Setting up Cross-device OpenXR on Godot For Windows PCVR

Goal of that document

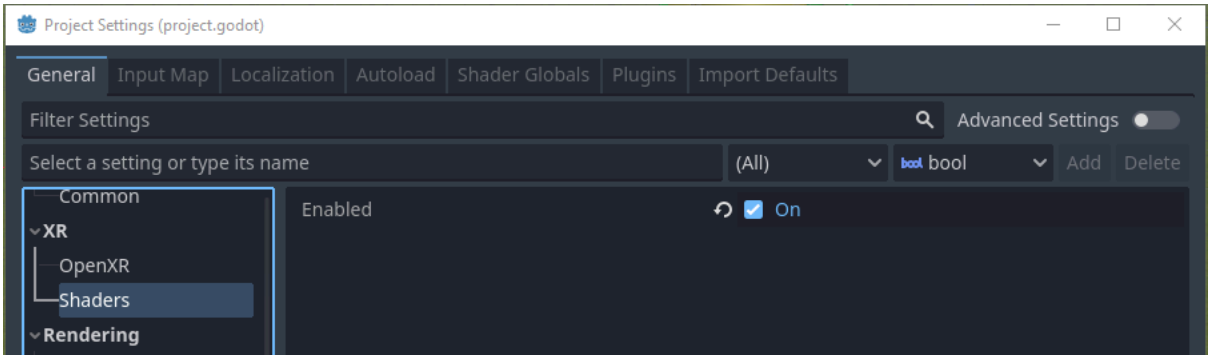
Step by step guide to create a Godot project based on OpenXR support that can build and run vendor-agnostic OpenXR application on Windows for PCVR

Create a PCVR cross-device Godot OpenXR project

Godot provides a modular XR system that abstracts many of the different XR platform specifics away from the user. At the core sits the XRServer which acts as a central interface to the XR system that allows users to discover interfaces and interact with the components of the XR system.

Prerequisites for XR in Godot 4

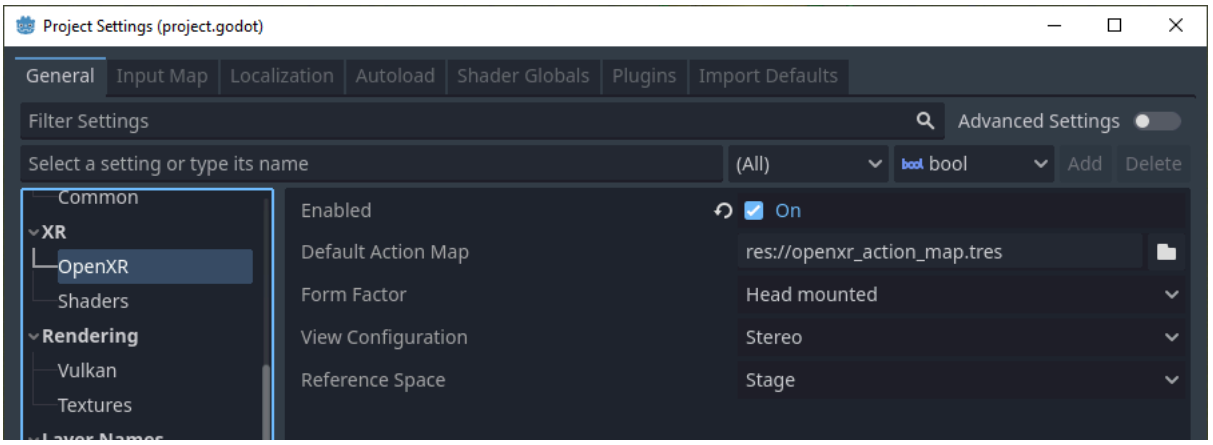
While in Godot 3 most things worked out of the box, Godot 4 needs a little more setup. This is mainly due to the more advanced nature of the Vulkan renderer. There are many rendering features in Vulkan the XR system uses that aren't enabled by default. They are very easy to turn on, simply open up your project settings and tick the XR shaders tickbox in the XR section:



OpenXR

The Vulkan implementation of OpenXR is closely integrated with Vulkan, taking over part of the Vulkan system. This requires tight integration of certain core graphics features in the Vulkan renderer which are needed before the XR system is setup. This was one of the main deciding factors to include OpenXR as a core interface.

