COMSM0129: Augmenting the Real World

Lab 1: Hello World

Overview

The purpose of this lab is to get you started with AR Foundation, the package that we will be relying heavily on to develop our AR applications for this unit. We will start by creating an AR "hello world" App for our Google Pixel phones (Android), while ensuring all the settings are correct for the platform. Next, we will get to know what AR *trackables* are and visualise them in our App.

After that, we will create another "hello world" app for our Meta Quest HMDs. This will be useful for you to develop AR apps for HMDs in the future.

Tasks

1. Build a "Hello World" AR app that displays a live background image from the smartphone's camera.
2. Visualise AR trackables: point clouds and planes.
3. Build a "Hello World" AR app for HMD.

Task 1: Hello World (Google Pixel)

In this task, you will take the first step for creating an AR application using Unity. Specifically, you will:

- Create an AR project and configure the environment settings, ensuring that all the necessary packages are installed and that the settings are correct for your target platform.
- Add AR Foundation components to your project.
- Build your App and test it on Google Pixel phones.

Subtask 1: create a Unity project

1. Open Unity Hub and create a new project.
2. Select the " Universal 3D" template and name your project "arunit", or another name.

   You may notice that there is an "AR" template available to choose, but starting with a simple 3D template allows you to exclude unwanted packages.

Subtask 2: update settings and install packages

1. The first thing you want to ensure is that you are building for the correct platform, so go to `File > Build Profiles`, where you will see several platforms to choose from. Select `Android` and `Switch Platform`.
2. Install AR-related packages. Go to `Window > Package Manager`, then click on the tab that says "Packages: In Project" and select "Unity Registry". Then select "AR" and install `AR Foundation` and `Google ARCore XR Plugin` only. *You may need to re-start the editor after this.*

   ARCore (for Android) and **ARKit** (for iOS) are two mobile AR platforms developed by Google and Apple respectively. **AR Foundation** is a cross-platform framework developed by Unity that provides a common API that supports core functionalities of both ARCore and ARKit.
3. **URP Configuration:** Since we are using URP, one additional configuration step is required for the camera background to work.
   - Go to the Project window and navigate to `Assets > Settings`.
   - Select the asset named `Mobile_Renderer` (or similar).
   - In the Inspector, click the `Add Renderer Feature` button at the bottom.
   - Select `AR Background Renderer Feature` from the list.
4. Enable AR in your project. Go to `File > Build Profiles > Player Settings > XR Plug-in Management >` under `Andriod` icon `>` tick `Google ARCore`.
5. Player settings. Go to `File > Build Profiles > Player Settings > Player`. Change the `Company Name` and `Product Name` to something unique, so that your build can be distinguished from those of other students.
6. Click and expand `Other Settings` in the same settings window, and
   - Deselect `Auto Graphics API`, then remove `Vulkan` from `Graphics API`.
   - Deselect `Multithreaded Rendering` , as ARCore does not support this.
   - In the `Identification` section, change `Minimum API Level` to `API Level 26`.
   - In the `Configuration` section, change `Scripting Backend` to `IL2CPP`, then in `Target Architectures` tick `ARM64`.
7. Lastly, we need to set up our Android phone to make it capable of running AR Foundation applications. This has been previously done for all the phones used in this class, but it's still worth pointing out. On the Android device:

- Install the ARCore package (Google Play Services for AR) from the Google Play store.
- Navigate to Settings > About phone and continuously tap Build number until a pop-up appears that asks you to confirm that you are a developer.
- After enabling developer mode, go to Settings > Developer options > Debugging and enable USB debugging.

Subtask 3: create a basic AR Foundation scene

1. Looking at the scene `Hierarchy`, we can see that there are already three components: `Main Camera` and `Directional Light`. Delete them.

2. Right-click on the empty space under `SampleScene`, and select `XR > AR Session` to add an AR Session.

   AR Session is the AR Foudation component that is responsible for controlling the lifecycle of the AR experience. It can be used to enable/disable certain AR features on the fly.

3. Now add `XR Origin` to your scene. You can see that `XR Origin (Mobile AR)` has a nested component called `Main Camera` (under `Camera Offset`).

   XR Origin is used to scale and transform your virtual AR content into the session space provided by the AR-Session data using the **transform** component of the **XR Origin**. It serves a way to bridge session space and unity coordinate space. The **Main Camera** doesn't have to be a child of **XR Origin**, but when it is, content can be scaled using the **scale** component of the **XR Origin** transform.

4. If you click `Main Camera`, the inspector will show its components.

   The **AR Camera Background** component is what replaces of the normal background of a Unity scene with live images that smartphone camera captures. Note the **Light Estimation** component here, it is disabled now but we will use them later in the course.

