Day 3
AR Foundation

Pre setup

Get 2 images that you can have in real life for image tracking and clean it up in any photo editing software (even MS paint will do in most cases)

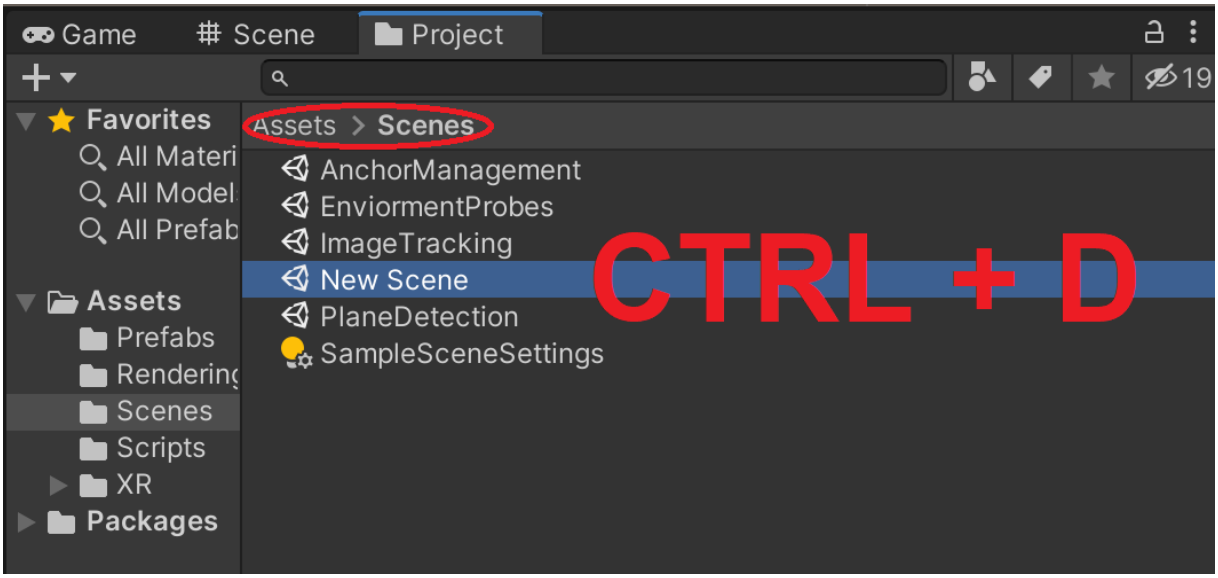
Additionally, measure the dimensions of the image in meters, this will be needed in Unity.

Try to avoid using reflective, shiny, warped, or curved surfaces.

Drag and drop the images into your project folder so unity can import them

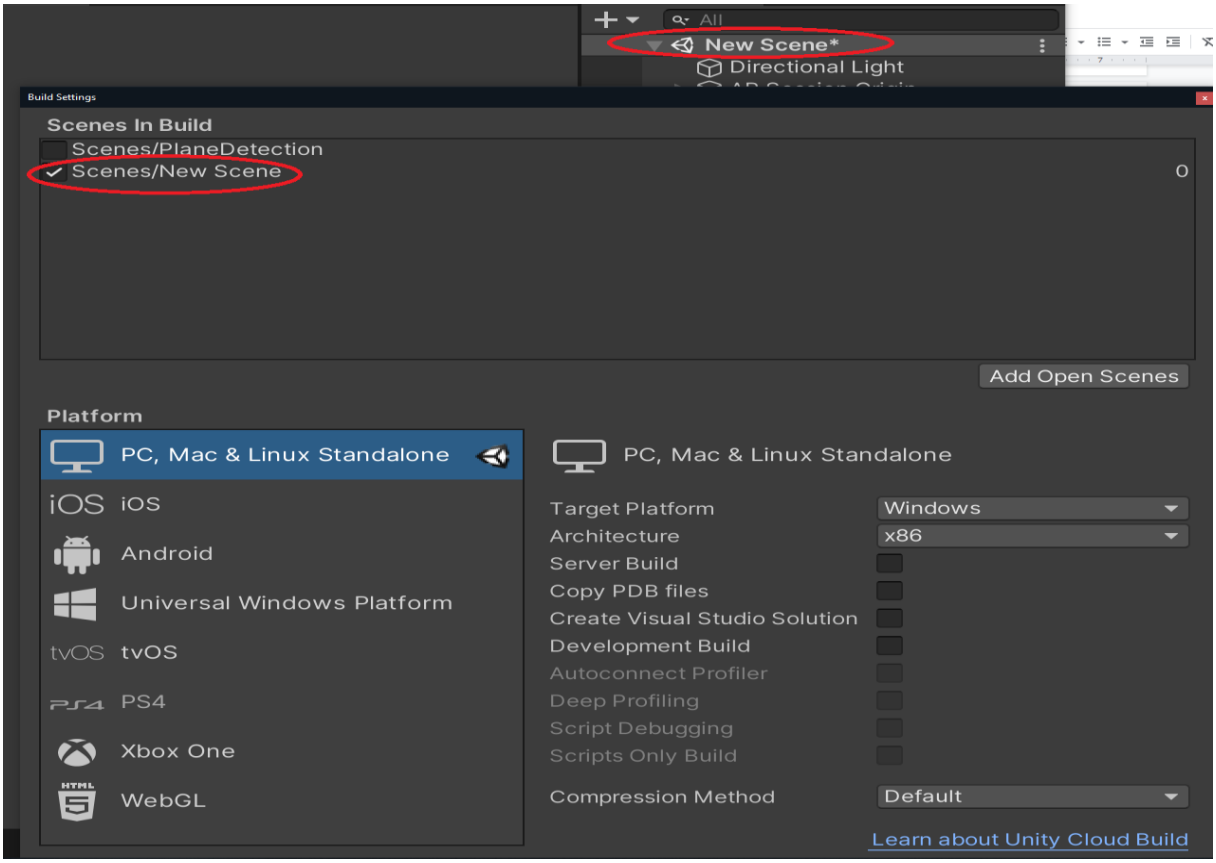
PART 1 – IMAGE TRACKING WITH SINGLE IMAGE

Duplicate the current scene by selecting the Scene in the project window CTRL+D and double click on it

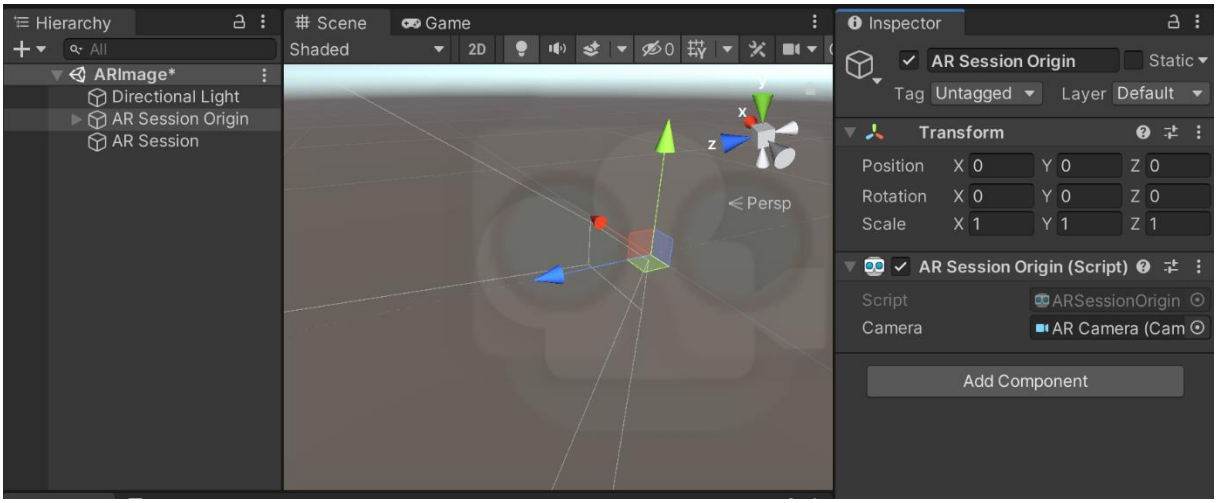


Make sure your new scene is active in the build settings.

Press Add open scenes if it is not on the list and deactivate all other scenes.

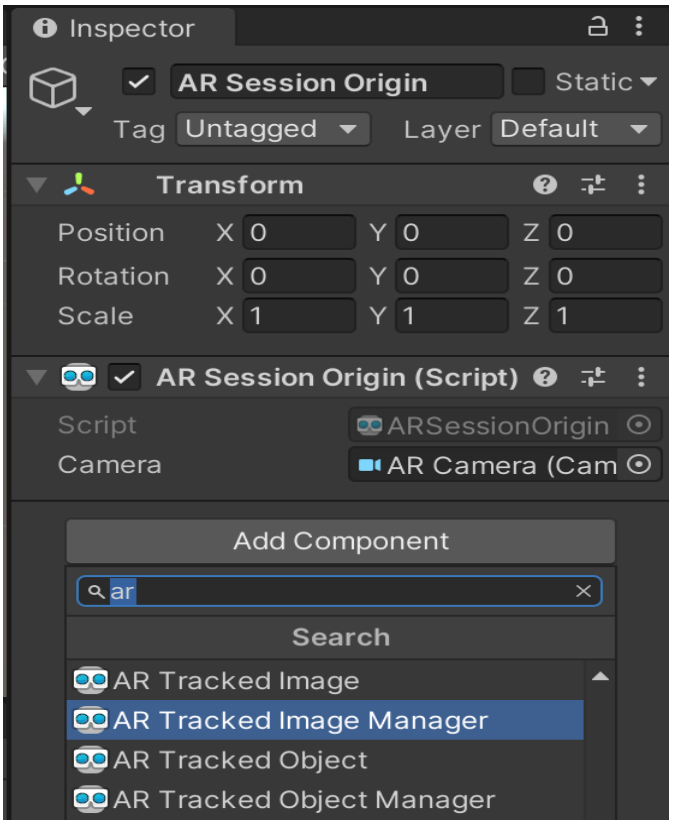


Delete all extra objects from the scene, leaving just the Light, AR Session, and AR Session origin. Also, delete any extra scripts on the AR session origin.

