Unity - basic setup

Tutorial for setting up Unity for VR using the XR Interaction Toolkit.

- Getting started with Unity and VR
- Setting the scene
- Basic Teleportation
- Customizing controls
- UI events
- Finishing touches

Getting started with Unity and VR

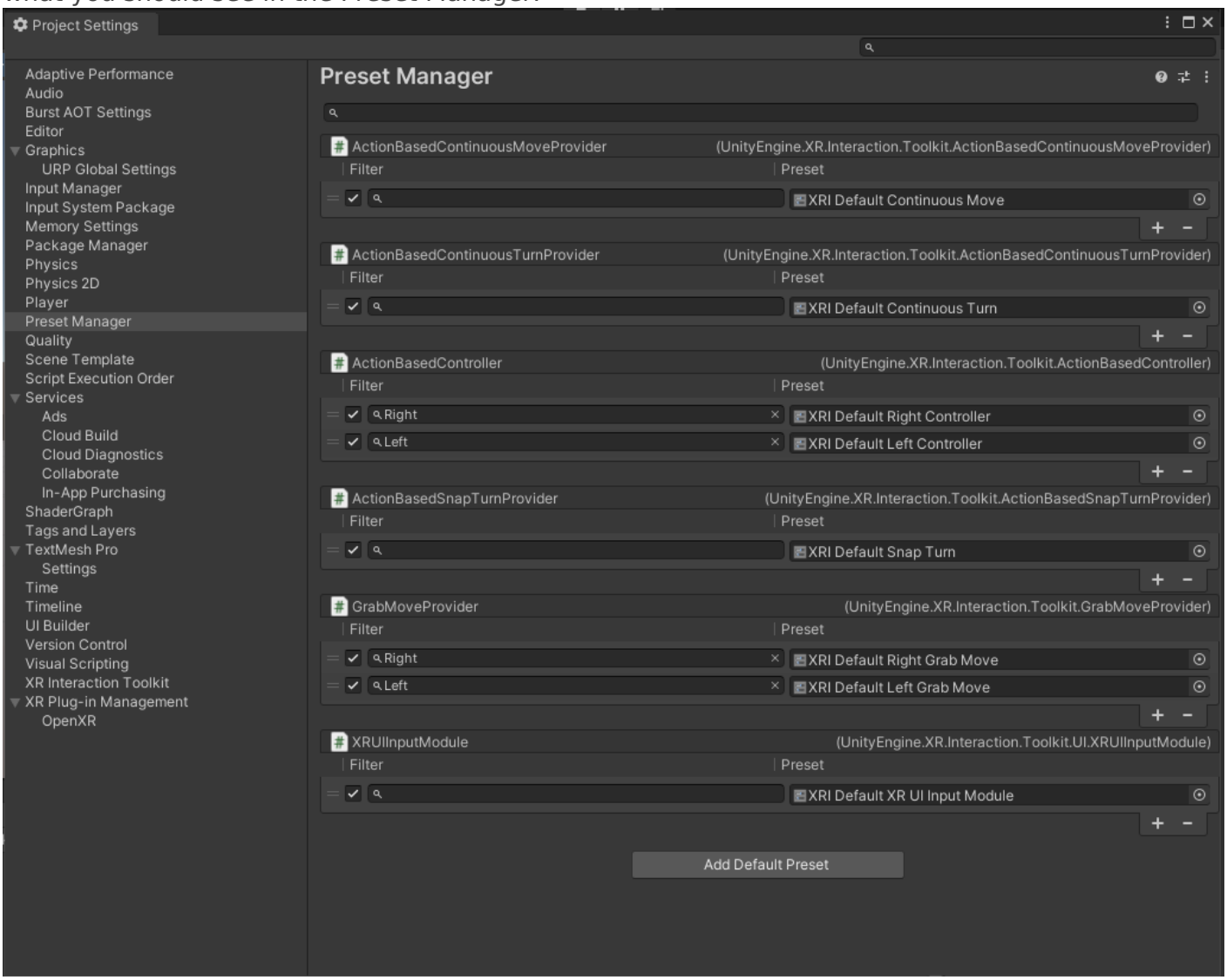
Before you get started, make sure that your VR headset and controllers are properly connected to your workstation. For additional tips on how to get started with the HTC Vive headset, consult the [hardware guide].

In the SteamVR settings, make sure that the **Current OpenXR Runtime** is set to SteamVR instead of Oculus, especially if a VR headset other than the HTC Vive was used on your current computer.

