

Session 3

Creating a User Interface in VR

Welcome to the session, **Creating a User Interface in VR**. This session illustrates the creation of a very simple User Interface in Virtual Reality.

In this session, students will learn to:

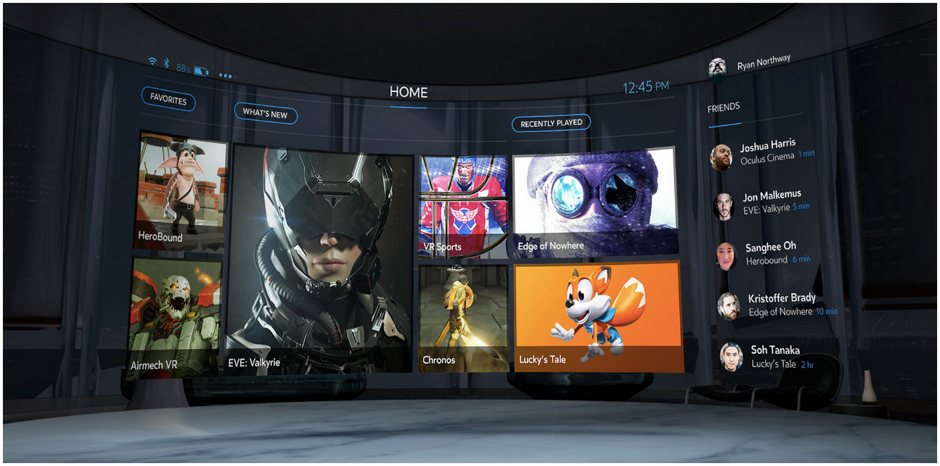
⮚ Discuss the basics of UI in VR

⮚ Explain how to create a canvas

⮚ Describe how to add a functional UI in VR

3.1 User Interface in Virtual Reality

**User interface (UI)** in virtual reality (VR) refers to the graphical and interactive elements that allow users to interact with the VR environment and **control their experience**. The UI in VR is unique (Figure 3.1) compared to traditional 2D interfaces, as it must be designed to work within a 3D space and to account for the user's movement and spatial awareness.



**Figure 3.1: User Interface in Virtual Reality**

In VR, UI is usually presented within the virtual environment itself, rather than as a **separate screen or window.** This means that UI must be integrated seamlessly into the environment and designed to **blend in with the surrounding objects.**

The UI in VR can include a variety of elements, such as menus, buttons, sliders, and text. These elements are typically designed to be interacted with **hand-held controllers** or hand-tracking technology, allowing users to point, click, and drag objects within the environment.

VR is a highly immersive experience and so, the UI in VR must be carefully designed to avoid breaking the **user's sense of presence in the environment**. This means that the UI should be non-intrusive and not distract from the experience, while still providing clear and easy-to-understand feedback and controls.

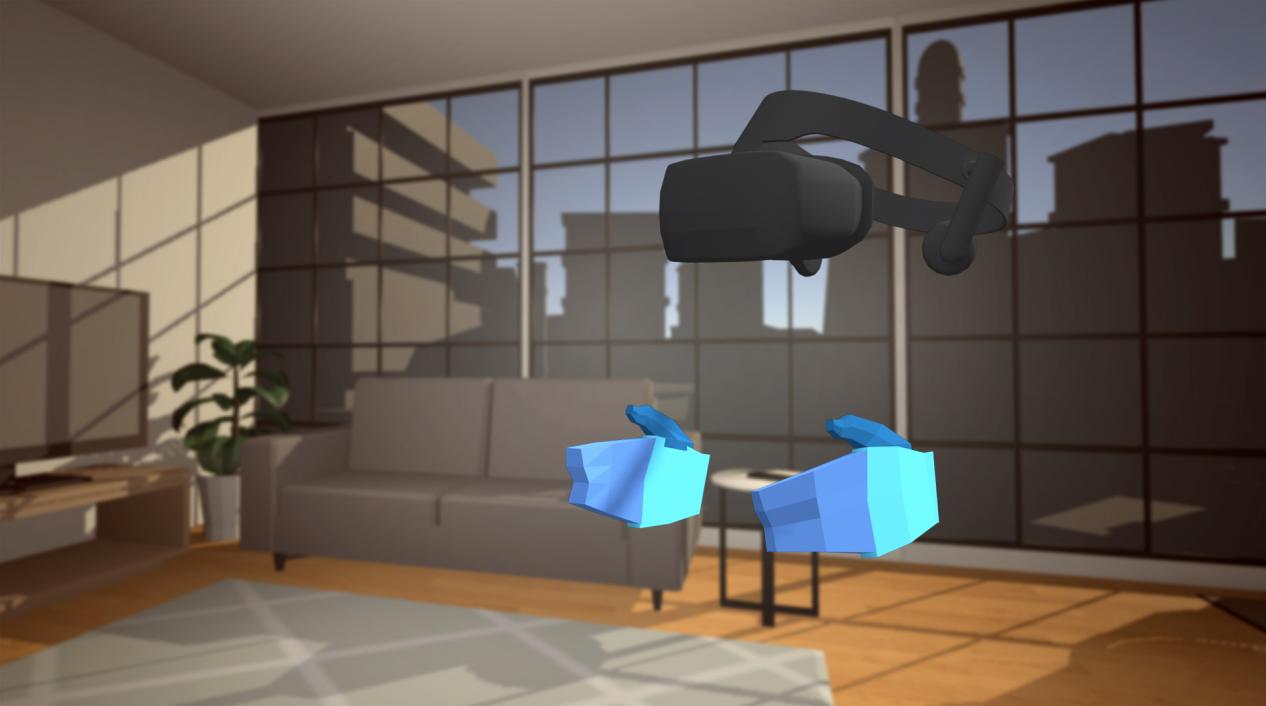
3.2 Working With VR



**Figure 3.2: User Interface in Virtual Reality Virtualization**

Working with user interfaces in virtual reality (VR) is a **complex and multifaceted task** that requires a deep understanding of both traditional user-interface design principles and the unique challenges and opportunities presented by the VR medium. Unlike traditional 2D interfaces, VR interfaces must be designed to work in a fully **immersive environment** where users are free to move their head and hands in 3D space. This means that designers must create interfaces that are intuitive and easy to use, while also being visually engaging and fitting seamlessly into the VR experience.

One of the key considerations when designing VR interfaces is **depth perception** (Figure 3.2).In a traditional 2D interface, objects appear on a flat screen, and the user perceives depth through perspective cues. However, in VR, objects have actual depth and **occupy a 3D space,** which can make it challenging to create interfaces that are easy to navigate and understand. Designers must carefully consider the placement and **orientation of interface elements**, such as buttons and menus, to ensure that they are visible and easy to interact with.



**Figure 3.3: User Interface and Virtual Reality in Virtual Space**

Another important consideration is scale. In VR, objects can appear **much larger or smaller than they would in the real world**, which can be disorienting for users. Designers must ensure that interface elements are appropriately sized and scaled to avoid confusion or discomfort.

