Fernando Chirici
Report for the exam of Computer Graphics and 3D
Professor: Marco Berretti

Placing and draggin an AR object from within the scene in Unity 3D

31st January, 2023

Abstract

In this report will be shown how has been developed a software able to create an AR experience where users can place an AR object on a plane in the scene and move it by grabbing it with their hand from within the scene. The project was implemented using *Unity 3D*, a popular game engine and development platform and utilizes the *ManoMotion* library to detect the user's hand movements to be then able to translate them into corresponding actions within the AR scene. The object can only be place within the actual planes (horizontal, like the floor or a table, and vertical like a wall) that are detected on the scene and are only visible if there is no other plane between the area of placement of the object and the viewer. This works just like natural object placing: if you would place an object beyond a wall or under a table you would not be able to see it.

The project serves as a proof of concept for the potential use of AR technology in various applications. The use of hand tracking technology such as ManoMotion allows for a more natural and intuitive interaction with the AR environment.

Index

1 Introduction to the development environment

The development of this project was carried out using Unity 3D version 2019.4.40f1. Unity 3D is a cross-platform engine that supports the development of 2D, 3D, AR, and VR applications. The engine provides a wide range of tools and features able to create high-quality and interactive experiences such as plane detection and occlusion. The project also required the use of various libraries and assets to support the development of augmented reality features. These libraries includes:

- AR Foundation 4.0.2
- ARCore XR Plugin 4.0.3
- ARKit XR Plugin 4.0.2
- Universal RP 7.7.1
- XR Plugin Management 4.2.0
- ManoMotion SDK Pro 1.4.9

where every AR, XR library is included in order to create and work with AR sessions, Manomotion, as already said is a hand-tracking software development kit that uses computer vision algorithms to detect and track hand gestures in real-time. It allows developers to create interactive and immersive experiences by incorporating hand gestures as input in their applications.

Finally this project has been developed and tested for Android devices only. To set up Android development on Unity 3D, the first step is to download and install the Android Build Support and Android SDK tools through the Unity Hub, next, you will need to set up the Android SDK and JDK paths in Unity. This can be done by going to "Edit" "Preferences" "External Tools" and specifying the paths for the Android SDK and JDK. You will also need to configure the player settings for your project by going back to "Edit" and then "Project Settings" "Player" and setting the bundle identifier and package name as defined by the Manomotion SDK configuration. In my case, since I've created a trial account for students, the package is set to *com.username.hellomano*.
At last, in order to grant compatibility with most android devices, the render pipeline of the project has been set to Universal RP.

It is to be noted that a significant portion of this project involved finding the correct versions of Unity and the necessary libraries that were compatible with each other. This proved to be a non trivial task as I needed to ensure that the versions of Unity, ARFoundation, ARSubsystems and Universal RP were compatible with each other, with the specific version of the UnityEngine being used in the project and with the specific Android device and AndroidAR services.

2 Setup the AR scene

To create an AR session in Unity 3D, several elements are needed and three are necessary. The first is the **AR Session** component, which is the foundation of the AR experience and is responsible for managing the device's camera and its relationship with the real world. The AR Session component is the entry point for the ARCore and ARKit frameworks, which provide the underlying functionality for tracking the device's position and orientation in the real world.

The second element is the **AR Session Origin** component, which is responsible for managing the coordinate system of the AR experience. It acts as the parent of the **AR Camera**, the third essential component, and is responsible for keeping track of the origin and scale of the AR experience. The AR Camera component acts as the main camera for the AR experience and is responsible for rendering the AR content and for providing the AR Session with the necessary information about the device's camera. It is also responsible for adjusting the camera's position and orientation based on the AR Session's tracking data.

In the AR Session Origin is also necessary to add a **Raycast Manager**. This is an important component responsible for managing the raycasting functionality of the AR experience, which is used to determine the point in the real world where the user is pointing the device's camera. It provides a number of options for controlling the raycasting behavior such as maximum distance and visualization of the ray but in this project will just be used to locate the pointing of the user.
In the following section will be shown other ways in which the AR Session Origin component has been altered in order to be able to perform plane detection and object placing.

Fig. 1. Basic configuration for AR development.

