CSE462/562 – Augmented Reality (Fall 2024)
Homework #3 Report

Berru Lafcı - 1901042681

PART1:

1. Calculating the Homography Matrix

In this project, I manually identified image points on the provided checkerboard pattern images. Here are the corners of checkerboard pattern on the images:

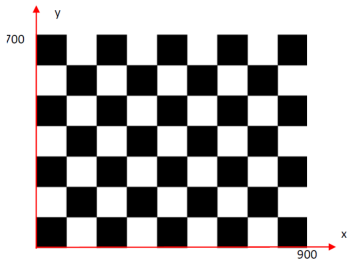
```
double[,] imagePoints = new double[4, 2]
{
    { 475, 514 },  // top-left
    { 2096, 532 }, // top-right
    { 454, 2612 }, // bottom-left
    { 2075, 2624 } // bottom-right
};

// Image2
double[,] imagePoints2 = new double[4, 2]
{
    { 276, 496 }, // top-left
    { 1954, 439 }, // top-right
    { 249, 2610 }, // bottom-left
    { 1936, 2692 } // bottom-right
};

// Image3
double[,] imagePoints3 = new double[4, 2]
{
    { 501, 593 }, // top-left
    { 2160, 721 }, // top-right
    { 490, 2683 }, // bottom-left
    { 1960, 2624 } // bottom-right
};
```

And this is the corner of the scene points:

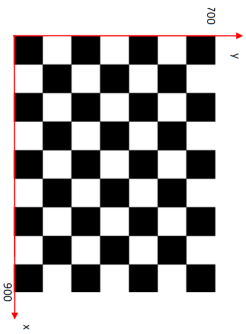
```
double[,] scenePoints = new double[4, 2]
{
    { 0, 0 }, // top-left
    { 0, 700 }, // top-right
    { 900, 0 }, // bottom-left
    { 900, 700 } // bottom-right
};
```

To ensure that the homography calculation was accurate, I arranged the order of the scene points to match the order of the identified image points for each image.

in pdf:



ordered coordinate:



To calculate the homography matrix, I implemented the `CalculateHomography` method. This function follows a structured pipeline:

1. **Point Normalization**: To improve numerical stability, I normalized the scene and image points.
2. **Direct Linear Transformation (DLT)**: Using the normalized points, I calculated the initial homography matrix with DLT, which involves constructing a linear system based on point correspondences and solving it using Singular Value Decomposition (SVD).
3. **Gradient Descent**: As a non-linear optimization method, I chose the gradient descent optimizer to iteratively refine the homography matrix. By minimizing the reprojection error between the projected source points and their corresponding destination points, the optimization ensures improved alignment accuracy.
4. **Denormalization**: Finally, I denormalized the homography matrix by reversing the normalization transformations.

```
Homography Matrix - Image1:
0.0134468984963485 -1.340561578313 -273.960481999911
-1.34455129201957 -0.0160880300024879 -296.454079469371
-2.3809231817723E-08 -2.36294476714322E-06 -0.576758909473558
UnityEngine.Debug:Log (object)
```

```
Homography Matrix - Image2:
0.0189313603629463 -1.33088210364166 -165.049666949509
-1.39426027363698 0.07175099525780006 -296.610995677369
3.9806392479734E-06 5.25198581814209E-05 -0.598006039672076
UnityEngine.Debug:Log (object)
```

```
Homography Matrix - Image3:
0.0269820299964267 1.69841808654019 164.977395444413
1.45116946238775 0.60617934852162 166.164502091105
0.000201400172067467 0.00044860147822844 0.392010886127683
UnityEngine.Debug:Log (object)
```

1.2 Handling Mismatched Correspondences

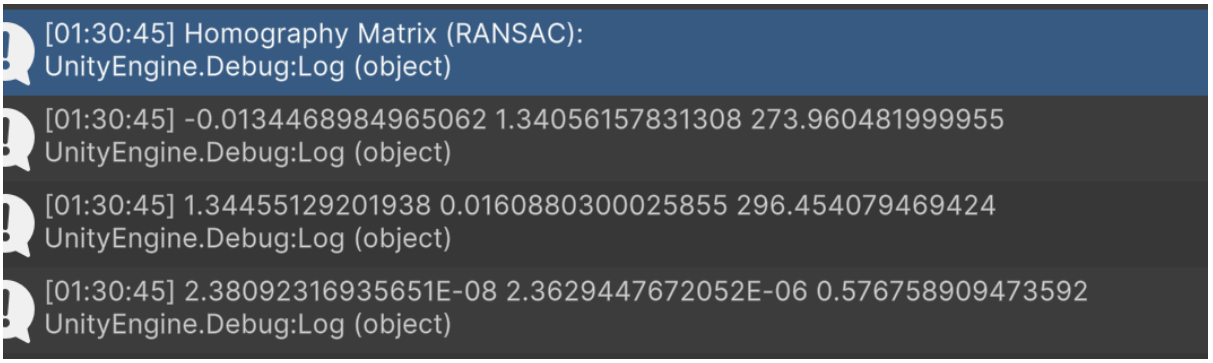
To handle mismatched correspondences, I implemented the `CalculateHomographyRANSAC` function, which follows the RANSAC algorithm:

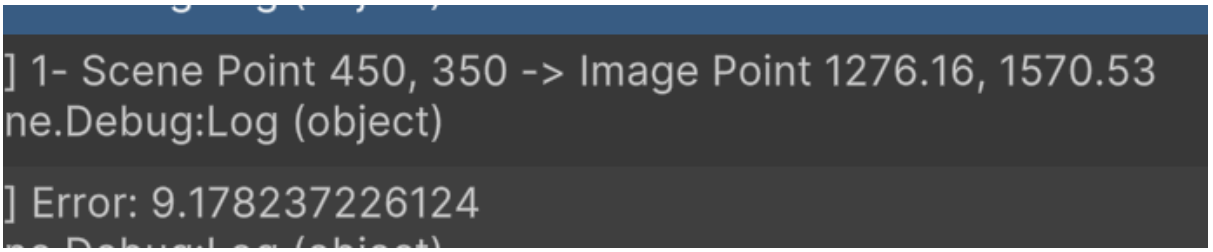
1. **Confidence Matrix**: I calculated a confidence matrix, where each entry indicates the degree of match between a scene point and an image point. I simply find their matching ratio to calculate the matrix.

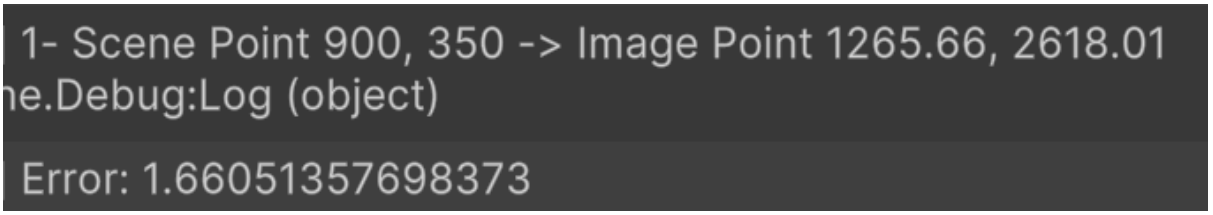
```
// 1.2 RANSAC
double[,] scenePointsRansac = new double[,]
{
    { 0, 0 },     // Top-left
    { 0, 700 },   // Top-right
    { 900, 0 },   // Bottom-left
    { 900, 700 }, // Bottom-right
    { 450, 350 }  // Center
};

// Image1
double[,] imagePointsRansac = new double[,]
{
    { 475, 514 },   // Top-left
    { 2096, 532 },  // Top-right
    { 454, 2612 },  // Bottom-left
    { 2075, 2624 }, // Bottom-right
    { 1265, 1585 }  // Center
};
```

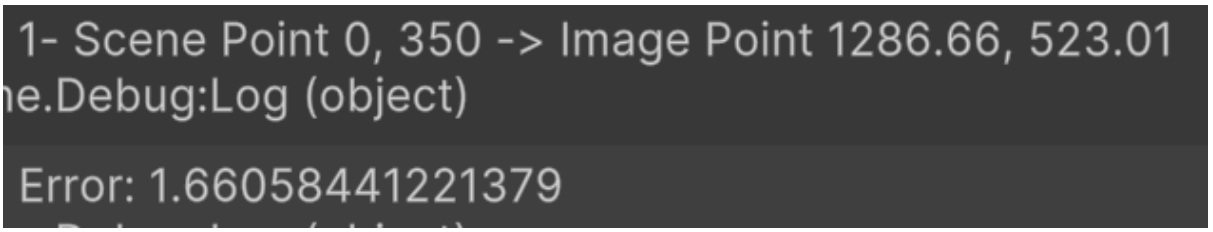
```
double[,] confidenceMatrix = new double[,]
{
    { 0.9, 0.1, 0.05, 0.02, 0.03 }, // Confidence for point 0
    { 0.1, 0.85, 0.03, 0.02, 0.05 }, // Confidence for point 1
    { 0.05, 0.02, 0.9, 0.03, 0.04 }, // Confidence for point 2
    { 0.02, 0.03, 0.04, 0.95, 0.03 }, // Confidence for point 3
    { 0.03, 0.05, 0.04, 0.03, 0.85 }  // Confidence for point 4
};
```

2. **Random Sampling**: I randomly sampled four point correspondences using the confidence matrix. This ensures that more likely matches are favored during sampling.
3. **Homography Estimation**: For each sample, I computed a homography matrix using the DLT method.
4. **Inlier Counting**: I calculated the reprojection error for all points using the estimated homography. Points with errors below a threshold were considered inliers.
5. **Best Model Selection**: After several iterations, I selected the homography matrix with the highest number of inliers as the best model.









