# HW2 Camera Relocalization

## Due: 2023/10/24 11:59 AM

3DCV 2023

Email: 3dcv@csie.ntu.edu.tw

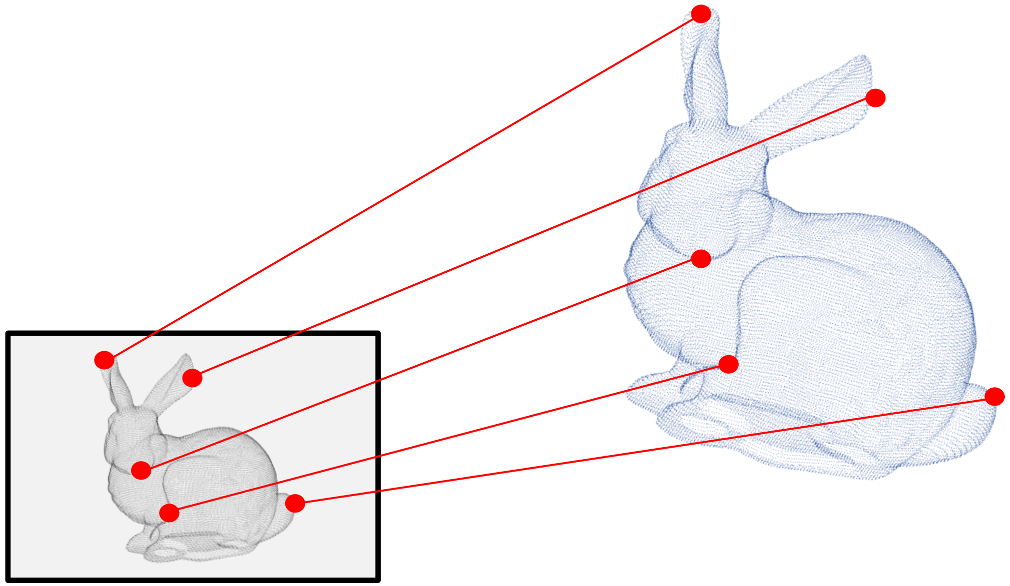
GitHub Classroom: https://classroom.github.com/a/cKfvp3Eo

GitHub Registration: https://forms.gle/R9JBiAAehcYyvoUu9

# Outline

The goal of this homework is to realize how a camera re-localization system works.

- Introduction
- Dataset
- Problem 1: 2D-3D Matching (Q1-1 ~ Q1-3)
- Problem 2: Augmented Reality (Q2-1 ~ Q2-2)
- Bonus List
- Grading Policy



# Introduction

- **Camera Relocalization**: Determine the camera pose from the visual scene representation. In other words, the scene is **seen** (and modeled) **beforehand**. Now, given a query image that is taken is this environment, we are able to find out where this image is taken.