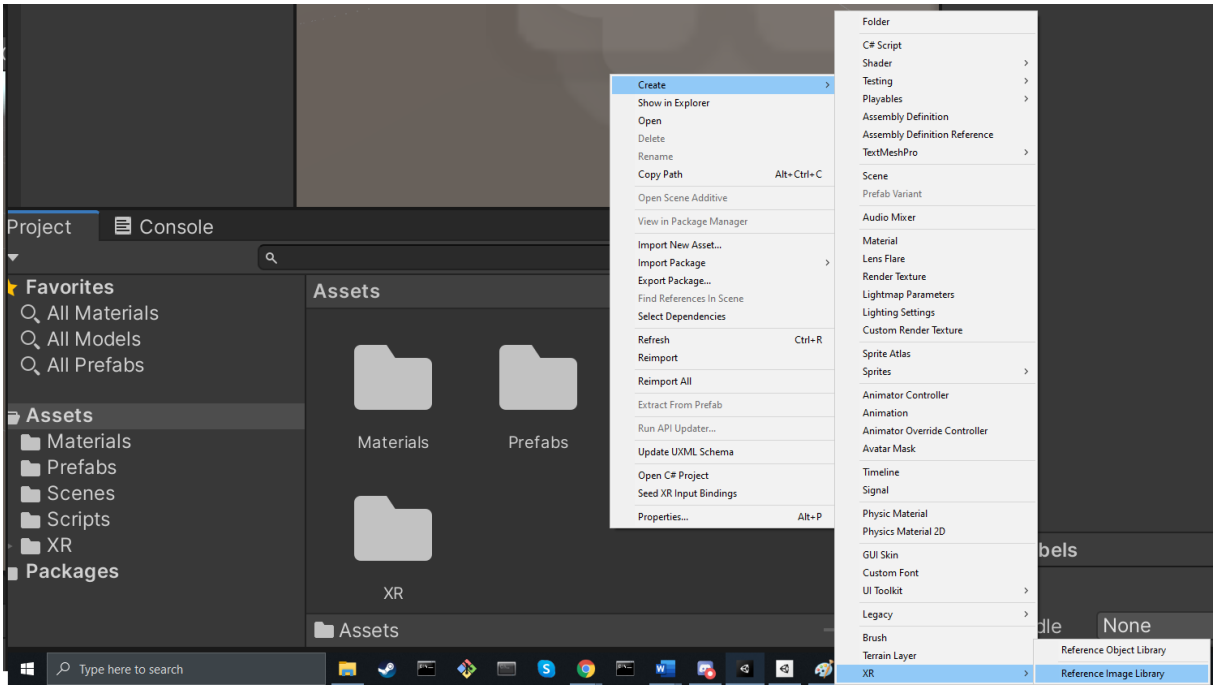


Select AR session origin.

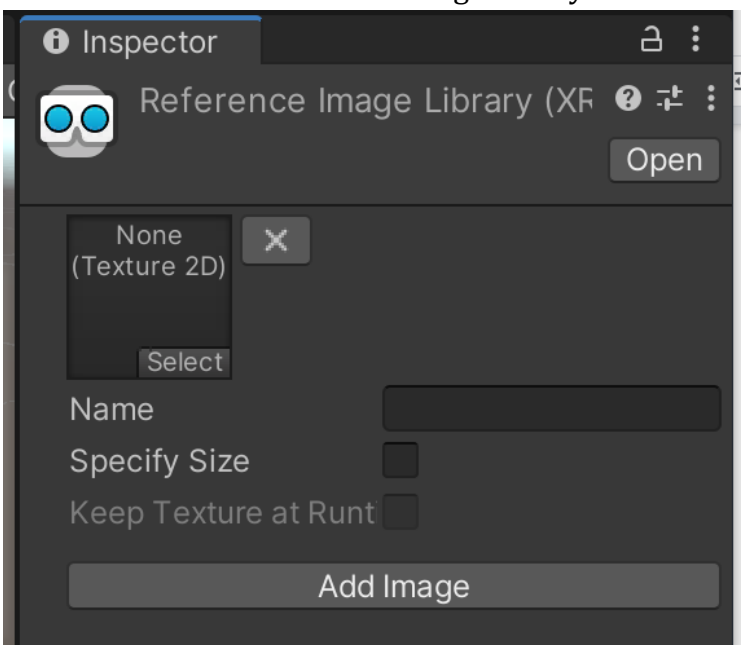
Add AR tracked image manager.



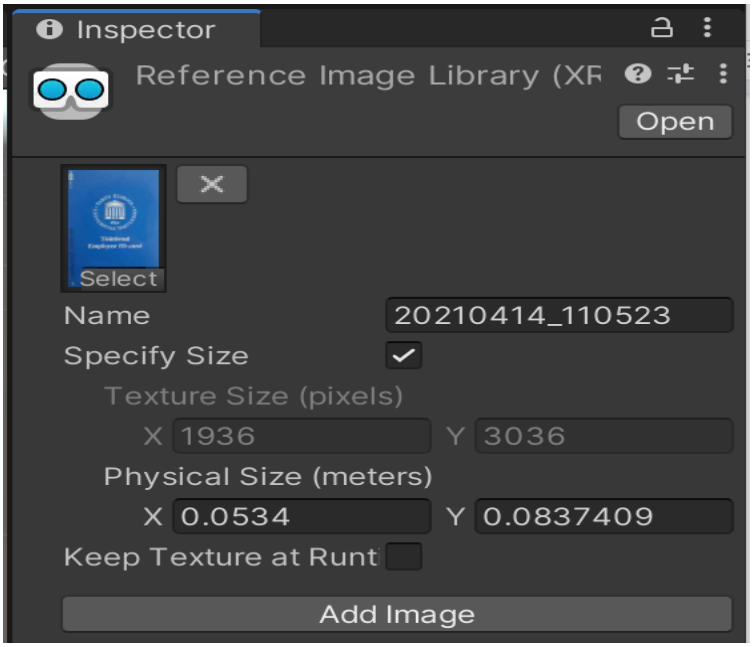
Go to Project window and Create>XR>Reference Image Library



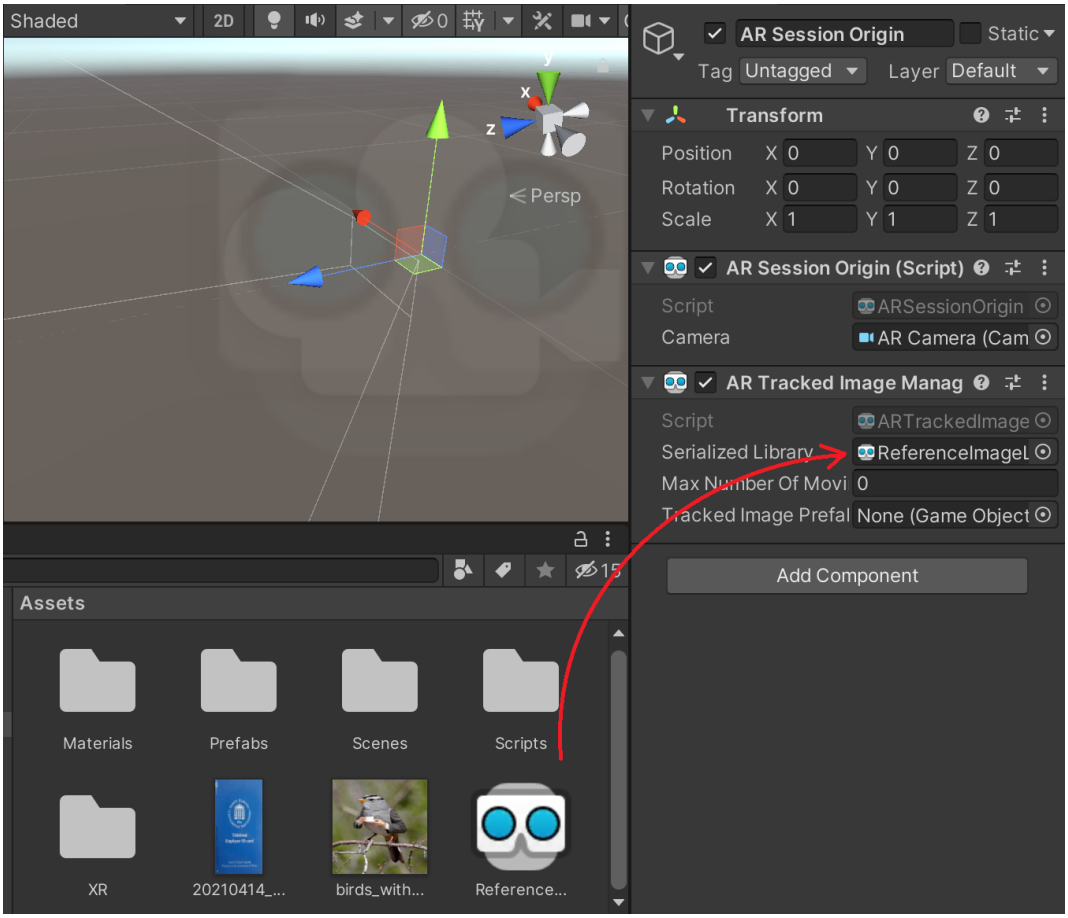
Click on the created refrence image library and in the inspector window click Add Image



