CMP3754M Virtual and Augmented Reality

Workshop 9 : AR Marker Detection

Objectives:

1. Create an app with marker-based detection
2. Respond to marker detection by instantiating a 3D model in the scene.

Duration: 1 hour
Platform: *Unity 6 (version = 6000.0.55f1 LTS)*

By completing this workshop you will build functionality that you can use to complete your second assessment for this module

---

1. Introduction

In this workshop we are going to create an image marker, detect it in the app, and use that event to spawn an object above it. We will test this using the editor simulated environment, and also by deploying it to the phone. We will then extend this to detecting multiple markers.

The marker images are already in the project assets, in the folder **Assets>CMP3754>Marker**. To test on the phone, you need to print these out. I have provided a Word document with the images in, for you to print.

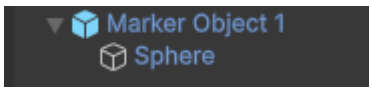
You will also notice that I have set up the simulation environment with both images, so that you can test your work in the Editor before deploying to the phone.

2. Creating the Marker Object

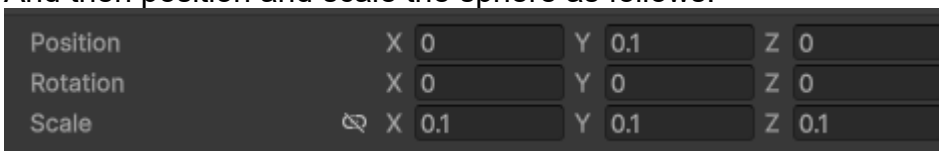
The marker object will appear above the marker image, when it is detected in the app. We will create this object in the hierarchy, and then turn it into a prefab.

Start by creating an empty object in the hierarchy. This is the base of the marker object, which will be positioned on top of the marker image. Rename it to “Marker Object 1” We want the visible object to appear above the object, so we will use the empty object as a reference position against which we will position the visible part of the object (which will be a sphere).

Create a sphere as a child object of the Marker Object 1 base object:



And then position and scale the sphere as follows:



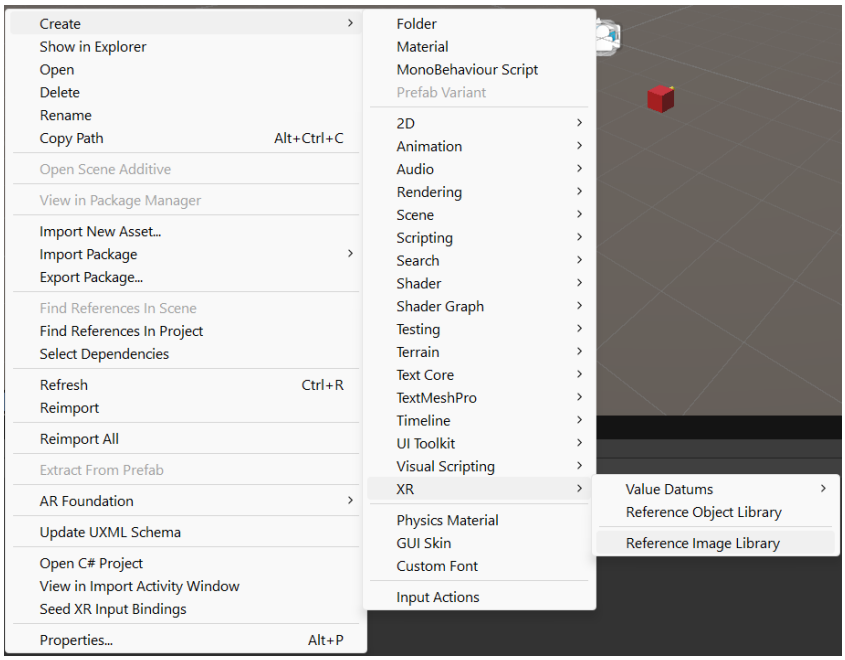
The local position of the sphere sets its relative offset with respect to the detected marker image in the scene: you can adjust this how you like.

