GEBZE TECHNICAL UNIVERSITY
COMPUTER ENGINEERING
DEPARTMENT

CSE462 Augmented Reality Fall 2024
Homework 3 Report

Osmancan Bozali
220104004011

Problem 1:

1.1: Function that calculates the homography matrix

```
Matrix<double> CalculateHomography(Vector2[] srcPoints, Vector2[]
dstPoints)
    {
        int n = srcPoints.Length;
        var A = DenseMatrix.Create(2 * n, 9, 0);

        for (int i = 0; i < n; i++)
        {
            double x = srcPoints[i].x;
            double y = srcPoints[i].y;
            double u = dstPoints[i].x;
            double v = dstPoints[i].y;

            A.SetRow(2 * i, new double[] { -x, -y, -1, 0, 0, 0, u
* x, u * y, u });
            A.SetRow(2 * i + 1, new double[] { 0, 0, 0, -x, -y, -
1, v * x, v * y, v });
        }

        var svd = A.Svd();
        var V = svd.VT.Transpose();
        var h = V.Column(V.ColumnCount - 1);

        var H = DenseMatrix.Create(3, 3, 0);
        for (int i = 0; i < 3; i++)
        {
            for (int j = 0; j < 3; j++)
            {
                H[i, j] = h[i * 3 + j];
            }
        }
        H = (DenseMatrix)H.Divide(H[2, 2]);
```

```
        return H;
    }
```

1.3: Function that takes a scene point and homography matrix and calculates the projection of the given point onto the target image.

```
Vector2 ProjectSceneToImage(Vector2 scenePoint, Matrix<double>
homography)
    {
        var p = Vector<double>.Build.Dense(3);
        p[0] = scenePoint.x;
        p[1] = scenePoint.y;
        p[2] = 1;

        var result = homography.Multiply(p);
        return new Vector2(
            (float)(result[0] / result[2]),
            (float)(result[1] / result[2])
        );
    }
```

1.4: Function that takes an image point and homography matrix and calculates the projection of the given point onto the scene.

```
Vector2 ProjectImageToScene(Vector2 imagePoint, Matrix<double>
homography)
    {
        var inverseH = homography.Inverse();
        var p = Vector<double>.Build.Dense(3);
        p[0] = imagePoint.x;
        p[1] = imagePoint.y;
        p[2] = 1;

        var result = inverseH.Multiply(p);
        return new Vector2(
```

```
            (float)(result[0] / result[2]),
            (float)(result[1] / result[2])
        );
    }
```

1.5: Manually calculated 5 point-correspondances in each image with 3 additional points. Calculated homography matrix with 5 points using the function from 1.1 and printed the results. After that tested homography matrix with 3 additional points for each image, calculated error as Euclidian Distance and printed the results.

Manually calculated points for each image and corresponding scene points:

```
scenePoints = new Vector2[]
        {
            new Vector2(100, 100),
            new Vector2(800, 100),
            new Vector2(800, 600),
            new Vector2(100, 600),
            new Vector2(500, 400)
        };

        imageData = new ImageData[3];

        // Manually calculated image points for IMG_1
        imageData[0].points = new Vector2[]
        {
            new Vector2(704, 755),
            new Vector2(686, 2378),
            new Vector2(1837, 2382),
            new Vector2(1853, 770),
            new Vector2(1384, 1687)
        };

        // Manually calculated image points for IMG_2
        imageData[1].points = new Vector2[]
```

```
        {
            new Vector2(507, 737),
            new Vector2(489, 2379),
            new Vector2(1670, 2426),
            new Vector2(1690, 708),
            new Vector2(1192, 1682)
        };

        // Manually calculated image points for IMG_3
        imageData[2].points = new Vector2[]
        {
            new Vector2(745, 876),
            new Vector2(716, 2463),
            new Vector2(1763, 2430),
            new Vector2(1901, 947),
            new Vector2(1398, 1827)
        };

        additionalScenePoints = new Vector2[] { new Vector2(100,
300), new Vector2(200, 400), new Vector2(300, 500) }; //
additional scene points [1.5]
        additionalImageData = new ImageData[3];
        additionalImageData[0].points = new Vector2[] { new
Vector2(1167, 762), new Vector2(1393, 996), new Vector2(1618,
1230) }; // additional image points for IMG_1 [1.5]
        additionalImageData[1].points = new Vector2[] { new
Vector2(969, 728), new Vector2(1202, 963), new Vector2(1438,
1202) }; // additional image points for IMG_2 [1.5]
        additionalImageData[2].points = new Vector2[] { new
Vector2(1228, 908), new Vector2(1442, 1159), new Vector2(1644,
1396) }; // additional image points for IMG_3 [1.5]
    }
```