1.3 Projecting Scene Points onto the Image

Using the homography matrix, I implemented `ProjectPoint` to project scene points onto image points. I simply applied the calculated homography matrix to scene points in this function.

1.4 Projecting Image Points onto the Scene

Using the inverse homography matrix, I mapped image points back onto the scene using the same projection formula. I applied the inverse of the calculated homography matrix to image points in this function.

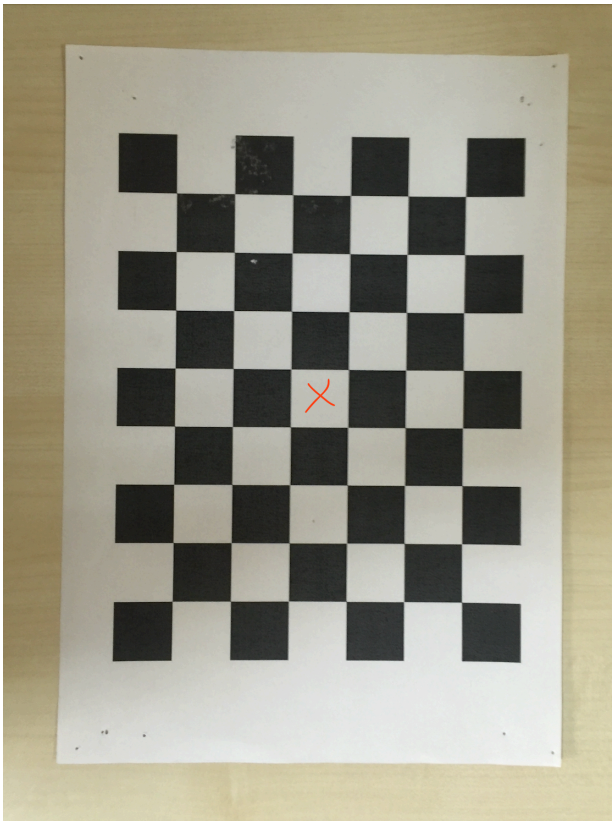
1.5 Calculating Errors for Additional Point Matches

To validate the accuracy of the homography, I approximately selected three additional points in the scene (my ordered scene) and their corresponding image points. I calculated the reprojection error with Euclidean error.

```
double[,] additionalImgPoints = new double[3, 2]
{
    { 1285,1573 }, // Center of the image
    { 1264,2618 },
    { 1285,523 }
};

double[,] additionalImgPoints2 = new double[3, 2]
{
    { 1105, 1553 }, // near center
    { 1100, 2651 }, // near bottom edge
    { 1115, 468 }   // near top edge
};

double[,] additionalImgPoints3 = new double[3, 2]
{
    { 1230, 1638 }, // near center
    { 1225, 2653 }, // near bottom edge
    { 1330, 657 }   // near top edge
};
```

```
double[,] additionalScenePoints = new double[3, 2]
{
    { 450,350 },
    { 900,350 },
    { 0,350 }
};
```

For example for Image 1 I calculated the center approximately like this. Pixel coordinate on image and 900/2, 700/2 on scene:



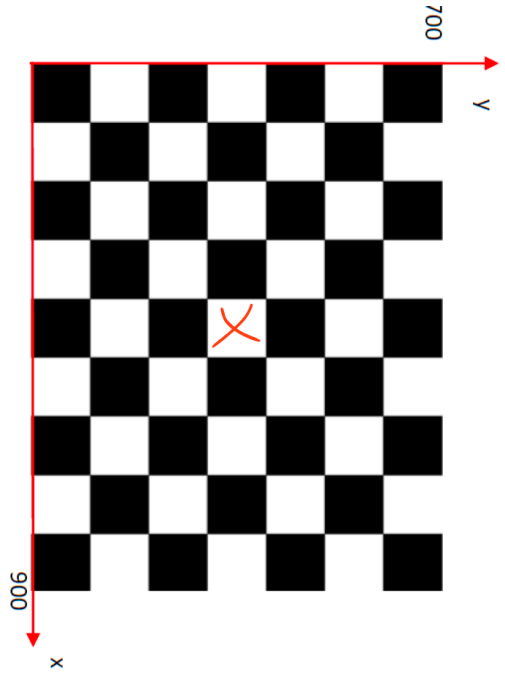


Image 1:

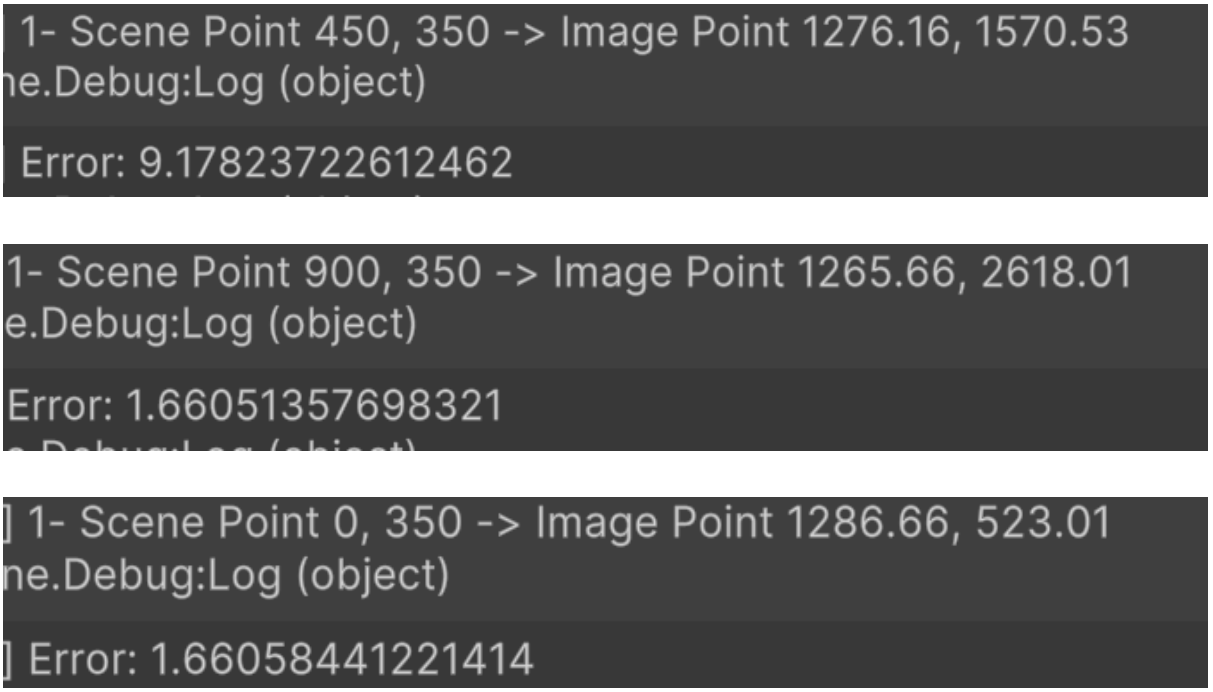


Image 2:

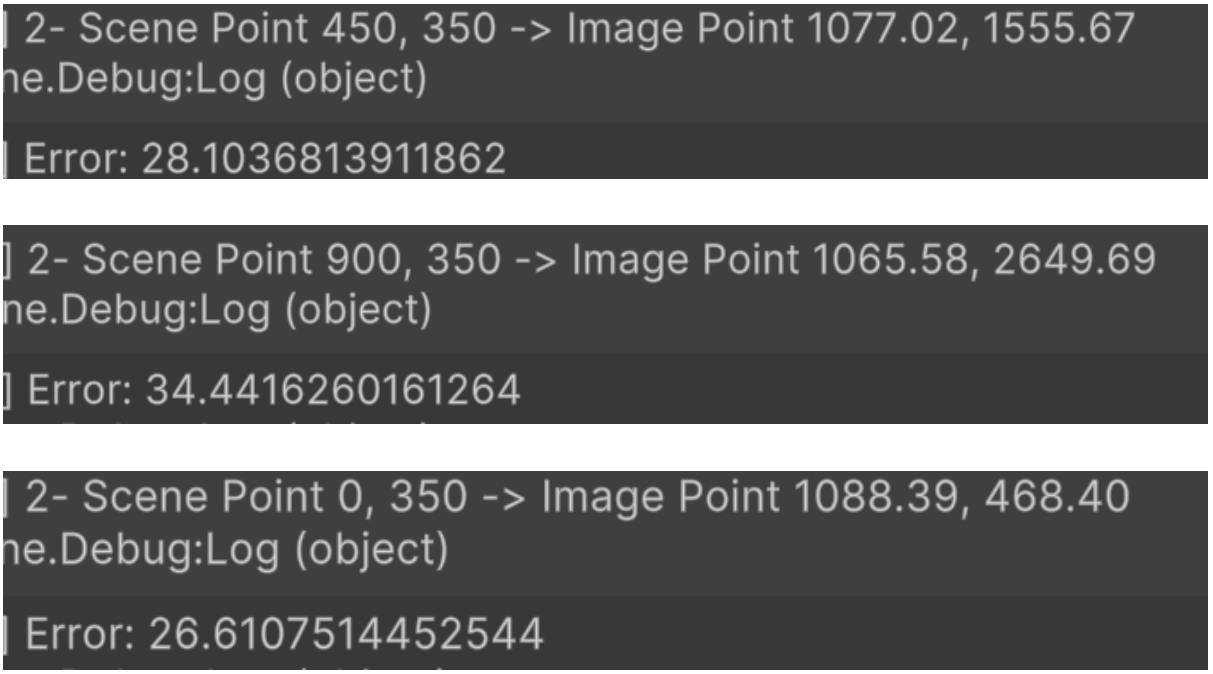
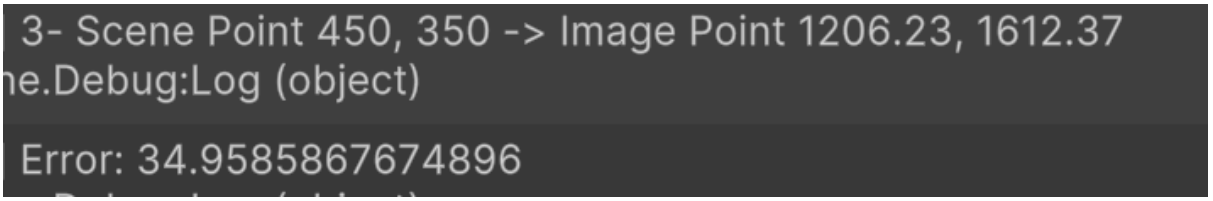
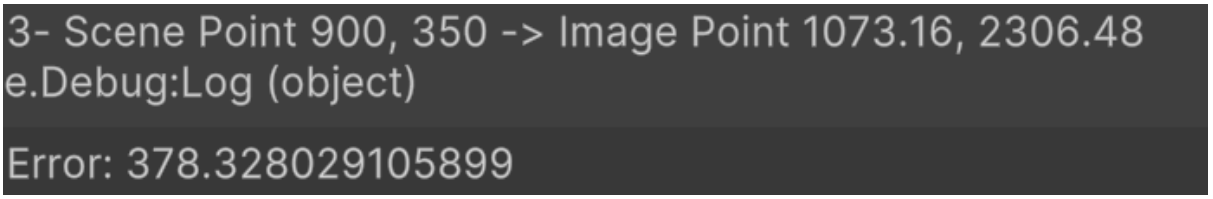
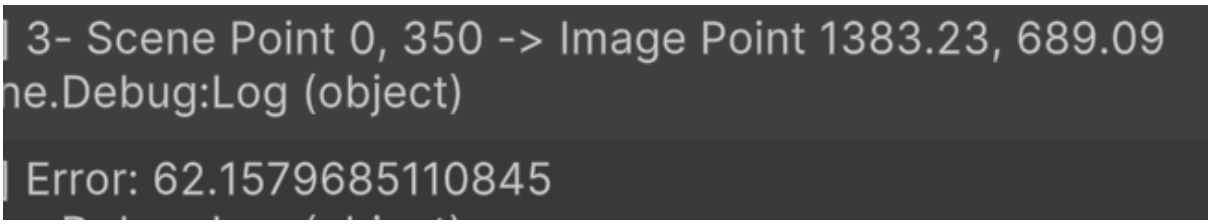


Image 3:



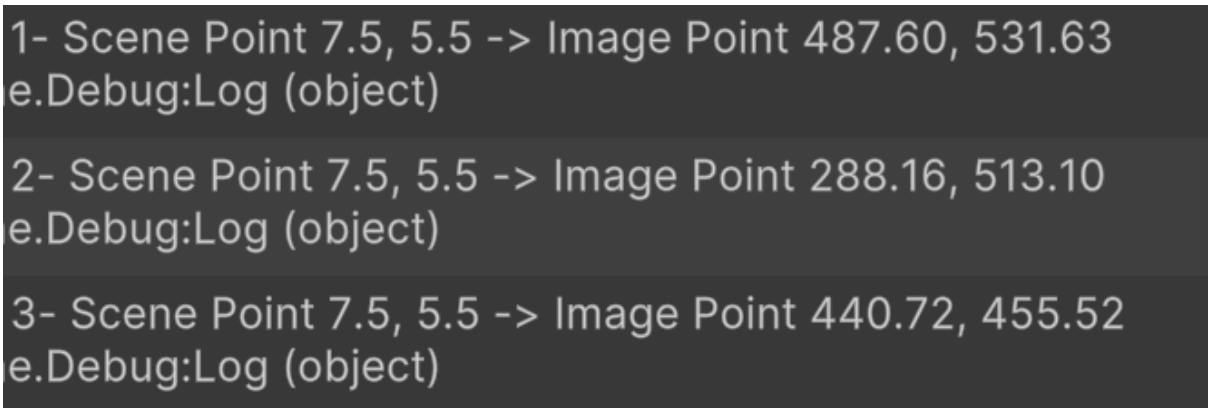




1.6 Projections of Scene Points onto the Image

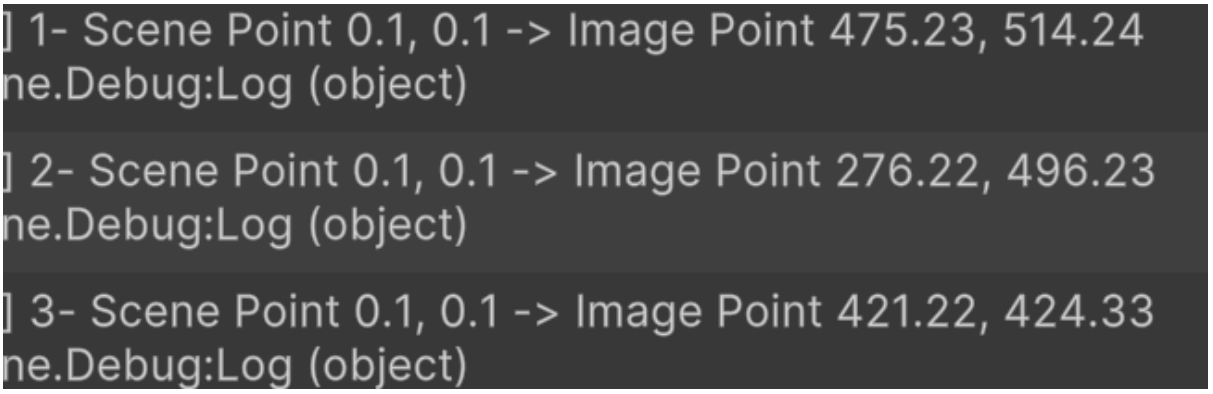
Using the computed homography matrices, I projected the scene points from the homework pdf onto the image for all three images.

S1:



S2:

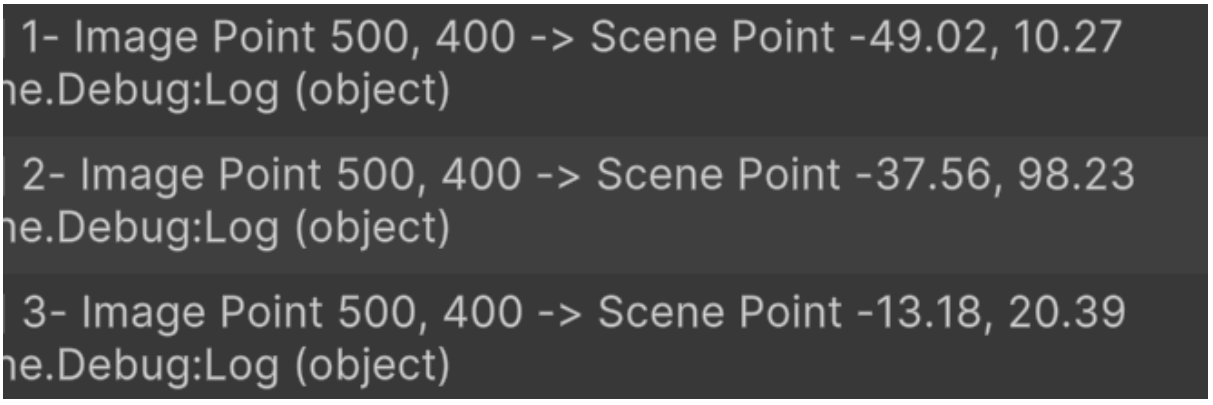
S3:



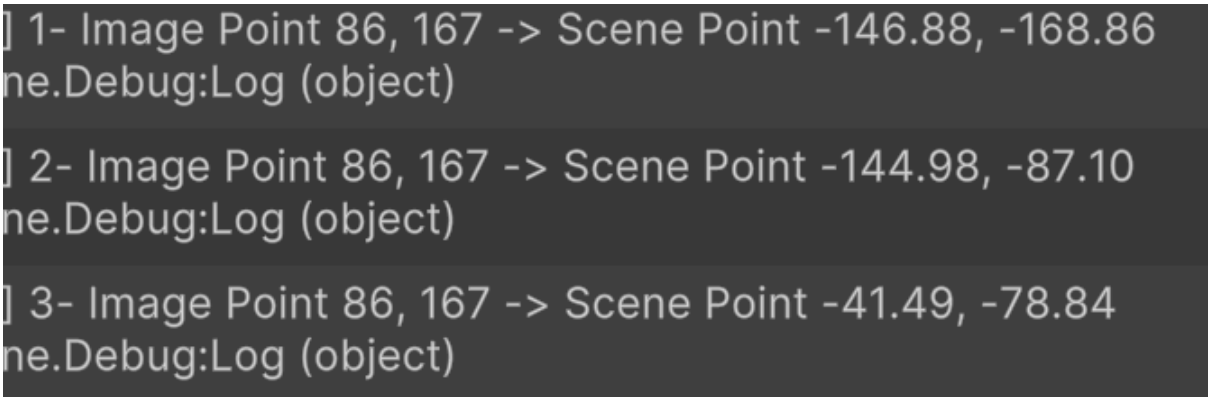
1.7 Projections of Image Points onto the Scene

Using the inverse homography matrices, I projected the image points from the homework pdf back onto the scene.

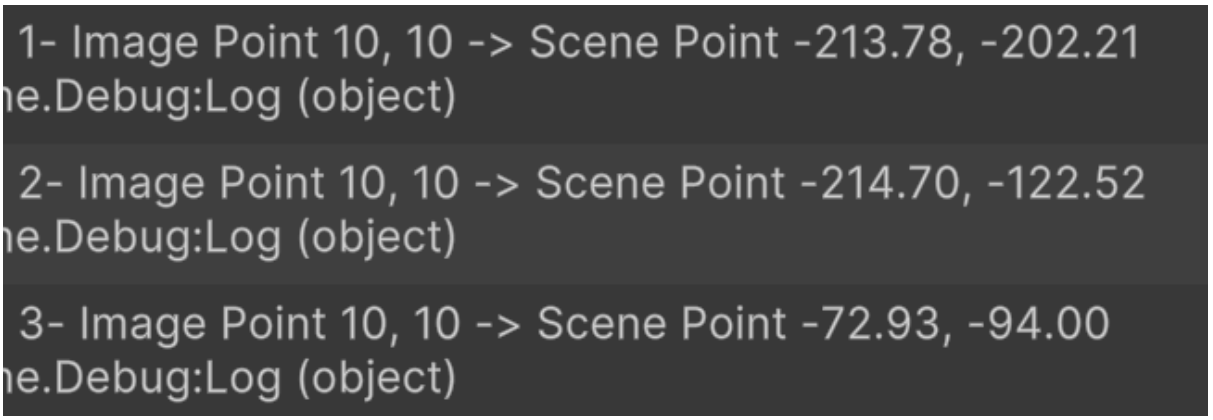
I1:



I2:


] 1- Image Point 86, 167 -> Scene Point -146.88, -168.86
ne.Debug:Log (object)
] 2- Image Point 86, 167 -> Scene Point -144.98, -87.10
ne.Debug:Log (object)
] 3- Image Point 86, 167 -> Scene Point -41.49, -78.84
ne.Debug:Log (object)

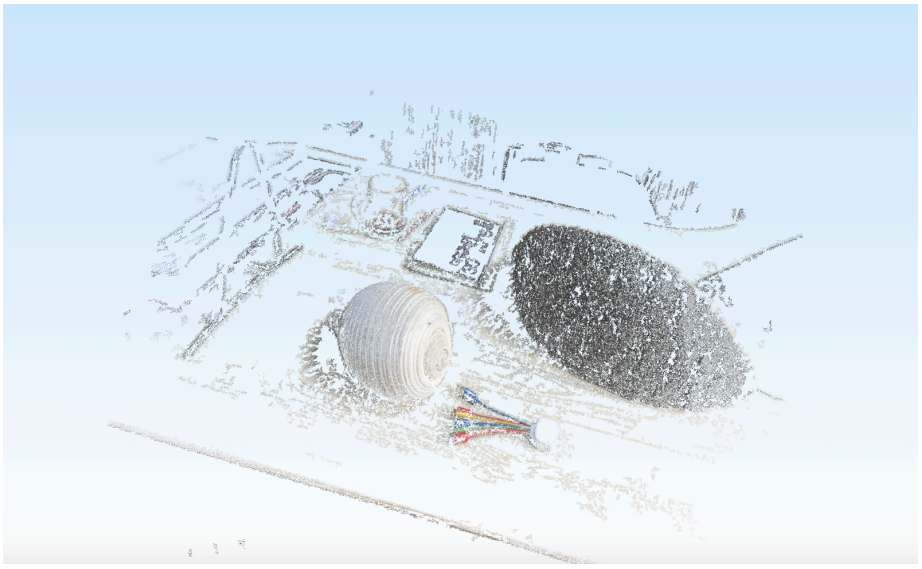
I3:


1- Image Point 10, 10 -> Scene Point -213.78, -202.21
ne.Debug:Log (object)
2- Image Point 10, 10 -> Scene Point -214.70, -122.52
ne.Debug:Log (object)
3- Image Point 10, 10 -> Scene Point -72.93, -94.00
ne.Debug:Log (object)

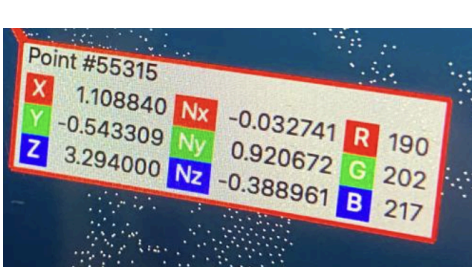
PART 2:

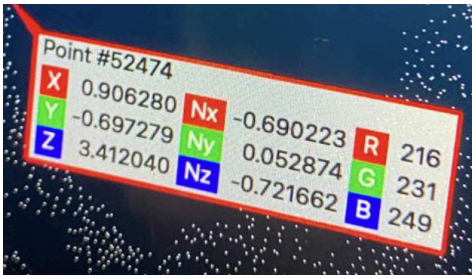
Resources and Preprocessing

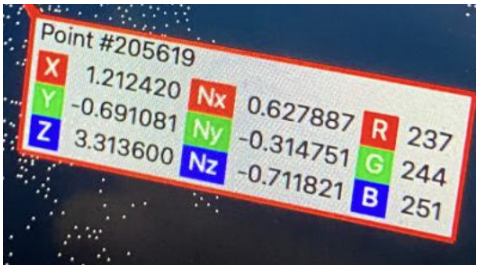
1. **Input Files from VisualSFM:**
    - **PLY file:** Contains the 3D point cloud data of the scene.



    - **NVM and bundle.out file:** Provides camera parameters such as focal length, rotation, and translation. They hardcoded into the code as `CameraParameters`.
2. **Tools Used:**
    - **VisualSFM:** To generate the PLY, NVM, and bundle.out files.
    - **Meshlab:** To identify the USB position (used as the teapot placement point) from the PLY file. I get the exact coordinate with getting the mean of USB corner points. This mean point is hardcoded into the code as `teapotWorldPosition`.

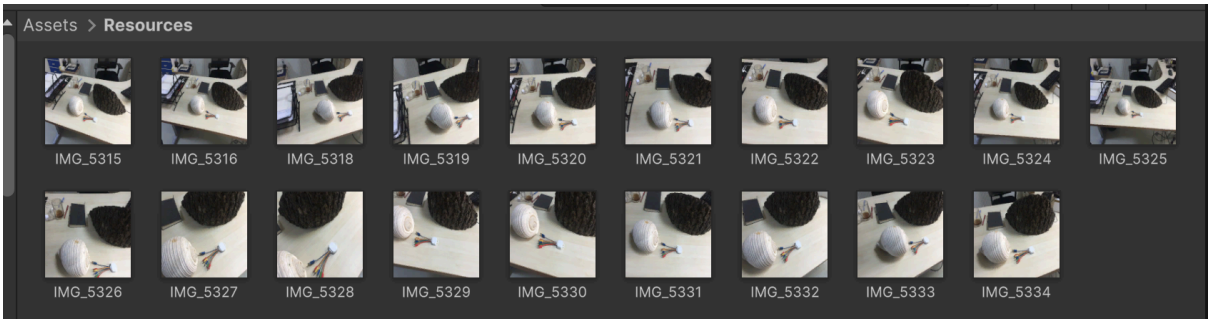






Implementation Details

1. **Image Plane:**
    - Each image was loaded from the `Resources` folder into Unity.