3 Plane detection, occlusion and interaction

To make the application as closer to reality as possible, before being able to place an object, the device needs to detect a plane to place it. By default this can be both an horizontal plane such as the floor or a table, or a vertical plane such as a wall. At first, an **AR Plane Manager** component must be added to the scene as children of the AR Session Origin, to be responsible for managing the detection and tracking of surfaces. If the only target was to just detect planes and plot them in the scene, the previous steps would have been enough since the those are action offered by default by the component. Instead, to create a more real experience, the plane prefab associated with the component needed to be custom in order to perform **plane occlusion**. To do so, a new default plane has been created in the scene to be then customized, exported as a new prefab and finally re-imported as a prefab in the AR Plane Manager (Fig. 2).

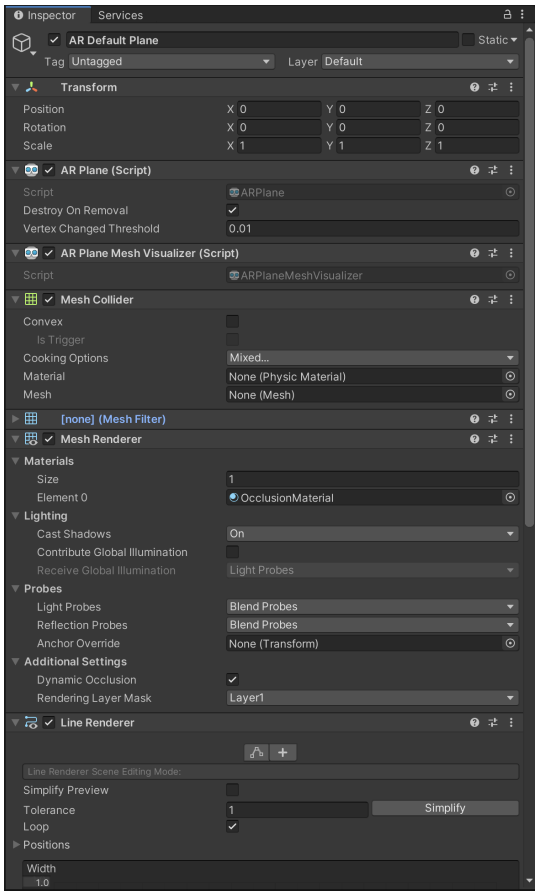

Fig. 2. AR Session Origin after the import of the AR Plane Manager Component.

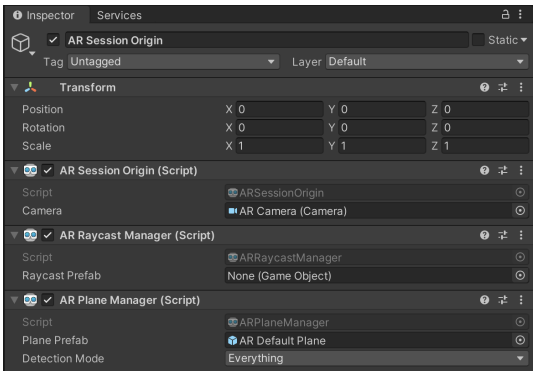

Fig. 3. AR Default Plane.

As visible in Fig 3, the AR Default Plane is composed by a default Place component, an AR Mesh Visualizer that is a script needed to plot the detected mesh of the plane, and custom Mesh Renderer where the material has been altered with a custom which inherit the render pipeline from an implemented **shader**: *PlaneOcclusionShader.shader*(Fig 4).

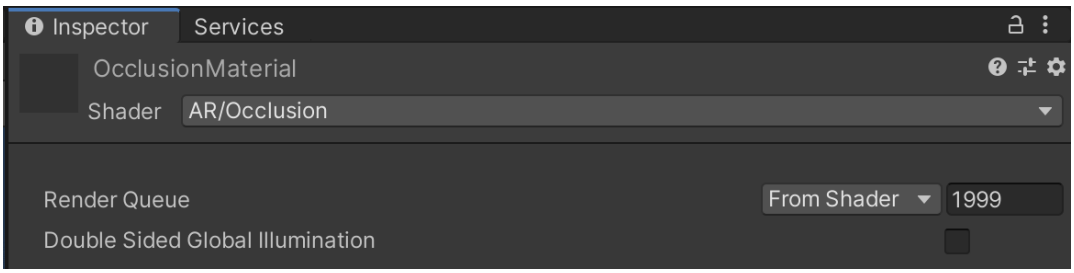

Fig. 4. Material used for the plane with inherit render queue.