Query Image

**Training Data**
**(Database Image)**

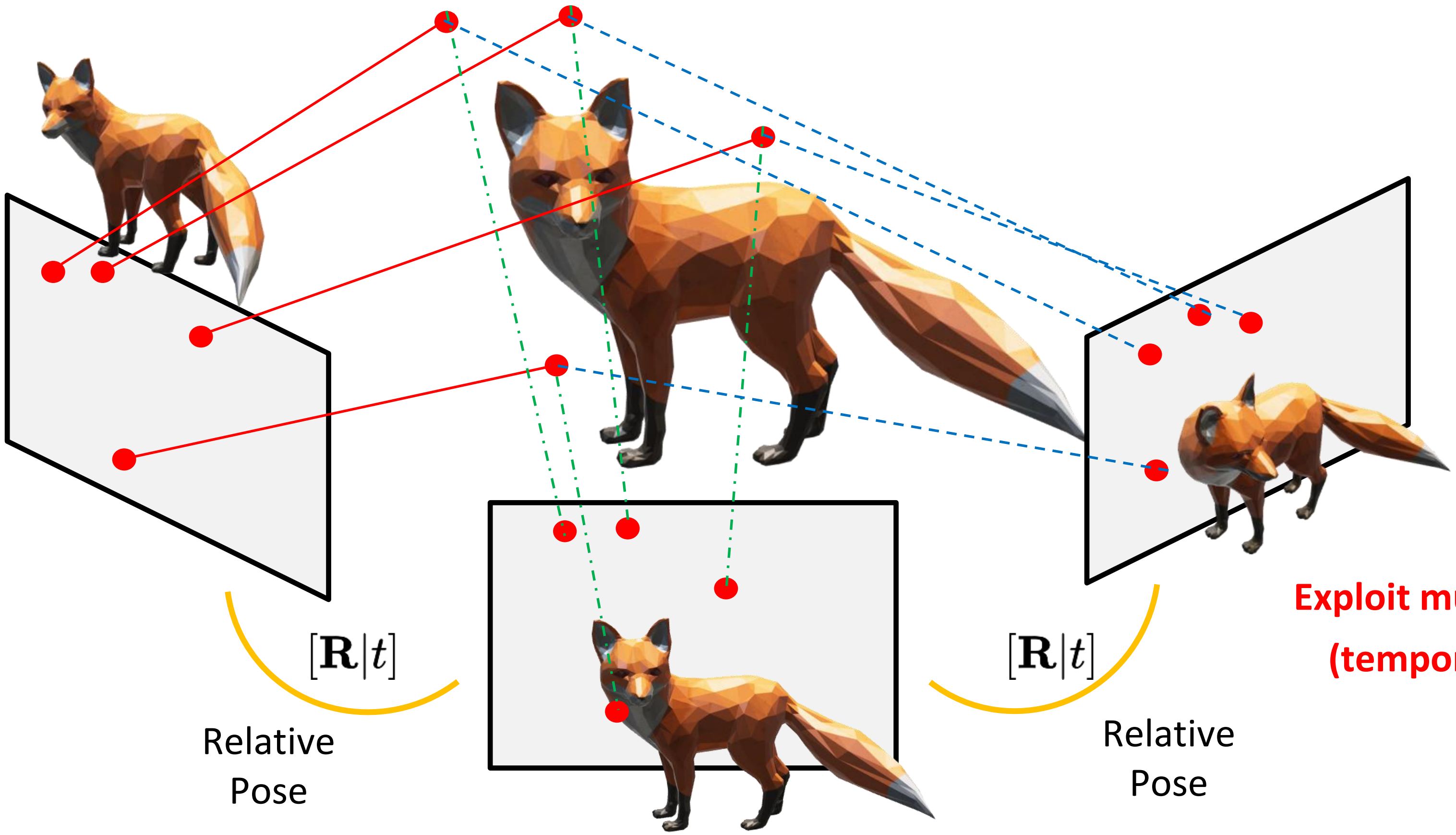
Projection

Recover

**Real World**
**(3D Information)**

# Introduction

- **One-shot relocalization**: focus on a finding the pose of still image.
- **Temporal camera relocalization**: estimates the poses of every frame in the video sequence



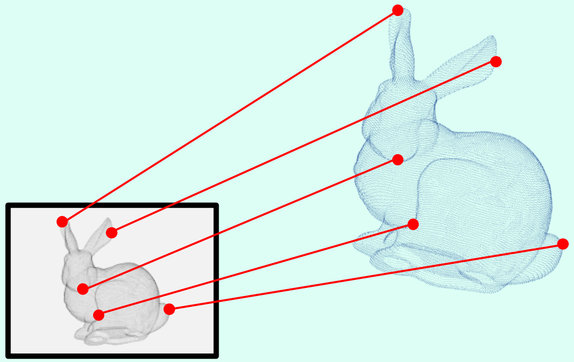

Exploit multi-view constraint (temporal information) !!

# Methodology

- Common strategies for camera relocalization. Note that there are some approaches utilize hybrid models to increase the efficiency and robustness.
- Metric localization can only be achieved by machine (or deep) learning models.

**2D-3D Matching Localization**

3D scene models with local feature descriptors



**Metric Localization (Absolute Pose Regression)**

regresses the position and orientation of the camera

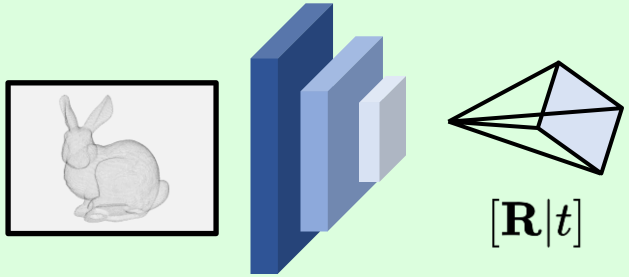
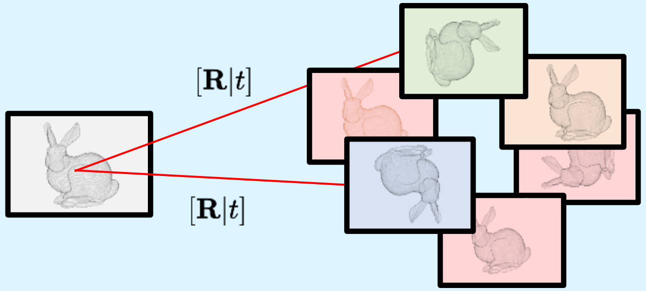



