

Session 4

Working with VR Player Character

Welcome to the session, **Working with VR Player Character.** This session illustrates the creation of a simple **Player Creation** in Virtual Reality.

⮚ Explain OVRPlayerController

⮚ Describe OVRCameraRig

⮚ Explain Left and Right Hand anchors

4.1Understanding OVRPlayerController

OVRPlayerController is a component in the Unity game engine that controls the player's movement in a virtual reality (VR) environment using an Oculus VR headset. This component is part of the Oculus Integration package that is provided by Oculus for Unity.

Features of **OVRPlayerController**:

1. TheOVRPlayerController is a prefab that can be added to a Unity project to create a VR player character. It includes a variety of components that work together to create a functional player controller, such as the **OVRManager, OVRInput, OVRHand**, and OVRPlayerController components.
2. The OVRPlayerController supports different types of movement modes, including smooth locomotion, snap turning, and teleportation. The smooth locomotion mode allows the player to walk or run in any direction by moving the **thumbstick** on the **Oculus Touch controllers**. The snap turning mode rotates the player in set increments to reduce motion sickness, and teleportation mode allows the player to move instantly to a new location in the game world.
3. TheOVRPlayerController includes a **hand tracking** system that allows the player to interact with objects in the game world using their virtual hands. The **OVRHand** component provides a set of scripts and prefabs that simulate hand gestures and finger movements, allowing players to pick up, hold, and manipulate objects.
4. The OVRPlayerController is designed to work with the **Oculus Rift** and **Oculus Quest VR headsets**. It includes support for the **Oculus Touch controllers**, which provide accurate hand tracking and input. The **OVRManager** component handles the communication between the headset and the Unity engine. **OVRInput** component handles the input from the Oculus Touch controllers.
5. The OVRPlayerController can be customized in many ways to suit the requirements of a particular VR experience. For example, developers can adjust the player's movement speed, jump height, and other physics parameters. They can also replace the default hand models with custom models that match the art style of the game.
6. The OVRPlayerController includes support for **networked multiplayer**, which allows players to connect and interact with each other in a shared game world.
7. TheOVRPlayerControllerincludes built-in support for spatialized audio, which allows sounds to be positioned in 3D space based on the position and orientation of the player. This enhances the immersion of the VR experience and provides important audio cues for gameplay.
8. The OVRPlayerController includes a collider that detects collisions with other objects in the game world. This can be useful for implementing physics-based interactions, such as picking up and throwing objects.
9. The OVRPlayerController integrates with Unity's physics system, allowing developers to create realistic physics-based interactions between objects in the game world. For example, objects can be picked up and thrown, or pushed and pulled around the environment.

4.2 Understanding OVRCameraRig

The **OVRCameraRig** is a prefab in the **Oculus Integration package** for Unity that provides a ready-to-use virtual reality camera rig for developing VR applications.

1. **Camera Setup:** The **OVRCameraRig** includes a left and right camera, which are positioned slightly apart to simulate the stereoscopic **view of the human eyes**. The cameras are also rotated to match the orientation of the player's head, which allows the player to look around the virtual environment in a natural way.
2. **Tracking:** The OVRCameraRig includes support for both positional and rotational tracking, which allows the player's head movements to be translated into movements within the virtual environment. This is achieved using the **Oculus tracking system**, which provides accurate and low-latency tracking of the VR headset.
3. **Controller Support:** The OVRCameraRigincludes support for the **Oculus Touch controllers,** which provide accurate hand tracking and input in VR. The controllers can be used to interact with objects in the virtual environment and perform actions such as grabbing, throwing, and shooting.
4. **Guardian System:** The OVRCameraRig includes support for the **Oculus Guardian system**, which allows developers to define a safe play area within the virtual environment. This helps prevent players from accidentally walking into physical objects while playing the game.
5. **Customization:** TheOVRCameraRigcan be customized in many ways to suit the requirements of a particular VR experience. For example, developers can adjust the camera settings such as the field of view, stereo separation, and render texture size. They can also modify the controller settings, such as button mappings and haptic feedback.
6. **Performance Optimization:** The OVRCameraRig includes several features to optimize performance and reduce latency in VR applications. For example, the cameras can be set to use single-pass stereo rendering, which reduces the number of draw calls and improves performance. Th**e** OVRCameraRigalso includes a dynamic prediction system, which uses previous frames to predict the player's head movement and reduce latency.
7. **Audio Support:** The OVRCameraRig includes built-in support for spatialized audio, which allows sounds to be positioned in 3D space based on the position and orientation of the player. This can enhance the immersion of the VR experience and provide important audio cues for gameplay.
8. **Networked Multiplayer:** TheOVRCameraRig includes support for networked multiplayer, which allows players to connect and interact with each other in a shared game world. The Photon Unity Networking (PUN) framework is used to handle the networking, making it easy for developers to implement multiplayer features.
9. **OVRManager** is a Unity component provided by the Oculus Integration package that manages the lifecycle of the Oculus VR system in your Unity project. It provides access to various Oculus SDK features and settings, including **tracking, rendering, and input**. The OVRManager component also handles device initialization, connection management, and device configuration.
10. The **OVRManager.cs** file is the script file that implements the OVRManager component. It contains the **main logic for initializing and managing the Oculus VR system** in your Unity project. You can use this script file as a reference to understand how the OVRManager component works and how it interacts with other components in your Unity project.