Interaction mechanics are also a crucial consideration when designing VR interfaces. Unlike traditional interfaces, where users interact with objects using a mouse or touch screen, VR interfaces require users to use their **hands or a controller to interact** with objects in 3D space(Figure 3.3). Designers must carefully consider the mechanics of user interaction, including how users will select and manipulate objects, to ensure that the interface is both intuitive and easy to use.

Overall, working with user interfaces in VR requires a deep understanding of the **unique constraints and possibilities** of the medium, as well as a willingness to experiment and iterate to create interfaces that are both functional and enjoyable to use. With careful planning and design, VR interfaces can enhance the immersive experience of virtual reality and make it more accessible to a wider range of users.

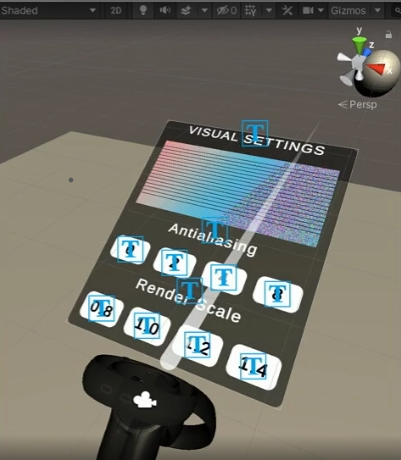
3.3 Best Practices of UI in VR



**Figure 3.4: User Interface in Virtual Reality**

Here are **some specific rules** and best practices to keep in mind when designing UI/UX for VR in Unity:

1. **Keep it aligned with the 3D environment**: The UI should be integrated seamlessly into the 3D environment and designed to blend with the surrounding objects. Use the same design language and style as the 3D environment to create a cohesive experience.
2. **Design for natural interactions**: Use Unity's VR tools and features, such as hand-tracking and controller input, to create natural and intuitive interactions. Design the UI to be responsive to the user's physical movements and gestures.Figure 3.5 shows UI of VR made in Unity.



**Figure 3.5: User Interface of Virtual Reality Made in Unity**

1. **Consider the user’s field of view**: In VR, the user’s field of view is limited by the VR headset. Design the UI to be easily visible and accessible within the user's field of view, without requiring them to turn their head too much.

4. **Use appropriate sizing and spacing:** The size and spacing of UI elements should be designed to be comfortable and readable in VR. Text should be large enough to read comfortably, and buttons should be spaced far enough to avoid accidental clicks.

1. **Test in VR:** Test the UI/UX within the VR environment to ensure that it works well and is comfortable to use. Test with a variety of users to get feedback on its usability, comfort, and effectiveness.

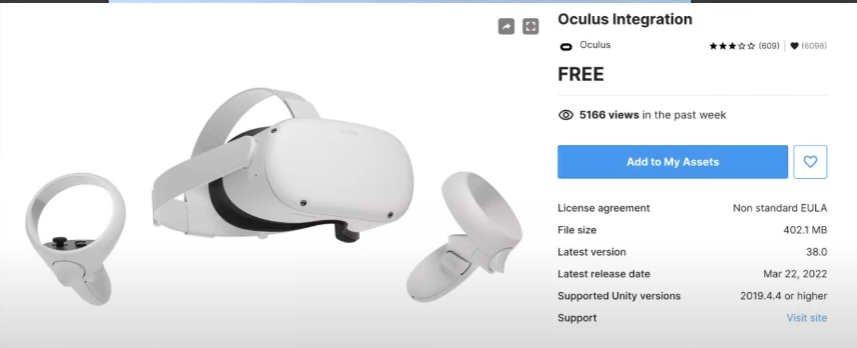
6. **Use audio and visual feedback**: Use audio and visual cues to provide feedback to the user on their actions, such as highlighting a button when it is selected or using sound effects to indicate a successful interaction.

7. **Optimize for performance:** VR applications can be demanding on system resources. Design the UI/UX to be efficient and optimized for performance to ensure a smooth and responsive experience.

By following these rules and best practices, designers can create effective and engaging UI/UX for VR in Unity that are intuitive, comfortable, and seamlessly integrated into the virtual environment.

3.4 Setting Up User Interface in Virtual Reality

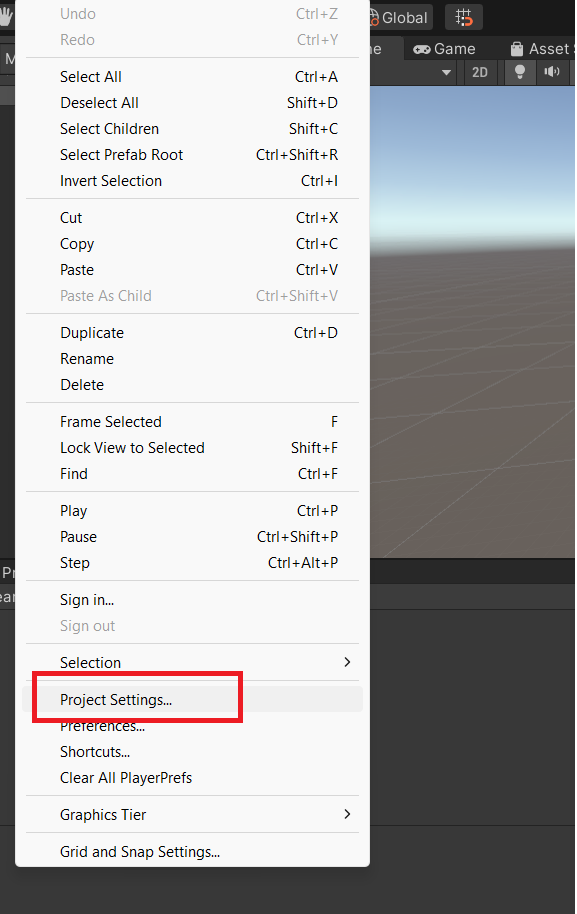
**Step 1**:We have installed all the necessary **Packages in Unity** for Making UI such as **XR Interaction Toolkit and** **Oculus Integration** Figure 3.6.



**Figure 3.6: Oculus Integration**

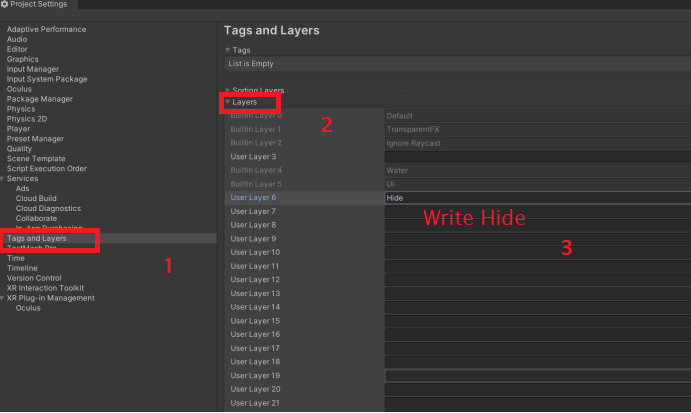
**Step 2**:After that we go to **Edit → Project Settings** as shown in

Figure 3.7.



