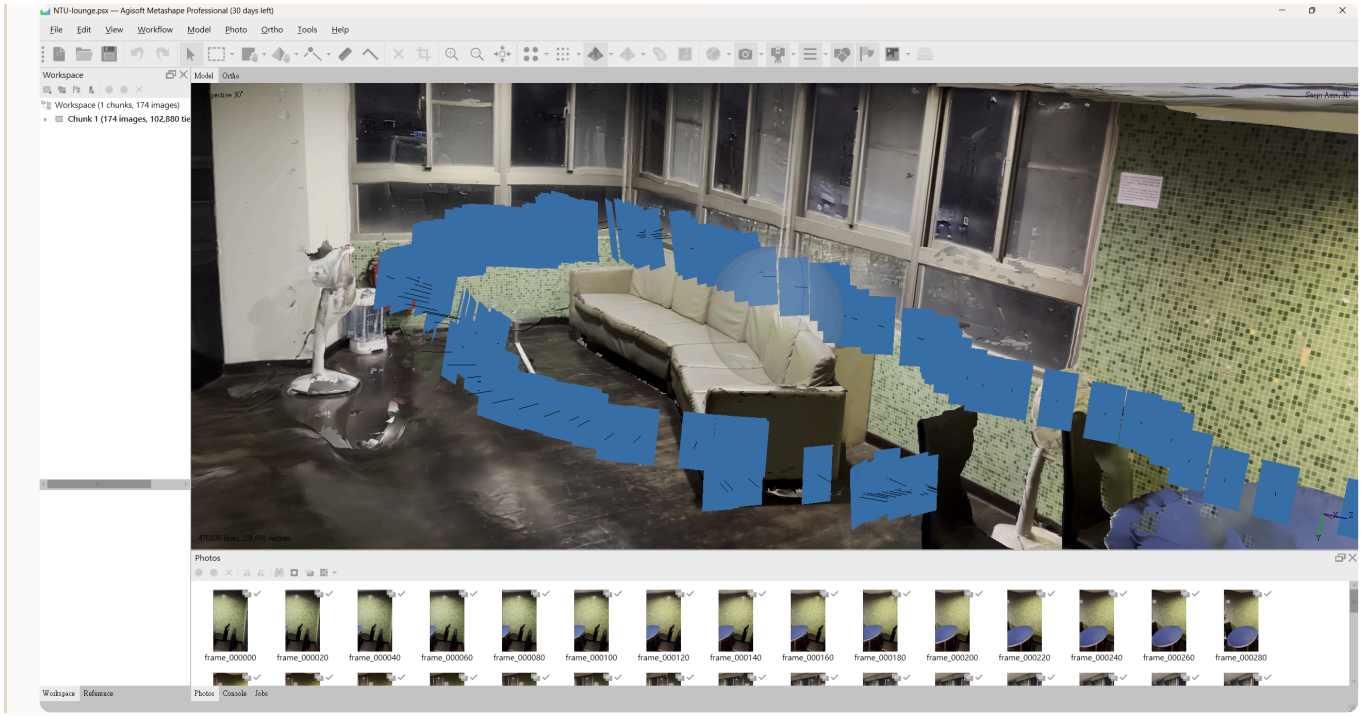
Resulting

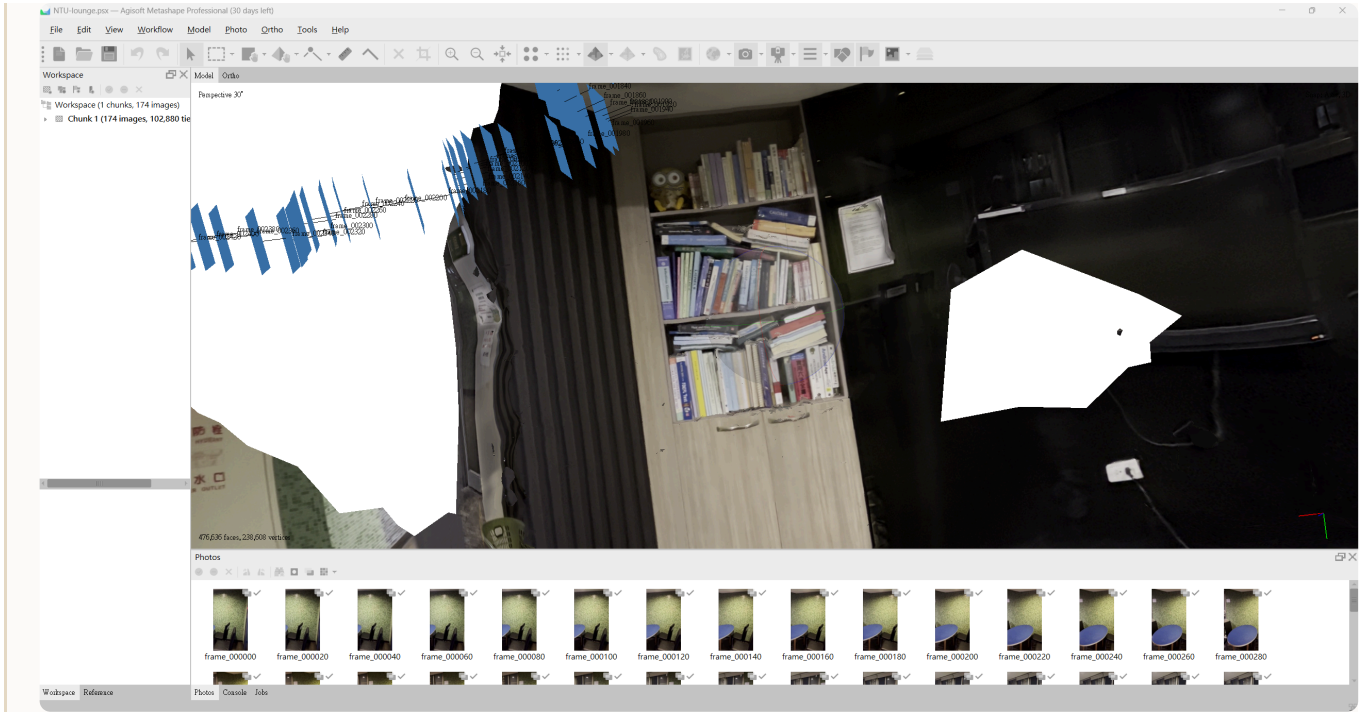
The resulting 3D point cloud, camera pose and trajectory generated by COLMAP:

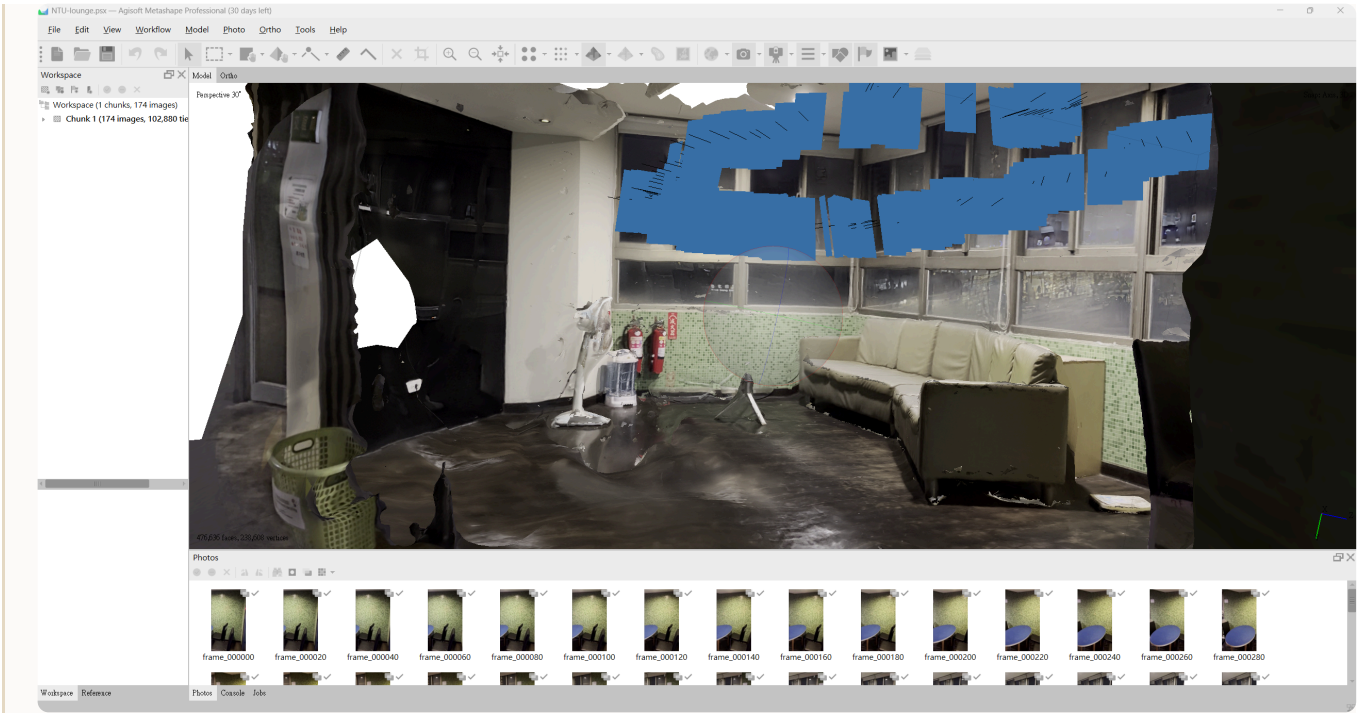


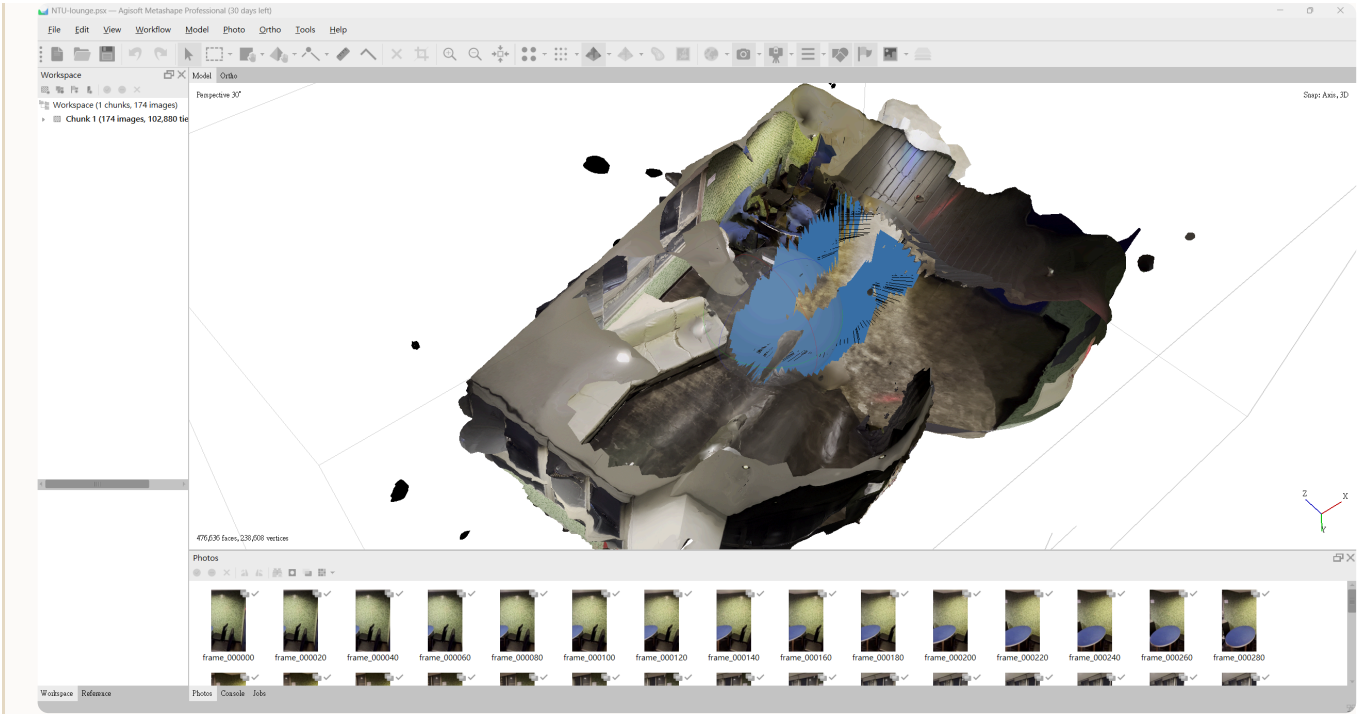


The resulting 3D triangle mesh model generated by Agisoft Metashape:


NTU-lounge.psx — Agisoft Metashape Professional (30 days left)
File Edit View Workflow Model Photo Ortho Tools Help
Workspace
Workspace (1 chunks, 174 images)
Chunk 1 (174 images, 102,880 tie
Model Ortho
Photos
frame_000000 frame_000020 frame_000040 frame_000060 frame_000080 frame_000100 frame_000120 frame_000140 frame_000160 frame_000180 frame_000200 frame_000220 frame_000240 frame_000260 frame_000280
Workspace Reference
Photos Console Jobs


NTU-lounge.psx — Agisoft Metashape Professional (30 days left)
File Edit View Workflow Model Photo Ortho Tools Help
Workspace
Workspace (1 chunks, 174 images)
Chunk 1 (174 images, 102,880 tie
Model Ortho
Perspective 30°
476,636 faces, 239,608 vertices
Photos
frame_000000 frame_000020 frame_000040 frame_000060 frame_000080 frame_000100 frame_000120 frame_000140 frame_000160 frame_000180 frame_000200 frame_000220 frame_000240 frame_000260 frame_000280
Workspace Reference
Photos Console Jobs





Problem 2

The whole process can be summerized as following:

1. Load 3D keypoints and 128D descriptors in `train.pkl`.
2. Load 2D keypoints and 128D descriptors in `images.pkl`.
3. For each image for validation,
    1. Use `cv2.FlannBasedMatcher().knnMatch` to match the descriptors.
    2. Use Lowe's ratio test to exclude some ambiguous descriptors.
    3. Use `cv2.solvePnPRansac` to solve the PNP problem.
    4. Get the *rotation matrix* and *translation vector* for the camera.
4. Convert rotation vector to quaternion.