4.3 Left and Right Hand Anchors in VR

In virtual reality development, **"left hand"** and **"right hand**" anchors refer to the virtual representations of the user's hands in the virtual environment. These **anchors** are typically created in the Unity game engine and are used to track the position and movement of the user's hands using input devices such as the Oculus Touch controllersas shown in Figure 4.1.



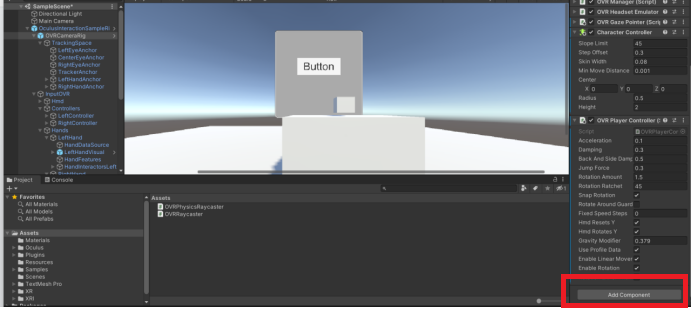
**Figure 4.1 : Left and Right Anchor Visualization**

The **left hand** anchor represents the position and orientationof the user's left hand, while the right hand anchor represents the position and orientation of the user's right hand. These anchors are used to attach virtual objects or tools to the user's hands, allowing them to interact with the virtual environment.

In Unity for Oculus, to create left and right hand anchors Oculus Integration package is used. The package provides a set of prefabs and scripts for working with Oculus input devices. To create the anchors, developer can add the **OVRHandPrefab** to the scene and then use the **OVRHand** component to access the left and right hand anchors.

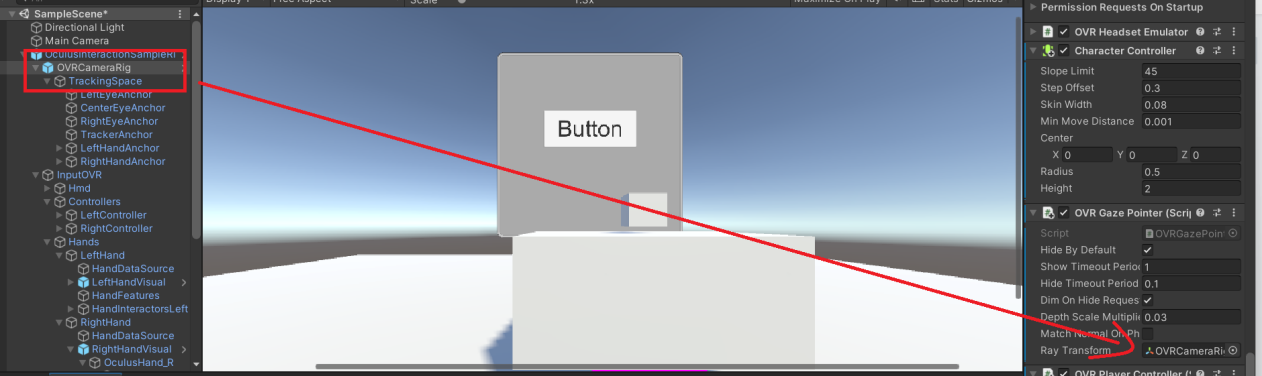
Once the left and right hand anchors are set up, developer can use them to track the user's hand movements and interact with the virtual environment. For example, developer can attach a virtual object to the user's hand anchor and allow them to manipulate the object using the **Oculus Touch controller**.