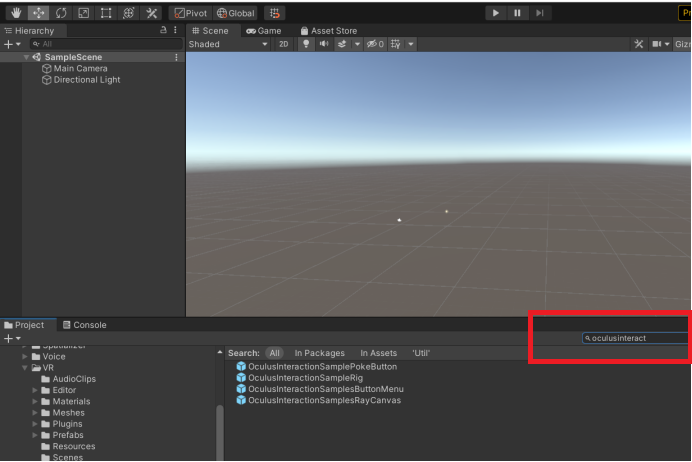
**Figure 3.7: Project Settings**

**Step 3 :** Go to **Project Setting**s and **Select Tags and Layer** as shown in Figure 3.8.

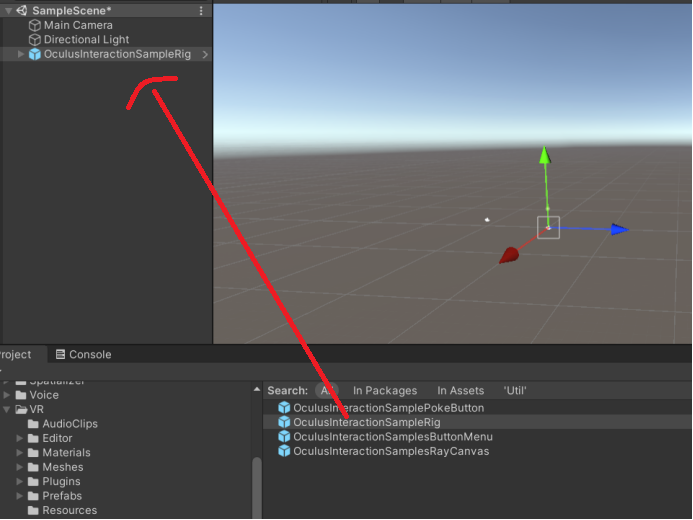
**Figure 3.8: Tags and Layer Panel**

**Step 4**:Go to the **Arrow with Layer Button** as shown inFigure 3.8.

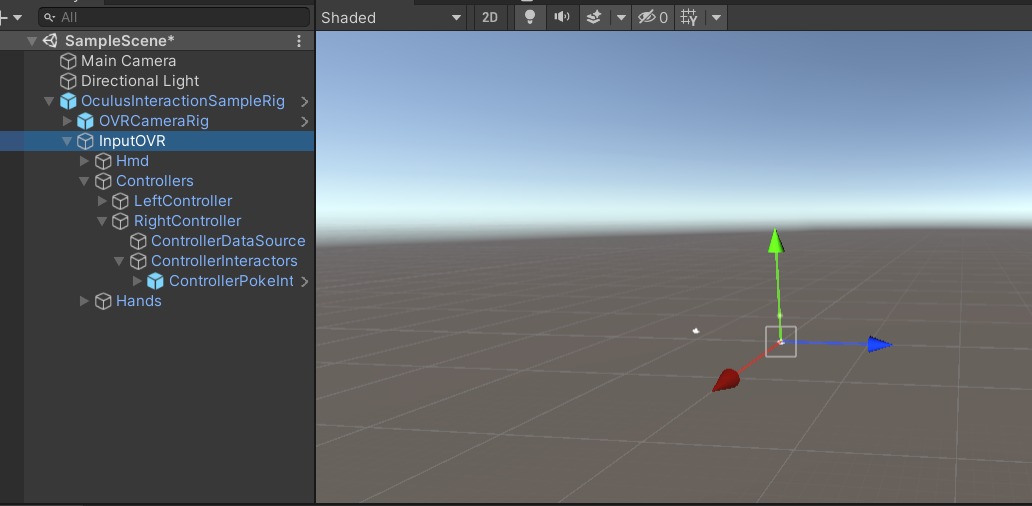
**Step 5**:Go to **Project → Search** and search for **OculusinteractionSampleRig** as shown inFigure 3.9.



**Figure 3.9: Search OculusinteractionSampleRig**

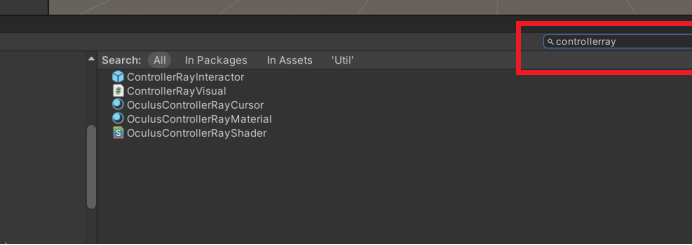
**Step 6**: Drag **OculusInteractinSampleRig** to the Hierarchy Panel as shown in Figure 3.10.

**Figure 3.10: Oculus IntegrationSampleRig is Dragged to Hierarchy Panel**

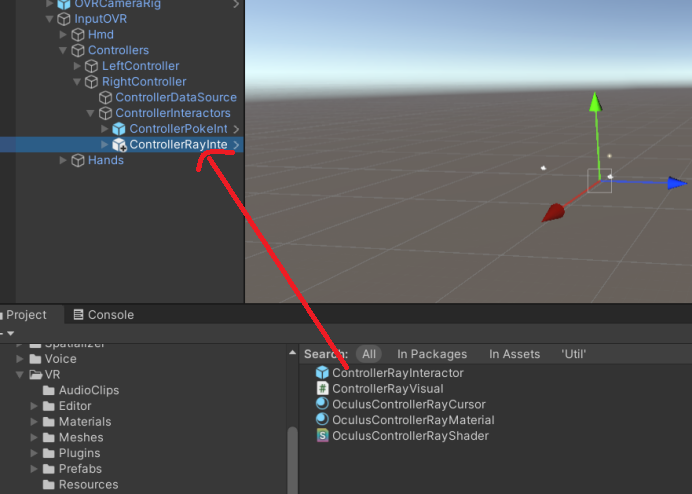
**Step 7**:Click arrow beside **InputOVR Controller → RightController → ControllerInteractors → ControllerPokeInt** in Hierarchy Panel as shown inFigure 3.11.