5. Save the scene you just constructed.

6. Go to `File > Build Profiles`.
   - Click `Open Scene List > Add Open Scene` to identify the scene to build.
   - Connect your Google Pixel phone to the PC and make sure your phone is in the Developer mode (Open Settings -> Navigate to About Phone -> Find Build Number -> Tap Build Number 7 Times)
   - In `Run Device`, select your device.
   - Build And Run. This will take a while, and when this is done, the app will launch on your device.

7. When the App is running on your phone, what you should see is the live video fed from the smartphone camera.

Task 2: Visualising trackables

As you may be aware, AR relies heavily on something called Spatial Computing, which allows the App to figure out its location, movement, device orientation, and relationship to the environment. While physical sensors like IMUs (Inertial Measurement Units) play an important role, the technology called SLAM (Simultaneous Localisation And Mapping) enables better performance of AR Apps. SLAM uses computer vision and photogrammetry techniques to analyse videos and find several types of visual features to build up spatial awareness. **Planes** and **point clouds** are among the possible features (or AR Trackables, in AR Foundation language) detected by AR Apps. The SLAM system matches the features detected in previous frames to those in the current frame in order to perform spatial reasoning. Therefore, the AR Apps tend to work better when they are fed images from different angles, positions, and the longer they run. In this task, we will build an AR App to visualise the planes and point clouds detected by the AR system.

1. Either continue working on your previous project or create a new project and go through the steps above again.

2. Select `XR Origin` in the scene, and in the inspector, click `Add Component` and type "arplane". Select "AR Plane Manager" to add it.

3. In the scene hierarchy, add an `AR Default Plane`. Inspect what components it has.

4. Drag the `AR Default Plane` from the scene hierarchy to the `Assets` folder to turn it into a prefab. Delete the `AR Default Plane` from the scene.

5. Select the `XR Origin` from the scene hierarchy, and from the inspector, you can see under the `AR Plane Manager` component that there is a slot for `Plane Prefab`. Drag the prefab you just created there.

6. Now, repeat the previous step to add an `AR Default Point Cloud` prefab, then add an `AR Point Cloud Manager` component under the `XR Origin` and attach the prefab to it.

7. Build and run your App on the Google Pixel phone. When the App starts, it may not be able to detect anything immediately. However, as you move around and point at floors, ceilings, walls, and other objects, you should be able to see the detected planes and point clouds. You may also notice that sometimes planes can get merged.

8. (Optional) Customise the visualisation prefabs, e.g., change their colour, texture, etc.

Task 3: Hello World (Quest HMD)

Now we move to the Quest 3 app development. The primary objective of this task is to gain a quick experience in app development on the Quest HMDs. We will create a basic app on Quest that provides pass-through features and simple interaction through hand gestures (grabbing). To speed up the development, we will also use the building blocks provided by Meta, which allow us to build the app without scripting.

Subtask 1: Create a Unity project and complete the configurations

1. Create a new project with the "**Universal 3D**" template. The project you created for Android can't be reused.
2. **Install Meta XR SDK:**
   - Open `Window > Package Manager`.
   - Click the "+" button and choose `Add package by name`.
   - Type `com.meta.xr.sdk.all` and click **Install**.
3. **Switch to Android Platform:**
   - Go to `File > Build Profiles`.
   - Select **Android** from the platform list.
   - Click **Switch Platform** and wait for Unity to reimport assets.
4. **Configure Player Settings:**
   - In Build Profiles, click **Player Settings** (or go to `Edit > Project Settings > Player`).
   - Under the **Android tab** (robot icon) `> Other settings`, configure the following:
     - **Minimum API Level:** Set to **Android 10.0 (API Level 29)** or higher.
     - **Install Location:** Set to **Automatic**.
     - **Scripting Backend:** Set to **IL2CPP**.
     - **Target Architectures:** Check **ARM64** only (uncheck ARMv7).
     - **Graphics APIs:** Remove **Vulkan** if present (select it and click the "-" button). Ensure only **OpenGLES3** remains.
     - **Color Space:** Set to **Linear** (under `Rendering` section).
     - **Texture Compression:** Set to **ASTC**.
5. **Configure XR Plug-in Management:**
   - Go to `Edit > Project Settings > XR Plug-in Management`.
   - Click **Install XR Plugin Management** if prompted.
   - Click the **Android tab** (robot icon).
   - Check the box for **OpenXR**.
   - A popup will appear asking to enable **Meta XR Feature Set** - click **Yes**.
   - Click the small **settings icon** (gear icon) next to OpenXR (or select `OpenXR` in the left sidebar under XR Plug-in Management).
   - Under the **Android tab**, in the **Interaction Profiles** section, ensure **Oculus Touch Controller Profile** is added.
   - In the **Feature Groups** section, ensure **Meta Quest Support** is checked.
6. **Fix Project Validation Issues:**
   - In `Project Settings`, select `Project Validation` in the left sidebar.
   - Click the **Android tab** (robot icon).
   - You will see a list of issues with warning icons (!). Review them and click **Fix All** to resolve automatically.
   - Common issues include: Color Space (should be Linear), Graphics API (OpenGLES3 only), IL2CPP scripting backend, ARM64 architecture.
7. **Run Meta Project Setup Tool:**
   - Go to `Meta > Project Setup Tool` (or `Tools > Meta XR > Project Setup Tool`).
   - In the Project Setup Tool window, switch to the **Android** tab.
   - Review any remaining "Outstanding Issues" (e.g., recommended quality settings).
   - Click **Fix All** to apply Meta's recommended settings for Quest development.
   - Click **Apply All** if there are recommended (but not required) improvements.