Till now we have added a UI in Our Scene and now we will Implement user Gaze in **VR Using OVR Gaze Pointer**.

**Step 1** :Select **OVRCameraRig** in **Hierarchy Panel → Select** Add Component and Search **OVR Gaze Pointer.**

**Figure 4.2 : Add Component in Inspector Panel**

**Step 2** :In **OVR Gaze Pointer** Select Drag **OVRCameraRig** to **Ray Transform** Component.

**Figure 4.3** : **Drag OVRCameraRig to OVRGazePointer**

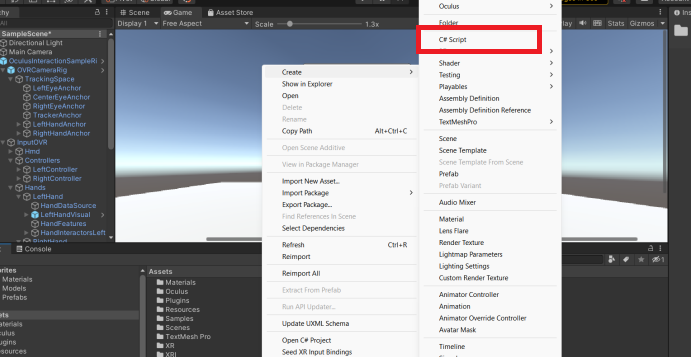
4.4 OVR Gaze Pointer

**OVR Gaze Pointer** is a component of the **Oculus Integration SDK** in Unity that provides a raycast from the user's gaze to thevirtual environment. The OVR Gaze Pointer is attached to the user's camera and generates a ray that extends from the center of the camera's view.

This ray is used to determine what the user is looking at in the virtual environment, which can be useful for implementing interactive elements such as buttons, menus, or other **UI** elements. When the raycast hits an interactive element, the OVR Gaze Pointer can trigger an event that can be used to perform an action or provide feedback to the user.

The **OVR Gaze Pointer** can be configured with various settings such as the length of the ray, the radius of the sphere that represents the pointer, and the layers that the pointer can interact with. By using the OVR Gaze Pointer in conjunction with other components of the Oculus Integration SDK, you can create immersive and interactive VR experiences that respond to the user's gaze and input.

**Step 3** :Create a new scrip**t UIFollow.cs** by **Right clicking→Create → C# Script.**

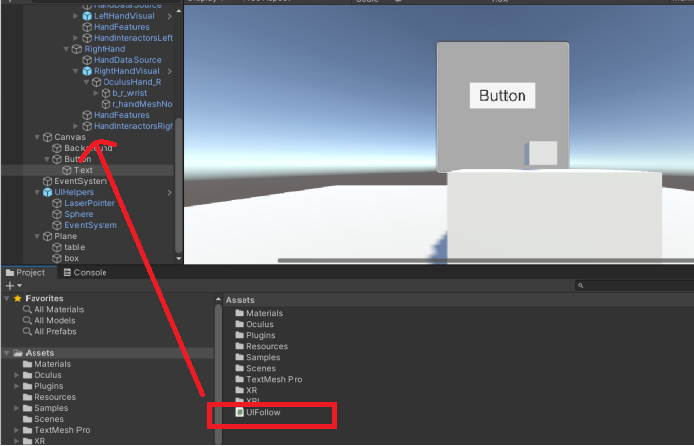
**Figure 4.4 :Create Script Name UIFollow**

**Step 4 :** Double click on Script name **UIFollow.cs** and insert Code Snippet 1.This code will position the UI element 2 units in front of the user's camera and make it face the camera.