**Figure 3.11: ControllerPokeInt in Hierarchy Panel**

**Step 8**: Search for **ControllerRayInteractor** as shown in Figure 3.12.

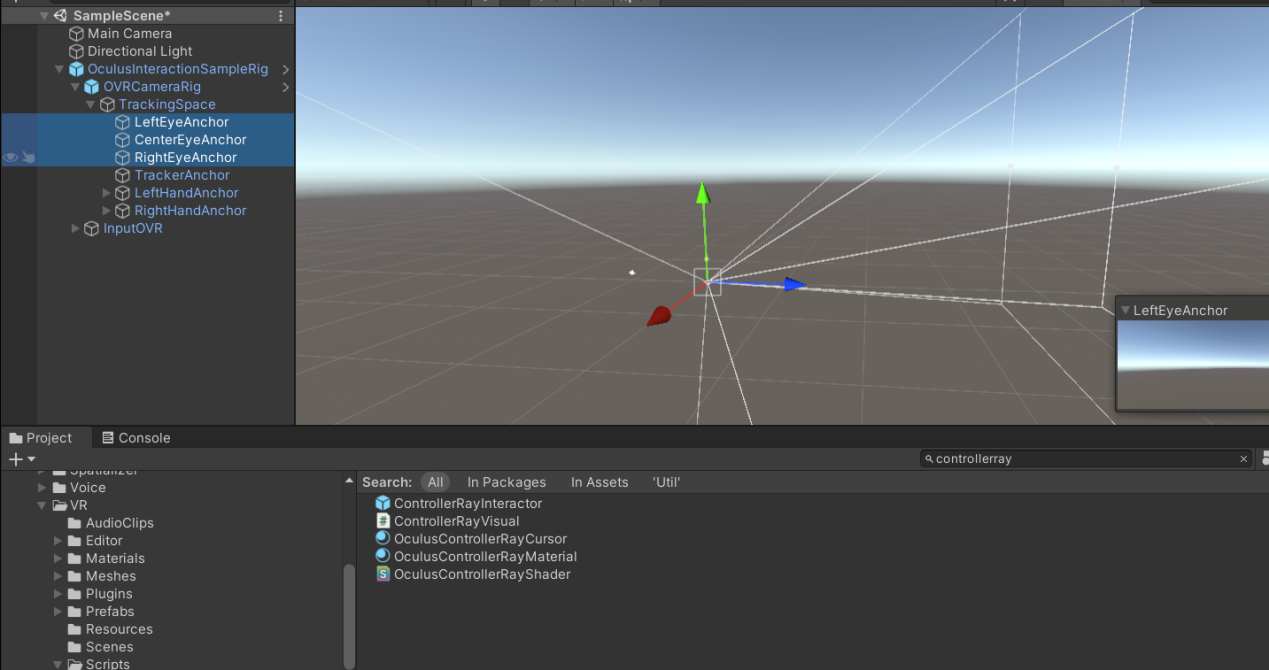
**Figure 3.12: ControllerRayInteractor**

**Step 9**: Drag it under **ControllerInteractor** as a child of **ControllerIntearactor GameObject** as showninFigure 3.13.

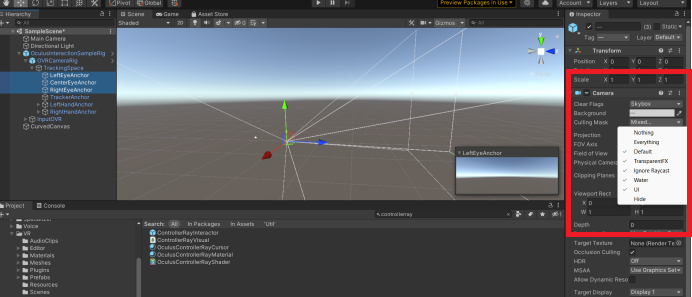


**Figure 3.13: ControllerRayIntearactor Dragged to Hierarchy**

**Step 10**:Now, go to **OVRCameraRig** and then **TrackingSpace** under this select **LeftEyeAnchor**, **CenterEyeAnchor**, **RightEyeAnchor** as shown inFigure 3.14.

**Figure 3.14: Select All Three Components**

**Step 11**: Then go to **Culling Masks settings** of the and select Hide this will remove the hide from culling masking as seen **in Figure 3.15.**



**Figure 3.15: Hide Culling Masks Settings**

3.5 Culling Masks Settings

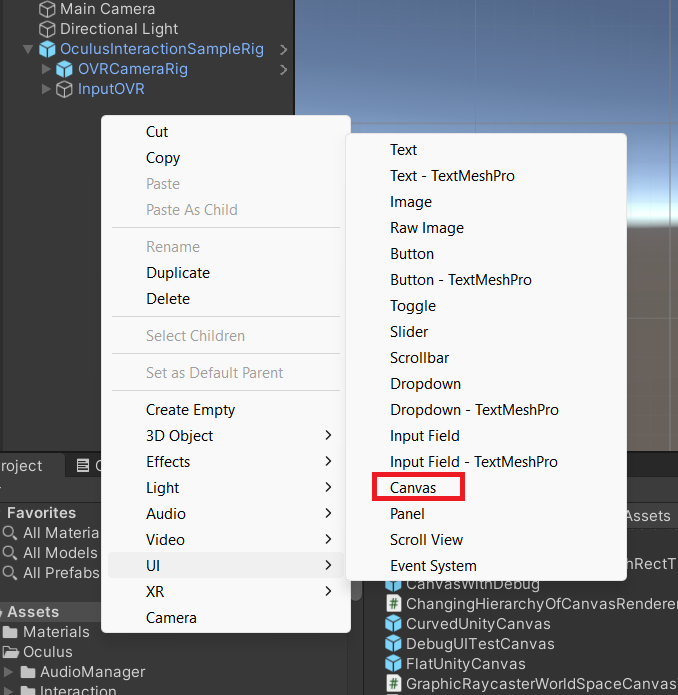
Culling Masks are a set of settings in Unity that control which layers of objects are rendered by a Camera in a Virtual Reality (VR) environment.

In Unity, objects can be assigned to different layers, and each Camera can be set to render only certain layers. This allows for better performance and optimization in complex scenes, where certain objects may not need to be rendered at all times.

In a VR context, culling masks can be particularly useful for improving performance and reducing latency, since VR headsets require high frame rates to prevent motion sickness and ensure a smooth user experience. By carefully controlling which objects are rendered by the camera in each eye of the VR headset, Unity can reduce the number of objects that need to be processed and rendered, resulting in a faster and more efficient VR application.