In the directory **Assets>CMP3754>Marker**, create a new material for the sphere. I made mine Yellow and called it “MarkerMat1”.

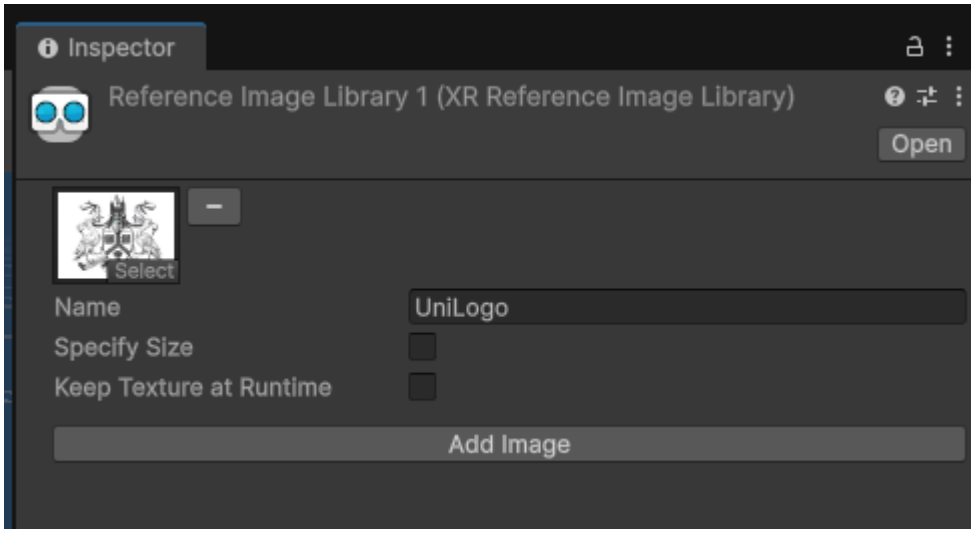
Drag the new object into the **Assets>CMP3754>Marker** directory in the Project window to create a prefab, and then delete the original object from the Hierarchy.

3. Enabling marker Detection

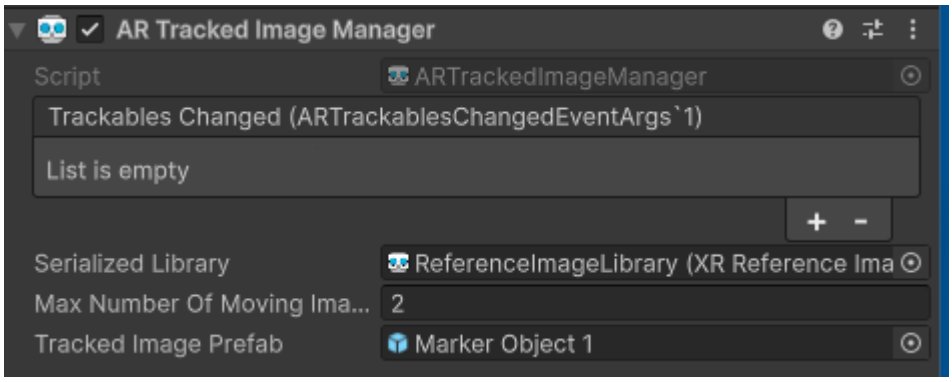
Right-click in the **Assets>CMP3754>Marker** directory, and select Create>XR>ReferenceImageLibrary:



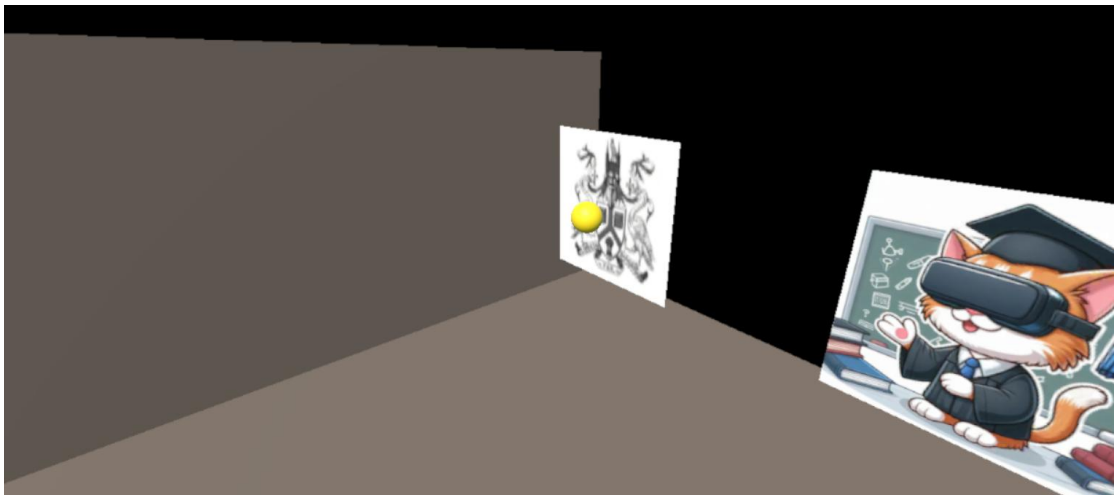
You can rename the new object as you like. With the new object selected, in the Inspector, click “Add Image”. Drag the UniLog image onto the Texture window. Your Reference Image Library object should look like this in the Inspector:



Next, select **XR Origin** in the Hierarchy window. Add an **AR Tracked Image Manager** component. Drag the marker object prefab into the **Tracked Image Prefab** field, and the Reference Image Library object into the **Serialized Library** field. Set the maximum number of tracked images to 2.



You can now test this in the editor simulator. You should see something like this:



Note that the simulator may also show the marker above the Cat image: I think this is just a limitation of the simulator. However, we will soon add a different marker for the cat image.

Now test to your phone: Select **Build and Run** to deploy the app to your phone. Place the printed UniLogo image on your desk, and point your phone at it. You should see something like this:

4. Tracking Multiple Images with Different Objects

What we have so far is a bit limited. The AR Tracked Image Manager component only allows us to instantiate one prefab, even if we have different tracked images. Often, we will want different prefabs or functions for different images.

To achieve this, we will need a script that processes events generated by the AR Tracked Image Manager, and then write our own functions to respond to different images.

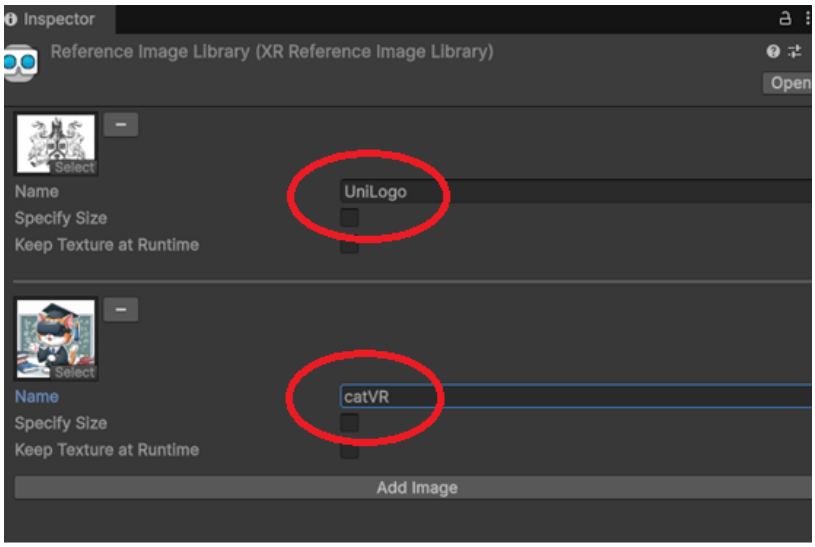
This script will create one prefab instance for each tracked image, and when the image is detected/tracked, it will place the correspond prefab over it. If the image is not visible, the prefab will be hidden. I have written and provided a script to do this, which you can use.