The shader has been composed in order to write to the depth buffer and performs a depth test on the scene. The color buffer is not specified while the fragment function has been set to return a fixed4 with all values set to 0, effectively making it transparent. Specifically what happen is that if a plane is detected and the depth test is passed ensuring that there is a surface at a given distance from the camera, the shader will write the material to the depth buffer and will be occluding other objects that are behind it in the 3D space while being invisible.

At this point the result is something like shown in Fig 5. The demonstration of the effectiveness of the occlusion is postpone after the introduction of the Placement script in the next section.

Fig. 5. Plane detection on surfaces.

3.1 Object Placing

Once the plane has been detected, the system is ready to place an object on the scene. In order to do so in Unity, it's necessary to add a script of **Placement Controller** in the AR Session Origin component. The script aim to place a single object pre-selected through a serialized field (Fig 6) and to allow the user to move it by touching or dragging in different parts of the screen. To do so the ARRaycastManager component is used to perform raycasts from the device's camera into the AR world. The raycasts are used to detect planes in the AR world, and the script uses the touch screen to allow the user to place the AR object on the detected plane. The script has a *TryGetTouchPosition* method that gets the position of the first touch, if there are any touches, and returns a boolean indicating whether or not it was successful. In the *Update* method, the script checks if a touch position was obtained and if there are any AR raycast hits on a trackable plane within the polygon. If both of these conditions are true, then it either instantiates a new object or updates the position of the existing object, depending on whether or not an object has already been placed.

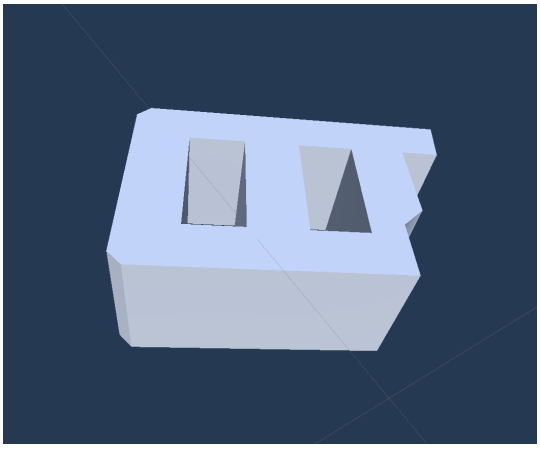

Fig. 6. The object to be placed.

Adding this script allow the software to place the brick on the scene, on the detected surface. Thanks to the occlusion shader, if an object is placed under or beyond a surface, it is then not possible to observe it from the camera. In Fig 7 an object has been set on the floor under the table, whereas in Fig 8 it's observable how in the same scene, the presence of the table in between the device and the object make it impossible to see the brick.

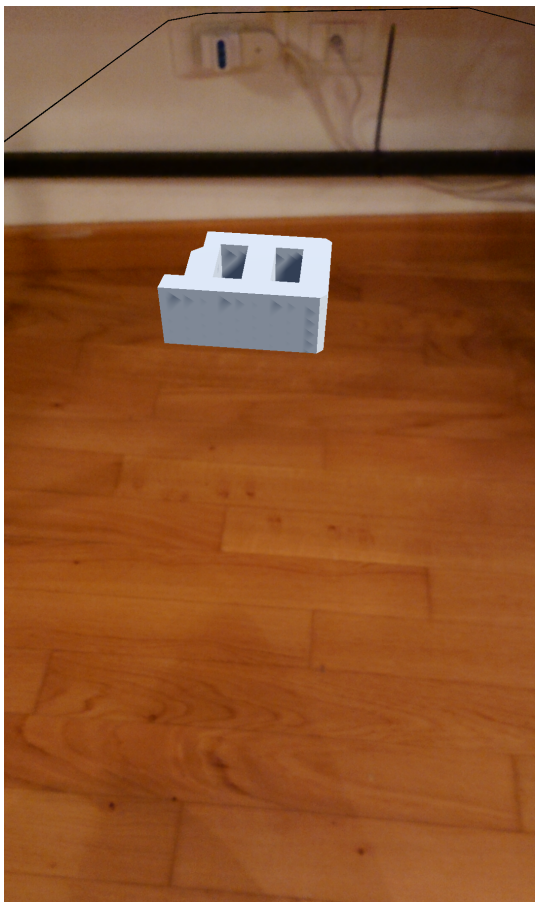

Fig. 7. Object placed under the table one the detected surface of the floor.