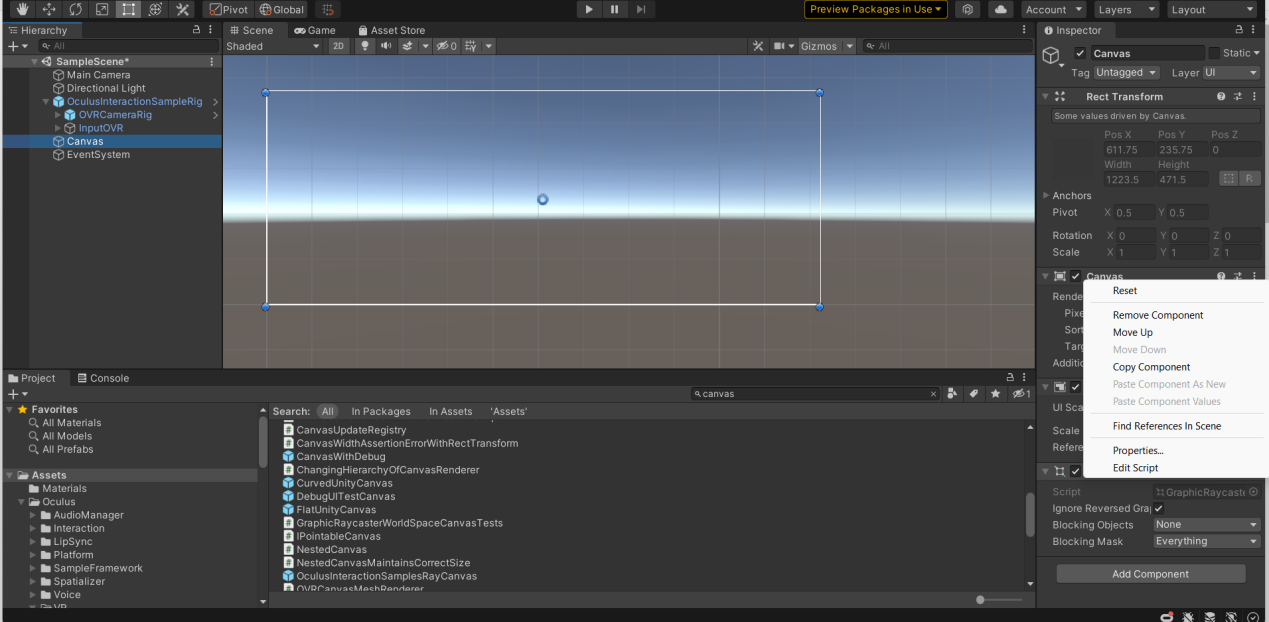
**Step 12**: To create a Canvas, right-click on **Hierarchy Panel**.

**Step 13**:Select **UI → Canvas**, this will add a Canvas element where our button will be present in **VR** as shown in Figure 3.16.



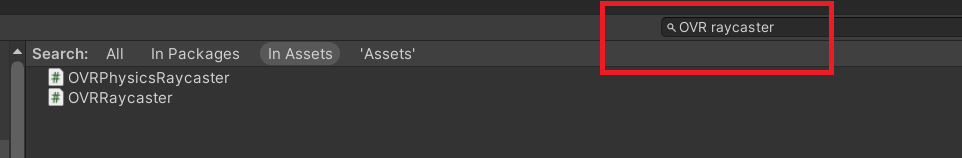
**Figure 3.16: Adding Canvas**

**Step 14**: After adding Canvas, remove **Graphic Ray Caster** Component as shown in Figure 3.17.



**Figure 3.17: Remove Graphic Ray Caster Component**

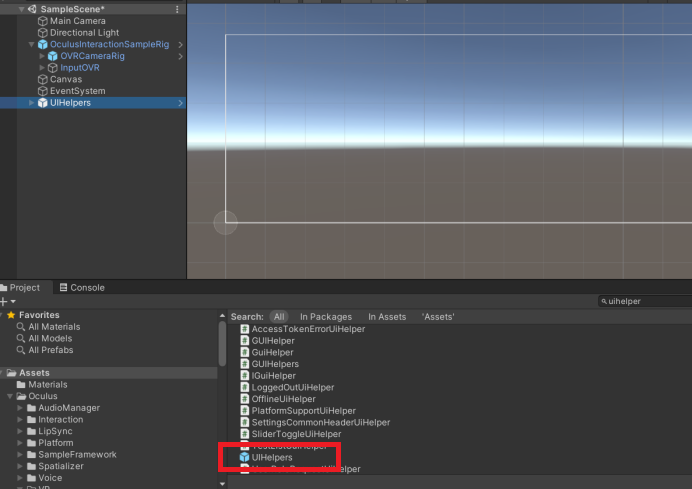
**Step 15**: Go to add component of canvas and select **Add Component → Search OVR RayCaster** and click as shown in Figure 3.18.



**Figure 3.18: Search OVR RayCaster**

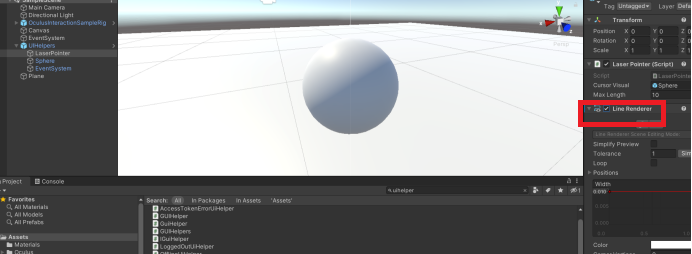
**Step 16**:Search for **UI Helper** in Project Panel and drag it to Hierarchy Panel.

**Step 17**: Click the arrow beside UIHelper and select Sphere as shown inFigure 3.19 and Figure 3.20.

**Figure 3.19: UIHelper Component**

**Figure 3.20: Select Sphere Under UIHelper**

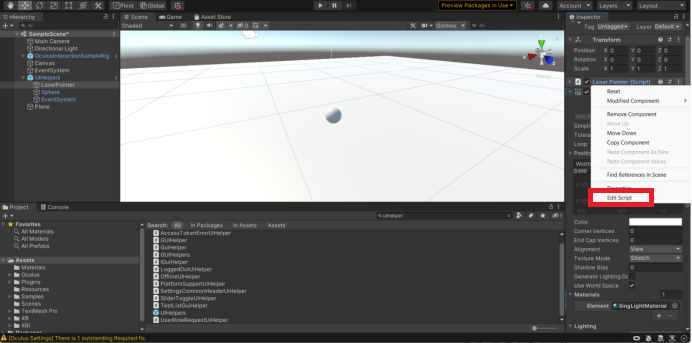
**Step 18**: Select the **line renderer** and **Click on Edit Script** and **open Script** Figure 3.21 and Figure 3.22.



**Figure 3.21: Line Renderer**

**Step 19**:Change the Script as follows -

Before private LaserBeamBehavior \_laserBeamBehavior; add [SerializeField] this will add the LaserBeamBehaviour function to the **Edit Panel** as shown inFigure 3.22 and Code Snippet 1.



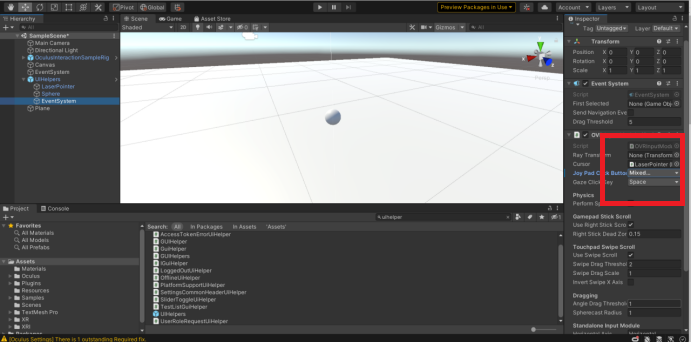
**Figure 3.22: Edit Script Option**

**Code Snippet 1:**

Shows to Serialized Field the LaserBeambehaviour.