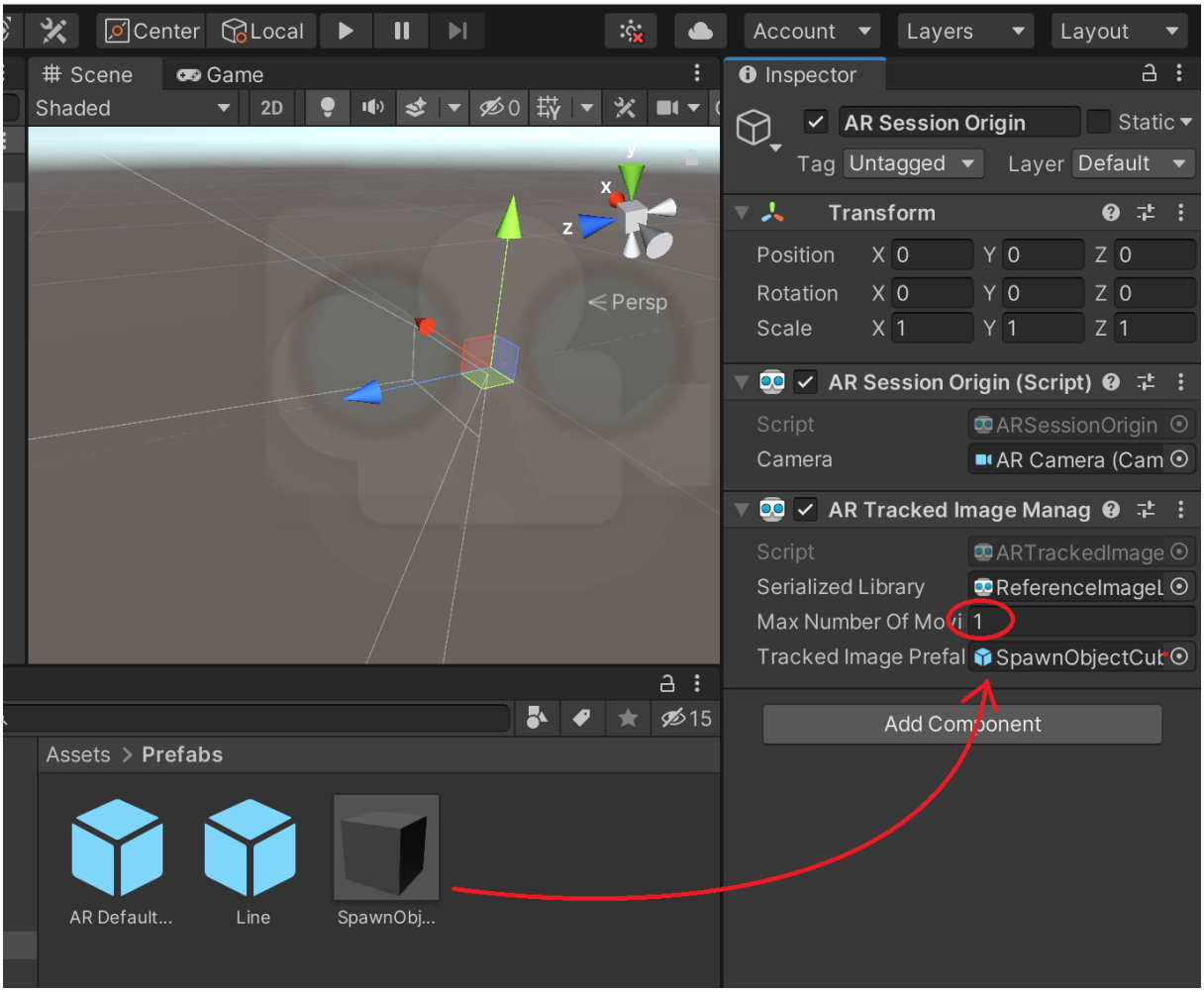
Here you can drag and drop your image and then specify the physical size in meters.



Go to the AR Session Origin and drag the refrence image library into the Serialized Library field



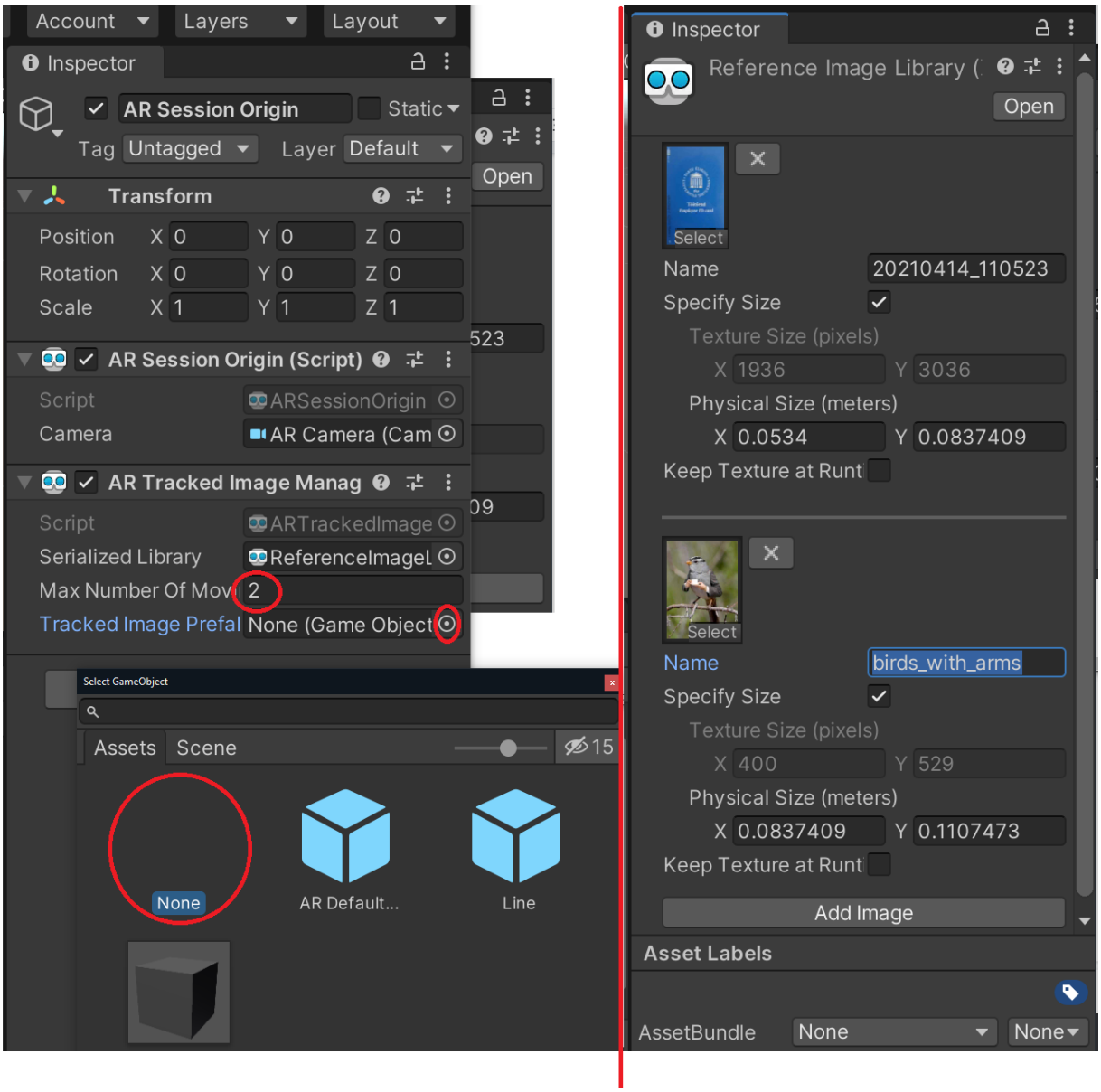
Add the reference image library to the ar tracked image manager Serialize field.
Specify the max number of tracked objects.
Add a prefab model to be placed in tracking (you can reuse the cube prefab from the last session or you can create a new prefab).