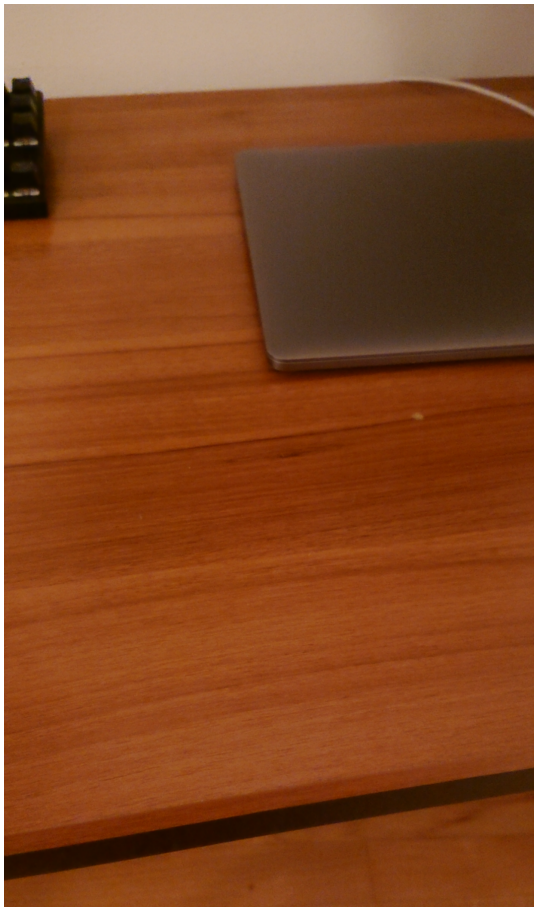

Fig. 8. Same scene observed from above the table.

4 Object manipulation with Manomotion

To achieve the goal of the project, to be able to manipulate a placed AR object from within the scene, it was necessary to use a tool able to recognise the structure of the hand and it's gesture, to link a specific movement to a specific action on the brick. Manomotion uses a combination of computer vision and machine learning algorithms to track the hand gestures. It works by analyzing the depth data from the camera and recognizing the hand's skeletal structure and movements. It is then possible to map the hand's gestures, such as grabbing and rotating, to specific actions which can be used to interact with the AR objects. In this project, three movement are tracked and used: grasping, holding and clicking with a finger. Specifically when the user makes a grasping gesture, the tool recognizes it and sends a signal to the AR scene to grab the object, when the user holds his hand around the object, the tool sends a signal to rotate the object in the AR scene and when the user click mid-air the tool send the signal to spawn a copy of the object at the coordinates indicated by the clicking.

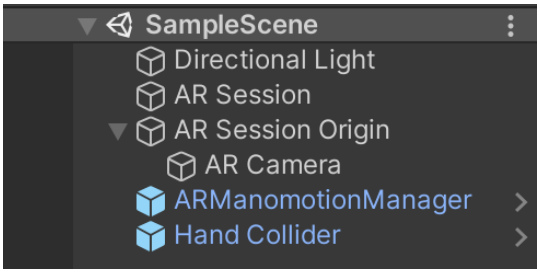

Fig. 9. The complete scene with Manomotion components.

First the components needs to be created. As visible in Fig. 9, the scene is completed adding an **ARManomotion Manager** and **Hand Collider** components. The first one, visible in Fig 10, provides an interface between the Unity project and the ManoMotion SDK, which allows to access the tracking data and use it to control the AR application.

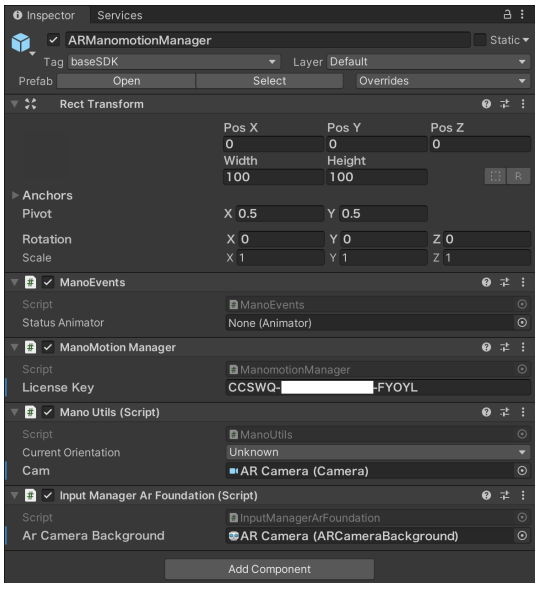

Fig. 10. ARManomotionManager component.