**Image Retrieval Localization**

pose approximation from top-k similar images



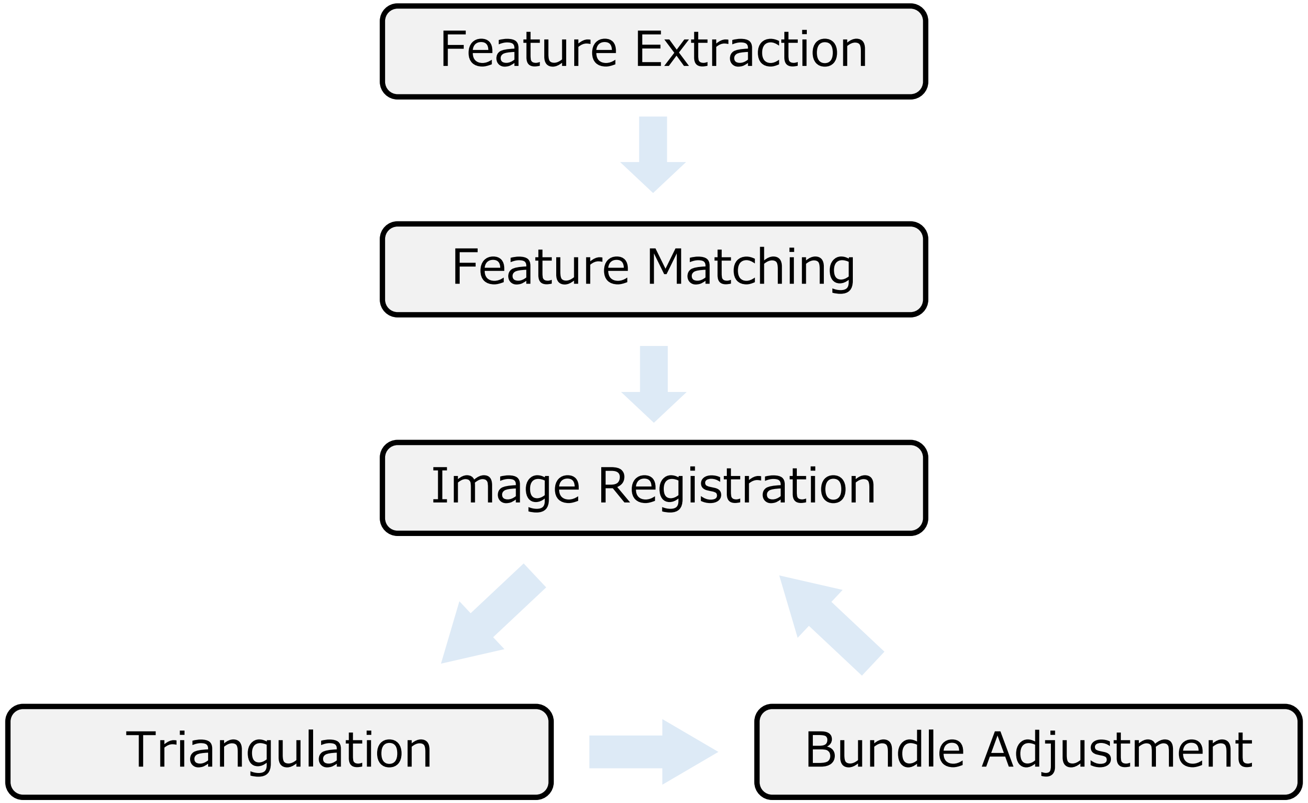
# Welcome to the NTU Front Gate

- We collect multiple images of the NTU front gate, and reconstruct its 3D point cloud model via structure from motion.