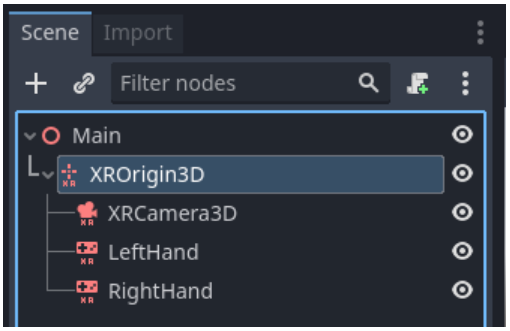
This also means OpenXR needs to be enabled when Godot starts in order to set things up correctly. The required setting can be found in your project settings:



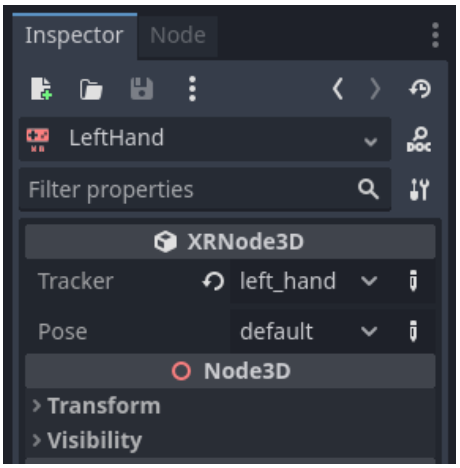
As setup is brought forward with OpenXR you can find several other settings related to OpenXR here as well. These can't be changed while your application is running. The default settings will get us started and we will go into detail in another section of the documentation.

Setting up the XR scene

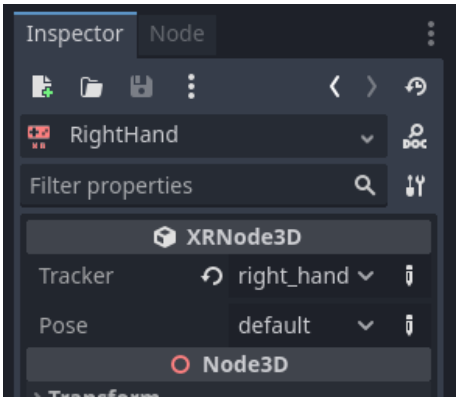
Every XR application needs at least an XROrigin3D and an XRCamera3D node. Most will have two XRController3D, one for the left hand and one for the right. Keep in mind that the camera and controller nodes should be children of the origin node. Add these nodes to a new scene and rename the controller nodes to LeftHand and RightHand, your scene should look something like this:



Next you need to configure the controllers, select the left hand and set it up as follows:

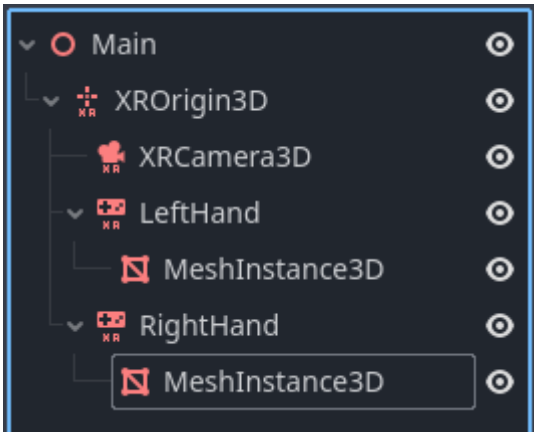


And the right hand:

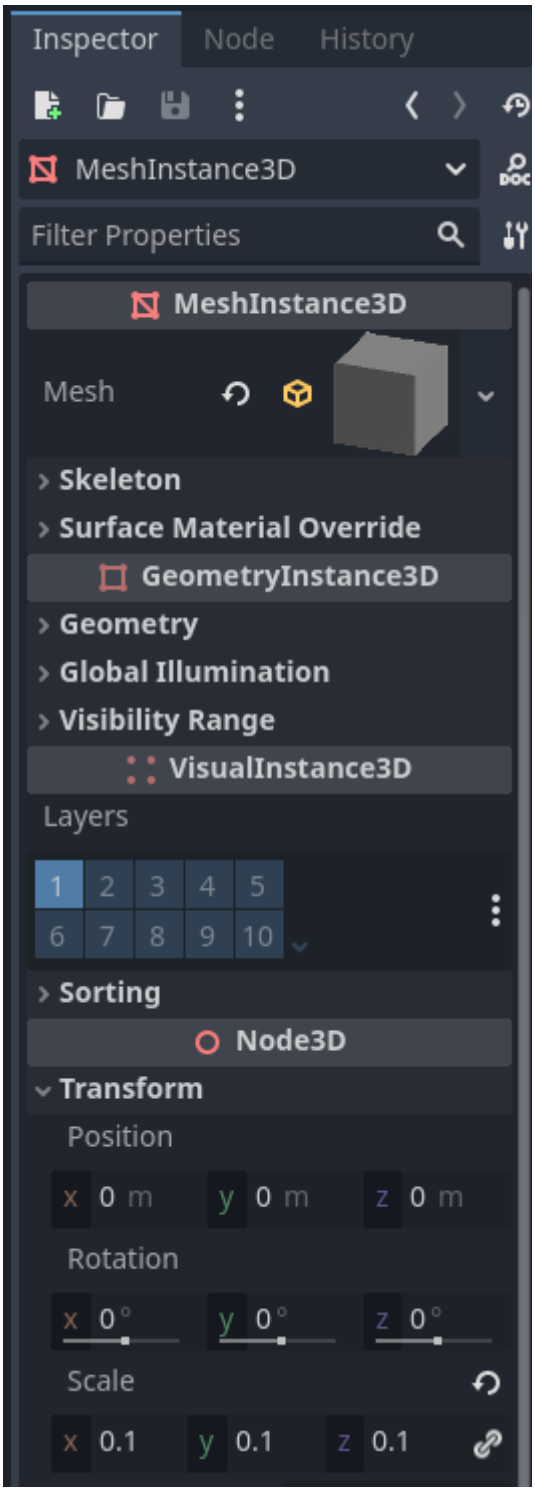


Right now all these nodes are on the floor, they will be positioned correctly in runtime. To help during development, it can be helpful to move the camera upwards so its y is set to 1.7, and move the controller nodes to -0.5, 1.0, -0.5 and 0.5, 1.0, -0.5 for respectively the left and right hand.

Next, we need to add a MeshInstance3D node under each hand node in order to be able to see our hands in XR :



And make those MeshInstance3D small boxes by selecting “New BoxMesh” in the Mesh property of the inspector. Using the Transform property of the inspector for both Meshes, set the Scale to 0.1, 0.1, 0.1.



Next we need to add a script to our root node. Add the following code into this script:

GDScript C#

```
extends Node3D

var xr_interface: XRInterface

func _ready():
    xr_interface = XRServer.find_interface("OpenXR")
    if xr_interface and xr_interface.is_initialized():
        print("OpenXR initialized successfully")

        # Turn off v-sync!
        DisplayServer.window_set_vsync_mode(DisplayServer.VSYNC_DISABLED)

        # Change our main viewport to output to the HMD
        get_viewport().use_xr = true
    else:
        print("OpenXR not initialized, please check if your headset is connected")
```

This code fragment sets xr_interface to OpenXR.

Warning

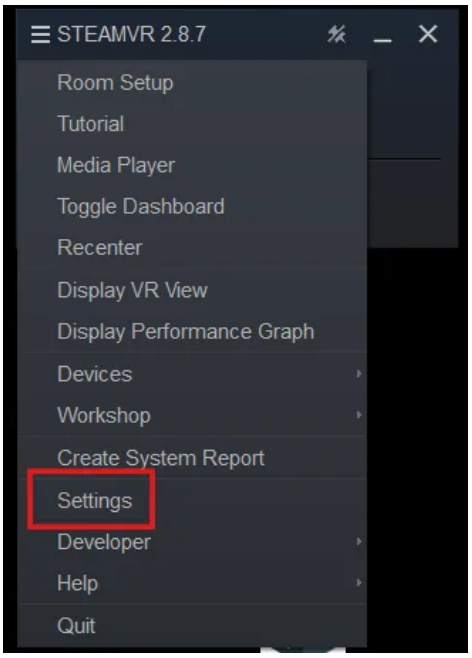
As you can see in the code snippet above, we turn off v-sync. When using OpenXR you are outputting the rendering results to an HMD that often requires us to run at 90Hz or higher. If your monitor is a 60hz monitor and v-sync is turned on, you will limit the output to 60 frames per second.