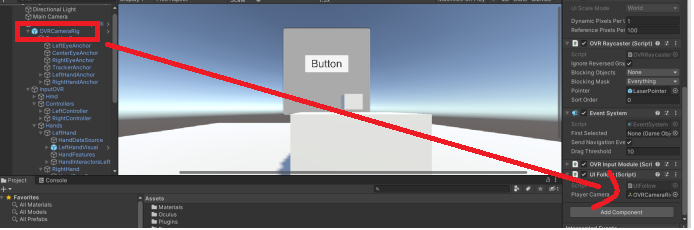
**Code Snippet1**:

|  |
| --- |
| using UnityEngine;  public class UIFollow : MonoBehaviour  {  public Transform playerCamera;  void Update()  {  transform.position = playerCamera.position + playerCamera.forward \* 2f;  transform.LookAt(playerCamera.position);  }  } |

**Step 5 :** Attach Script name **UIFollow.cs** to the **UI Canvas.**

**Figure 4.5 : Attach UIFollow.cs to canvas, Button, Background, Button and Text**

**Step 6** : Attac**h OVRCameraRig** to Script Name **UIFollow.cs** Script **Edit Panel**



**Figure 4.6 : Drag OVRCameraRig to Script name UIFollow.cs → PlayerCamera**

4.5 Oculus Pointer Input Module

Th**e Oculus Pointer** Input Module is a Unity script that allows user to use the Oculus Touch controllers to interact with UI elements in the VR application. The module provides functionality such as pointer highlighting, button selection, and click events for UI elements that are compatible with theOculus Touch controllers.

Working of Oculus Pointer Input Module:

**Pointer:** When the user points with the Oculus Touch controller, a pointer is projected from the controller's tip. The pointer is visible in the VR environment and indicates the location where the user is pointing.

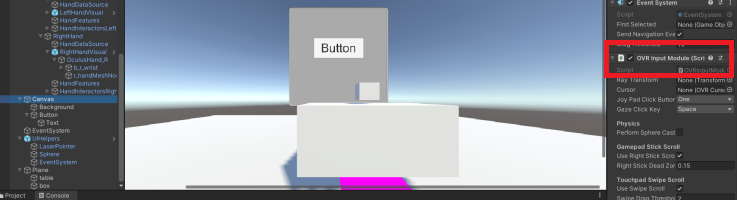
**Hovering:** When the pointer is hovering over a UI element, the element is highlighted to indicate that it can be selected.

**Selection:** When the user presses the primary button on the Oculus Touch controller while pointing at aUI element, the element is selected.

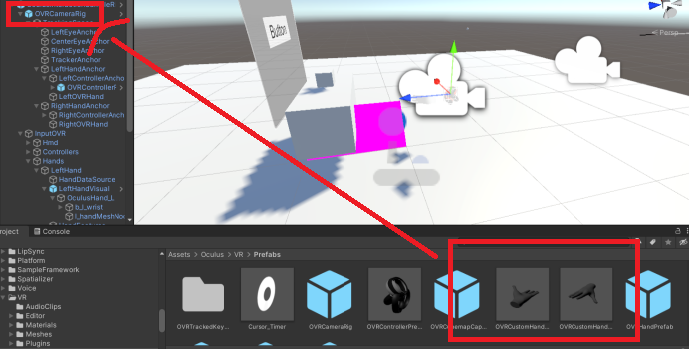
**Click Events:** Once a UI element is selected, click events can be triggered when the user releases the primary button on the Oculus Touch controller. These events can be used to perform actions such as opening a menu, navigatingto a new scene, or toggling a setting.

The **Oculus Pointer Input Module** simplifies the process of implementing VR UI interactions with the Oculus Touch controllers in **Unity**. By adding the module to the canvas, developer can easily enable pointer highlighting, button selection, and click events for your UI elements, making it easier for users to interact with your VR application.

**Step 7** : Got to **Canvas → Add Component** **→ OVR Pointer Input Module.**

**Figure 4.7 : Add a OVR Input Module Component**

**Step 8** :Add hand models to the scene that will be used as the VR controllers. The **OVRHandPrefab** from the **Oculus Integration package** into the **OVRCameraRig** object in the **Hierarchy window.** This will add **two hand models** to the scene, one for each hand.