# About Dataset

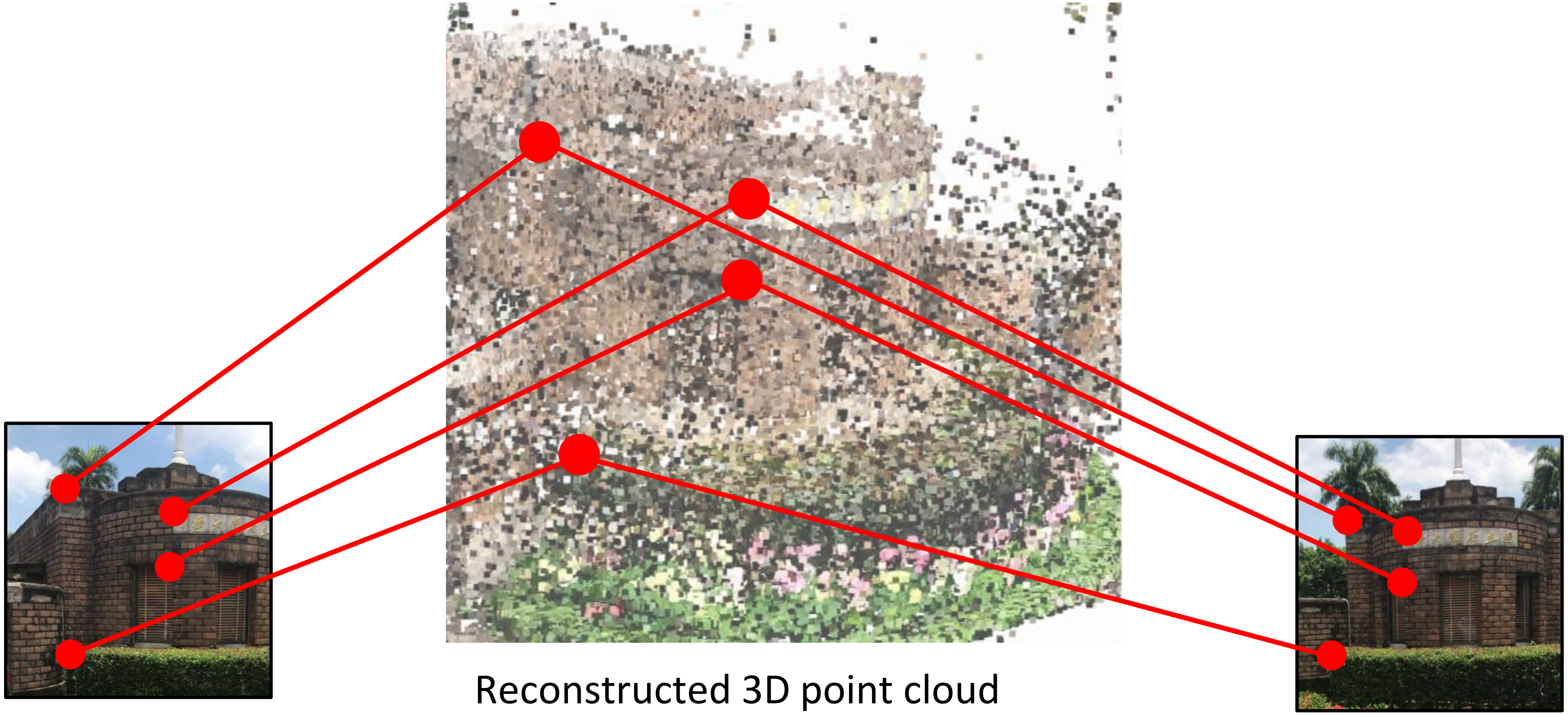
- 293 color images (1920x1080x3): 163 images for training, 130 images for testing
- 111,518 points (in world coordinate) with 682,467 local image descriptors

Dataset images



# About Dataset

- 293 color images (1920x1080x3): 163 images for training, 130 images for testing
- 111,518 points (in world coordinate) with 682,467 local image descriptors



Reconstructed 3D point cloud

# Data/image.pkl

⚠ The pose of an image is represented as the projection **from world to the camera** coordinate system. That is, p=K[R|T]X.

Camera Position(x,y,z) — Rotation (in quaternion)