Results:

Img1:

```
[1.1] Homography Matrix for Image 1:
-0.027461  2.323521   474.914744
2.314784   0.038899   520.617057
-0.000002  0.000013   1.000000
Results for Image 1:
[1.5]
Scene point (100.00, 300.00) -> Projected Image point (1165.09, 761.06) --> Actual Image point (1167.00, 762.00)
Error: 2.132271 pixels
Scene point (200.00, 400.00) -> Projected Image point (1392.45, 994.58) --> Actual Image point (1393.00, 996.00)
Error: 1.524837 pixels
Scene point (300.00, 500.00) -> Projected Image point (1619.35, 1227.62) --> Actual Image point (1618.00, 1230.00)
Error: 2.741933 pixels
```

Img2:

```
[1.1] Homography Matrix for Image 2:
-0.027087  2.194103   286.102628
2.322763   -0.120640  510.158221
-0.000001  -0.000088  1.000000
Results for Image 2:
[1.5]
Scene point (100.00, 300.00) -> Projected Image point (967.29, 725.49) --> Actual Image point (969.00, 728.00)
Error: 3.040167 pixels
Scene point (200.00, 400.00) -> Projected Image point (1200.86, 960.47) --> Actual Image point (1202.00, 963.00)
Error: 2.772825 pixels
Scene point (300.00, 500.00) -> Projected Image point (1438.78, 1199.83) --> Actual Image point (1438.00, 1202.00)
Error: 2.306146 pixels
```

Img3:

```
[1.1] Homography Matrix for Image 3:
0.075260   2.666007   494.366101
2.746593   0.286144   600.372542
0.000166   0.000149   1.000000
Results for Image 3:
[1.5]
Scene point (100.00, 300.00) -> Projected Image point (1226.64, 905.47) --> Actual Image point (1228.00, 908.00)
Error: 2.872994 pixels
Scene point (200.00, 400.00) -> Projected Image point (1442.20, 1156.96) --> Actual Image point (1442.00, 1159.00)
Error: 2.054591 pixels
Scene point (300.00, 500.00) -> Projected Image point (1645.69, 1394.36) --> Actual Image point (1644.00, 1396.00)
Error: 2.356088 pixels
```

1.6 and 1.7:

```
void TestProjections()
    {
        // Scene Points from [1.6]
        Vector2[] testScenePoints = new Vector2[]
        {
            new Vector2(7.5f, 5.5f),
            new Vector2(6.3f, 3.3f),
            new Vector2(0.1f, 0.1f)
        };
        // Image Points from [1.7]
        Vector2[] testImagePoints = new Vector2[]
        {
            new Vector2(500, 400),
            new Vector2(86, 167),
            new Vector2(10, 10)
        };

        for (int i = 0; i < 3; i++)
        {
            string matrixString = "\n";
            matrixString += $"Results for Image {i + 1}:\n";
            matrixString += ProjectWithErrorCalculation(i);
            matrixString += "\n[1.6] Scene to Image
Projections:\n";
            foreach (var point in testScenePoints)
            {
                Vector2 projected = ProjectSceneToImage(point,
imageData[i].homography);
                matrixString += $"Scene point {point} ->
Projected Image point {projected}\n";
            }
            matrixString += "\n[1.7] Image to Scene
Projections:\n";
            foreach (var point in testImagePoints)
            {
```

```
                Vector2 projected = ProjectImageToScene(point,
imageData[i].homography);
                matrixString += $"Image point {point} ->
Projected Scene point {projected}\n";
            }

            Debug.Log(matrixString);
        }
    }
```

