COMSM0129: Augmenting the Real World

Lab 5: Using Quest for AR and Vuforia

Overview

This lab is designed to deepen students' understanding of Augmented Reality (AR) technology, focusing on developing AR applications using Quest headsets with Unity. The primary objective is to explore basic AR interactions, allowing students to learn how to use the head tracking and spatial mapping features of Quest 3 for interactive AR experiences. The lab consists of two main tasks: Object Selection and Object Movement, which demonstrate the fundamentals of interacting with virtual objects in an AR environment.

Note: make sure you have finished the previous lab tasks, as we will continue to work on the same project.

Tasks

1. Enable users to select a virtual object in the scene using their gaze direction.
2. Allow users to move a selected object based on head movements and gaze direction.

Task 1: Object Selection

The primary objective of this task is to enable users to interact with virtual objects within an AR environment using their gaze (head direction). Since Quest 3 does not have eye-tracking by default, we will implement gaze as a ray from the camera center. This task builds upon the Quest setup from Lab 1 Task 3.

Prerequisites: Ensure you have completed Lab 1 Task 3 (Quest Hello World) and your project has:

- Meta XR SDK installed (`com.meta.xr.sdk.all`)
- OVRCameraRig in the scene (via Building Blocks)
- OpenXR configured with Meta Quest Support

Subtask 1: Project Setup

1. **Verify Meta XR SDK Installation:**
   - Open `Window` > `Package Manager`.
   - Click "+" and select `Add package by name`.
   - Enter: `com.meta.xr.sdk.all`
   - If already installed, verify it's up to date.
2. **Verify XR Plug-in Management:**
   - Go to `Edit` > `Project Settings` > `XR Plug-in Management`.
   - Under the **Android tab**, ensure **OpenXR** is checked.
   - Click the settings icon next to OpenXR.
   - Under **Interaction Profiles**, ensure **Oculus Touch Controller Profile** is added.
   - Under **Feature Groups**, ensure **Meta Quest Support** is checked.
3. **Verify Player Settings:**
   - Go to `Edit` > `Project Settings` > `Player`.
   - Under the **Android tab** > `Other Settings`:
     - **Color Space:** Linear
     - **Graphics APIs:** OpenGLES3 only (remove Vulkan)
     - **Scripting Backend:** IL2CPP
     - **Target Architectures:** ARM64

Subtask 2: Scene Setup

1. **Create or Open Your Quest Scene:**
   - Open the scene from Lab 1 Task 3, or create a new scene.
   - Ensure you have **OVRCameraRig** in the scene (from Meta Building Blocks).
   - If not, go to `Meta` > `Building Blocks` and add **Camera Rig**.
2. **Add Interactable Objects:**
   - Create a `3D Object` > `Cube` in the scene.
   - Position it at approximately `(0, 1.5, 2)` so it's visible in front of the user.
   - Add a **Box Collider** component (should be added automatically).
   - Tag the cube as `Interactable`:
     - Select the cube.
     - In the Inspector, click the **Tag** dropdown (top area).
     - Select `Add Tag`, create a new tag named `Interactable`.
     - Return to the cube and set its tag to `Interactable`.

Subtask 3: Implement Gaze Selection

1. **Create GazeSelection Script:**
   - In `Assets/Scripts`, create a new C# script named `GazeSelection`.
   - Find the **CenterEyeAnchor** under `OVRCameraRig` > `TrackingSpace`.
   - Attach the `GazeSelection` script to **CenterEyeAnchor**.
2. **Implement Raycasting Logic:**

```
using UnityEngine;

public class GazeSelection : MonoBehaviour
{
    [SerializeField] private float rayDistance = 100f;
    private GameObject currentGazedObject = null;

    void Update()
    {
        RaycastHit hit;
        Ray ray = new Ray(transform.position, transform.forward);

        if (Physics.Raycast(ray, out hit, rayDistance))
        {
            if (hit.collider.CompareTag("Interactable"))
            {
                if (currentGazedObject != hit.collider.gameObject)
                {
                    if (currentGazedObject != null)
                    {
                        currentGazedObject.GetComponent<InteractableObject>()?.OnGazeExit();
                    }

                    currentGazedObject = hit.collider.gameObject;
                    currentGazedObject.GetComponent<InteractableObject>()?.OnGazeEnter();
                }
            }
        }
        else
        {
            if (currentGazedObject != null)
            {
                currentGazedObject.GetComponent<InteractableObject>()?.OnGazeExit();
                currentGazedObject = null;
            }
        }
    }
}
```

Subtask 4: Visual Feedback for Selection

1. **Create InteractableObject Script:**

```
using UnityEngine;

public class InteractableObject : MonoBehaviour
{
    [SerializeField] private Material highlightMaterial;
    private Material originalMaterial;
    private Renderer objectRenderer;

    void Start()
    {
        objectRenderer = GetComponent<Renderer>();
        originalMaterial = objectRenderer.material;
    }

    public void OnGazeEnter()
    {
        if (highlightMaterial != null)
        {
            objectRenderer.material = highlightMaterial;
        }
        Debug.Log("Gaze entered: " + gameObject.name);
    }

    public void OnGazeExit()
    {
        objectRenderer.material = originalMaterial;
        Debug.Log("Gaze exited: " + gameObject.name);
    }
}
```

2. **Setup Materials:**
   - Create a new **Material** in `Assets/Materials` named `HighlightMaterial`.
   - Set its color to bright yellow or cyan.
   - Attach the `InteractableObject` script to your cube.
   - Drag the `HighlightMaterial` into the **Highlight Material** field.
3. **Build and Test:**
   - Connect your Quest 3 and ensure Developer Mode is enabled.
   - Go to `File` > `Build Profiles` > `Build And Run`.
   - Put on the headset and look at the cube - it should highlight when you gaze at it.