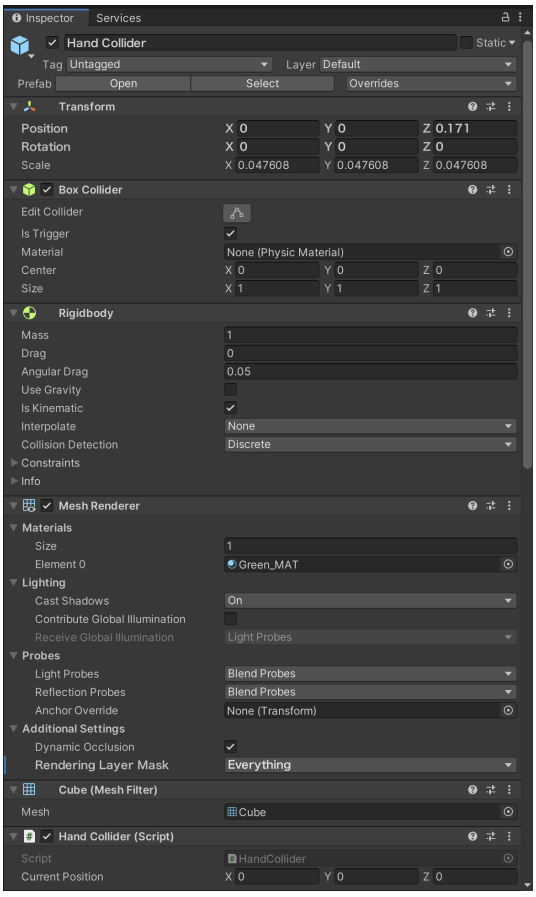

Fig. 11. Hand Collider component.

This component is mainly used to setup the usage of the SDK in the project and in order to do so is necessary to enter the licence key and to setup the correct camera that will use the tool. Note that to be able to works, Manomotion needs an internet connection to establish the validity of the licence key. In Fig. 11 instead, is shown the Hand Collider, a component that I've physically represented by a small cube (*collider object*) characterized by a rigid body with

kinematic properties and a mesh renderer with a opaque visible color. This component works with a Manomotion script that allows for the hand's movement to be tracked in the 3D space and updates the position of the collider object accordingly, allowing for the hand to interact with other objects in the scene with the collider. This collider works in fact as a pointer laying over the hand and is needed in order to interact with the brick. To be precise the hand is not exactly the one that grabs the brick but it is just used to recognise the movement of the finger based on the skeleton extraction to give the correct action input. It's the collider cube that going over the brick actually attaches (Fig 12).

Fig. 12. Result of the creation of the collider cube.

So far the Manomotion environment has just been integrated on the project but it's not linked to the interaction with the brick. To do that, it's necessary to extend the component of the object brick itself.