| | IMAGE_ID | NAME | TX | TY | TZ | QW | QX | QY | QZ |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | train_img100.jpg | -3.12923 | -0.273371 | 3.17218 | 0.969363 | -0.003488 | 0.244797 | 0.019927 |
| **1** | 2 | train_img104.jpg | -3.10598 | -0.264036 | 3.12049 | 0.972423 | -0.005048 | 0.232322 | 0.019880 |
| **2** | 3 | train_img108.jpg | -3.06986 | -0.270274 | 3.08285 | 0.975032 | -0.004203 | 0.221007 | 0.021220 |
| **3** | 4 | train_img112.jpg | -3.02027 | -0.290710 | 3.07195 | 0.976940 | -0.003627 | 0.212336 | 0.022091 |
| **4** | 5 | train_img116.jpg | -2.98028 | -0.307973 | 3.05439 | 0.979017 | -0.002989 | 0.202524 | 0.022389 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **288** | 289 | valid_img75.jpg | -2.86676 | -0.366566 | 3.79563 | 0.931094 | 0.002295 | 0.363172 | 0.034118 |
| **289** | 290 | valid_img80.jpg | -2.86618 | -0.323873 | 3.72239 | 0.937160 | -0.001973 | 0.347476 | 0.031431 |
| **290** | 291 | valid_img85.jpg | -2.91426 | -0.300918 | 3.59808 | 0.945271 | -0.004261 | 0.325035 | 0.028210 |
| **291** | 292 | valid_img90.jpg | -2.99320 | -0.267023 | 3.46717 | 0.954254 | -0.004443 | 0.298019 | 0.023733 |
| **292** | 293 | valid_img95.jpg | -3.08001 | -0.259334 | 3.30072 | 0.962891 | -0.003862 | 0.269045 | 0.021006 |

293 rows × 9 columns

⚠ Note that the order is (QW, QX, QY, QZ)

# Data/point_desc.pkl

- point_desc.pkl

Source Info

128D Descriptors

| | POINT_ID | IMAGE_ID | XY | DESCRIPTORS |
|---|---|---|---|---|
| 0 | 1 | 1 | [94.94650268554688, 284.02899169921875] | [46, 43, 12, 11, 10, 5, 19, 37, 24, 16, 8, 9, ... |
| 1 | 1 | 2 | [99.05780029296875, 290.6889953613281] | [39, 42, 34, 14, 15, 12, 13, 31, 29, 11, 8, 7,... |
| 2 | 1 | 3 | [110.51899719238281, 291.7560119628906] | [47, 57, 39, 12, 12, 11, 9, 20, 43, 26, 13, 7,... |
| 3 | 1 | 4 | [131.70199584960938, 286.4880065917969] | [38, 58, 39, 12, 11, 11, 13, 16, 35, 20, 12, 8... |
| 4 | 1 | 7 | [156.52499389648438, 279.2149963378906] | [32, 38, 31, 19, 15, 6, 11, 32, 28, 14, 6, 10,... |
| ... | ... | ... | ... | ... |
| 1234453 | 129081 | 276 | [816.5590209960938, 353.6910095214844] | [28, 20, 11, 16, 23, 18, 22, 25, 42, 11, 8, 24... |
| 1234454 | 129081 | 278 | [892.0490112304688, 384.6050109863281] | [30, 30, 15, 22, 28, 14, 15, 23, 47, 13, 10, 2... |
| 1234455 | 129081 | 279 | [965.5770263671875, 397.2950134277344] | [29, 22, 12, 18, 28, 16, 20, 30, 40, 12, 9, 27... |
| 1234456 | 129081 | 280 | [1039.56005859375, 405.864990234375] | [27, 24, 14, 15, 26, 16, 25, 33, 45, 12, 10, 2... |
| 1234457 | 129081 | 280 | [1045.989990234375, 404.6090087890625] | [23, 38, 24, 33, 28, 7, 3, 7, 52, 12, 12, 26, ... |

⚠ If Point_ID is -1, then its 3D position is not available.