**Figure 4.8 : Drag hand Models to OVRCameraRig**

**Step 9** :Select one of the hand models in the Hierarchy window, and then add a new script component to it by clicking the Add Component button in the Inspector window. Name the script **"HandController.".**



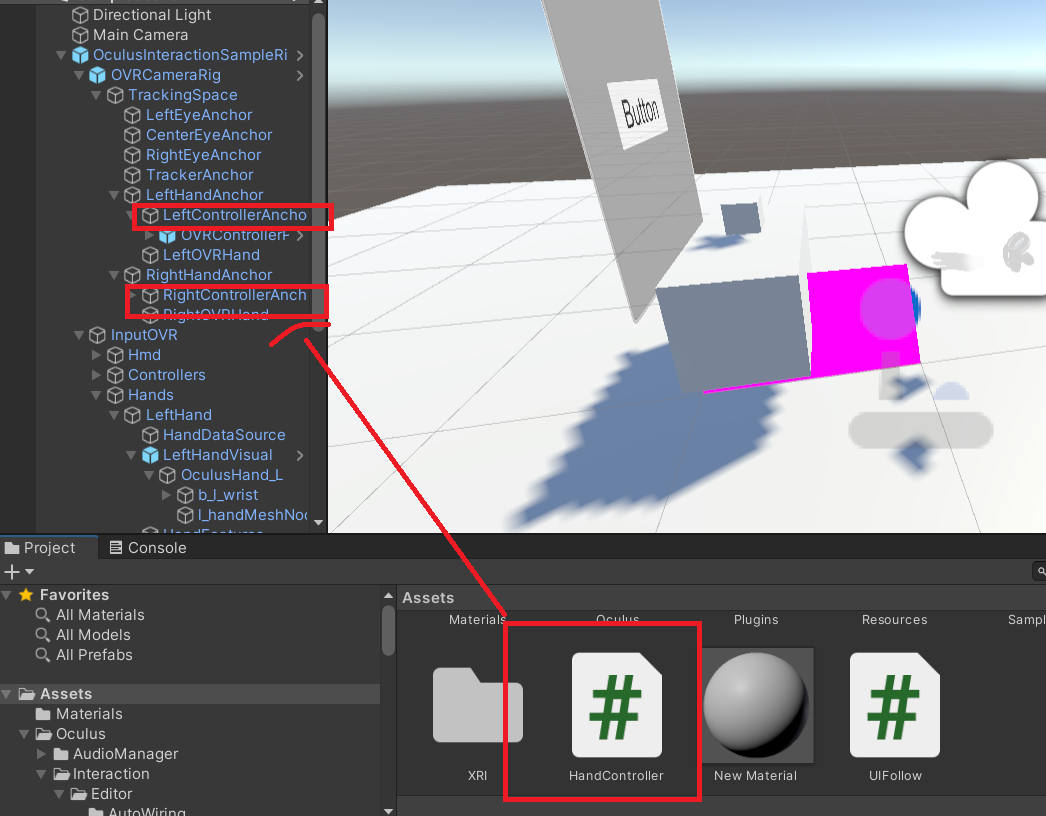
**Figure 4.9 AddComponent → HandController.cs Script**

**Code Snippet 2:** Used to create Hand Controller

|  |
| --- |
| using UnityEngine;  using OculusSampleFramework;  public class HandController : MonoBehaviour  {  private OVRHand hand;  void Start()  {  hand = GetComponent<OVRHand>();  }  void Update()  { // TODO: Add code here to handle hand gestures and input  }  } |

