AR Mobile Application Documentation

Libraries & Technologies

Unity	https://unity.com/
Google ARCore	https://developers.google.com/ar/
NuGet	https://www.nuget.org/
SQLite	https://www.sqlite.org/index.html
YoutubeExplode	https://github.com/Tyrrrz/YoutubeExplode

App Permission

The mobile app requires camera and storage permission in order to function properly.

Navigation

The main menu contains two tabs. The first tab displays a list of ‘portals’ (or ‘worlds’) that the user had downloaded. User can click on the green ‘Enter’ button and the app would switch to the AR user interface, which would be discussed later. The second tab is the search tab where users can download portals by entering their map codes. Map codes are 8-character strings containing letters and numbers (e.g. ABCD 1234). Capitalisation does not matter. If a password is set by the portal creator, users must enter the correct password to download the portal. Currently, the third tab, settings, is just a placeholder for potential additional features in the future.

AR Interface

After the user choose to enter a portal, the camera is turned on and the camera image is shown on screen. The user is advised to walk around to facilitate ground surface detection. Once the app detects a ground surface, a grid would be displayed on that surface. The user can then click on the grid and a virtual portal would be generated at the clicked position.

The user can walk into the virtual portal and explore the room. Note that Google ARCore makes use of the camera to track relative position of the ground surface. It is advised that the user should keep some part of the ground surface within the view of the camera (avoid the camera facing upward). Otherwise, the app might lose track of the ground surface and the portal would disappear.

Objects are generated in the virtual room according to the configuration in the web app. Currently, users can interact with objects in two ways.

Firstly, the user can tap and hold on an object for around 2 seconds. After 2 seconds, the user is now ‘holding’ the object. The user can move the object around by sliding finger or walking until the finger is lifted.

Secondly, if an interaction is assigned to the object in the web app, the user can single tap on that object to trigger it. For instance, the ‘Play 360-degree video’ script is assigned to a TV object. Whenever the user taps on that TV object, the video would be played. Users can exit the video by pressing the Android back button.

Portal Download

For each portal, the mobile app downloads two files from the web server. The first file is a json file containing information of all objects in the portal, including coordinate, scale, rotation and attached script. The second file is an AssetBundle containing user uploaded models.

Currently, the web server IP is hard coded in the mobile app in *Assets/Scripts/UI/MapDownloader*. Please change the IP addresses accordingly if the server is relocated.

Data Storage

Downloaded AssetBundles are saved in Unity’s StreamingAssets directory.

Object information of each portal is stored in the SQLite database at *Assets/StreamingAssets/portal.db*. The (x, y, z) coordinate of each object is the relative coordinate to the portal, not the absolute coordinate in the world space.

Room Generation

The room generation logic is found in *Scripts/AR/ObjectSpawner.cs*. The app loads user custom 3D models from the AssetBundle and object information from the database.

After objects are instantiated, shaders and colliders are applied to each object.

Shader

Shader is an essential part of the AR Portal. With shader, users can only view the virtual room through the portal when they are out of the room. And users can view the real world through the portal when they are inside the room.

Shaders are applied to each object at runtime during the room generation process. All custom shaders are located under the *Assets/Shaders* directory. The *PortalWindow* shader is applied to the portal plane at the portal. This plane defines the area from which users can view the virtual room or real world.

The *SpecularStencilFilter* shader is applied to every object at runtime. The value of *StencilTest* determines how the object is displayed. When the user is out of the portal, it has a value of *CompareFunction.Equal,* which means the user can only view objects via the portal plane. *When* the user is inside the room, it has a value of *CompareFunction.Always*, which means objects are always shown regardless of whether the user is looking through the portal.

To decide whether the user is inside the portal, there are colliders attached to the portal plane and the AR camera (the phone). Whenever they collide with each other, the app calculates the relative position of them and changes the *StencilTest* value accordingly.

Collider

Mesh colliders are applied to every object and all its children with mesh renderer at runtime. Colliders are used to detect whether users tap on an object and to trigger appropriate interactions.

360-degree Video

Playing 360-degree video is one of the object interactions. This feature supports playing Youtube videos given the Youtube video links (e.g. https://www.youtube.com/watch?v=QKm-SOOMC4c&t=2s). When a 360-degree video is played, the AR camera is surrounded by a sphere and the video would be shown on the inner wall of the sphere.

The app first converts the Youtube video link into .mp4 link using the YoutubeExplode library.

The *Video360* shader is applied to the video sphere to play the video on the inner wall of the sphere and to invert the video laterally. Without the shader, the video would be shown on the outer wall of the sphere.