# Data/train.pkl

- train.pkl

3D Point Position(x,y,z) | Source Info | 128D Descriptors

| | POINT_ID | XYZ | RGB | IMAGE_ID | XY | DESCRIPTORS |
|---|---|---|---|---|---|---|
| **0** | 1 | [1.6093346, -1.1848674, 1.610395] | [87, 87, 77] | 1 | [94.94650268554688, 284.02899169921875] | [46, 43, 12, 11, 10, 5, 19, 37, 24, 16, 8, 9, ... |
| **1** | 1 | [1.6093346, -1.1848674, 1.610395] | [87, 87, 77] | 2 | [99.05780029296875, 290.6889953613281] | [39, 42, 34, 14, 15, 12, 13, 31, 29, 11, 8, 7,... |
| **2** | 1 | [1.6093346, -1.1848674, 1.610395] | [87, 87, 77] | 3 | [110.51899719238281, 291.7560119628906] | [47, 57, 39, 12, 12, 11, 9, 20, 43, 26, 13, 7,... |
| **3** | 1 | [1.6093346, -1.1848674, 1.610395] | [87, 87, 77] | 4 | [131.70199584960938, 286.4880065917969] | [38, 58, 39, 12, 11, 11, 13, 16, 35, 20, 12, 8... |
| **4** | 1 | [1.6093346, -1.1848674, 1.610395] | [87, 87, 77] | 7 | [156.52499389648438, 279.2149963378906] | [32, 38, 31, 19, 15, 6, 11, 32, 28, 14, 6, 10,... |
| **...** | ... | ... | ... | ... | ... | ... |
| **682463** | 129081 | [0.66382873, -1.3121917, 5.433149] | [33, 30, 28] | 141 | [834.9459838867188, 363.7510070800781] | [32, 26, 15, 19, 28, 14, 18, 30, 37, 12, 11, 2... |
| **682464** | 129081 | [0.66382873, -1.3121917, 5.433149] | [33, 30, 28] | 142 | [867.6019897460938, 366.8039855957031] | [33, 16, 6, 11, 25, 16, 18, 36, 41, 10, 7, 23,... |
| **682465** | 129081 | [0.66382873, -1.3121917, 5.433149] | [33, 30, 28] | 144 | [981.5599975585938, 398.8039855957031] | [25, 14, 7, 12, 27, 21, 24, 28, 50, 13, 8, 24,... |
| **682466** | 129081 | [0.66382873, -1.3121917, 5.433149] | [33, 30, 28] | 145 | [1039.56005859375, 405.864990234375] | [27, 24, 14, 15, 26, 16, 25, 33, 45, 12, 10, 2... |
| **682467** | 129081 | [0.66382873, -1.3121917, 5.433149] | [33, 30, 28] | 145 | [1045.989990234375, 404.6090087890625] | [23, 38, 24, 33, 28, 7, 3, 7, 52, 12, 12, 26, ... |

682468 rows × 6 columns

# About Dataset: Camera Parameters

- Review the Pinhole camera model:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \approx \begin{bmatrix} f_x & s & o_x \\ 0 & f_y & o_y \\ 0 & 0 & 1 \end{bmatrix} [R \quad t] \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

- Intrinsic Parameters:

$$K = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1868.27 & 0 & 540 \\ 0 & 1869.18 & 960 \\ 0 & 0 & 1 \end{bmatrix}$$

- Distortion Parameters (Brown-Conrady Model):

$$D = [k_1 \quad k_2 \quad p_1 \quad p_2] = [0.0847023, -0.192929, -0.000201144, -0.000725352]$$

# Problem1

**Q1-1** For each validation image, compute its camera pose with respect to world coordinate. Find the 2D-3D correspondence by descriptor matching, and solve the camera pose. Implement at least one kind of algorithm that solves a PnP problem. Briefly explain your implementation and write down the pseudo code in your report.

**Notes**:

- Expected Solution: P3P + RANSAC. You have to implement RANSAC by yourself.
- You cannot use calib3d module in OpenCV. That is, solvePnP and solvePnPRansac is forbidden. However, you are encouraged to try them beforehand.
- You may also try DLT, EPnP, AP3P, or any kinds of solutions.

# Problem1

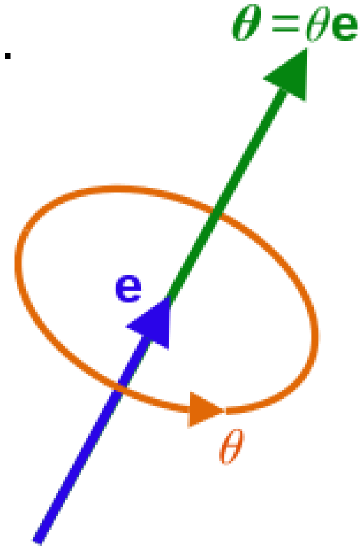
**<u>Q1-2</u>** For each camera pose you calculated, compute the median pose error (translation, rotation) with respect to ground truth camera pose. Provide some discussion.

**Notes**:

- Translation: median of all absolute pose differences (Euclidean Distance).

$$t_e = \|\mathbf{t} - \hat{\mathbf{t}}\|_2$$

- Rotation: median of relative rotation angle between estimation and ground-truth.
  (1. Find out the relative rotation and represent it as axis angle representation.
  2. Report the median of angles.)