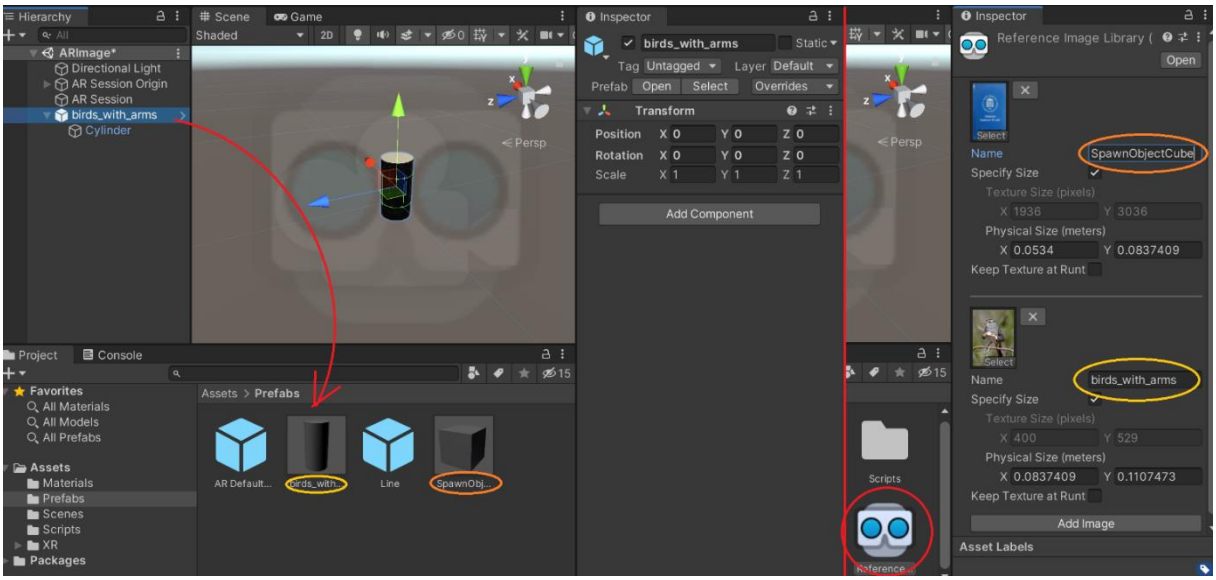
BUILD AND RUN THE CODE

PART 2 - UPGRADING TO MULTIPLE IMAGE TRACKING

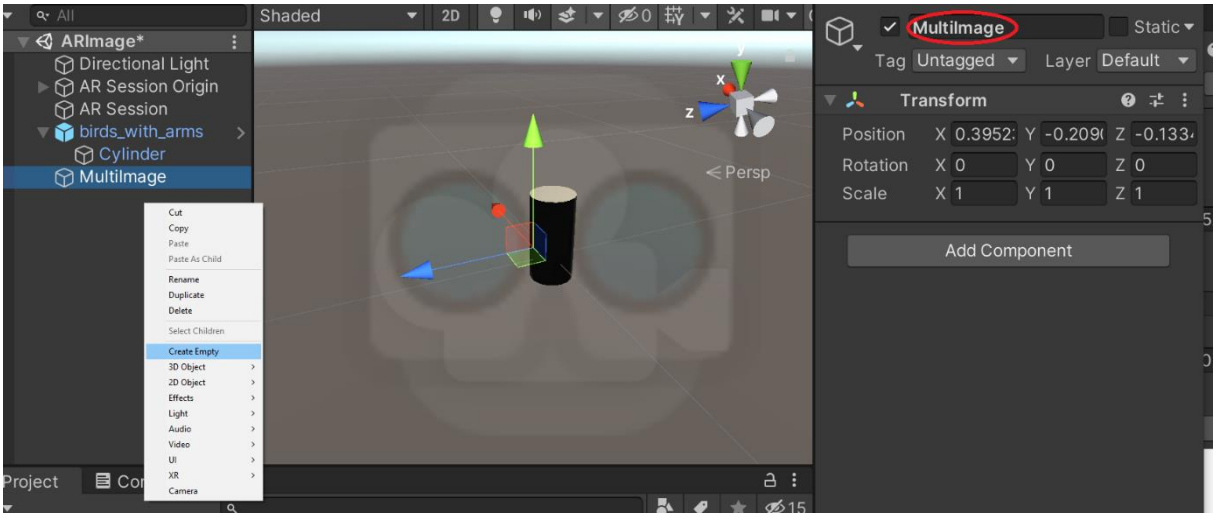
Remove the prefab model from ar tracked image manager.
Specify the new max number of tracked objects.
Add a new image to reference the library.



Create 2 Prefab models that you want to display and name them the same as the images (Or rename the images in the reference image library, this is case sensitive)



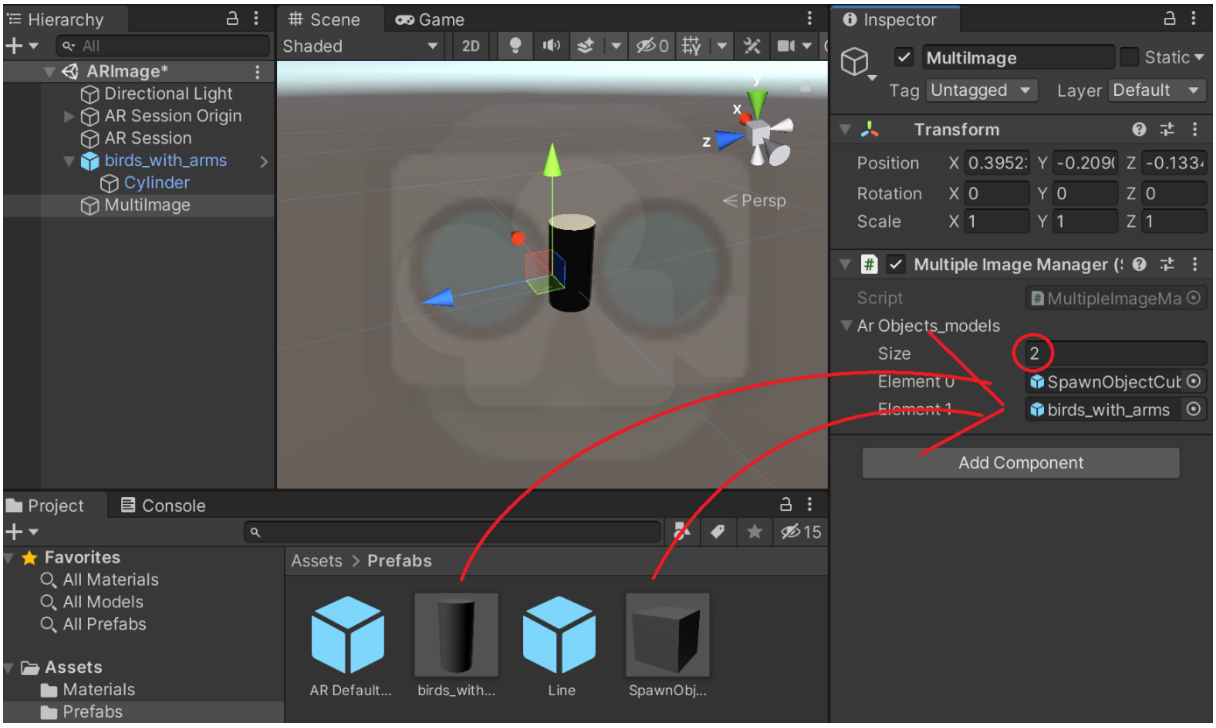
Create an empty object that will hold our new script



Create script

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.XR.ARFoundation;
using UnityEngine.XR.ARSubsystems;

public class MultipleImageManager : MonoBehaviour
{
    public GameObject[] arObjects_models;
    private ARTrackedImageManager imageManager;
    private Dictionary<string, GameObject> arObjects = new Dictionary<string, GameObject>();

    void Awake()
    {
        imageManager = FindObjectOfType<ARTrackedImageManager>();
        foreach (GameObject arObject in arObjects_models)
        {
            GameObject newArObject = Instantiate(arObject, Vector3.zero, Quaternion.identity);
            newArObject.name = arObject.name;
            arObjects.Add(arObject.name, newArObject);
        }
    }

    void OnEnable()
    {
        imageManager.trackedImagesChanged += OnTrackedImagesChanged;
    }

    void OnDisable()
    {
        imageManager.trackedImagesChanged -= OnTrackedImagesChanged;
    }

    void OnTrackedImagesChanged(ARTrackedImagesChangedEventArgs eventArgs)
    {
        foreach (ARTrackedImage trackedImage in eventArgs.added)
        {
            UpdateARImage(trackedImage);
        }

        foreach (ARTrackedImage trackedImage in eventArgs.updated)
        {
            UpdateARImage(trackedImage);
        }

        foreach (ARTrackedImage trackedImage in eventArgs.removed)
        {
            arObjects[trackedImage.name].SetActive(false);
        }
    }

    private void UpdateARImage(ARTrackedImage trackedImage)
    {
        string name = trackedImage.referenceImage.name;
        Vector3 position = trackedImage.transform.position;
        Quaternion rotation = trackedImage.transform.rotation;