    - A plane was created for each image and positioned in the scene according to the camera's parameters.
    - The image plane was scaled to match the field of view (FOV) of the camera.

```
// Position the image plane
imagePlane.transform.position = unityCamera.transform.position + unityCamera.transform.forward * planeDistance;
imagePlane.transform.LookAt(unityCamera.transform.position);
imagePlane.transform.Rotate(0f, 180f, 0f);

// Scale the image plane to match the camera's FOV
float planeHeight = 2f * planeDistance * Mathf.Tan(unityCamera.fieldOfView * 0.5f * Mathf.Deg2Rad);
float planeWidth = planeHeight * unityCamera.aspect;
imagePlane.transform.localScale = new Vector3(planeWidth, planeHeight, 1f);
```

2. **Camera Configuration:**
    - A Unity camera was created for each image.
    - Camera parameters (focal length, rotation, and translation) were applied to position and orient the camera in the Unity scene. The camera is inverted because the image was looking upside down on the game.

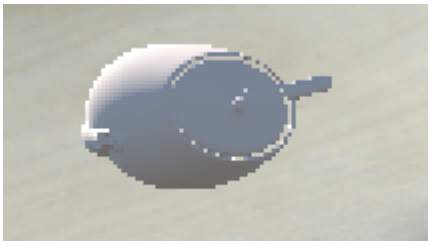
```
// Set camera position and rotation
unityCamera.transform.position = cam.translation;
Matrix4x4 rotationMatrix = MatrixFrom2DArray(cam.rotation);
unityCamera.transform.rotation = RotationFromMatrix(rotationMatrix);

// Invert the camera's Z direction to align with Unity's coordinate system
unityCamera.transform.Rotate(0f, 0f, 180f);
```

3. **Teapot Placement:**
    - A 3D teapot was instantiated and placed programmatically using the `teapotWorldPosition`.
    - The teapot's position was projected into each camera's screen space using the Unity function `WorldToScreenPoint()`. This ensured that the teapot appeared correctly placed in all images when viewed through their respective cameras.
    - The teapot's orientation was adjusted to face the image plane, ensuring that it aligned with the scene's visual perspective.

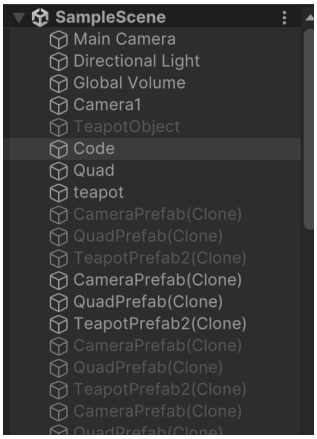
```
// Project and position the teapot marker
Vector3 projectedTeapot = unityCamera.WorldToScreenPoint(teapotWorldPosition);
teapotMarker.transform.position = unityCamera.ScreenToWorldPoint(
    new Vector3(projectedTeapot.x, projectedTeapot.y, planeDistance)
);
```

```
teapotMarker.transform.LookAt(imagePlane.transform.position);
teapotMarker.transform.Rotate(0f, 0f, 90f); // Rotate the teapot marker to face the image plane
```

In game:

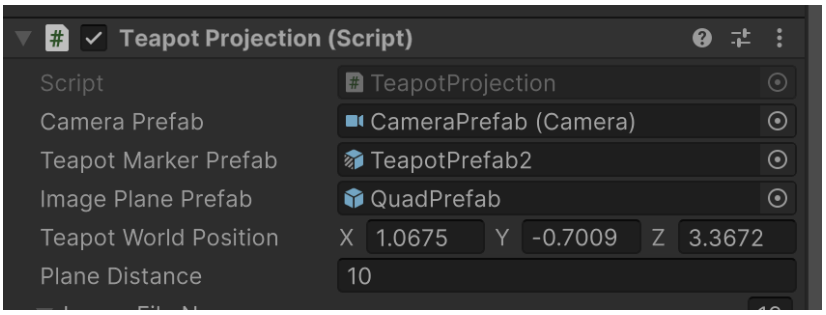


4. **Cloning Setup:**
    - Each camera, image plane, and teapot was instantiated as a separate object in Unity for each of the 19 images. The objects were stored in lists to make activation and deactivation based on the image index being displayed.

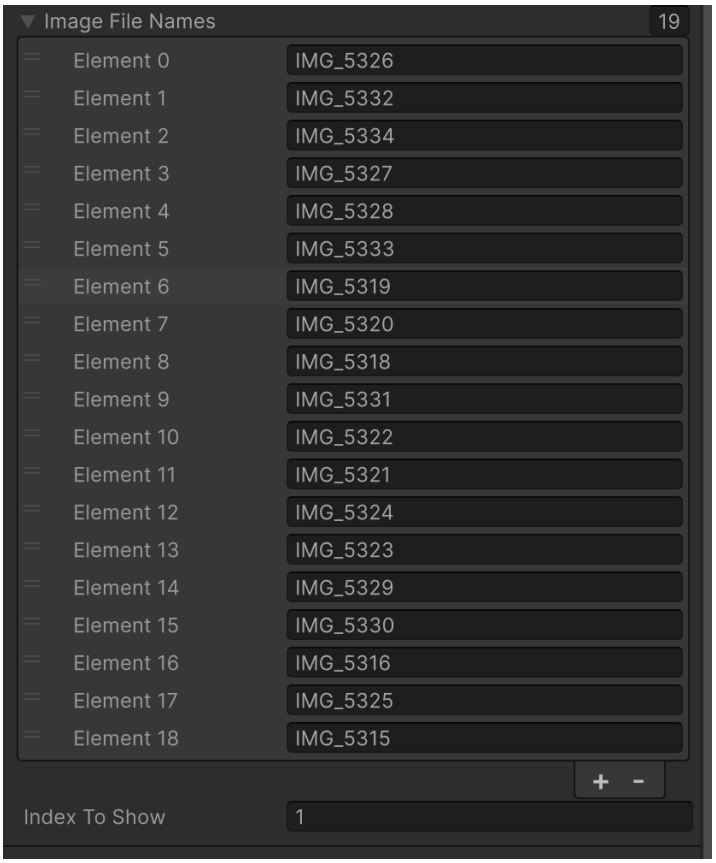



```
7 references
private List<GameObject> cameras = new List<GameObject>(); // 19 cameras
3 references
private List<GameObject> imagePlanes = new List<GameObject>(); // 19 image planes
3 references
private List<GameObject> teapotMarkers = new List<GameObject>(); // 19 teapot markers
```

C# Script inspector:



- I am changing the scene in here with Index To Show:

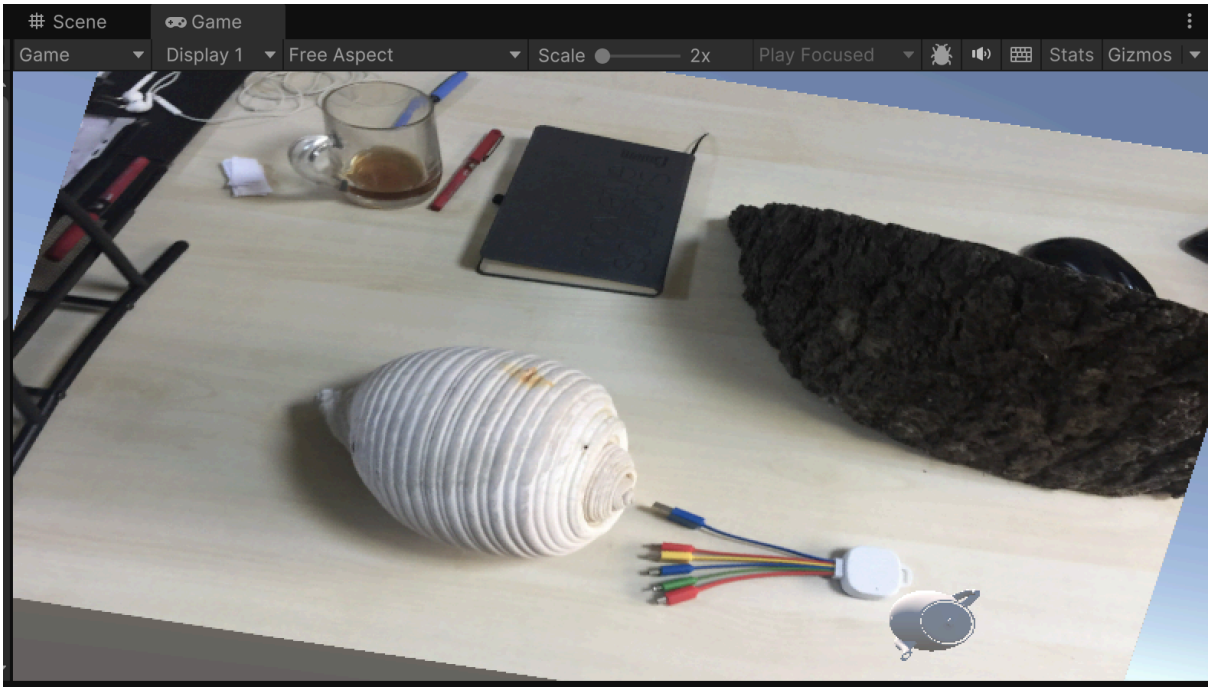


Results

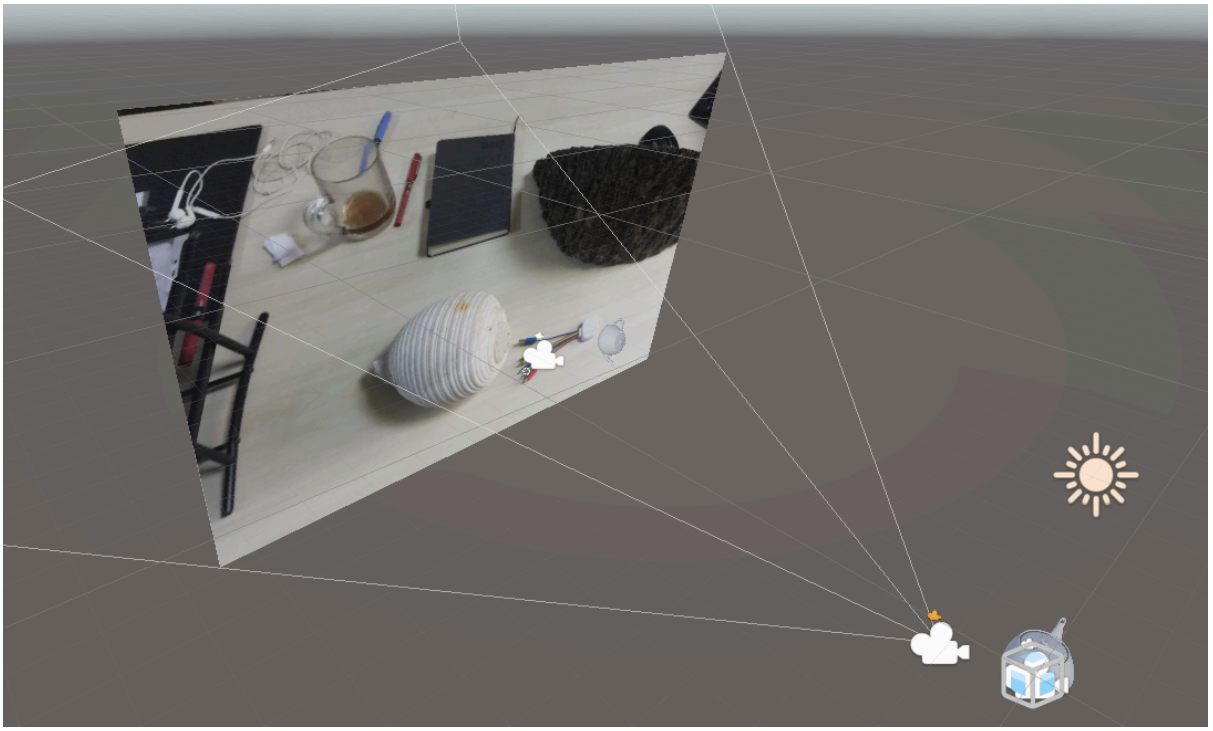
I couldn’t achieve a perfect projection alignment directly onto the USB connector, though it is generally close across the images. To solve this, I tried modifying the Z-axis values, changing camera parameters, and applying various offsets to the teapot’s position, yet none of these attempts gave me the precise result. Despite these challenges, the projections remain visually consistent, and the teapot placement closely matches with the USB position in most cases.

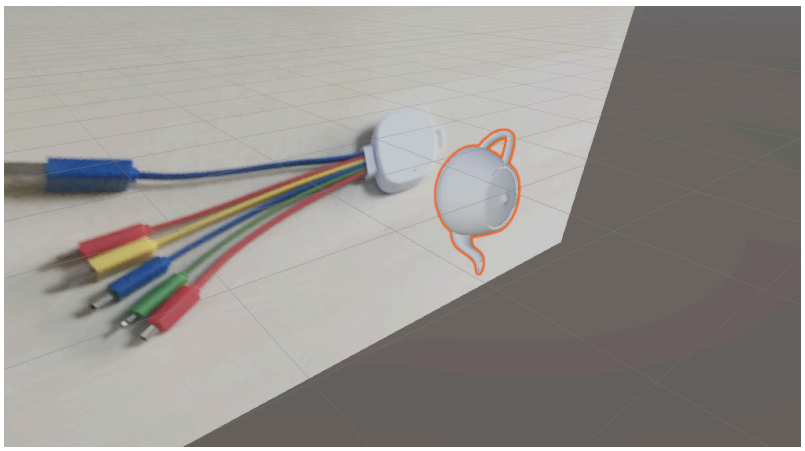
Example result for IMG_5321:

Game Screen:



Scene:





Debugs:

```
[03:43:53] Camera 12 Parameters:
UnityEngine.Debug:Log (object)
[03:43:53] Focal Length: 2879.719
UnityEngine.Debug:Log (object)
[03:43:53] Rotation Matrix:
UnityEngine.Debug:Log (object)
[03:43:53] 0.978119 -0.198142 0.063438
UnityEngine.Debug:Log (object)
[03:43:53] -0.208040 -0.934442 0.289031
UnityEngine.Debug:Log (object)
[03:43:53] 0.002009 -0.295904 -0.955216
UnityEngine.Debug:Log (object)
[03:43:53] Translation: (-0.20, -0.56, -1.27)
UnityEngine.Debug:Log (object)
```

```
[03:43:53] Camera 12: Teapot projected at (-3.08, -0.23, -11.83)
UnityEngine.Debug:Log (object)
[03:43:53] Camera 12: Image plane at (0.43, 2.33, -10.82)
UnityEngine.Debug:Log (object)
[03:43:53] Camera 12: Camera at (-0.20, -0.56, -1.27)
UnityEngine.Debug:Log (object)
```

All Projections:

IMG_5326:

```
[03:58:17] Camera 1: Teapot projected at (-3.57, 0.19, -10.40)
UnityEngine.Debug:Log (object)
[03:58:17] Camera 1: Image plane at (-0.17, 1.48, -10.14)
UnityEngine.Debug:Log (object)
[03:58:17] Camera 1: Camera at (-0.18, -0.46, -0.33)
UnityEngine.Debug:Log (object)
```

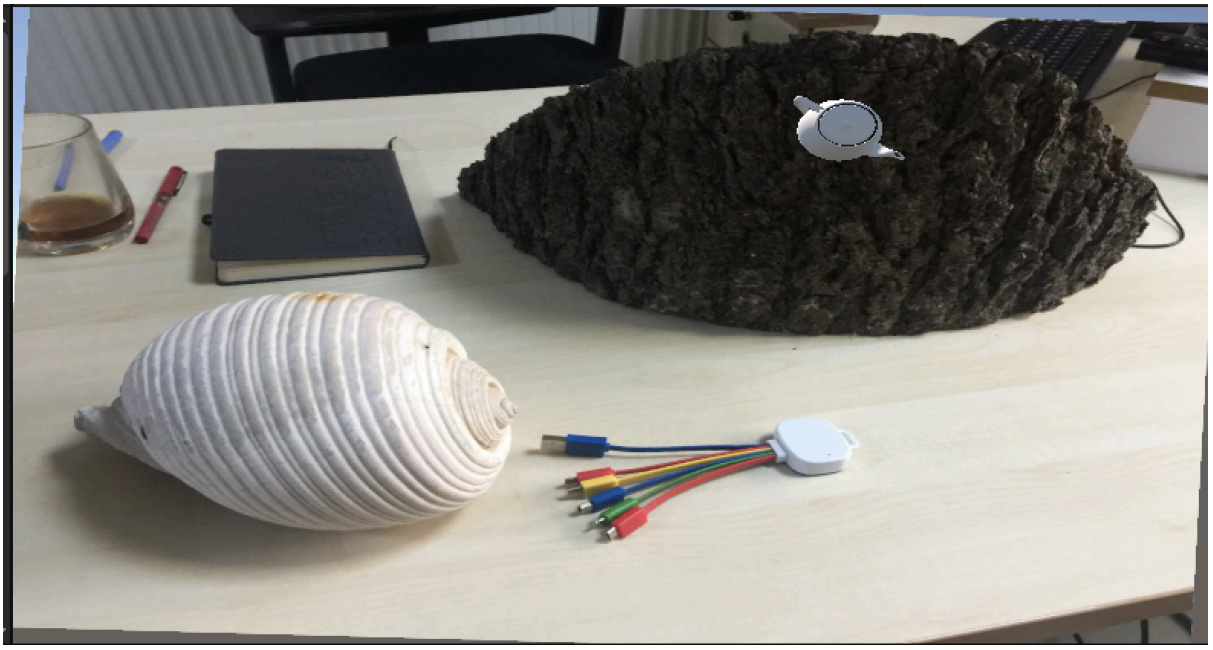
IMG_5332:

[03:58:17] Camera 2: Teapot projected at (-2.96, 1.28, -9.80)
UnityEngine.Debug:Log (object)

[03:58:17] Camera 2: Image plane at (-2.86, 1.64, -9.77)
UnityEngine.Debug:Log (object)

[03:58:17] Camera 2: Camera at (-0.07, -0.14, -0.33)
UnityEngine.Debug:Log (object)

IMG_5334:

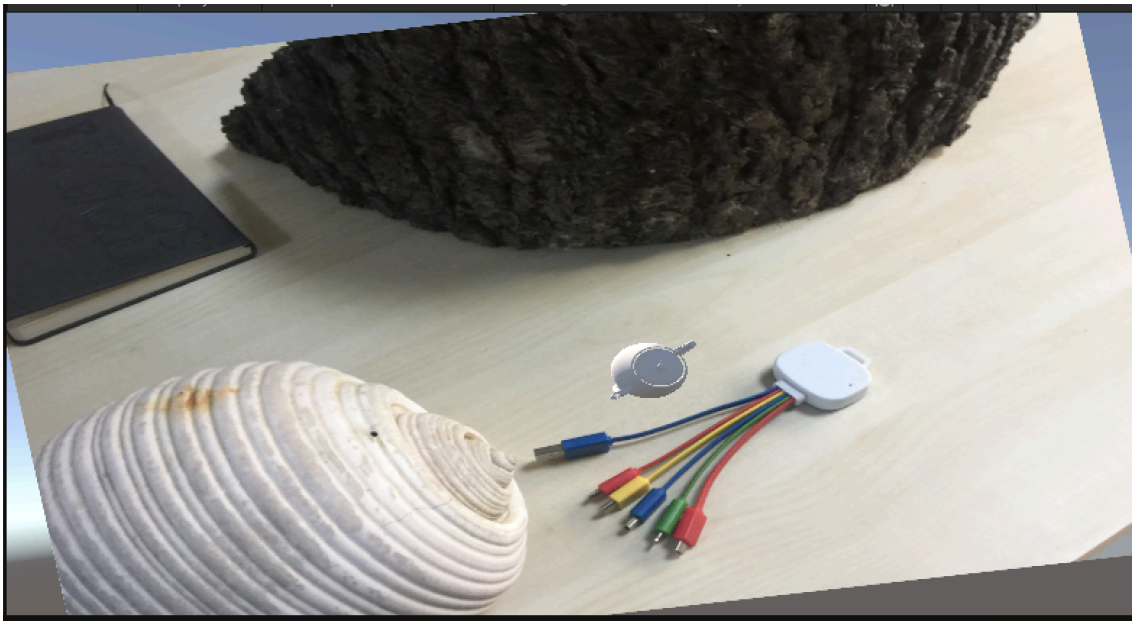


[03:58:17] Camera 3: Teapot projected at (-3.67, 2.79, -10.45)
UnityEngine.Debug:Log (object)

[03:58:17] Camera 3: Image plane at (-1.64, 0.14, -10.66)
UnityEngine.Debug:Log (object)

[03:58:17] Camera 3: Camera at (-0.34, 0.34, -0.75)
UnityEngine.Debug:Log (object)

IMG_5327:



[04:00:29] Camera 4: Teapot projected at (-3.35, 0.72, -9.35)
UnityEngine.Debug:Log (object)

[04:00:29] Camera 4: Image plane at (-2.74, 1.65, -9.32)
UnityEngine.Debug:Log (object)

[04:00:29] Camera 4: Camera at (-0.07, -0.34, 0.10)
UnityEngine.Debug:Log (object)

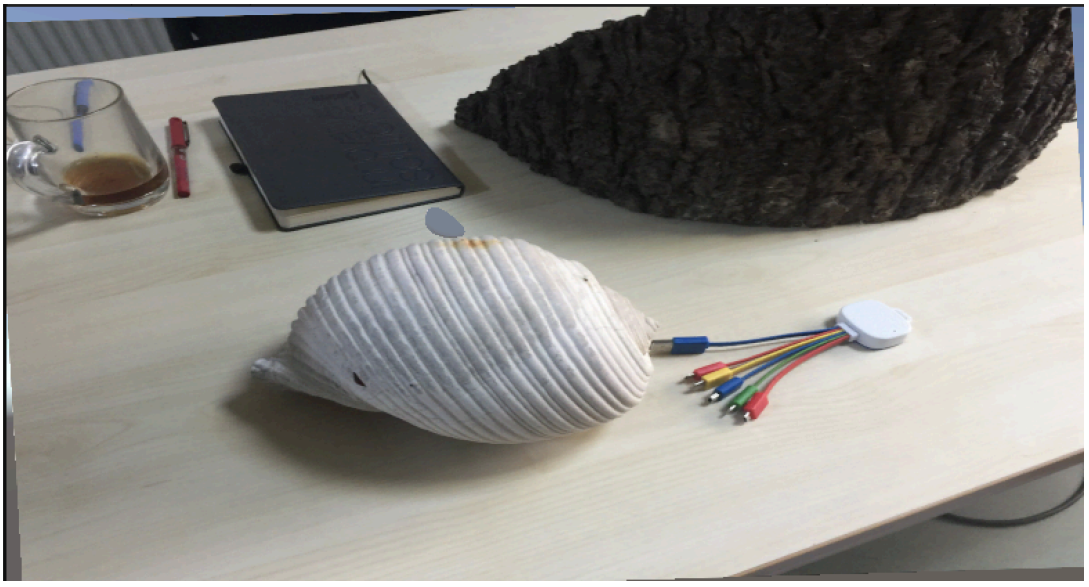
IMG_5328:

[04:00:29] Camera 5: Teapot projected at (-3.64, 0.87, -8.71)
UnityEngine.Debug:Log (object)

[04:00:29] Camera 5: Image plane at (-3.92, 0.59, -8.62)
UnityEngine.Debug:Log (object)

[04:00:29] Camera 5: Camera at (-0.03, -0.33, 0.55)
UnityEngine.Debug:Log (object)

IMG_5333:



[04:00:29] Camera 6: Teapot projected at (-0.93, 2.42, -10.57)
UnityEngine.Debug:Log (object)

[04:00:29] Camera 6: Image plane at (-1.87, 1.24, -10.46)
UnityEngine.Debug:Log (object)

[04:00:29] Camera 6: Camera at (0.47, 0.23, -0.79)
UnityEngine.Debug:Log (object)

IMG_5319:

[04:00:29] Camera 7: Teapot projected at (0.37, 1.57, -11.41)
UnityEngine.Debug:Log (object)

[04:00:29] Camera 7: Image plane at (-0.77, 0.49, -11.28)
UnityEngine.Debug:Log (object)

[04:00:29] Camera 7: Camera at (0.84, 0.03, -1.42)
UnityEngine.Debug:Log (object)

IMG_5320:

[04:00:29] Camera 8: Teapot projected at (-2.33, 1.69, -11.44)
UnityEngine.Debug:Log (object)

[04:00:29] Camera 8: Image plane at (-0.05, 0.41, -11.48)
UnityEngine.Debug:Log (object)

[04:00:29] Camera 8: Camera at (-0.05, 0.08, -1.49)
UnityEngine.Debug:Log (object)

IMG_5318:

[04:00:29] Camera 9: Teapot projected at (0.09, 1.53, -12.17)
UnityEngine.Debug:Log (object)

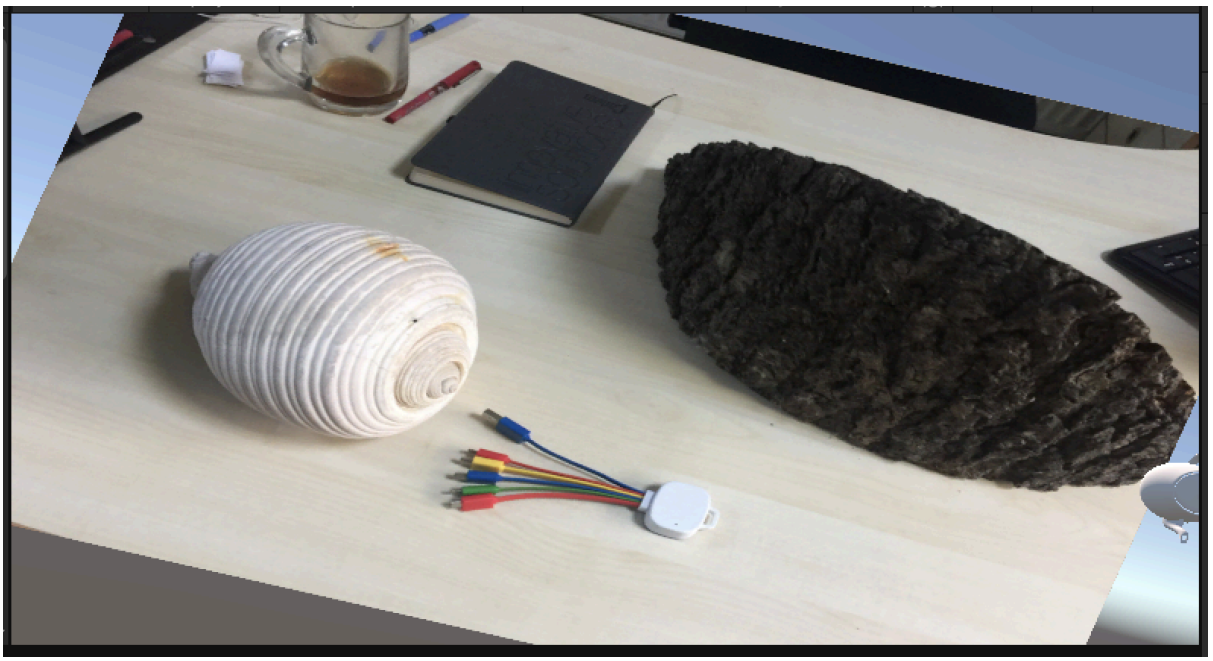
[04:00:29] Camera 9: Image plane at (0.20, 0.80, -12.23)
UnityEngine.Debug:Log (object)

