3DCV HW2

R13631030 陳澔平

HW2 Problem 1 Demo Video

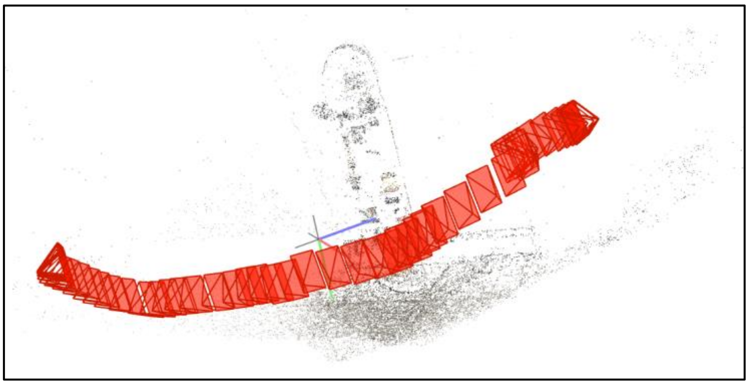
HW2 Problem 2 Demo Video

Problem 1

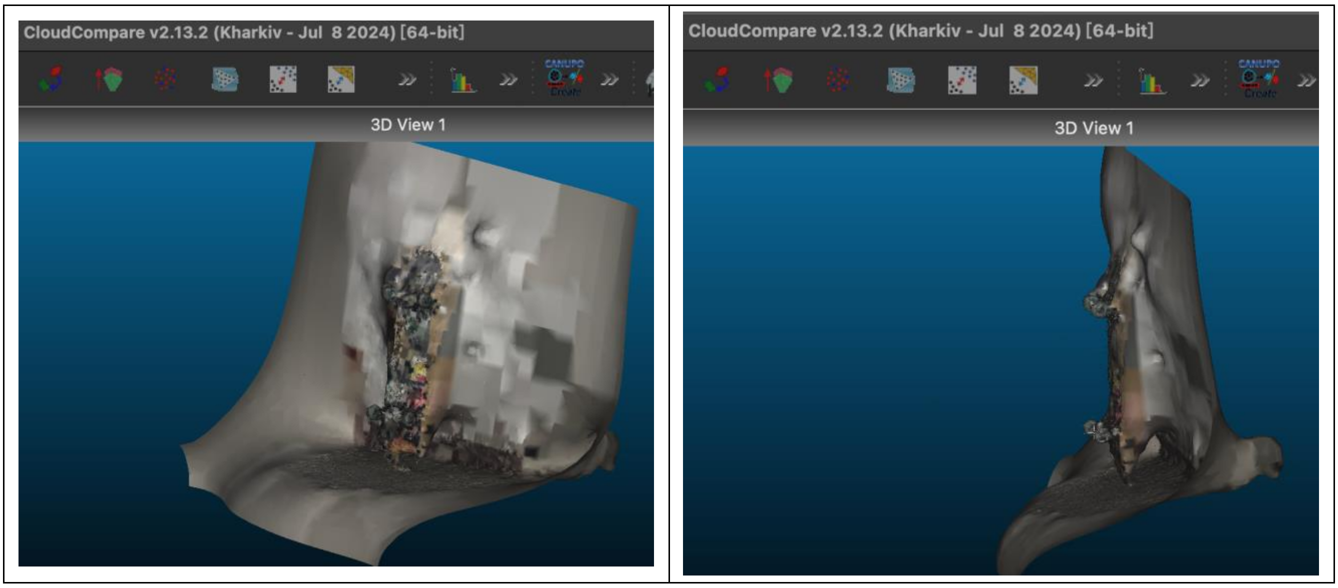
- **Q1-1**

1. Use ffmpeg to extract still frames at a fixed rate (5 fps):

2. Run COLMAP (Structure-from-Motion), save to .ply



- **Q1-2 Mesh: open .ply in CloudCompare**
  1. Plugins -> Hough Normals Computation
  2. Plugins -> PoissonRecon



Problem 2

- **Q2-1 Camera Relocalization**

Objective:

Estimate the camera extrinsic parameters (R, t) of each validation image by solving the PnP (Perspective-n-Point) problem using SIFT feature correspondences between 2D keypoints and known 3D map points.