Subtask 2: Build the basic Quest app

1. **Create and prepare the scene:**
   - Create a new Scene: `File > New Scene > Basic (Built-in)` or `Basic (URP)`.
   - Save the scene: `File > Save As` (e.g., "HelloQuest").
   - In the Hierarchy, delete the default components (we'll use Meta's Camera Rig instead).
2. **Add Meta Building Blocks:**
   - Go to `Meta > Building Blocks` (or `Tools > Meta XR > Building Blocks`).
   - This opens the Building Blocks window/panel (it may appear as an overlay or dockable window).
   - From the Building Blocks panel, drag the following blocks into your scene Hierarchy:
     a) **Camera Rig:** This sets up the OVRCameraRig prefab for head and controller tracking. It will appear as "OVRCameraRig" in your Hierarchy.
     b) **Passthrough:** This configures passthrough (mixed reality mode) automatically. It adds an OVRManager if not present and configures camera passthrough settings.
     c) **Grab Interaction:** This adds hand tracking capability and a grabbable cube object as a demonstration.
3. **Adjust the grabbable cube position:**
   - In the Hierarchy, expand the `Grab Interaction` object (or look for an object named "Grabbable Cube" or similar).
   - Select the `Cube` child object.

- In the Inspector, set the Transform **Position** to `(0, 1, 0.5)` so it appears floating in front of you at a comfortable height.

4. **Prepare your headset:**
   - Ensure your headset is powered on and Developer Mode is enabled.
   - Connect your headset to your computer using a USB-C cable.
   - Put on the headset briefly - you should see a prompt asking to "Allow USB debugging". Select **Always allow from this computer** and tap **OK**.
5. **Build and Run:**
   - Go to `File > Build Profiles`.
   - Verify that **Android** is selected as the platform.
   - Under "Run Device", click the **Refresh** button if your headset doesn't appear.
   - Select your **device** from the "Run Device" dropdown menu (it may appear as "Meta Quest 3" or a serial number).
   - (Optional) Under "Scenes in Build", ensure your current scene is added. If not, click **Add Open Scenes**.
   - Click **Build And Run**.
   - Choose a location and filename to save the APK file (e.g., "HelloQuest.apk").
   - Unity will build the APK and automatically install it on your Quest 3 (this may take several minutes for the first build).
6. **Test the app:**
   - Once the build completes, the app should launch automatically on your Quest 3.
   - Put on your headset. You should see:
     - Your real-world surroundings (passthrough mode).
     - A virtual cube floating in front of you at position (0, 1, 0.5).
   - Reach out with your hands - you should see hand tracking working.
   - Use a pinch gesture to grab the cube and move it around.
   - If hand tracking doesn't work, ensure you've enabled hand tracking in Quest settings: go to Settings > Hands and Controllers > Hand Tracking.
7. **(Optional) Add more interactions:**
   - Return to Unity and open the Building Blocks panel again.
   - Try adding other blocks such as:
     - **Synthetic Hands:** Shows virtual hand models.
     - **Poke Interaction:** Allows poking UI elements or objects.
     - **Distance Grab:** Grab objects from far away using ray casting.
   - After adding new blocks, click **Build And Run** again to update the app on your Quest.
8. For more details, refer to the official documentation: Meta Horizon Unity SDK - Hello VR Tutorial.

Conclusion

In this lab session, we first learnt how to build and run a basic AR App with minimal functionalities. We also became familiar with several important AR Foundation components, including ARSession, XR Origin, and ARCamera. We learnt what trackables are and visualised them in the App. We also created a simple app to experience the features provided by Quest HMDs. In the next session, we will explore the AR Trackables as well as their Managers and interact with them through C# scripts.