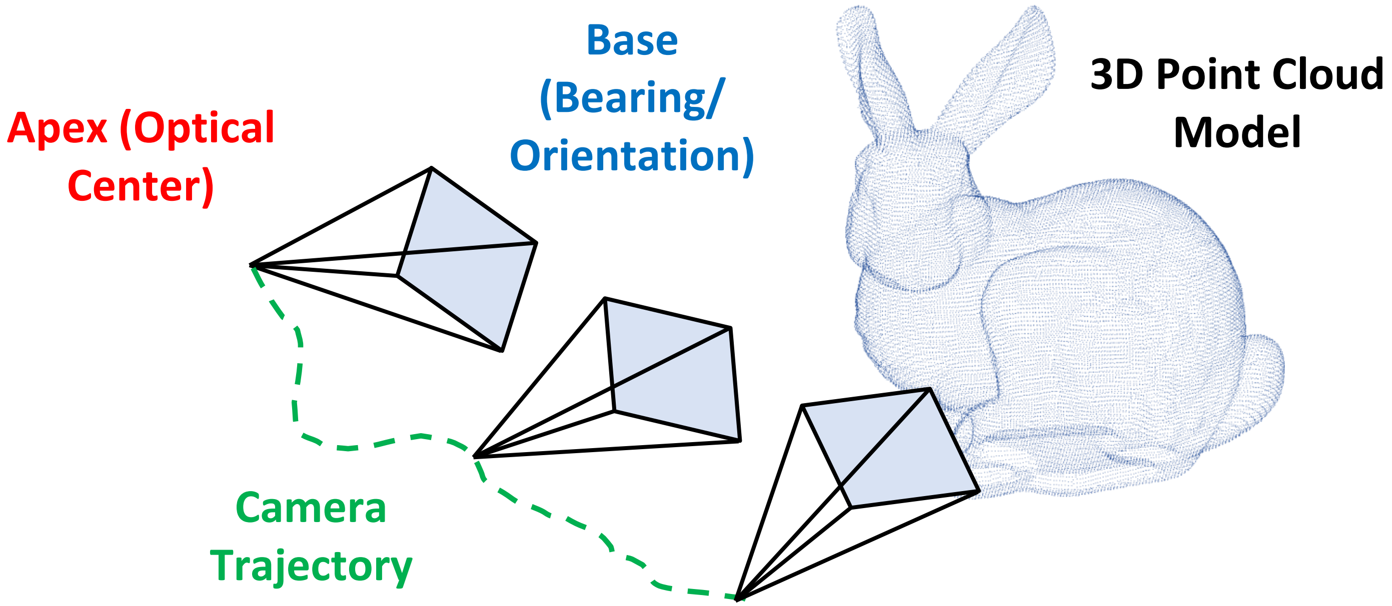

$$\mathcal{R} = R_e \widehat{\mathcal{R}}$$

# Problem1

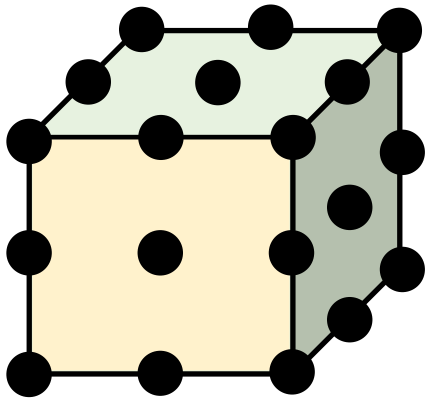
**Q1-3** For each camera pose you calculated, plot the trajectory and camera poses along with 3d point cloud model using Open3D.  Explain how you draw and provide some discussion.

**Notes**:

- Draw the camera pose as a quadrangular pyramid, where the apex is the position of the optical center, and the normal of base is the bearing (orientation) of the camera.



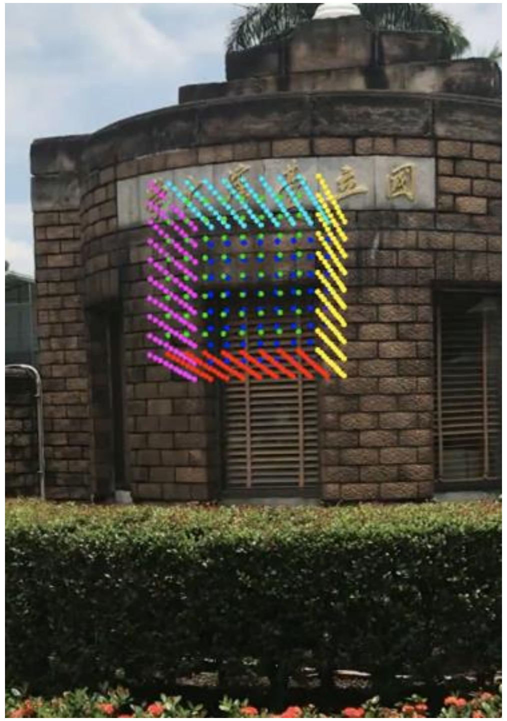
# Problem2



**<u>Q2-1</u>** With camera intrinsic and extrinsic parameters, place a virtual cube in the the validation image sequences to create an Augmented Reality video. Draw the virtual cube as a point set with different colors on its surface. Implement a simply but efficient painter's algorithm to determine the order of drawing.