Methodology:

Step 1 – Camera Pose Estimation

1. Descriptor Averaging

- Since each 3D point could be observed from multiple training images, multiple 128-D descriptors were associated with the same 3D coordinate.
- All descriptors of the same POINT_ID were averaged, producing one representative descriptor per 3D point.

2. Feature Matching (2D–3D Association)

- For every validation image, its 2D descriptors were matched to the averaged 3D descriptors using GPU-accelerated cosine similarity implemented in PyTorch.
- A ratio test (0.8) was applied to remove ambiguous correspondences, ensuring reliable 2D–3D pairs

3. Pose Estimation PnP + RANSAC

- The filtered correspondences were fed into **cv2.solvePnPRansac()** with the given intrinsic matrix and distortion coefficients.
- It produced the optimal rotation vector *rvec* and translation vector *tvec*, representing the camera's orientation and position in world coordinates.

Step 2 – Pose Error Evaluation

Translation Error

The Euclidean distance between estimated and true translation vectors was used:

```
def translation_error(t_est, t_gt):
    """ translation error = ||t_est - t_gt||_2 """
    return float(np.linalg.norm(t_est.reshape(-1) - t_gt.reshape(-1)))
```

Rotation Error

The relative rotation between the estimated and ground-truth orientations was expressed as an axis–angle representation:

```
def rotation_error(R_est_quat_xyzw, R_gt_quat_xyzw):
    """
    Compute the rotation error (in degrees) between estimated and ground-truth orientations.
    1. Convert both quaternions (xyzw format) to Rotation objects.
    2. Compute the relative rotation: R_rel = R_est * R_gt.inverse()
    3. Convert the relative rotation to axis□angle form (rotvec) and take its magnitude.
       The magnitude represents the angular difference (in radians).
    4. Return the angle in degrees.
    """
    r_est = R.from_quat(R_est_quat_xyzw)
    r_gt  = R.from_quat(R_gt_quat_xyzw)
    r_rel = r_est * r_gt.inv()
    ang_rad = np.linalg.norm(r_rel.as_rotvec())
    return np.degrees(ang_rad)
```

The angle of this relative rotation (in degrees) was used as the rotational error.

The median values across all validation images were reported to suppress the influence of outliers.

Result Discussion:

The median rotation and translation errors were extremely small (≈ 0.002° and 0.000 units), indicating that the estimated camera poses were almost identical to the ground truth.

This demonstrates that the feature matching and PnP–RANSAC pipeline successfully recovered accurate camera poses.

Step 3 – Visualization of Camera Trajectory and 3D Model

All camera centers and orientations were visualized along with the 3D point-cloud model using Open3D's O3DVisualizer.

Discussion:

The resulting visualization showed that the estimated camera poses aligned perfectly with the 3D structure and followed a smooth trajectory.

The small pyramid size and consistent orientation confirmed that the pose estimation pipeline produced geometrically coherent results.

- **Q2-2 Virtual Cube in AR**

Objective

The goal of this task was to render a virtual cube into the sequence of validation images to form an Augmented Reality (AR) video.

Given the camera intrinsics and extrinsics (either ground-truth or estimated poses from Q2-1), each cube point was projected into the image plane using the perspective camera model.

A simple painter's algorithm was implemented to correctly determine the drawing order according to depth.