Create a new Unity project using the latest official version. It is beneficial at this stage to check for an updated version, but the rest of this guide will assume you're working with version `2021.3.19f1`. Select the **Universal Render Pipeline** as the project type. This format provides the highest graphics fidelity.

Create a new empty scene in your project. Go to Window>Package Manager, choose Packages:Unity Registry from the drop-down menu, then search for **XR Interaction Toolkit** and install the package. The version used in this guide is `2.2.0`. From the samples in the XR Interaction Toolkit, import the **Starter Assets**. Install the XR Plugin Manager as well.

Navigate to Edit>Project Settings>Preset Manager. In the Project file explorer, navigate to Assets > Samples > XR Interaction Toolkit > 2.2.0 > Starter Assets, open a preset in the inspector, then click Add to *ScriptName* Default. You should see the preset appear in the preset manager. Repeat this step for all the other presets available in the Starter Assets folder. Make sure that you label the presets for the Left and Right controllers using the filter. If everything is set up correctly, this is what you should see in the Preset Manager:



Navigate to XR Plug-in Management from the Project Settings. Under Plug-in Providers, check the Open XR box, and wait for the scripts to reload. An Open XR tab should appear under Plug-in Management, where you should add the HTC Vive controller profile to the list of interaction profiles. In the OpenXR tab, also make sure that the **Play Mode OpenXR Runtime** is set to SteamVR.

You' re now ready to start building your VR scene in Unity!

Setting the scene

Now that all the project settings have been updated to work with our VR setup, it's time to build our VR scene. First, create a plane game object at coordinates `0, 0, 0` to serve as the floor. Then, create an **XR Origin (VR)** object. This object contains a camera offset, which defines the height of the camera corresponding to the headset in space, the main camera object, and the two controllers. If you run the game now, you' ll notice that your head movements are tracked, but you cannot see the controllers. That's because we haven' t assigned them an asset yet!

To fix that, you can import the `vr_controller_vive_1_5` asset from the VR Starter Template, simply by dragging and dropping the asset from the asset folder to the controller objects. Rotate the assets (not the parent controllers!) by 180 degrees around the Y-axis to make sure that they' re matching their real-world orientation. To make the controllers appear more realistic, we can create a new material and set the Base Map to black or dark gray. Then, we drag this new material onto the lowest-level controller objects, which should be called whole_model_group1.

For each controller, you will also need to add the input handling preset. To do so, click on the slider icon in the top-right corner of the **XR Controller (Action-based)** dropdown, and select the XRI Default Left/Right Controller preset, depending on which controller you've selected.

Each controller should have an **XR ray interactor**, which allows us to use the controller buttons to interact with game objects and UI components. To test this, add a Sphere or Cube game object to the scene, and assign it a Rigidbody and an XR Grab Interactable component. When running the game, you should be able to grab the object by using the controller's trigger.

Basic Teleportation

Teleportation allows you to instantly move from a location to another in your VR environment. To get a teleportation system up and running, you first need to add a **Locomotion System** and a **Teleportation Provider** to your XR Origin object. Make sure that the XR Origin is assigned to the Locomotion System, and that the Locomotion System is assigned to the Teleportation Provider

From the Unity main menu, click GameObject > XR > Teleportation Area or GameObject > XR > Teleportation Anchor to create a plane on which teleportation is possible. On a Teleportation Area, you teleport to your pointed target on the plane's child collider, whereas a Teleportation Anchor specifies a pre-determined position and/or rotation in addition to the Teleportation Area. You can also add a Teleportation Area component to the Plane object we've previously set up as our floor.

You should now have a basic scene with the ability to teleport using your controllers. By default, teleportation is activated by the *grip* buttons on the HTC Vive controllers. The default ray interactors will turn from red to white if you're pointing at an area where you can teleport.

Customizing controls

In order to make your game or simulation more intuitive or accessible, you can modify the controller bindings inside your Unity project. Here, we will show how to change the teleportation controls from the default grip buttons to the trigger.

From the Assets folder, click on **Samples > Interaction Toolkit > 2.2.0 > Starter Assets**, then open the XRI Default Input Actions Import Settings. Select the XRI LeftHand Interaction menu, in the drop-down properties of Select you should see gripPressed [LeftHand XR Controller]. Change that path to triggerPressed [LeftHand XR Controller]. Repeat this step for Select Value, changing the path from grip [LeftHand XR Controller] to trigger [LeftHand XR Controller].
To avoid having two actions mapped to the same button input, you can also change Activate to gripPressed, and Activate Value to grip.