5. Compute the rotation error and translation error for given ground truth rotation matrix and translation vector. And report the median of errors.
6. Visualize the camera position, pose, trajectory and 3D point cloud. (Q2-1)
7. Use the api functions of open3d to place a virtual cube.
8. Outputs the video, `./AR_camera_trajectory.mp4`. (Q2-2)

I tried some methods provided by openCV to solve PNP problem, the errors is organized as the following table:

	SOLVEPNP_ITERATIVE	SOLVEPNP_EPNP	SOLVEPNP_P3P
rotation error (median)	0.00008002	0.00027875	0.00027927
translation error (median)	0.00032930	0.00144841	0.00151145
total cost time	89s	88s	89s

It can be seen that iterative method is better than EPNP and P3P, and the time cost is basically the same, while the difference of error is not significant. It means in this dataset, the results are highly consistent between different methods, which implies the quality of this dataset is good (there are only few outliers).

As for the painter's algorithm, `open3d` provides an api to generate the depth image, if the pixel (x, y) on the depth image is 0, it is not visible. We can use that to determine the visibility.
Let $D_{\text{cube}}(x, y)$ be the cube's depth image, and $D_{\text{real}}(x, y)$ be the real photo's depth image. The painter's algorithm can be described as:

$$I_{\text{cube}}(x,y) = \begin{cases} \text{True} & , D_{\text{cube}}(x,y) > 0 \wedge D_{\text{real}}(x,y) = 0 \\ \text{True} & , D_{\text{cube}}(x,y) > 0 \wedge D_{\text{real}}(x,y) > 0 \wedge D_{\text{cube}}(x,y) < D_{\text{real}}(x,y) \\ \text{False} & , \text{otherwise.} \end{cases}$$