Results:

Img1:

```
[1.6] Scene to Image Projections:
Scene point (7.50, 5.50) -> Projected Image point (487.46, 538.16)
Scene point (6.30, 3.30) -> Projected Image point (482.40, 535.31)
Scene point (0.10, 0.10) -> Projected Image point (475.14, 520.85)

[1.7] Image to Scene Projections:
Image point (500.00, 400.00) -> Projected Scene point (-52.24, 10.23)
Image point (86.00, 167.00) -> Projected Scene point (-150.05, -169.22)
Image point (10.00, 10.00) -> Projected Scene point (-217.19, -202.67)
```

Img2:

```
[1.6] Scene to Image Projections:
Scene point (7.50, 5.50) -> Projected Image point (298.11, 527.17)
Scene point (6.30, 3.30) -> Projected Image point (293.26, 524.55)
Scene point (0.10, 0.10) -> Projected Image point (286.32, 510.38)

[1.7] Image to Scene Projections:
Image point (500.00, 400.00) -> Projected Scene point (-43.93, 95.04)
Image point (86.00, 167.00) -> Projected Scene point (-151.96, -92.75)
Image point (10.00, 10.00) -> Projected Scene point (-221.95, -128.53)
```

Img3:

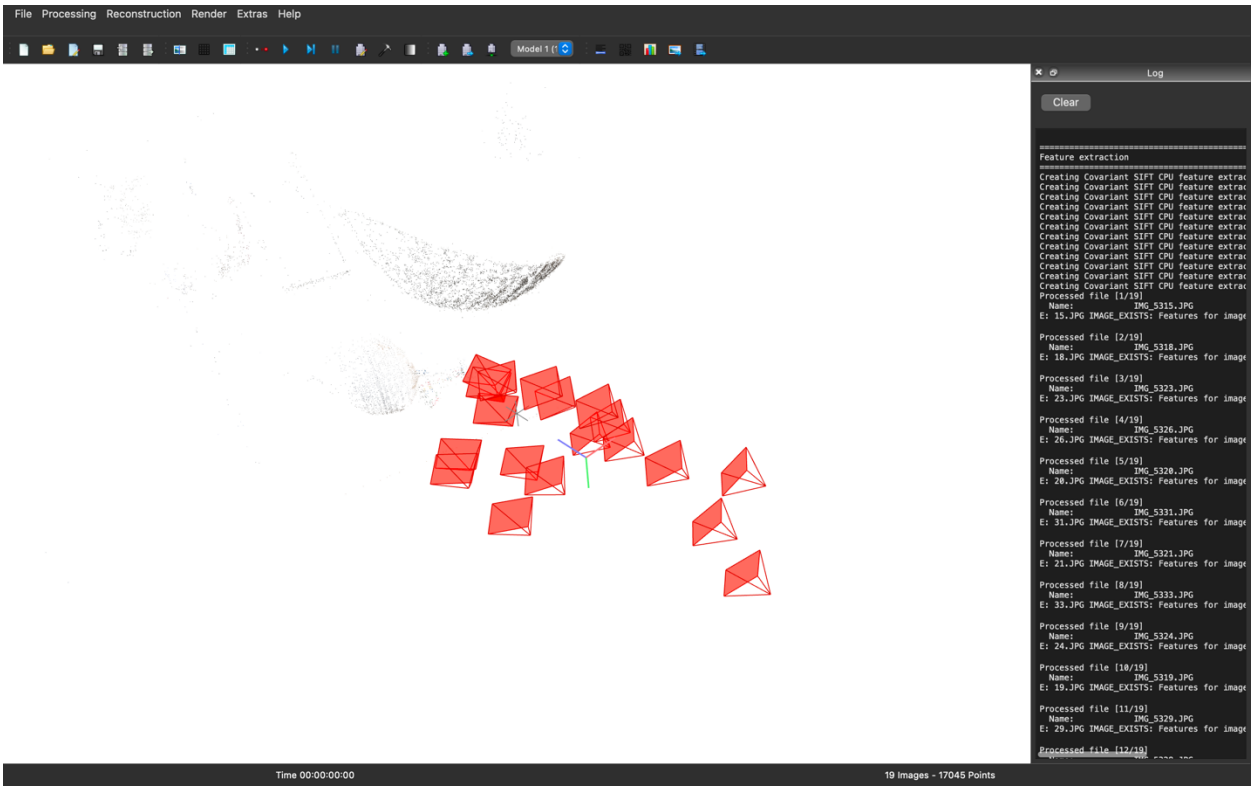
```
[1.6] Scene to Image Projections:
Scene point (7.50, 5.50) -> Projected Image point (508.54, 621.26)
Scene point (6.30, 3.30) -> Projected Image point (502.87, 617.67)
Scene point (0.10, 0.10) -> Projected Image point (494.62, 600.66)

[1.7] Image to Scene Projections:
Image point (500.00, 400.00) -> Projected Scene point (-74.93, 1.95)
Image point (86.00, 167.00) -> Projected Scene point (-144.92, -150.58)
Image point (10.00, 10.00) -> Projected Scene point (-196.79, -176.35)
```