Go back to your LeftHand Controller in your scene, and under the XR Ray Interactor properties, check the box "allow Hovered activate". This way, you won't have to grip the controller to activate objects you're pointing at.

For the right controller, you can keep the default input actions. To make the right controller only interact with UI elements, the XR Ray Interactor can be modified. Change the Raycast Mask from Everything/Default to UI, and the right ray will only be able to interact with UI elements. This can be useful if you want to avoid accidental teleportation when pressing a UI button, for example.

UI events

To interact with Unity's built-in UI elements, you need to perform extra steps, particularly if you're dealing with 3D-tracked devices. The XR Interaction Toolkit package provides a number of new components that you can use to convert an XR controller to work seamlessly with the UI, as well as helper menu options that handle basic configuration settings.

Before adding a button, a slider, or any other UI element, click on GameObject > UI > Canvas to generate a Canvas which will contain all UI elements, as well as an EventSystem component. In the Canvas properties, change the Render Mode to **World Space**, and set the Event Camera to the Main Camera. Then, use the Scale property to make the Canvas fit your scene.

> **Important:** If you directly modify the Width and Height of your Canvas instead of scaling it, the Canvas might not work as intended. This also applies to Buttons and other UI elements.

The Event System component acts as a central dispatch for UI events to process input, and update individual active canvases. Additionally, each Event System needs an Input Module to process input. The EventSystem component might have a Standalone Input Module or an Input System UI Module, which will prevent proper input processing. Remove the component, and add an **XR UI Input Module** to the event system.

Finally, add a Tracked Device Graphic Raycaster component to the Canvas, and set the Raycast Trigger Interaction to "Collide". This will allow your Raycast to actually hit elements on the Canvas instead of passing through them.

To test your setup, add a Button object as a child of the Canvas. If you select the button by pressing the trigger on your Right controller, the button should change color. Now, you can add your own script to the Button object, and change the `On Click ()` property to run that script when the button is pressed.

Finishing touches

Now that all the basic elements are in place, you can add some features to the scene to make it more user-friendly.

First of all, change the color of the left and right controller object to easily tell them apart. To do this, click on the `whole_model_group1` object, then on Material > Surface Inputs. Change the Base Map to a different color for each controller. In the Demo Project, you can see that the left controller is blue, and the right controller is red.

Now you can improve the looks of teleportation and raycasting. In the XR Ray Interactor properties, under Raycast Configuration, change the Line Type to Projectile Curve. You can also change the ray color: in the template, the Invalid Color Gradient is set to transparent. This makes the ray invisible, until it is pointed at a valid target.

A teleportation reticle can also be added to the ray. To prototype this, create a new cylinder, rename it Teleport Reticle, remove the collider, scale it down to 0.5 and lower the Y axis to create a disk shape. Create a new material called Reticle Base, set the shader to URP/unlit, set the surface type to transparent, choose a color, and decrease the alpha value to about 70. Drag the new material onto the Teleport Reticle object and assign the Reticle to the Reticle property of the Line Visual component. You can see a transparent reticle at the end of our ray which appears when hovering over an eligible object and provides some additional feedback to the player on their teleport location.

To make this element more interesting, you can use a ShaderGraph instead of a solid color. Right click in the Project folder and select Create > Shader Graph > URP > Unlit Shader Graph, then open the new shader in the shader editor. In the Graph Inspector, change the Surface Type to Transparent and the Render Face to Both. Now let's create two colors in the left-hand panel for the shader effect. Switch the Graph Inspector to the Node Settings window to view the properties for each color. Make the TopColor fully transparent, and the BottomColor your preferred shade. To create a gradient based on the height of the model, right click in the open space and select Create Node then search for Position. Update the Space value to Object. Create another node for Split. Connect the nodes by dragging the node marker from the output of Space to the input of Split. Create a Lerp node, then connect BottomColor to A, TopColor to B, and the output of Split to T. Next drag the Lerp Output to the “Base Color” Fragment. Create a new Split node and drag in the Lerp output, then drag the alpha channel from the split to the alpha on the Fragment. This will allow you to separately control the alpha values. The Shader setup is now complete. Click on Save Asset and return to the Scene. Right click on the Shader and select Create -> Material. This will create a new material with the shader already applied to it. Next apply the material to our teleportation reticle cylinder. Once applied, choose some colors and play around with the alpha channel.