        GameObject goARObject = arObjects[name];
        goARObject.SetActive(true);
        goARObject.transform.position = position;
        goARObject.transform.rotation = rotation;
    }
}

```

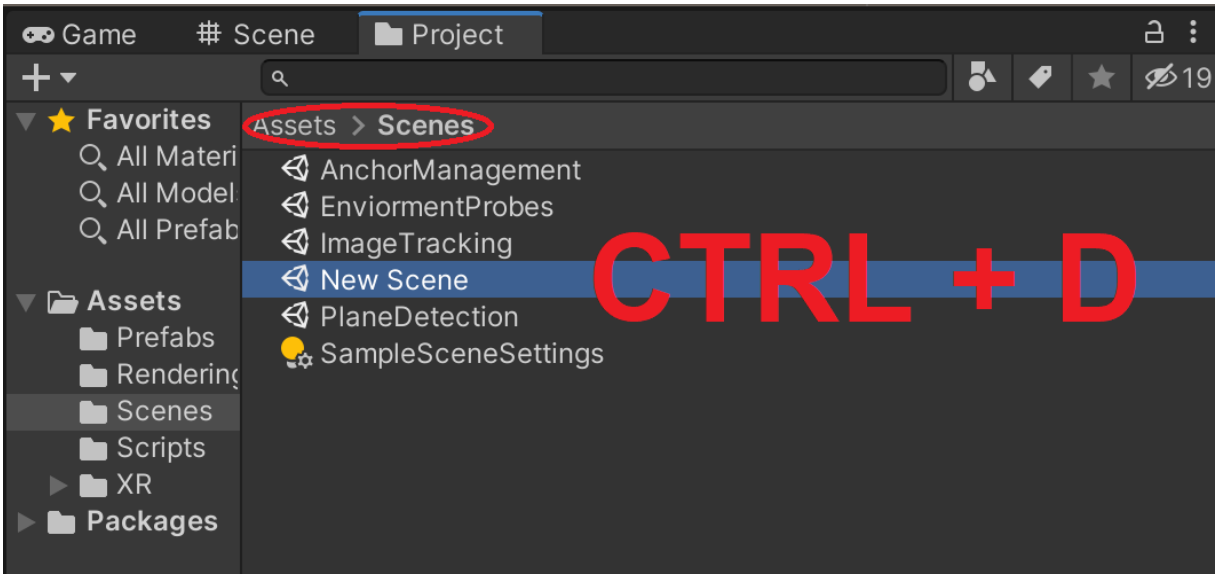
Attach the script to your empty gameobject and add the prefabs to the script array



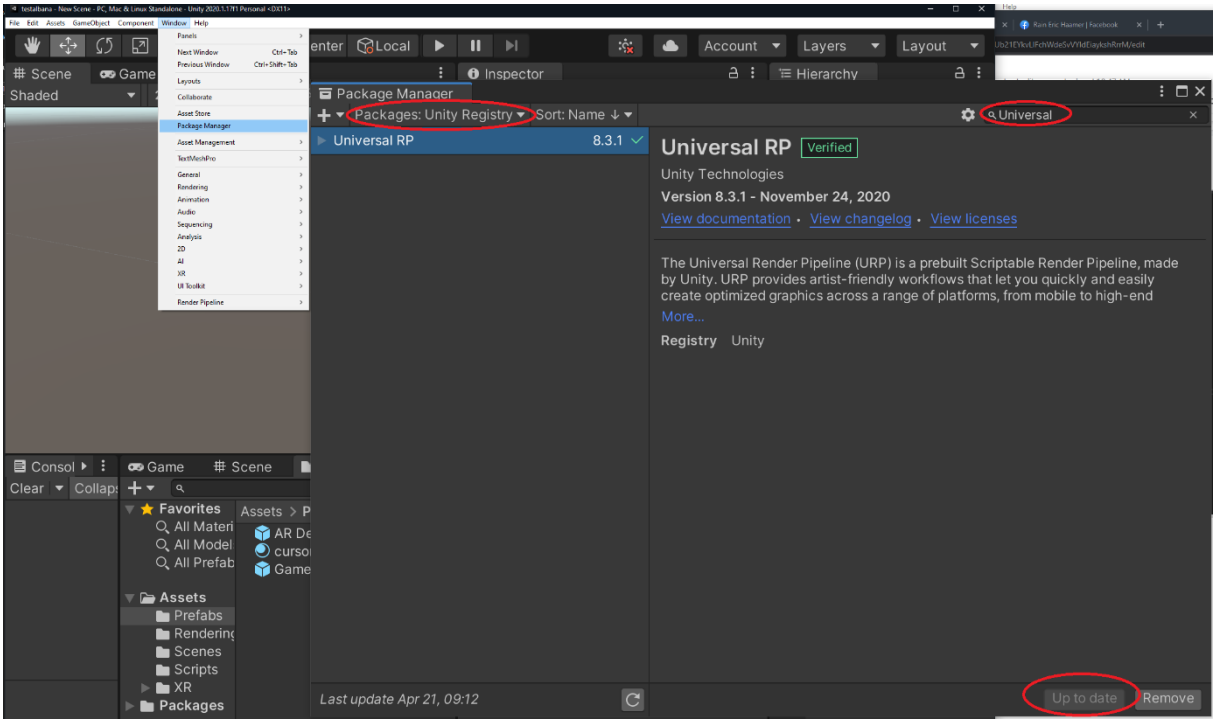
BUILD AND RUN THE CODE

PART 3 – USING THE UNIVERSAL RENDER PIPELINE

Duplicate the first scene that has plane detection and the ability to spawn models on click by selecting the Scene in the project window CTRL+D and double-click on it.

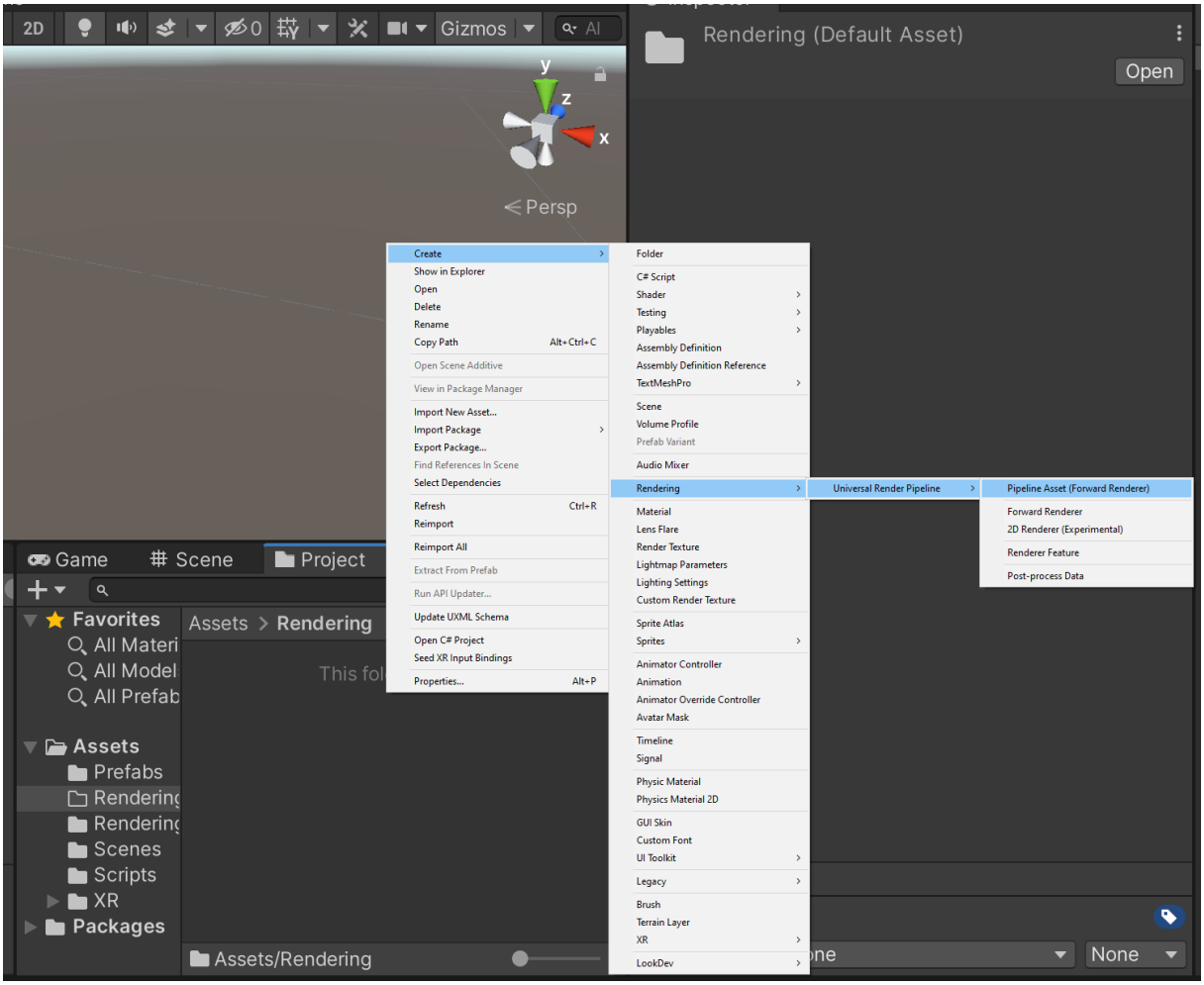


Go to window->package manager
Select Packages in the Unity registry.
Add Universal RP (verified version).

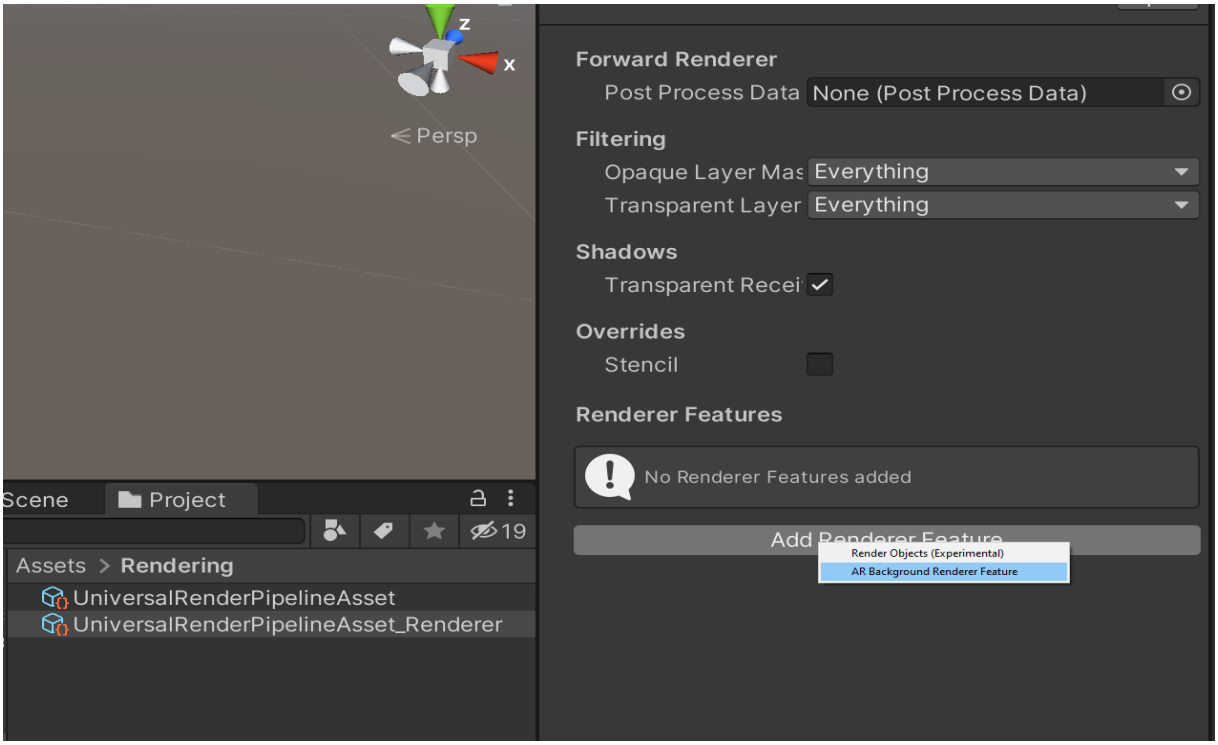


Create folder Rendering

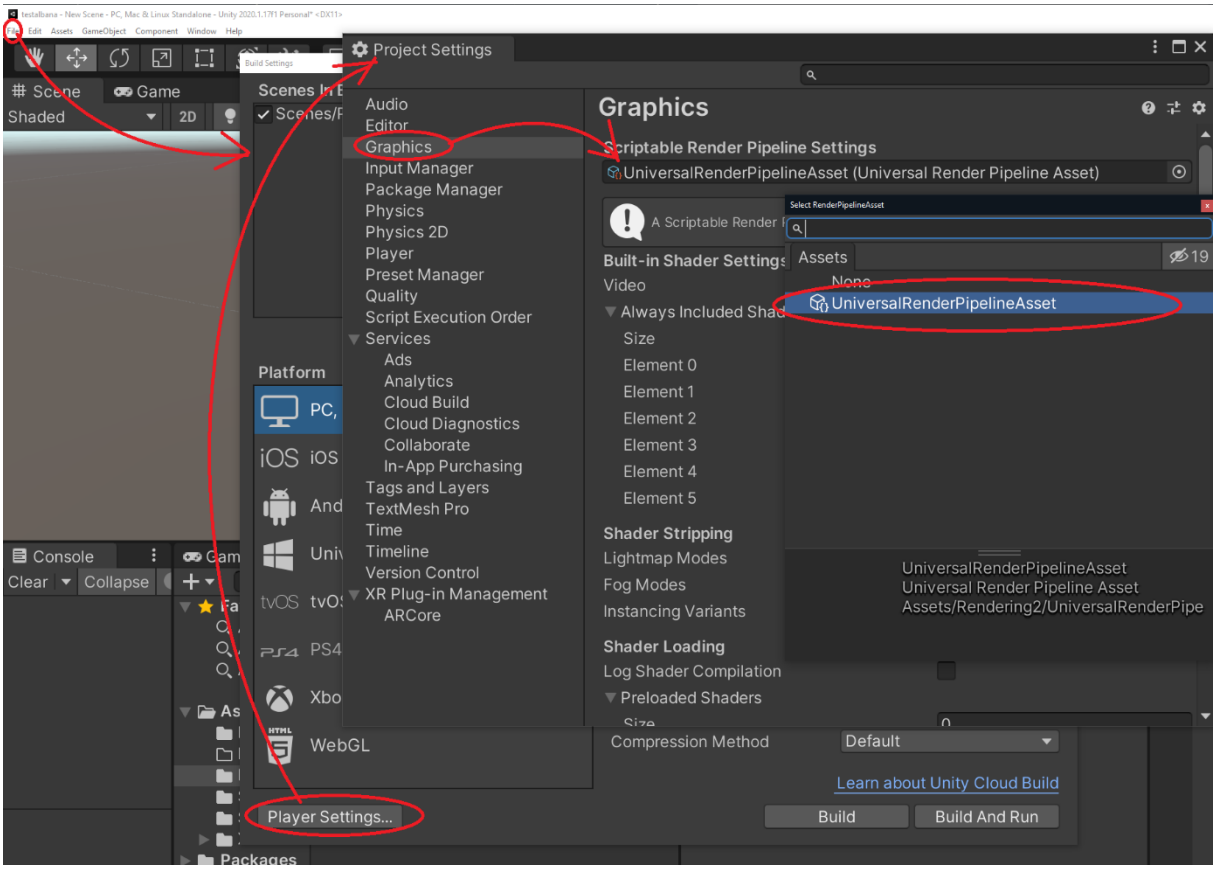
Create>Rendering>Universal Rendering Pipeline>Pipeline Renderer(Forward Renderer)