Problem 2:

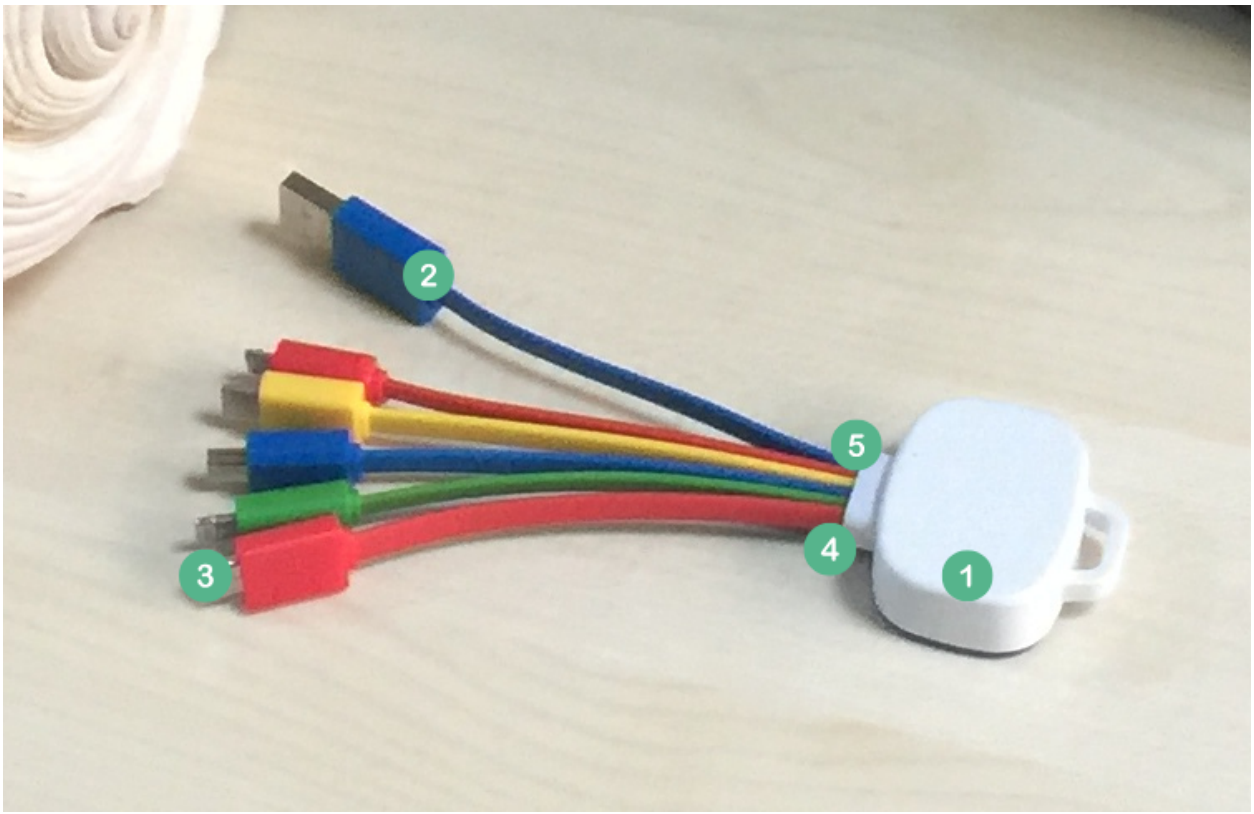
To calculate the placement geometry, I used camera data obtained using COLMAP, which provided both intrinsic and extrinsic parameters These parameters used to understand how the camera views the scene.