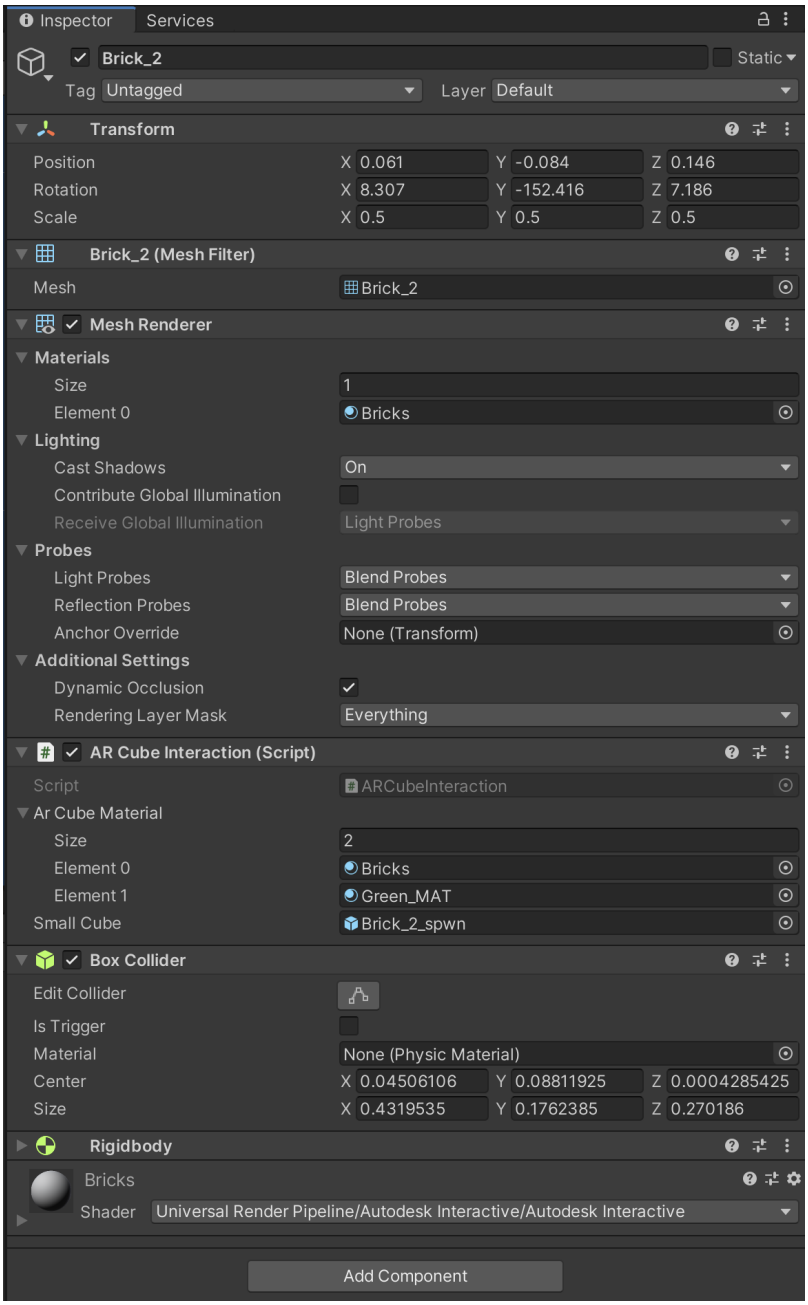

Fig. 13. Properties of the object brick.

In Fig. 13 are visible the characteristics of the object brick, in particular the AR Cube Interaction script. That is another MonoBehaviour that allows interaction with an AR cube in Unity. It has three main functionalities:

1. **MoveWhenGrab**: when the user makes a grab gesture, the brick will move to accordingly position of the hand as long as the grab is maintained.

2. **RotateWhenHolding**: when the user makes a pinch gesture, the cube will rotate around its up axis as long as the pinch is maintained

3. **SpawnWhenClicking**: when the user makes a click gesture, another brick will be spawned at the position of the big cube

This functionalities are implemented taking full advantage on the Manomotion library that allow to extract the hand position of the user and compare it with a standard position such as "grab" (Fig 14)

```
private void MoveWhenGrab(Collider other)
{
    if (ManomotionManager.Instance.Hand_infos[0].hand_info.gesture_info.mano_gesture_continuous == grab)
    {
        transform.parent = other.gameObject.transform;
    }

    else
    {
        transform.parent = null;
    }
}
```

Fig. 14. Implementation in C# taking advantage of the skeleton detection.

The AR Cube Interaction script has also two more functionalities:

1. **OnTriggerEnter**: when the hand enters the collider of the cube, the cube will change its material and the device will vibrate
2. **OnTriggerExit**: when the hand leaves the collider of the cube, the cube will change its material again

Fig. 15. Example of grabbing the AR object to make it move.

Fig. 16. Example of holding the AR object to make it rotate.

In Fig. 15 and 16 are visible the examples of grabbing and rotating the object. An image is not the ideal media to show the effectiveness of the scripts but the changing of the color is

already indicative. In fact, when there is a cube interaction event, following what's reported in Fig 14, the object inherit the color of the cube pointer for the all interaction time.
Note that the previous images show a brick placed midair and not placed on a plane. This was only done to provide a more clear view of the object and the interaction.