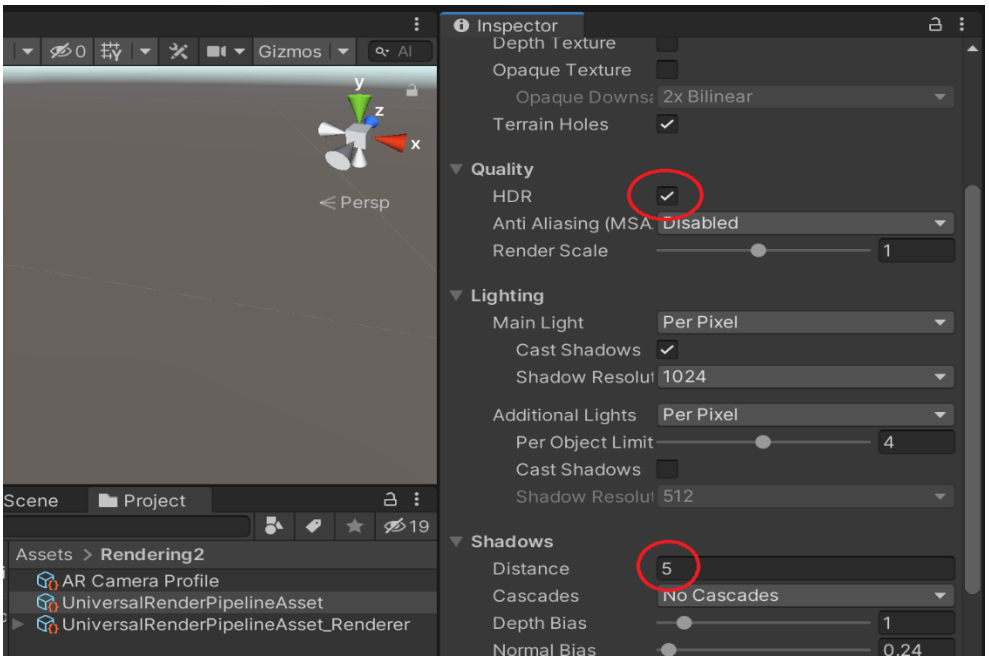
Select the _Renderer file, inspector Add render feature>AR background render feature



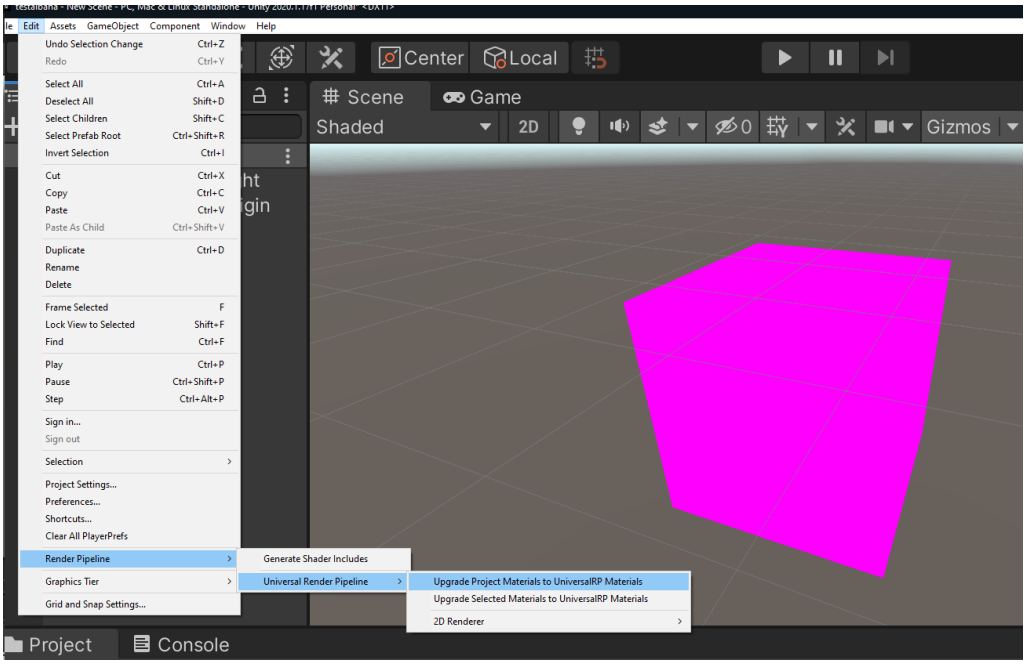
player settings>graphics>first thing select the universal render pipeline



Open up the UniversalRendererPipelineAsset and go to the inspector window
Set the Quality>HDR true if your device supports it
Set Shadows>Shadow distance ~5m



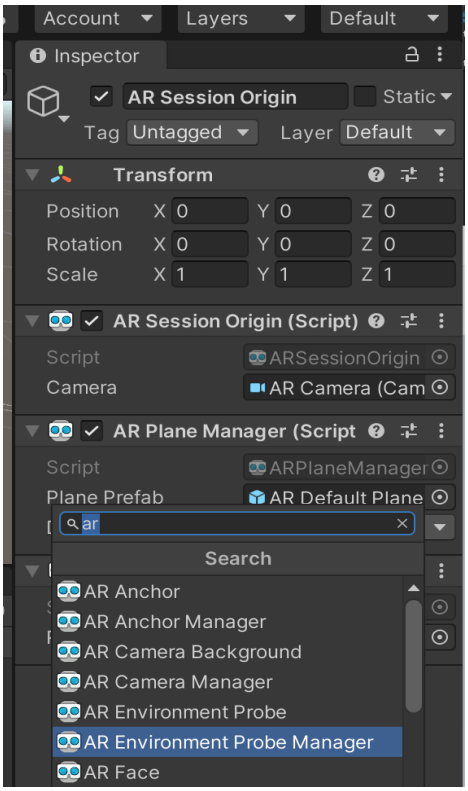
If you have old materials in your scene that are now pink.
Edit>Render pipeline> universal render pipeline>upgrade project materials to URP Materials.



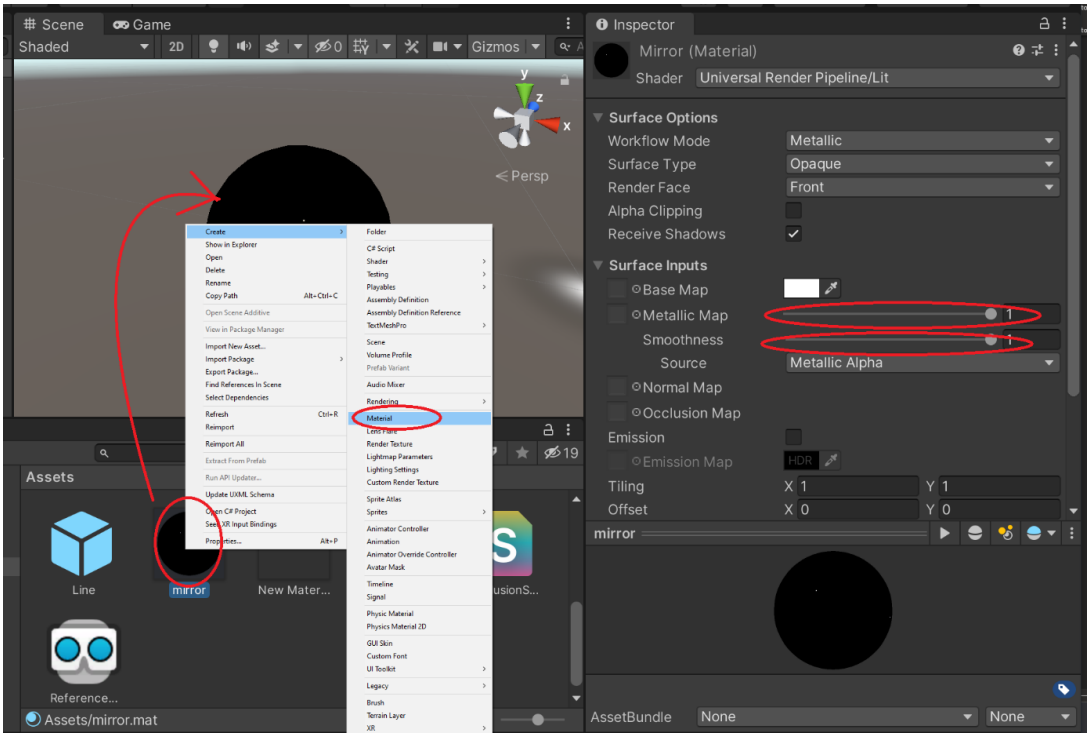
Now the project is running the Universtal Rendering Pipeline

PART 4 – ENVIRONMENT PROBES AND POST-PROCESSING

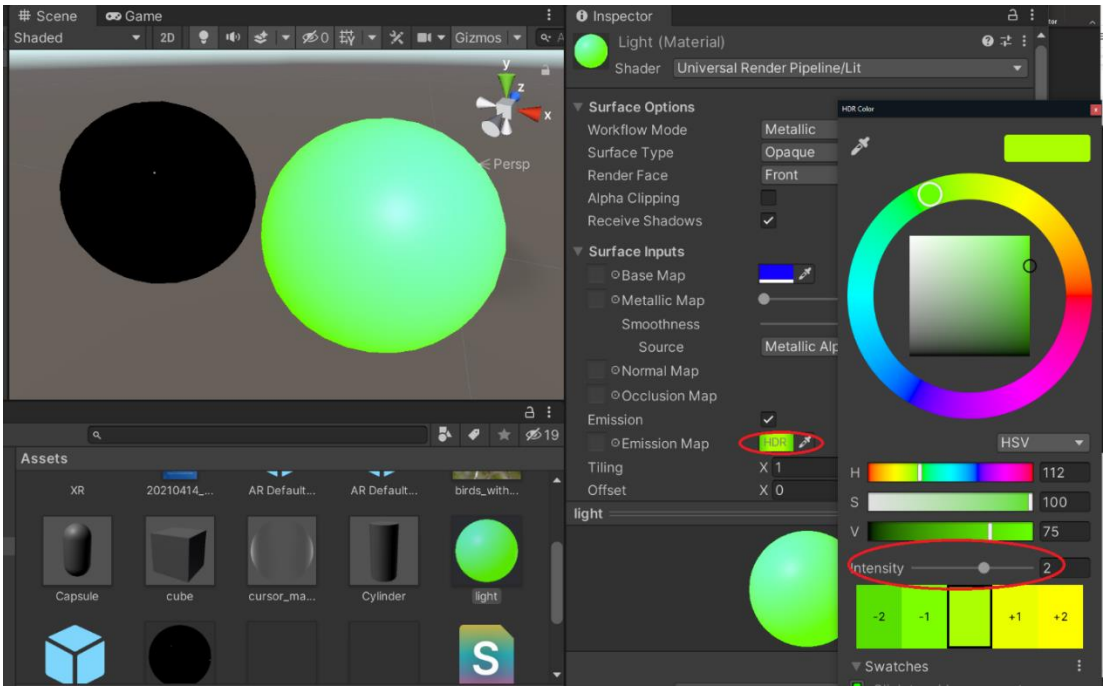
Select AR session origin.
Add AR environment probe manager.



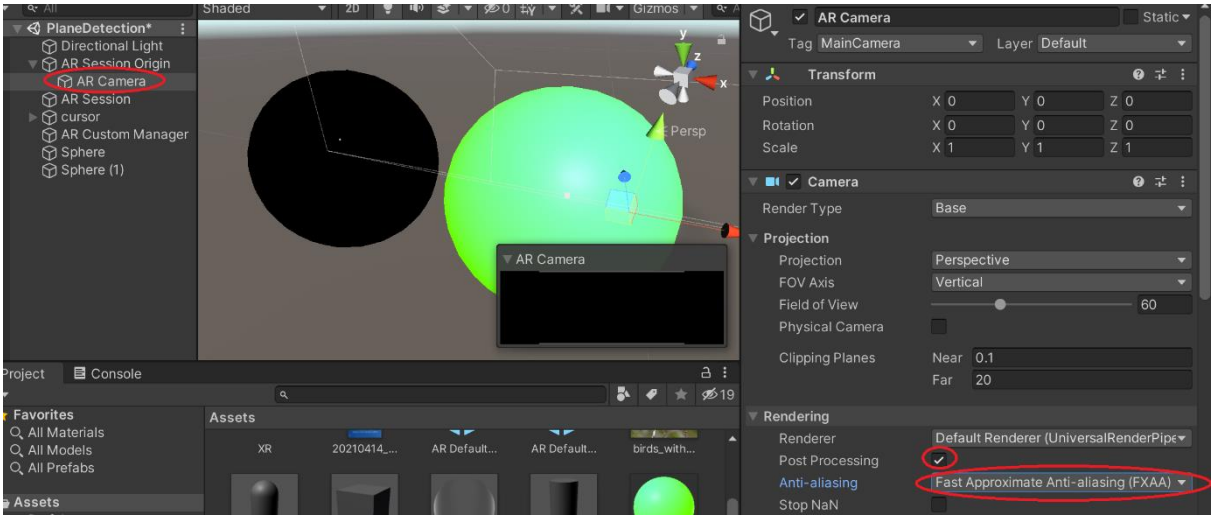
Create a sphere (size 0.1, 0.1, 0.1, and near the camera) and new material.