COLMAP for obtaining camera parameters:



I started by selecting the first image as the reference image, where I manually chose 5 specific points that were easy to identify and consistent across other images. Additionally, I selected a placement point as the sixth point, which would be used to position the object. For every new image, I calculated the homography matrix by matching these same 5 points selected in the Unity UI to the corresponding points in the reference image. This allowed me to transform the placement point from the reference image into the target image coordinates. To ensure the object's position and orientation were accurate, I also used camera data obtained from COLMAP. This data included details about how the camera sees the image. Using this information, I adjusted the rotation of the object to better match the perspective of the camera. Finally, I used the calculated placement point and the adjusted rotation to project the teapot into the scene. Although this approach worked well in some cases, I noticed that there were errors in the projection when the perspective and the scale of the target image was very different from the reference image.

Reference Points I Used:



Results (selected points are indicated with red dots):

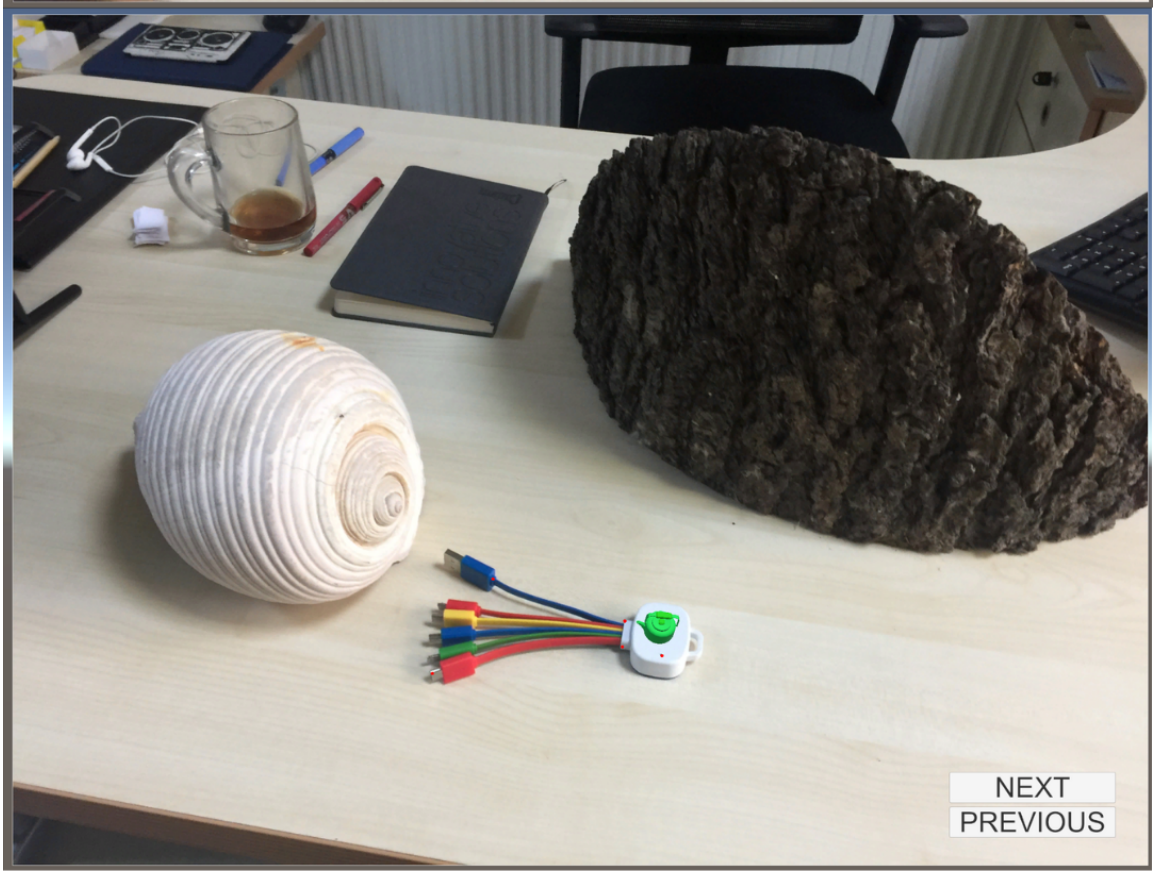

NEXT
PREVIOUS

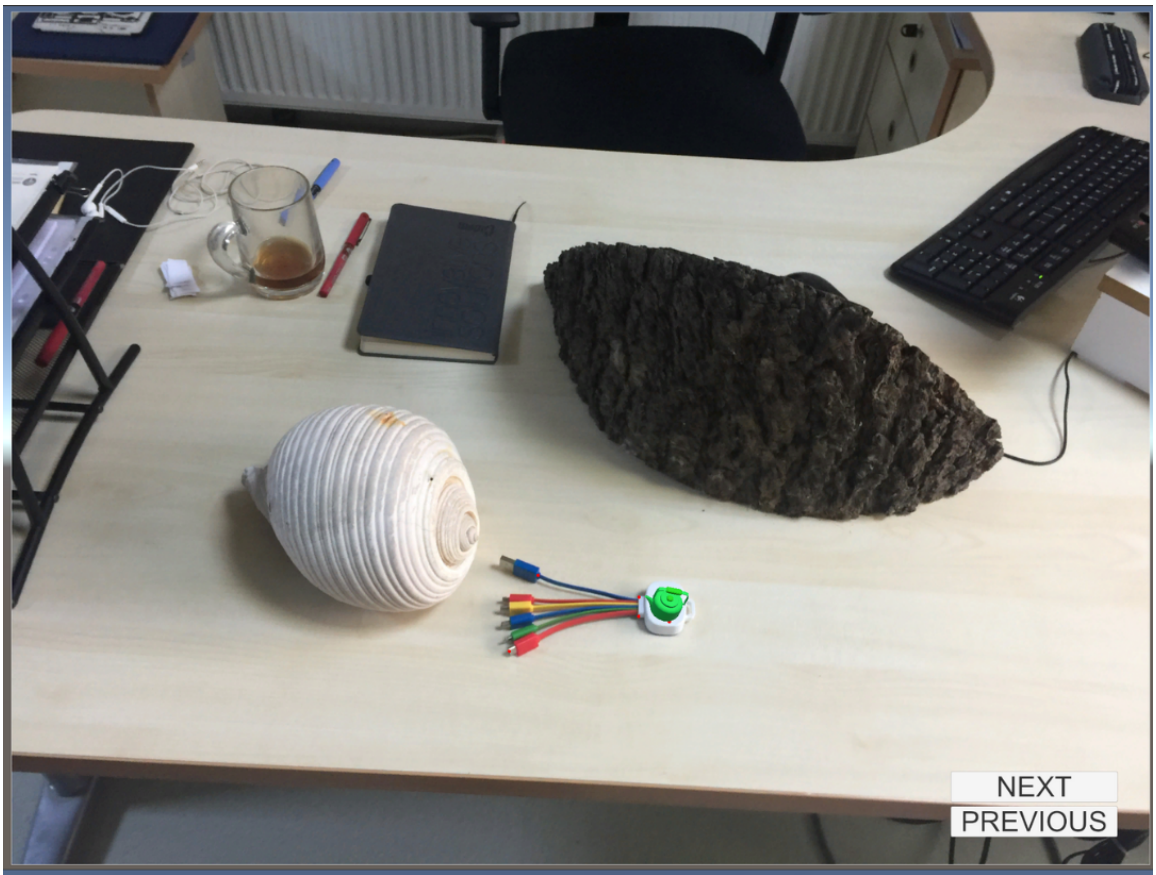

NEXT
PREVIOUS

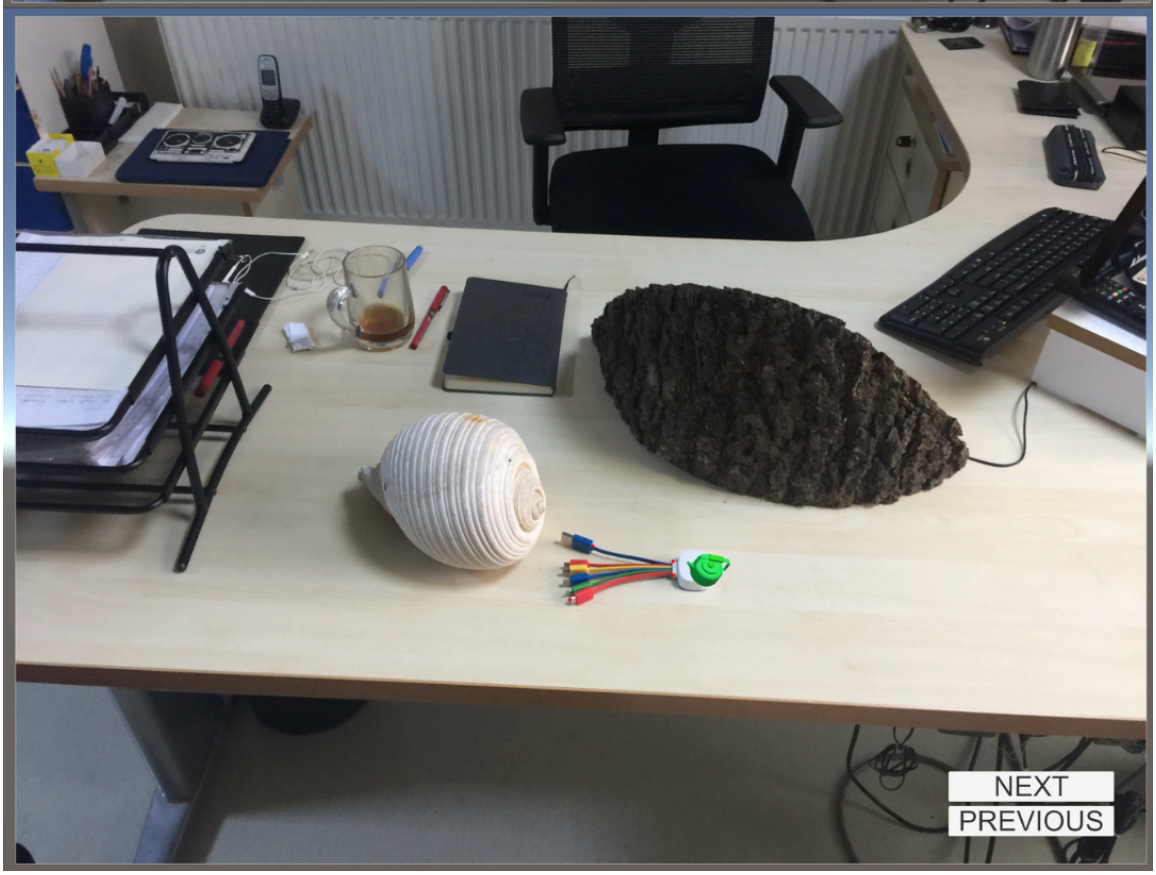

NEXT
PREVIOUS


NEXT
PREVIOUS


NEXT
PREVIOUS

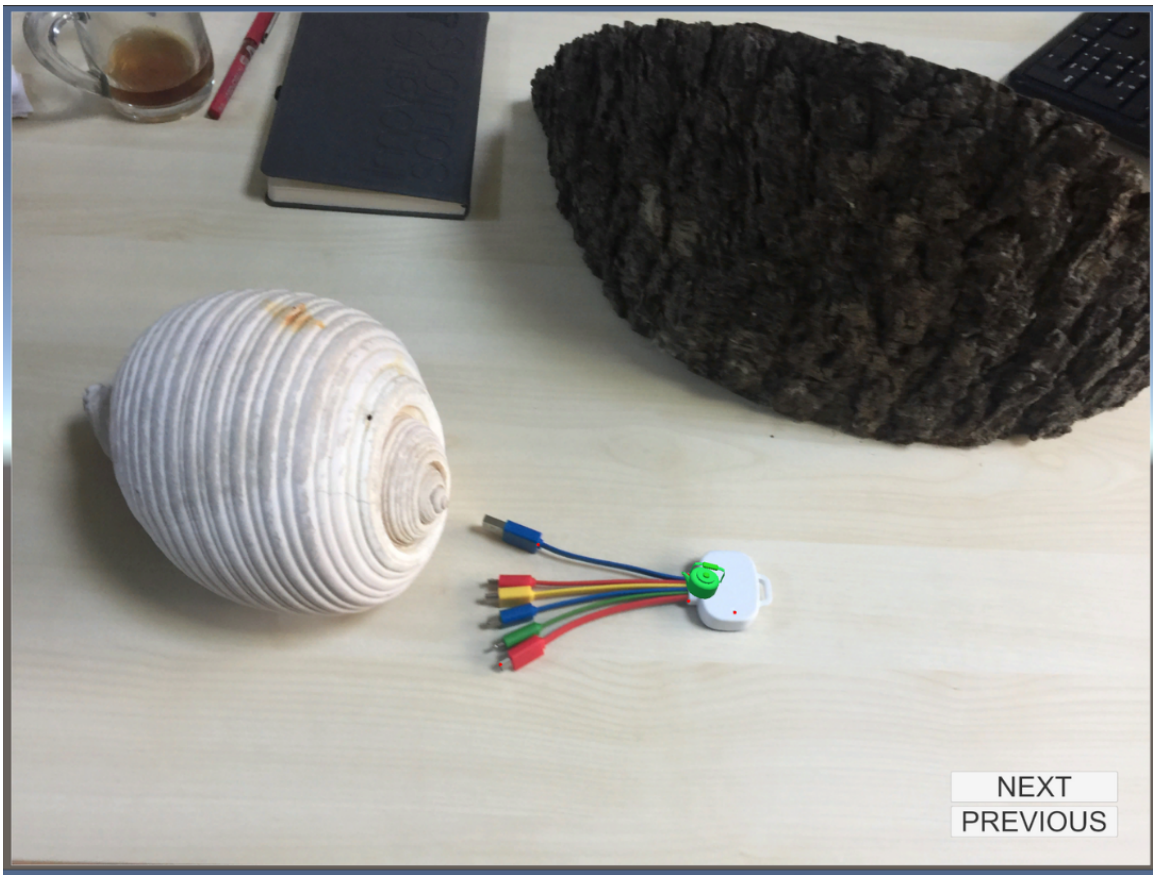

NEXT
PREVIOUS


NEXT
PREVIOUS


NEXT
PREVIOUS


NEXT
PREVIOUS


NEXT
PREVIOUS


NEXT
PREVIOUS


NEXT
PREVIOUS