5 Building and testing the software

As already anticipated, the software has been developer for Android devices only. Specifically, I've tested it on a Huawei P20 Pro and a Sony Xperia Z7. These devices have the necessary sensors and hardware to support the AR functionality of the software. It is worth noting that the testing environment should have a good amount of ambient light in order to achieve optimal performance. This is because the software relies on the device's camera to detect and track hand gestures, and a lack of light can negatively affect the accuracy of the hand tracking.

In order to build and run the software on an Android device, the following steps needs to be followed.
At first, **on the Android phone** is necessary to enable the developer options, the possibility to install application by other sources that are not the official store, to enable "debug usb", and to install the Android AR Service application. An Android of recent generation AR-compatible is obviously needed.
Once the previous steps are completed, it's possible to connect the phone with a cable directly to the computer and set the mode to "file transfer".

In Unity:

1. Select "File" and "Build Settings" to open the Build Settings window

2. In the Platform section, select "Android" and click "Switch Platform" to set Android as the target platform.

3. It is then possible in the "Player Settings" to configure details of the build such as the name.

4. Once everything has been set, click on the "Run device" menu and select your device. If everything has been enabled as previously described, it should be there (Fig 17).

5. Press "Build and run" to build the software directly on the Android device. At the end of the build, the app will open automatically.

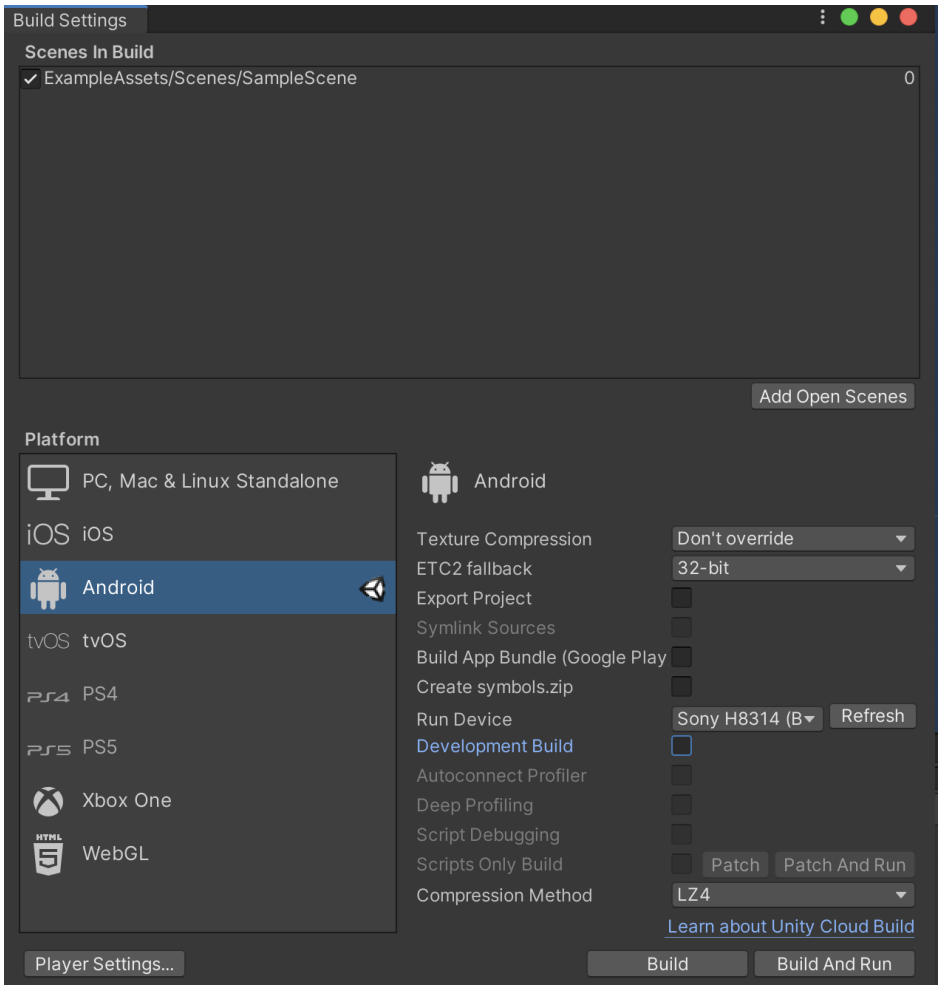

Fig. 17. Build settings view before pushing "Build and run".