**Notes**:

- You don't have to consider whether virtual cube will be occluded.
- Manually select the location, orientation, and scale of the virtual cube. (We provide a code that allows you to adjust the cube by keyboard.)
- Painter's Algorithm:
  1. Sort each voxel by depth
  2. Place each voxel from the furthest to the closest



# Sample Code

- You should read the pickle files with **pandas**.

```
>>> import pandas as pd
>>> images_df = pd.read_pickle("dataframes/images.pkl")
```

- You may use **Scipy** to deal with 3D rotation representations.

```
>>> from scipy.spatial.transform import Rotation as R
>>> r = R.from_quat([0, 0, np.sin(np.pi/4), np.cos(np.pi/4)])
>>> r.as_rotvec()
array([0. , 0. , 1.57079633])
```

⚠ Be aware of the order.

**Parameters:** **quat** : ***array_like, shape (N, 4) or (4,)***

Each row is a (possibly non-unit norm) quaternion in scalar-last (x, y, z, w) format. Each quaternion will be normalized to unit norm.

**Returns:** **rotation** : ***Rotation instance***

Object containing the rotations represented by input quaternions.

# Introduction to Open3D

- Install open3D

```
pip install open3d
```

- Basic manipulation in open3D (Example Drawing):

```
points = [[0, 0, 0], [1, 0, 0], [0, 1, 0], [1, 1, 0],
        [0, 0, 1], [1, 0, 1], [0, 1, 1], [1, 1, 1]]
lines  = [[0, 1], [0, 2], [1, 3], [2, 3], [4, 5], [4, 6],
        [5, 7], [6, 7], [0, 4], [1, 5], [2, 6], [3, 7]]

import open3d as o3d
line_set = o3d.geometry.LineSet()
line_set.points = o3d.utility.Vector3dVector(points)
line_set.lines = o3d.utility.Vector2iVector(lines)

vis = o3d.visualization.Visualizer()
vis.create_window()
vis.add_geometry(line_set) o3d.visualization.ViewControl.set_zoom(vis.get_view_control(), 0.8)
vis.run()
```

⚠ Please refer to the document to find the property you need.

http://www.open3d.org/docs/latest/tutorial/visualization/visualization.html

# Bonus List

To get extra credits, you can try the following things:
(including, but not limited to)

- **Local Features:** Try different kinds of local features (including deep features)
- **Make it faster:** Come up with faster matching or image registration strategy. (prioritized matching, approximate nearest neighbor, coarse-to-fine strategy, image retrieval, …)
- **Make it more accurate**: Make the pose estimation more accurate. (Different PnP solving methods, outlier rejection strategies, …)
- **Absolute Pose Regression:** Train a deep neural network to regress the absolute camera pose. (For example, PoseNet: A Convolutional Network for Real-Time 6-DOF Camera Relocalization, ICCV 2015)

# Grading

- We will evaluate both **the functionality of the code** and **the quality of the report**.
- **[*] Youtube link :**
  - You should record your demonstration, including the start time and the GitHub clone action
  - Example : https://youtu.be/-VnjVda7c8o?si=zowfe7vjvCMMFrOk
- **Functionality**: Can it run? How's the performance?
- **Quality**: theoretical/experimental analysis, observation, discussion, …
- Note that it **might be curved** based on overall performance of students.
- Grade
  - Meet the basic requirement (programming & report) → A
  - Basic requirement + advanced studies (programming & report) → A+

# Grading Policies

- Push your code and report to the GitHub classroom.
- Programming Languages: Python (Python>=3.8), (C++)
- Report Format: PDF or Markdown
  (Warning for Markdown users: Latex equations cannot be rendered properly in GitHub)
- Late Submission: **-10% from your score** / day
- Plagiarism: You have to **write your own codes**.
- Discussion: We encourage you to discuss with your classmates, but remember to **mention their names and contributions in the report**.

# Thanks

If you have any question, please email 3dcv@csie.ntu.edu.tw