Painter's Algorithm:

1. Transform cube points to the camera frame

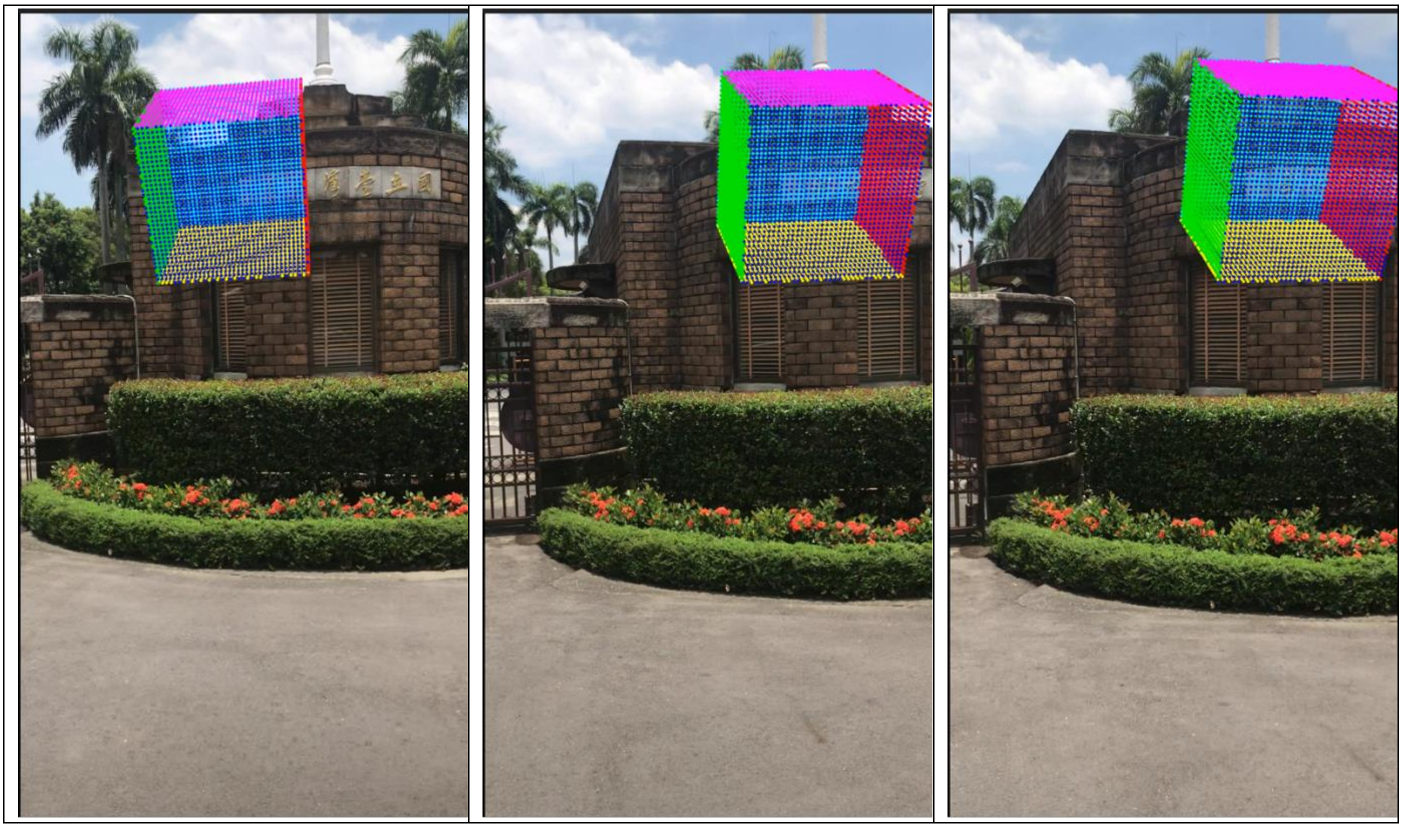
- $X_c = R_{w2c} X_{world} + t_{w2c}$, where R_{w2c} and t_{w2c} are from the current camera pose

2. Depth sorting

- The z-values of X_c were used to sort all cube points from furthest to nearest
- This ensures that nearer points overwrite further ones when drawn (simplified painter's algorithm)

3. Perspective projection and drawing

- Each point was projected to pixel coordinates using OpenCV's **cv2.projectPoints()**
- Each projected point was drawn as a small filled circle (**cv2.circle**)



*LLM used: ChatGPT5 is used for code debugging, implementation guidance, and report polishing.