The first case means the cube is visible and the real scene is not visible, the second case means the they are both visible, but the cube is in the front of the scene.
And the resulting augmented reality photo is

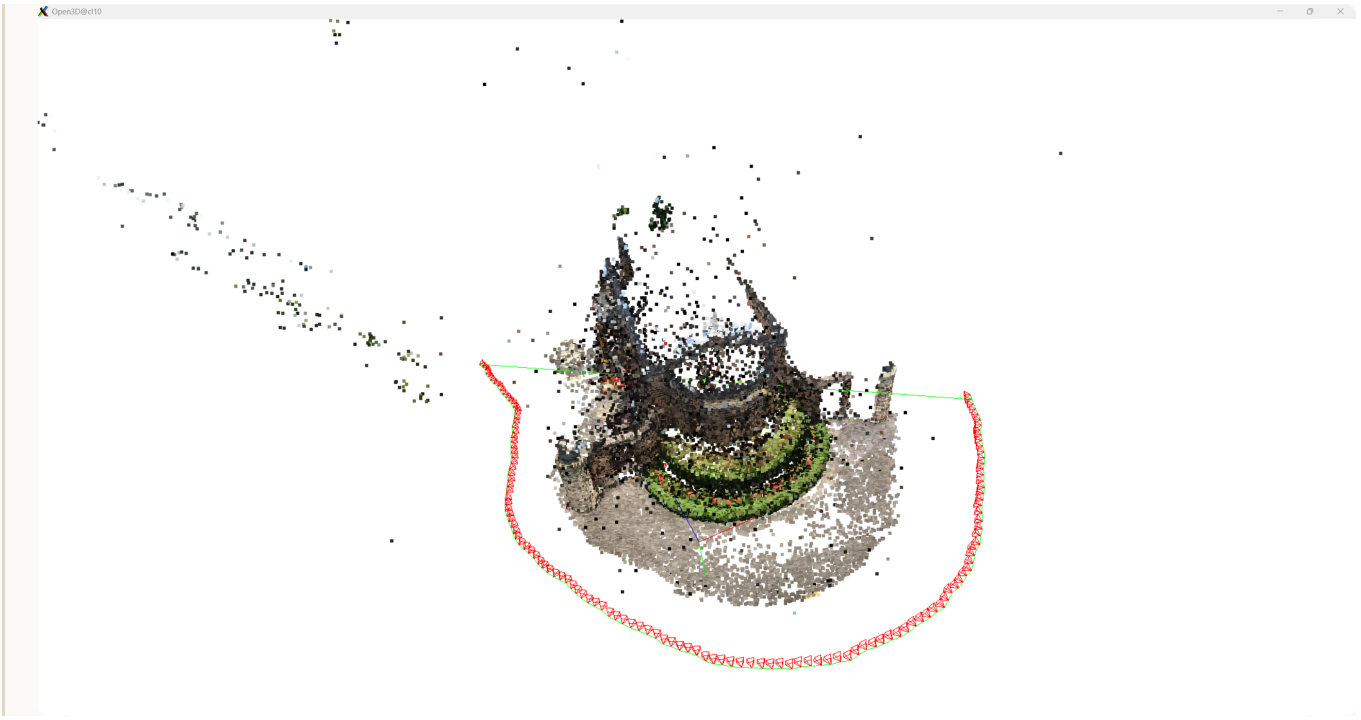
$$I_{\text{result}}(x,y) = \begin{cases} \text{cube}(x,y) & , I_{\text{cube}}(x,y) = \text{True}, \\ \text{real}(x,y) & , I_{\text{cube}}(x,y) = \text{False}. \end{cases}$$