[04:00:29] Camera 9: Camera at (0.71, 0.11, -2.27)
UnityEngine.Debug:Log (object)

IMG_5331:

```
[04:00:29] Camera 10: Teapot projected at (-5.20, 1.35, -10.97)
UnityEngine.Debug:Log (object)
[04:00:29] Camera 10: Image plane at (-0.24, 2.61, -10.36)
UnityEngine.Debug:Log (object)
[04:00:29] Camera 10: Camera at (-0.73, -0.11, -0.75)
UnityEngine.Debug:Log (object)
```

IMG_5322:



[04:00:29] Camera 11: Teapot projected at (-5.41, 1.23, -11.84)
UnityEngine.Debug:Log (object)

[04:00:29] Camera 11: Image plane at (0.41, 1.67, -10.99)
UnityEngine.Debug:Log (object)

[04:00:29] Camera 11: Camera at (-0.89, -0.12, -1.23)
UnityEngine.Debug:Log (object)

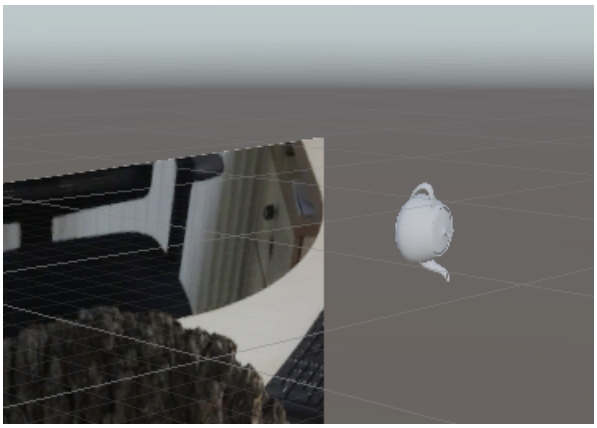
IMG_5324:

[04:00:29] Camera 13: Teapot projected at (-4.22, 1.14, -12.83)
UnityEngine.Debug:Log (object)

[04:00:29] Camera 13: Image plane at (-0.08, 1.17, -12.49)
UnityEngine.Debug:Log (object)

[04:00:29] Camera 13: Camera at (-0.88, -0.02, -2.59)
UnityEngine.Debug:Log (object)

IMG_5323:

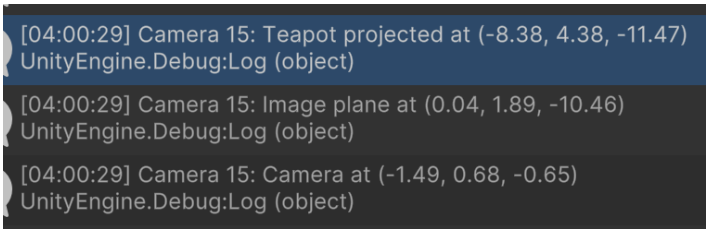


[04:00:29] Camera 14: Teapot projected at (-6.65, 3.49, -12.60)
UnityEngine.Debug:Log (object)

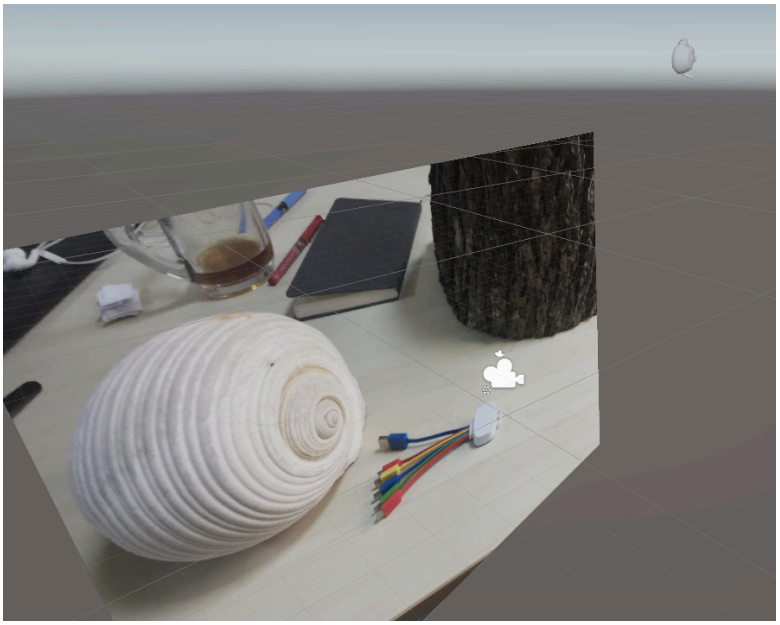
[04:00:29] Camera 14: Image plane at (0.72, 0.38, -11.12)
UnityEngine.Debug:Log (object)

[04:00:29] Camera 14: Camera at (-1.19, 0.53, -1.30)
UnityEngine.Debug:Log (object)

IMG_5329:



IMG_5330:



[04:00:29] Camera 16: Teapot projected at (-8.87, 6.59, -13.02)
UnityEngine.Debug:Log (object)

[04:00:29] Camera 16: Image plane at (1.23, 0.74, -10.20)
UnityEngine.Debug:Log (object)

[04:00:29] Camera 16: Camera at (-1.30, 1.03, -0.53)
UnityEngine.Debug:Log (object)

IMG_5316:

[04:00:29] Camera 17: Teapot projected at (-3.72, 2.45, -14.26)
UnityEngine.Debug:Log (object)

[04:00:29] Camera 17: Image plane at (1.71, 0.24, -12.86)
UnityEngine.Debug:Log (object)

[04:00:29] Camera 17: Camera at (-0.70, 0.46, -3.16)
UnityEngine.Debug:Log (object)

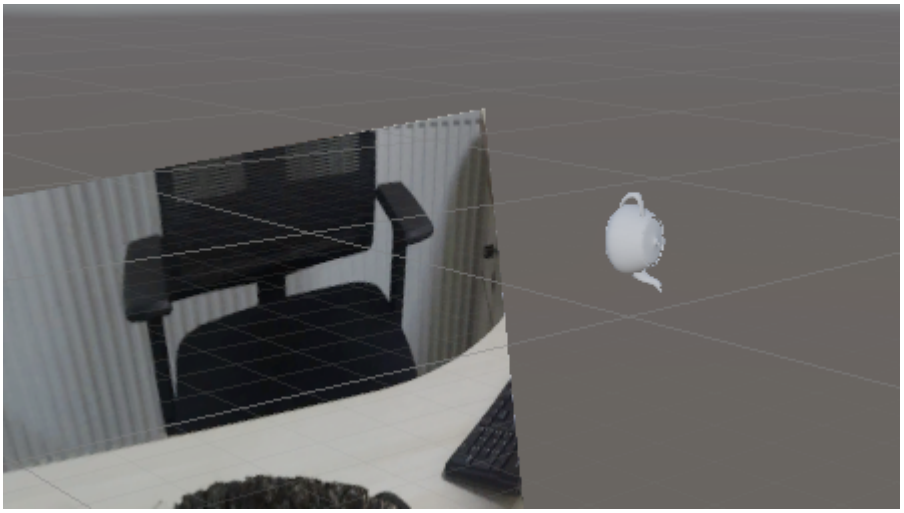
IMG_5325:

[04:00:29] Camera 18: Teapot projected at (-3.12, 1.69, -14.49)
UnityEngine.Debug:Log (object)

[04:00:29] Camera 18: Image plane at (0.46, 0.28, -14.07)
UnityEngine.Debug:Log (object)

[04:00:29] Camera 18: Camera at (-0.69, 0.30, -4.14)
UnityEngine.Debug:Log (object)

IMG_5315:



[04:00:29] Camera 19: Teapot projected at (-5.51, 2.58, -15.14)
UnityEngine.Debug:Log (object)

[04:00:29] Camera 19: Image plane at (1.97, 0.05, -12.52)
UnityEngine.Debug:Log (object)

[04:00:29] Camera 19: Camera at (-1.21, 0.44, -3.05)
UnityEngine.Debug:Log (object)