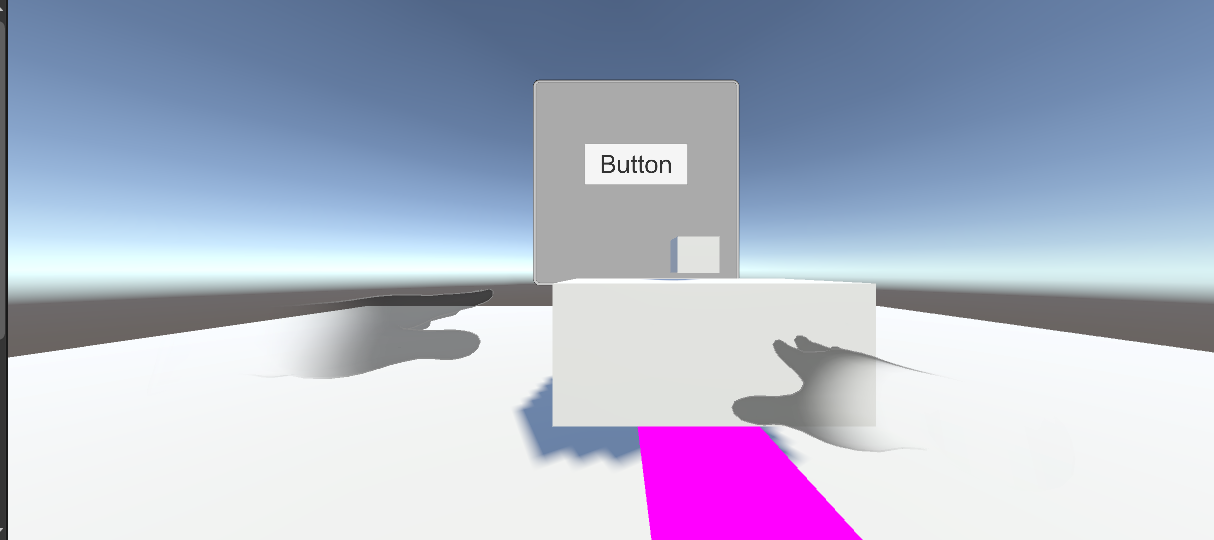
Save the script, and then go back to the Unity editor.

**Step 10** : In the **HandController** component in the Inspector window, Drag **OVRHand** Component to Hand variables.



**Figure 4.10 : Drag Script Name HandController.cs to LeftControllerAnchor and RightController Anchor**

**Step 11** :Test the App on **Oculus HMD,Final** **Output.**



4.6 Summary

⮚ **OVRPlayerController** is a component in the Unity game engine that is used to control the player's movement in a virtual reality (VR) environment using an Oculus VR headset. This component is part of the Oculus Integration package that is provided by Oculus for Unity

⮚ **OVRCameraRig** provides a convenient and customization way to set up the camera rig for a VR project in Unity.

⮚ **Left and Right Hand anchor**s are used to track the user's hands in VR and enable interaction. They are part of the VR camera rig and can be controlled in Unity along with VR controllers.

⮚ **OVR Gaze Pointer** is a Unity component that enables gaze-based interaction in VR by generating a ray from the user's gaze direction and detecting intersections with selectable objects in the scene.

⮚ **Oculus Pointer Input Module** is a Unity component that enables interaction with VR objects using the Oculus Touch controllers. It provides various input options and modes for customization.

* 1. Check Your Progress



1. Which component enables gaze-based interaction in VR by generating a ray from the user's gaze direction?

a) Hand Anchors

b) OVRCameraRig

c)OVR Gaze Pointer

d)OVR Gaze Input Module

2. Which SDK should be installed for OVRCameraRig?

a) XR Interaction Toolkit

b) Oculus Intersection SDK

c) Oculus Interaction SDK

d) XR Toolkit

1. Where to place Hand Model to set up hand as Controller?

a) OVRGazeController

b) OVRPlayerController

c) OVRCameraRig

d) OVRHand

4. Where is UIFollow Script inserted

a) Canvas

b) Button

c) Evevnt System

d) OVRPlayerController

5. Which component provides locomotion to VR Player Character?

a) OVR{PlayerController

b) OVRCameraRig

c) OVRGazePointer

d)OVRInputSystem

4.7.1 Answers

|  |  |
| --- | --- |
| 1 | c |
| 2 | c |
| 3 | c |
| 4 | a |
| 5 | b |

4.8 Try It Yourself

⮚ Add OVR Gaze Pointer Hands

⮚ Add Gaze Input Module

⮚ Add Controller as VR Hands.