For the material set the metallic and smoothness both to 1 and attach the material to the sphere.



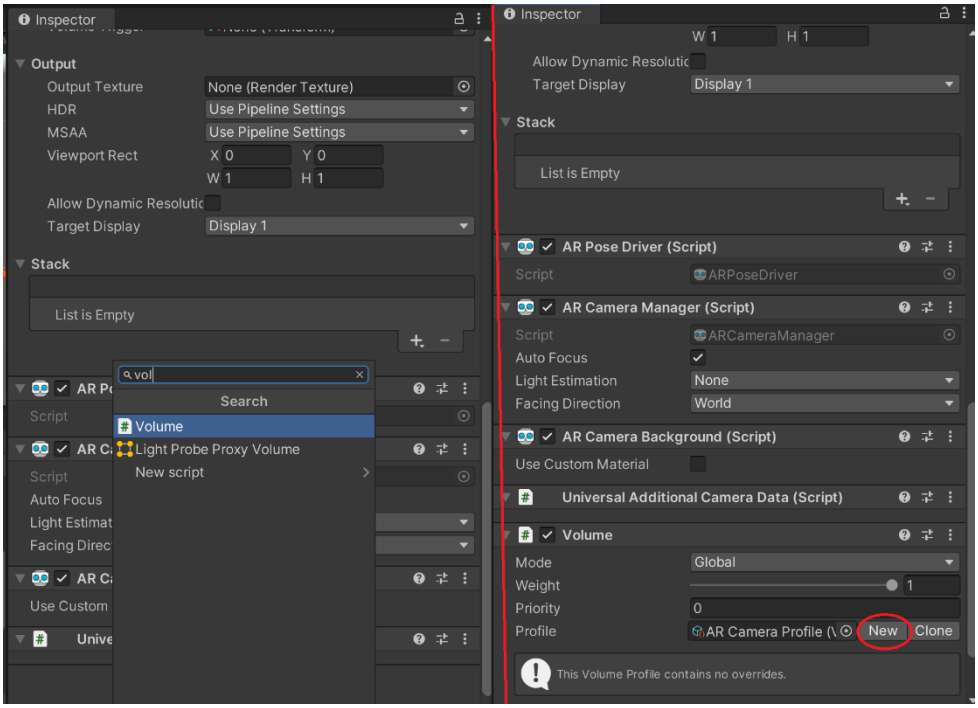
Repeat the previous step with a new sphere and material but give the new material an emission intensity instead of reflectivity.



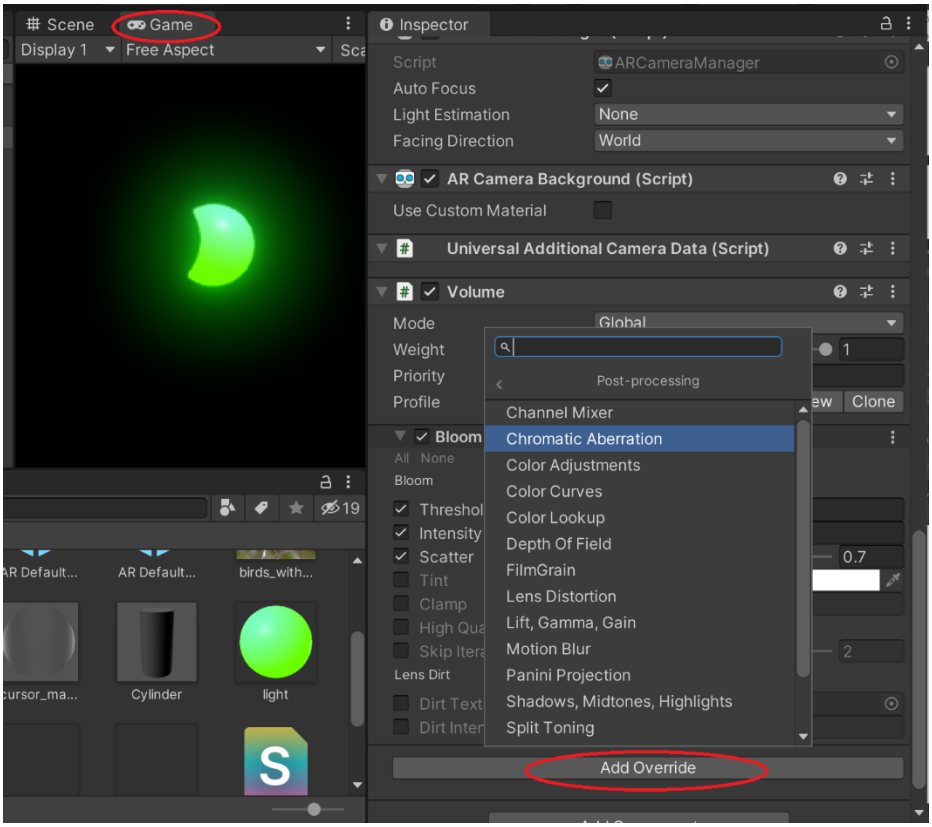
In the scene select Camera>Rendering>Post processing enable.
For anti-aliasing add the FXAA.



Add component Volume to the AR Camera and press new.



Now click add override in volume and you can add post-processing effects.
Switch to the camera and make sure the spheres are in view.



BUILD AND RUN THE CODE

EXTRA TOPICS
PART 5 – LIGHT ESTIMATION

In the Scene Under AR Session Origin select AR Camera.
In the Inspector find AR Camera Manager>Light Estimation>Everything

Write a light estimation script and add it to the Directional Light in the scene.
Attach AR Camera to the ARCameraManager property.