|  |
| --- |
| using UnityEngine;  using System.Collections;  using UnityEngine.EventSystems;  using UnityEngine.UI;  using System;  public class LaserPointer : OVRCursor  {  public enum LaserBeamBehavior  {  On, // laser beam always on  Off, // laser beam always off  OnWhenHitTarget, // laser beam only activates when hit valid target  }  public GameObject cursorVisual;  public float maxLength = 10.0f;  **[SerializeField**  **private LaserBeamBehavior \_laserBeamBehavior;**  bool m\_restoreOnInputAcquired = false;  public LaserBeamBehavior laserBeamBehavior  {  set  {  \_laserBeamBehavior = value;  if (laserBeamBehavior == LaserBeamBehavior.Off || laserBeamBehavior == LaserBeamBehavior.OnWhenHitTarget)  {  lineRenderer.enabled = false;  }  else  {  lineRenderer.enabled = true;  }  }  get  {  return \_laserBeamBehavior;  }  }  private Vector3 \_startPoint;  private Vector3 \_forward;  private Vector3 \_endPoint;  private bool \_hitTarget;  private LineRenderer lineRenderer;  private void Awake()  {  lineRenderer = GetComponent<LineRenderer>();  }  private void Start()  {  if (cursorVisual) cursorVisual.SetActive(false);  OVRManager.InputFocusAcquired += OnInputFocusAcquired;  OVRManager.InputFocusLost += OnInputFocusLost;  }  public override void SetCursorStartDest(Vector3 start, Vector3 dest, Vector3 normal)  {  \_startPoint = start;  \_endPoint = dest;  \_hitTarget = true;  }  public override void SetCursorRay(Transform t)  {  \_startPoint = t.position;  \_forward = t.forward;  \_hitTarget = false;  }  private void LateUpdate()  {  lineRenderer.SetPosition(0, \_startPoint);  if (\_hitTarget)  {  lineRenderer.SetPosition(1, \_endPoint);  UpdateLaserBeam(\_startPoint, \_endPoint);  if (cursorVisual)  {  cursorVisual.transform.position = \_endPoint;  cursorVisual.SetActive(true);  }  }  else  {  UpdateLaserBeam(\_startPoint, \_startPoint + maxLength \* \_forward);  lineRenderer.SetPosition(1, \_startPoint + maxLength \* \_forward);  if (cursorVisual) cursorVisual.SetActive(false);  }  }  // make laser beam a behavior with a prop that enables or disables  private void UpdateLaserBeam(Vector3 start, Vector3 end)  {  if (laserBeamBehavior == LaserBeamBehavior.Off)  {  return;  }  else if (laserBeamBehavior == LaserBeamBehavior.On)  {  lineRenderer.SetPosition(0, start);  lineRenderer.SetPosition(1, end);  }  else if (laserBeamBehavior == LaserBeamBehavior.OnWhenHitTarget)  {  if (\_hitTarget)  {  if (!lineRenderer.enabled)  {  lineRenderer.enabled = true;  lineRenderer.SetPosition(0, start);  lineRenderer.SetPosition(1, end);  }  }  else  {  if (lineRenderer.enabled)  {  lineRenderer.enabled = false;  }  }  }  }  void OnDisable()  {  if (cursorVisual) cursorVisual.SetActive(false);  }  public void OnInputFocusLost()  {  if (gameObject && gameObject.activeInHierarchy)  {  m\_restoreOnInputAcquired = true;  gameObject.SetActive(false);  }  }  public void OnInputFocusAcquired()  {  if (m\_restoreOnInputAcquired && gameObject)  {  m\_restoreOnInputAcquired = false;  gameObject.SetActive(true);  }  }  private void OnDestroy()  {  OVRManager.InputFocusAcquired -= OnInputFocusAcquired;  OVRManager.InputFocusLost -= OnInputFocusLost;  }  } |

**Step 20 :** Go to Event System Edit Panel and **Change Joy Pad Edit Button to Secondary Index Button** Figure 3.23.

**Figure 3.23:Change Joy Pad**

**Step 21**:Go to Canvas in Hierarchy Panel and Change **Render Mode to World Space** as shown in Figure 3.24.

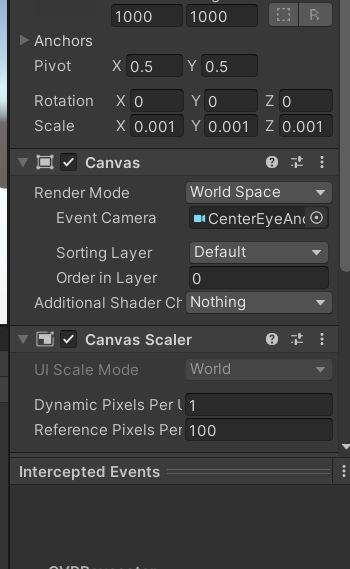
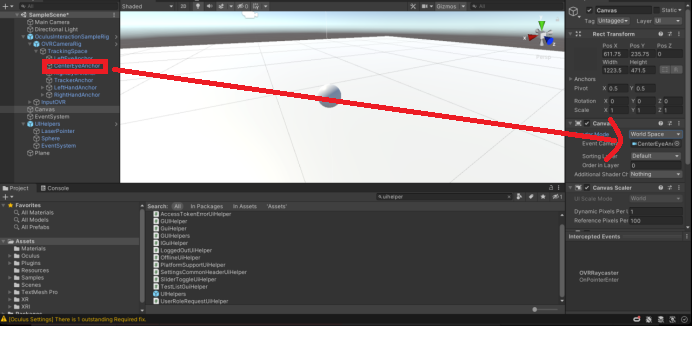


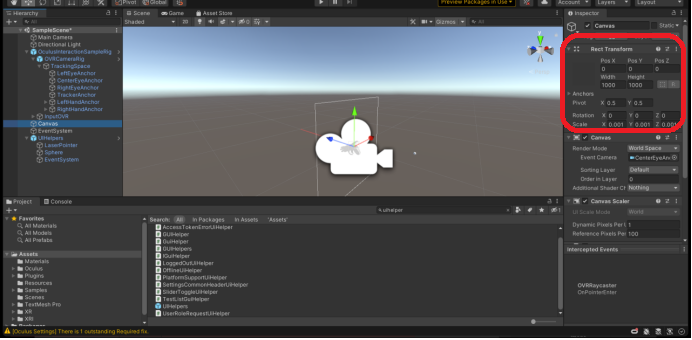
Figure 3.24: Render Mode to World Spa**ce**

**Step 22**: Go to **OVRPlayerController → OVRCameraRig → TrackingSpace → Drag CenterEyeAnchor** from Hierarchy Panel to Event Camera in **Edit Panel** as shown in Figure 3.25.