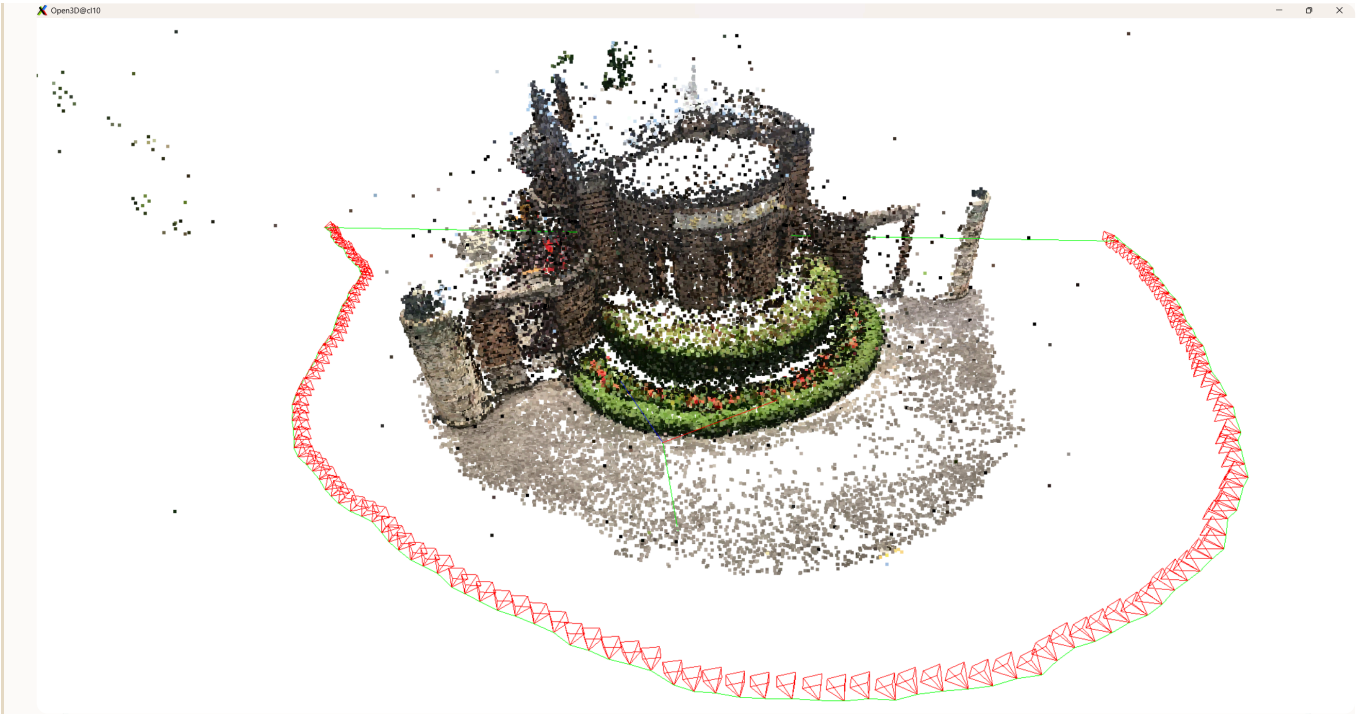
The pesudo implementation:

```
for each (c2w, photo_path) in (Camera2World_Transform_Matrixs, photo_paths):
    set the camera pose by inverse(c2w)

    depth_real = render_depth(scene_without_cube)
    depth_cube = render_depth(cube)

    real_img = load(photo_path)

    # painter's algorithm
    for each pixel (x, y):
        if depth_cube(x,y) > 0:
            if depth_real(x,y) == 0 or depth_cube(x,y) < depth_real(x,y):
                visible_cube(x,y) = True
            else:
                visible_cube(x,y) = False
        else:
            visible_cube(x,y) = False

    for each pixel (x, y):
        if visible_cube(x,y):
            result(x,y) = img_cube(x,y)
        else:
            result(x,y) = real_img(x,y)

    output(result)
```

The visualization results of camera position, pose, trajectory and 3D poinst cloud.


Open3D@cl10


Open3D@cl10



Some screenshots of AR video:


國立臺灣大學

LLM's contribution

1. Show the process of solving PNP problem.
2. Explain usage of open3d api. Generate the implementation of AR video.
3. Recommand the tools for converting point cloud to 3D triangle mesh model.
4. Show the steps to converting point cloud to 3D triangle mesh model.