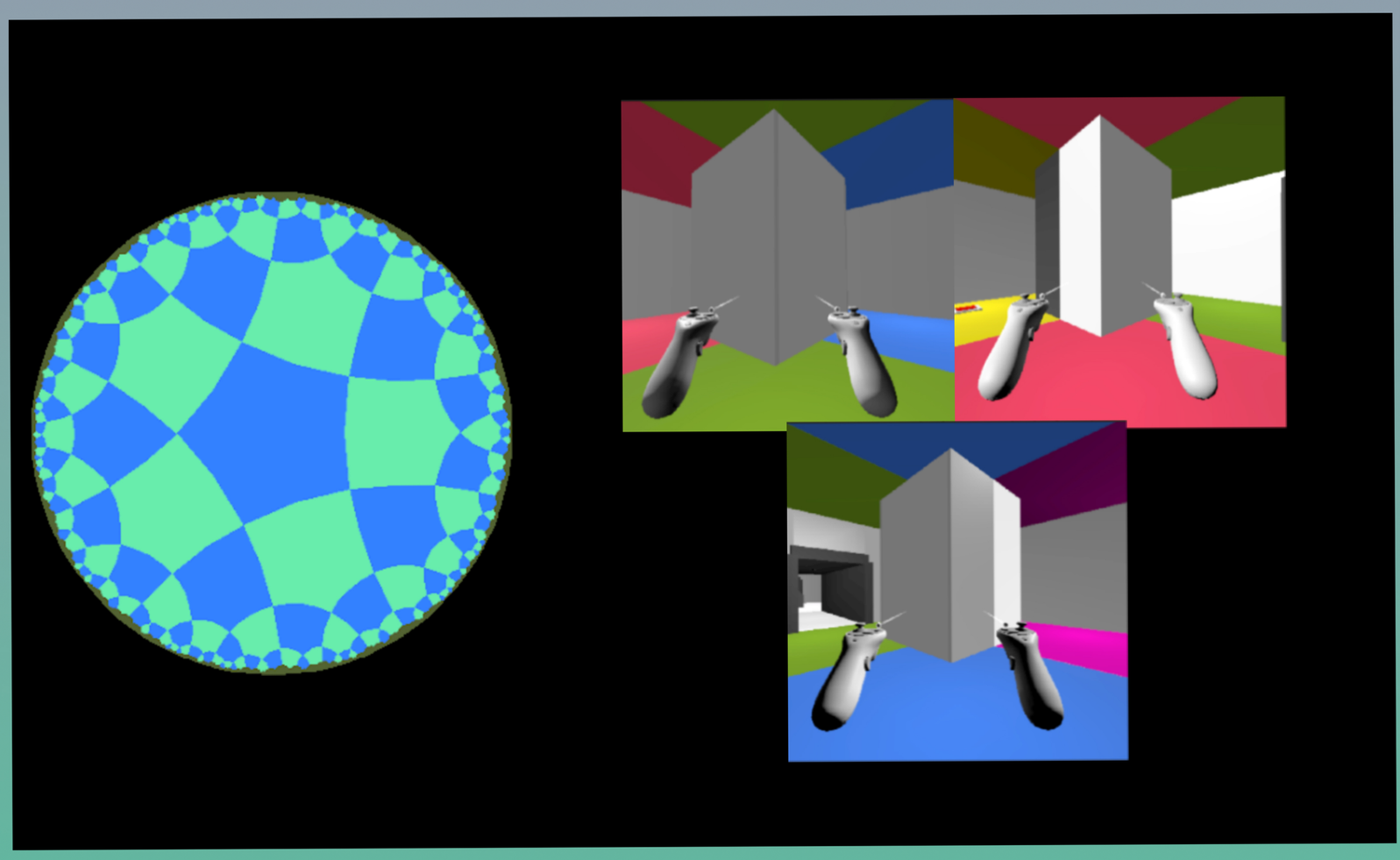
LYRA EDWARDS

BSC (HONS) COMPUTER GAMES PROGRAMMING

CREATING AN ENGAGING VIRTUAL REALITY EXPERIENCE USING NON-EUCLIDEAN GEOMETRY


```gdscript
    public void Render () {

        // Skip rendering the view from this portal if player is not looking at the linked portal
        if (!CameraUtility.VisibleFromCamera (linkedPortal.screen, playerCam)) {
            return;
        }

        CreateViewTexture ();

        var localToWorldMatrix = playerCam.transform.localToWorldMatrix;
        var renderPositions = new Vector3[recursionLimit];
        var renderRotations = new Quaternion[recursionLimit];

        int startIndex = 0;
        portalCam.projectionMatrix = playerCam.projectionMatrix;
        for (int i = 0; i < recursionLimit; i++) {
            if (i > 0) {
                // No need for recursive rendering if linked portal is not visible through this portal
                if (!CameraUtility.BoundsOverlap (screenMeshFilter, linkedPortal.screenMeshFilter, portalCam)) {
                    break;
                }
            }
            localToWorldMatrix = transform.localToWorldMatrix * linkedPortal.transform.worldToLocalMatrix * localToWorldMatrix;
            int renderOrderIndex = recursionLimit - i - 1;
            renderPositions[renderOrderIndex] = localToWorldMatrix.GetColumn (3);
            renderRotations[renderOrderIndex] = localToWorldMatrix.rotation;

            portalCam.transform.SetPositionAndRotation (renderPositions[renderOrderIndex], renderRotations[renderOrderIndex]);
            startIndex = renderOrderIndex;
        }

        // Hide screen so that camera can see through portal
        screen.shadowCastingMode = UnityEngine.Rendering.ShadowCastingMode.ShadowsOnly;
        linkedPortal.screen.material.SetInt ("displayMask", 0);