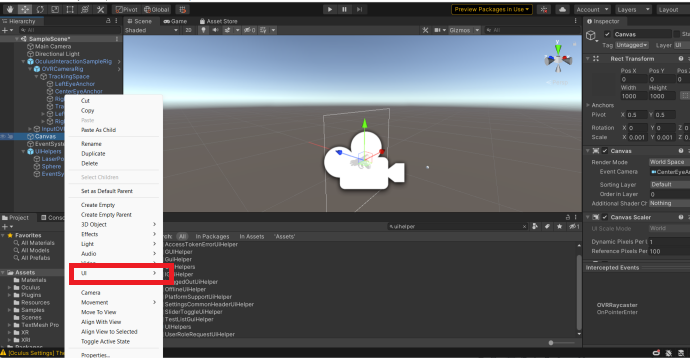
**Figure 3.25: Drag CenterEyeAnchor to Hierarchy Panel**

**Step 23**: Adjust **Canvas** Size to following as shown in Figure 3.26.

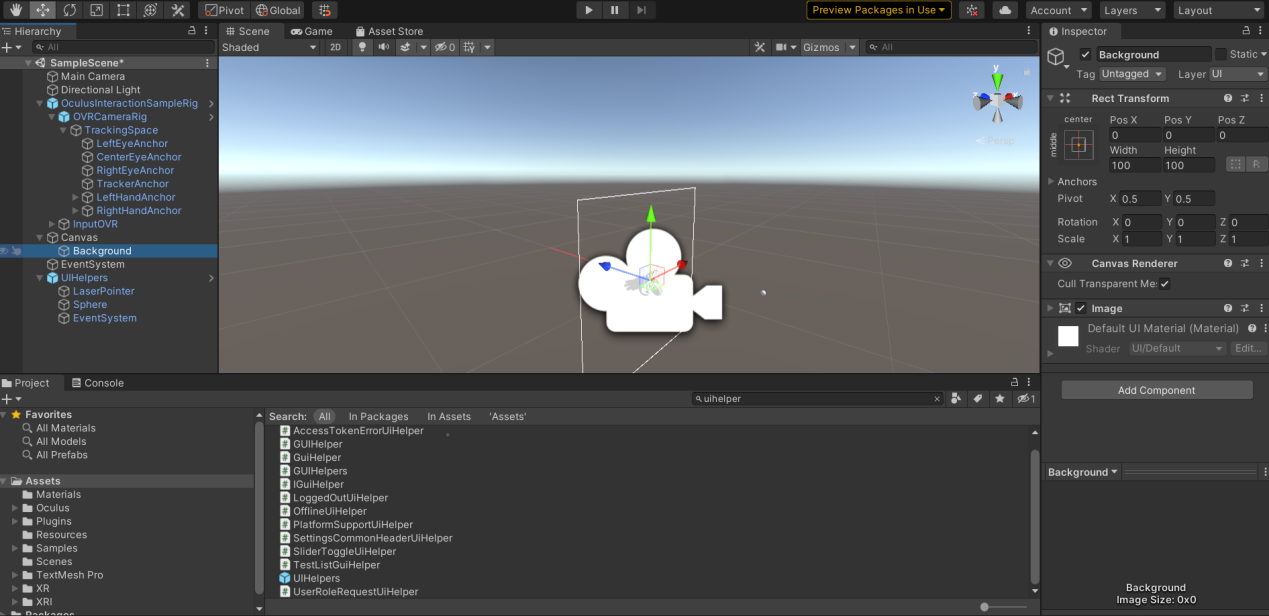


**Figure 3.26: Add Canvas and Change Size**

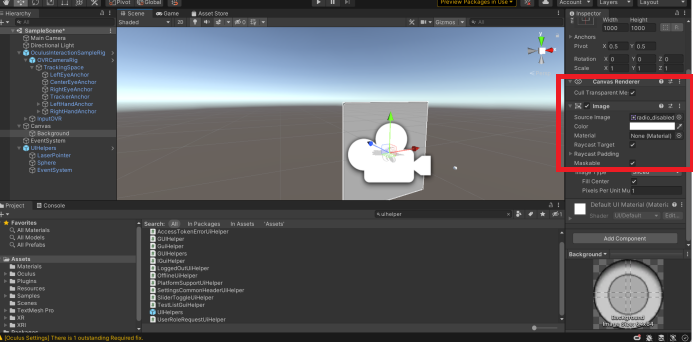
**Step 24**:Add image to canvas by right clicking on it and select **UI → canvas** as shown inFigure 3.27.

**Figure 3.27: User Interface Settings**

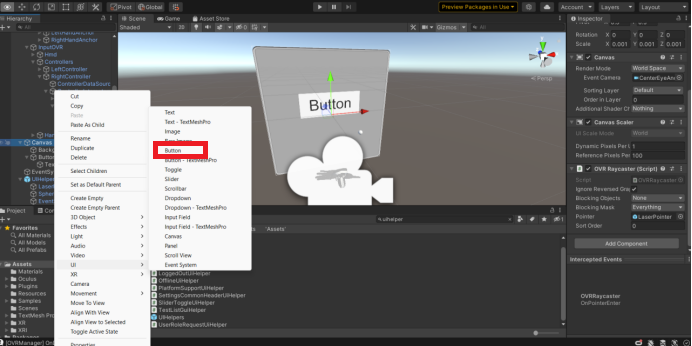
**Step 25**:Rename the image to **Background** as shown in Figure 3.28.

**Figure 3.28: Background Image Change**

**Step 26**: Add a Background to the image by clicking **Image → Source Image** and select the image of your choiceas shown in Figure 3.29.