```
public class LightEstimation : MonoBehaviour
{
    public ARCameraManager arCameraManager;
    Light mainLight;

    void Awake()
    {
        mainLight = GetComponent<Light>();
    }

    void OnEnable()
    {
        if (arCameraManager != null)
            arCameraManager.frameReceived += FrameChanged;
    }
    void OnDisable()
    {
        if (arCameraManager != null)
            arCameraManager.frameReceived -= FrameChanged;
    }

    void FrameChanged(ARCameraFrameEventArgs args)
    {
        if (args.lightEstimation.averageBrightness.HasValue)
        {
            mainLight.intensity = args.lightEstimation.averageBrightness.Value;
        }
        if (args.lightEstimation.averageColorTemperature.HasValue)
        {
            mainLight.colorTemperature = args.lightEstimation.averageColorTemperature.Value;
        }

        if (args.lightEstimation.colorCorrection.HasValue)
        {
            mainLight.color = args.lightEstimation.colorCorrection.Value;
        }
        if (args.lightEstimation.mainLightDirection.HasValue)
        {
            mainLight.transform.rotation = Quaternion.LookRotation(args.lightEstimation.mainLightDirection.Value);
            //Quaternion targetRotation = Quaternion.LookRotation(args.lightEstimation.mainLightDirection.Value);
            //mainLight.transform.rotation = Quaternion.RotateTowards(mainLight.transform.rotation, targetRotation, Time.deltaTime * 5f);
        }
        if (args.lightEstimation.mainLightColor.HasValue)
        {
            mainLight.color = args.lightEstimation.mainLightColor.Value / Mathf.PI;
            mainLight.color = mainLight.color.gamma;
        }
        if (args.lightEstimation.mainLightIntensityLumens.HasValue)
        {
            mainLight.intensity = args.lightEstimation.averageMainLightBrightness.Value;
        }
        if (args.lightEstimation.ambientSphericalHarmonics.HasValue)
        {
            RenderSettings.ambientMode = AmbientMode.Skybox;
            RenderSettings.ambientProbe = args.lightEstimation.ambientSphericalHarmonics.Value;
        }
    }
}
```

PART 6 – SHADOWS AND DEPTH

For shadows Create a new material.
In the project, window click Create>Shader>Standard Surface Shader.
Write the script in the shader and this will be programmed in HLSL(**High-Level Shader Language**).
Do not overwrite the very first line

```
Shader "Custom/shadow"
{
    // The properties block of the Unity shader. In this example this block is empty
    Properties
    {
    }

    // The SubShader block containing the Shader code.
    SubShader
    {
        // SubShader Tags define when and under which conditions a SubShader block or
        // a pass is executed.
        Tags
        {
            "RenderType" = "Transparent"
            "RenderPipeline" = "UniversalPipeline"
            "IgnoreProjector" = "True"
        }
        //  Bumped Specular level of detail
        LOD 300

        Pass
        {
            Name "AR Proxy"
            Tags
            {
                "LightMode" = "LightweightForward"
            }

            // the end shader value is {multiplied by source alpha value} {then multiplied by 1 - source alpha}
            //// its the code for Traditional transparency
            Blend SrcAlpha OneMinusSrcAlpha

            //This forces all objects behind the shader to not be rendered
            ZWrite On

            //render shader from both sides
            Cull Off

            // The HLSL code block. Unity SRP uses the HLSL language.
            HLSLPROGRAM

        // -------------------------------------
        // Lightweight Pipeline keywords
#pragma multi_compile _ _MAIN_LIGHT_SHADOWS
#pragma multi_compile _ _MAIN_LIGHT_SHADOWS_CASCADE
#pragma multi_compile _ _ADDITIONAL_LIGHTS_VERTEX _ADDITIONAL_LIGHTS
#pragma multi_compile _ _ADDITIONAL_LIGHT_SHADOWS
#pragma multi_compile _ _SHADOWS_SOFT

            // This line defines the name of the vertex shader.
#pragma vertex HiddenVertex
            // This line defines the name of the fragment shader.
#pragma fragment HiddenFragment


            // The Core.hlsl file contains definitions of frequently used HLSL
            // macros and functions, and also contains #include references to other
            // HLSL files (for example, Common.hlsl, SpaceTransforms.hlsl, etc.).
#include "Packages/com.unity.render-pipelines.universal/ShaderLibrary/Core.hlsl"
```

```
61
62          // Need this for things like shadow coordinates
63  #include "Packages/com.unity.render-pipelines.universal/ShaderLibrary/Lighting.hlsl"
64
65          struct Attributes
66          {
67              //this improves performance if this shader is drawn on multiple meshes
68              UNITY_VERTEX_INPUT_INSTANCE_ID
69
70              // The positionOS variable contains the vertex positions in object
71              // space.
72              float4 positionOS   : POSITION;
73          };
74
75          struct Varyings
76          {
77              // The positions in this struct must have the SV_POSITION semantic.
78              float4 positionCS   : SV_POSITION;
79              // TEXCOORD0 is a texture coordinate
80              float4 shadowCoord  : TEXCOORD0;
81          };
82
83          // The vertex shader definition with properties defined in the Varyings
84          // structure. The type of the vert function must match the type (struct)
85          // that it returns.
86          Varyings HiddenVertex(Attributes input)
87          {
88              // Declaring the output object (OUT) with the Varyings struct.
89              Varyings output = (Varyings)0;
90
91              //this improves performance if this shader is drawn on multiple meshes
92              UNITY_SETUP_INSTANCE_ID(input);
93
94              //This takes a position and transforms it from object space to a bunch of spaces like worldspace/viewspace/clipspace
95              VertexPositionInputs vertexInput = GetVertexPositionInputs(input.positionOS.xyz);
96
97              //assigns the values to the varyings output. clips space coordinate to positioncs and shadow coordinate to shadowcoord
98              output.shadowCoord = GetShadowCoord(vertexInput);
99              output.positionCS = vertexInput.positionCS;
100
101             return output;
102         }
103
104         // The fragment shader definition.
105         half4 HiddenFragment(Varyings input) : SV_Target
106         {
107             //this gets the shadow value on the material (1 - no shadow, 0 - shadow)
108             //the half is like a float just 16bit instead of 32bit
109             half s = MainLightRealtimeShadow(input.shadowCoord);
110
111             //we assign the shadow to the alpha value in reverse and set the albedo to black
112             //this makes it so everything is transparent part from the shadow which is black
113             return half4(0, 0, 0, 1 - s);
114         }
115         ENDHLSL
116     }
117 }
118 }
119
```

Drag and drop the Shader onto the Material.
Drag and drop the Material to the AR Plane Prefab.

Now the AR Planes are invisible but disable rendering behind them and allow shadows to be cast on their surface.