        for (int i = startIndex; i < recursionLimit; i++) {
            portalCam.transform.SetPositionAndRotation (renderPositions[i], renderRotations[i]);
            SetNearClipPlane ();
            HandleClipping ();
            //UniversalRenderPipeline.RenderSingleCamera(src, portalCam);
            portalCam.Render ();

            if (i == startIndex) {
                linkedPortal.screen.material.SetInt ("displayMask", 1);
            }
        }
}
```



```gdscript
 public static MinMax3D GetScreenRectFromBounds (MeshFilter renderer, Camera mainCamera) {

     MinMax3D minMax = new MinMax3D (float.MaxValue, float.MinValue);

     Vector3[] screenBoundsExtents = new Vector3[8];
     var localBounds = renderer.sharedMesh.bounds;
     bool anyPointIsInFrontOfCamera = false;

     for (int i = 0; i < 8; i++) {
         Vector3 localSpaceCorner = localBounds.center + Vector3.Scale (localBounds.extents, cubeCornerOffsets[i]);
         Vector3 worldSpaceCorner = renderer.transform.TransformPoint (localSpaceCorner);
         Vector3 viewportSpaceCorner = mainCamera.WorldToViewportPoint (worldSpaceCorner);


         if (viewportSpaceCorner.z > 0) {
             anyPointIsInFrontOfCamera = true;
         } else {
             // If point is behind camera, it gets flipped to the opposite side
             // So clamp to opposite edge to correct for this
             viewportSpaceCorner.x = (viewportSpaceCorner.x <= 0.5f) ? 1 : 0;
             viewportSpaceCorner.y = (viewportSpaceCorner.y <= 0.5f) ? 1 : 0;
         }

         // Update bounds with new corner point
         minMax.AddPoint (viewportSpaceCorner);
     }

     // All points are behind camera so just return empty bounds
     if (!anyPointIsInFrontOfCamera) {
         return new MinMax3D ();
     }

     return minMax;
 }
```

Unity

University of Staffordshire