Task 3: Vuforia Engine — Image Target Demo (In-class Practical)

Goal

In this task, you will run a **Vuforia Image Target** demo on an Android phone and verify that:

1. the **real camera feed** is visible on the phone, and
2. when the camera points at a **correct printed target image**, a **3D augmentation appears anchored** to it and remains stable as you move the phone.

Task 3.1: Import Vuforia Engine + Core Samples into Unity

Package Manager — My Assets: (If you have already added Vuforia to your Unity Asset list)

1. Open Unity.
2. Go to **Window** → **Package Manager**.
3. In the top-left dropdown, select **My Assets**.
4. Find **Vuforia Engine** and **Vuforia Engine Core Samples** (names may vary slightly).
5. Click **Download** (if needed) and then **Import** the Core Samples.

Package Manager — My Assets: (If you have not added Vuforia to your Unity Asset list)

1. Register or log in to your Vuforia account.
2. Download **Vuforia Core Samples** and **Vuforia Engine** from: Download Page
3. After downloading, the packages can be added to your personal asset page and will be visible in **Package Manager** → **My Assets** (see the corresponding figure in the lab handout).

Required license acquisition:

1. Follow the official guidance to generate a license: vuforia-license-manager
2. Open the license generation page: licenses After logging in or registering, select **Plan & Licenses** in the top bar. You should see a **Generate Basic License** button in the **Basic Plan** box. Click it and give your license a name to generate it.
3. After generation, click the license you just created. On the detailed page, locate the `License Key` (a grey text block). You will use this key later in the Vuforia configuration.

Task 3.2: Open the correct demo scene (Image Targets)

Step 1 (Locate the demo scene):

1. In the Project window search bar, type `3-ImageTargets`.
2. Confirm it is a Unity scene file (scene icon / `.unity`).
3. Double-click it to open.

Task 3.3: Configure Vuforia (License + Targets)

Task 3.3.1: Paste a Vuforia license key

Step 1 (ARCamera configuration):

1. Select `ARCamera` in the Hierarchy.
2. In the Inspector, find the **Vuforia Configuration** section.
3. Paste your **App License Key** into the **App License Key** field (a Basic plan key is sufficient for this demo).

Task 3.3.2: Confirm each ImageTarget is correctly set

Step 1 (Check one target):

1. Select `ImageTarget_Astronaut`.
2. In the Inspector, locate **Image Target Behaviour**.
3. Confirm:
   - **Type:** `From Database`
   - **Database:** `VuforiaMars_Images` (or the sample database used by your project)
   - **Image Target:** `Astronaut`

Step 2 (Repeat):

- Repeat the same check for `ImageTarget_Drone`, `ImageTarget_OxygenTank`, and `ImageTarget_Fissure`.

Task 3.4: Print / display the correct target images (Critical)

Important concept: Vuforia will only recognize **the exact images** that exist in its target database. Showing a random picture on a monitor will **not** trigger recognition.

Print the official sample target PDF

1. Use the sample PDF that contains the demo target images. Download the Core Samples from: downloads samples
2. Print the pages containing the targets used in the scene (e.g., Astronaut/Drone/OxygenTank/Fissure).
3. Print at **100% scale** (do *not* use "Fit to page").

Task 3.5: Android build setup + run

Task 3.5.1: Switch platform to Android

1. Go to **File** → **Build Settings**.
2. Select **Android**.
3. Click **Switch Platform**.

Task 3.5.2: Player Settings

Go to **Edit** → **Project Settings** → **Player** → **Android**:

- **Target Architectures:** enable **ARM64**.
- **Minimum API Level:** API 29+ is a safe choice.

Task 3.5.3: Camera permission

- When you run the app on the phone, **allow Camera permission**.
- If you previously denied it: Android **Settings** → **Apps** → **(your app)** → **Permissions** → **Camera** → **Allow**.

Task 3.5.4: Build & Run

1. Connect your Pixel 8 to the computer (USB debugging enabled).
2. In **Build Settings**, click **Build and Run**.
3. When the app opens, point the camera at a **correct printed target**.

Task 3.6: Expected result (What you should observe)

1. With no target in view: you see only the camera feed (and any UI overlays).
2. When you bring a correct Image Target into view:
   - the augmentation appears on top of it,
   - tracking remains stable as you move the phone,
   - the augmentation stays aligned with the target plane.
3. If you cover the target or move it out of view: the augmentation disappears or tracking is lost.

Optional extension (5–10 minutes)

1. Replace the 3D child object under `ImageTarget_Astronaut` with your own prefab.
2. Adjust scale/rotation so the object sits nicely on the target.
3. Add a simple animation (e.g., slow rotation) to show that the augmentation is dynamic.

Conclusion

This lab session provides an opportunity to develop an AR application using Quest 3 headsets with Unity, focusing on the fundamental interactions such as object selection and movement. Through hands-on tasks, students will not only grasp the core principles of AR development but also acquire practical skills in implementing interactive features that enhance user engagement with virtual content