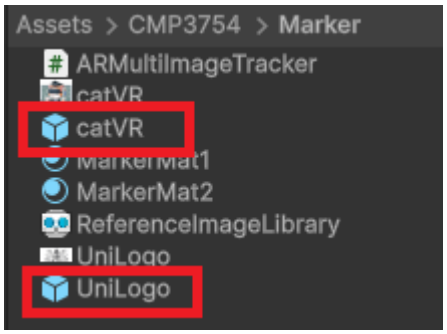
Firstly, delete the contents of the Tracked Image Prefab field of the AR Tracked Image Manager.

Add the catVR image to the Reference Image Library, so that you now have the 2 images in the library.

Create a new Marker prefab to display. I copied the existing yellow sphere prefab, and gave it a new name. I then created a new material (blue), and added that to the sphere on the new marker. But you could create a different shape, or import a model from the Asset Store.

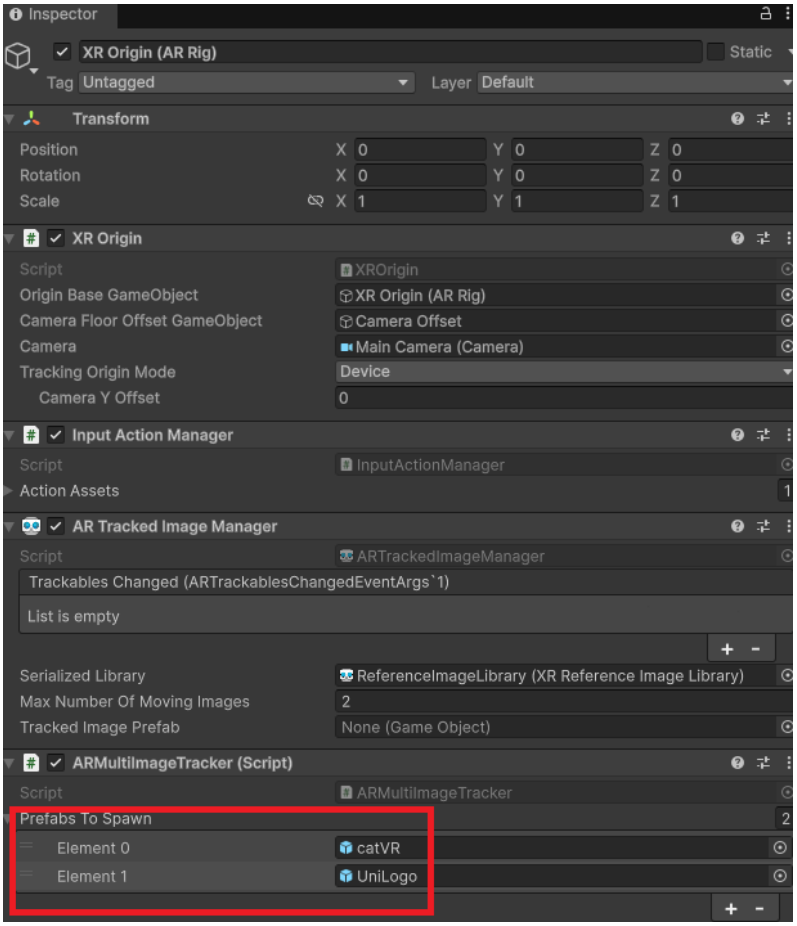
Now, change the names of the marker prefabs so that they match the names of the images in the reference library. This is how the script will know which marker to display for which image. For example:





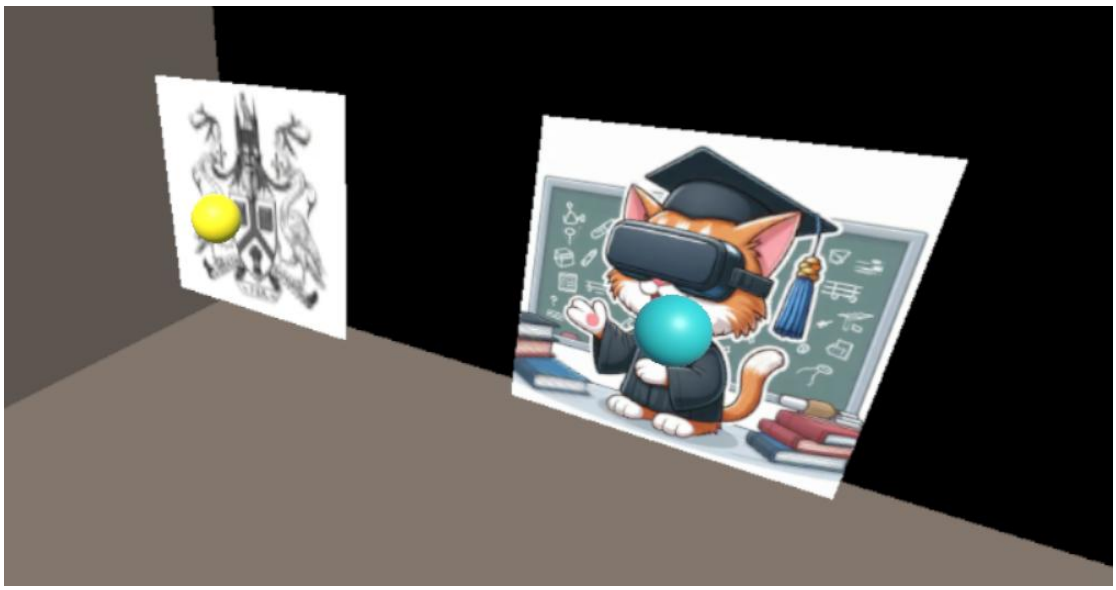
Download the provided script “ARMultiImageTracker.cs”, and place it in the **Assets>CMP3754>Marker** directory. **Read through the script carefully, and make sure you understand how it works**.

Add the script to the XR Origin object in the hierarchy. Create two entries in the **Prefabs To Spawn field**, and drag the catVR and UniLogo prefabs into those entries:

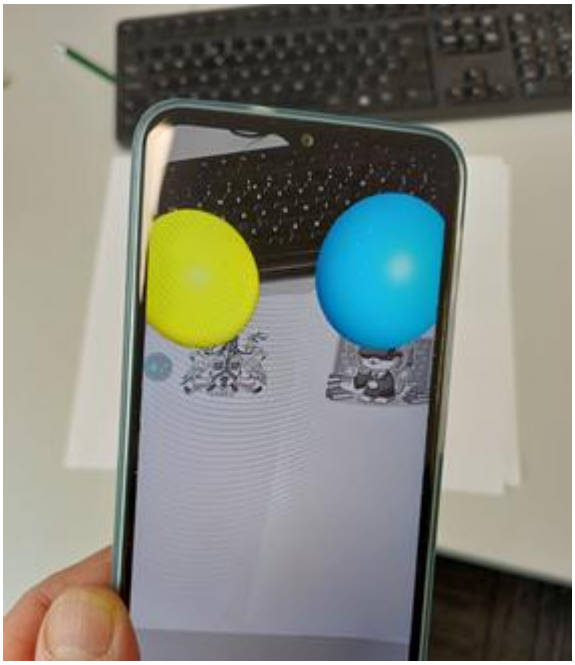


5. Testing

You can test this code in the simulator. You should see something like this:



You can then deploy to the phone:



6. Notes

- Currently you can only spawn one primitive for each tracked image: if there are multiple instances of the same image, the primitive will jump between them.

- You can add more images and primitives, as you wish.

- You can add other code to ARMultiImageTracker.cs, other than controlling the prefabs. If you want to respond to one of the images specifically, you could add code similar to: if (image.referenceImage.name == "catVR") {….}

- Debugging on device can be tricky. However, you can log your own debugging messages from the device using the Unity Debug.Log() command – when running on device, these messages appear in the Logcat window, which can be a useful way of trapping errors. There are a lot of system messages, but you can filter these out using the message search function.