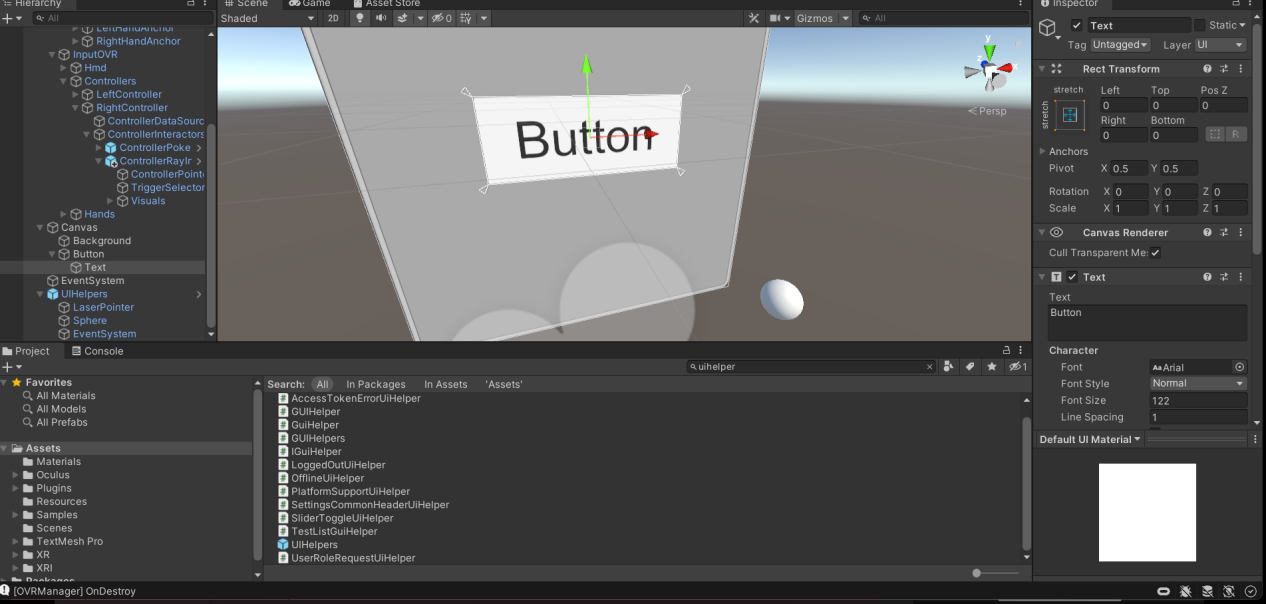
**Figure 3.29: Changing Source Image**

**Step 27**:Right-click **Canvas → UI → Button** as shown in Figure 3.30.



**Figure 3.30: Adding Button**

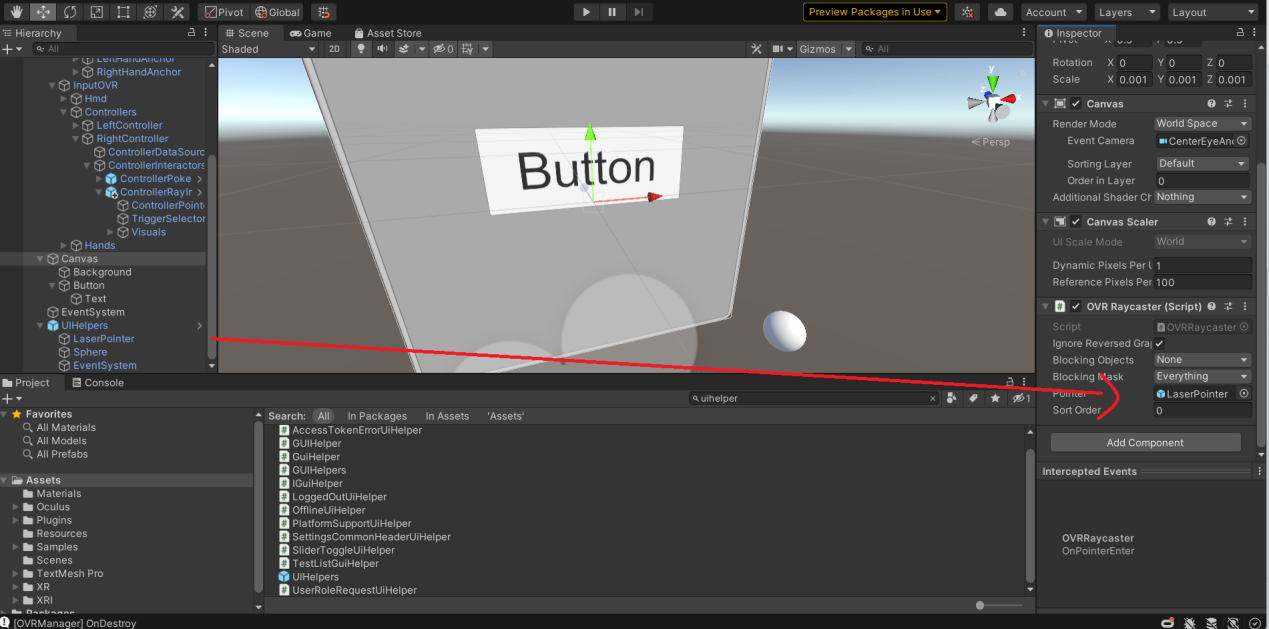
**Step 28**: Change the **Button Size** as shown in Figure 3.31.



**Figure 3.31: Button Added and Size Changed**

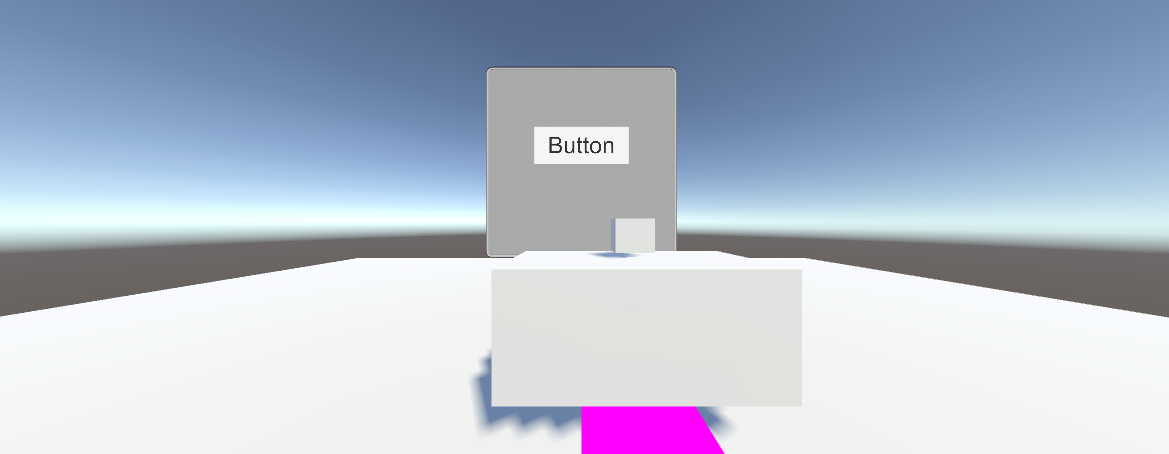
**Step 29**:Go to **Canvas → and Add LaserPointer** GameObject to the **OVR RayCaster** Component in **Edit Panel of Canvas** Figure 3.32, Drag Laserpointer to **Canvas → OVR RayPointer** as shown

in Fig 3.33.

****

**Figure 3.32 : Laser Pointer to Pointer Settings**

**Step 30**: The scene is ready to test now Figure 3.33.



**Figure 3.33 : Final Output**

Note : make Sure that all have access to OculusVR/Meta quest hardware and controllers

# 3.6 Summary

* Create a new Canvas GameObject by right-clicking in the Hierarchy window and selecting **UI → Canvas.**
* Set the Render Mode of the Canvas component to **World Space.**
* Add UI elements to the canvas by right-clicking in the Hierarchy window and selecting **UI *Working with VR Player Character* Text, UI → Image**, or any other UI element.
* Position the UI elements within the **VR environment by adjusting** their transform properties.
* Attach the canvas to the appropriate **VR camera.**
* If required, add scripts to the **UI elements** to control their behavior and interaction with the user.
* **Test the VR projec**t with the UI to ensure everything is working correctly.

# 3.7 Check Your Progress



1. The Sphere Component appears under which component in Hierarchy Panel?
2. OVRCameraRig
3. OVR Ray Caster
4. UI Helper
5. All of these
6. Which component(s) must be selected under OVRCameraRig ?
7. LeftEyeAnchor

b) CenterEyeAnchor

c) RightEyeAnchor

d) All of the above

1. Which is the right way to create a Canvas in Unity?
2. Right-click → UI → canvas
3. Right-click → canvas
4. Settings → canvas
5. Left-click → canvas
6. Which Render Mode is preferred in VR?
7. World
8. Space
9. Mixed
10. All of these
11. Which component acts as a central camera in VR view?
12. LeftEyeAnchor
13. RightEyeAnchor
14. CentreEyeAnchor
15. All of these

# 3.8.1 Answers

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

3.9 Try It Yourself

⮚ Add Controller Ray Interactor

⮚ Add OVRPlayerController and OVRCameraRig and make a simple VR Scene.

⮚ Test the VR Scene in Oculus HMD.