XR interfaces like OpenXR perform their own sync.

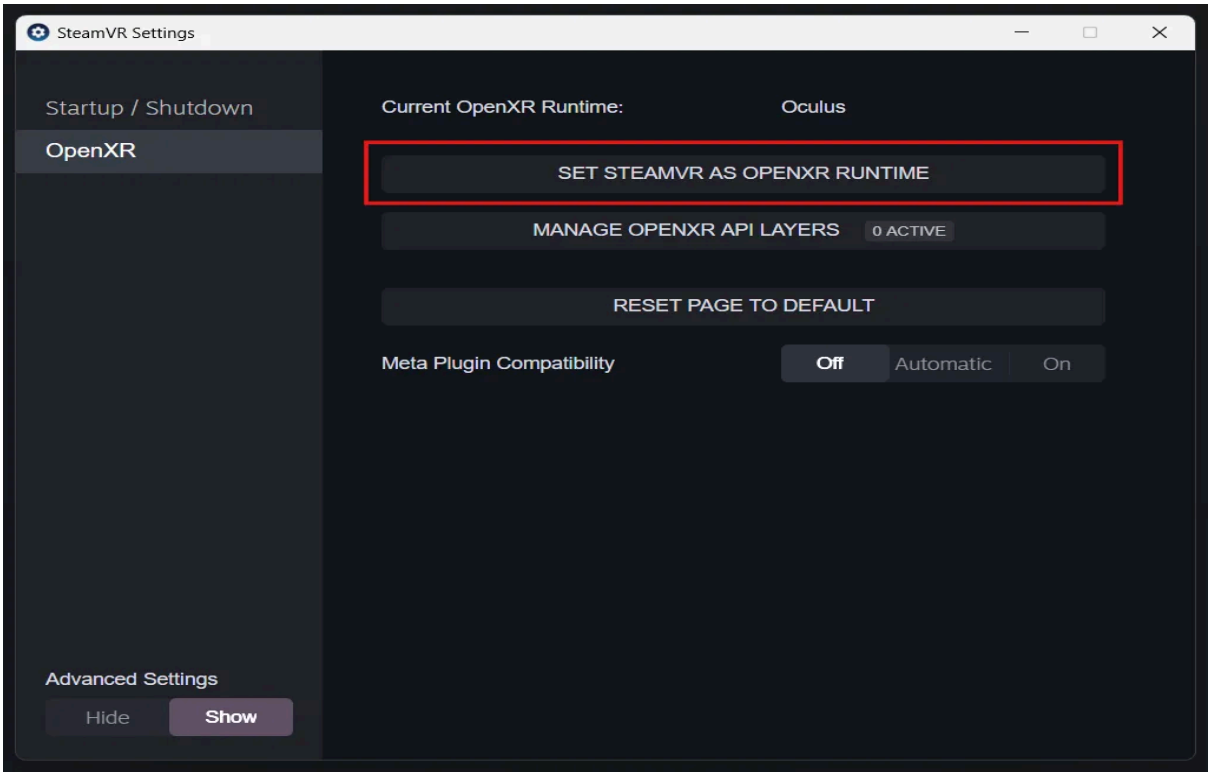
Also note that by default the physics engine runs at 60Hz as well and this can result in choppy physics. You should set `Engine.physics_ticks_per_second` to a higher value.

Set default OpenXR Runtime

Now make sure your favorite OpenXR runtime on Windows's set as default runtime. For SteamVR for example, hit the three-line top-left button and click the "Settings" menu.

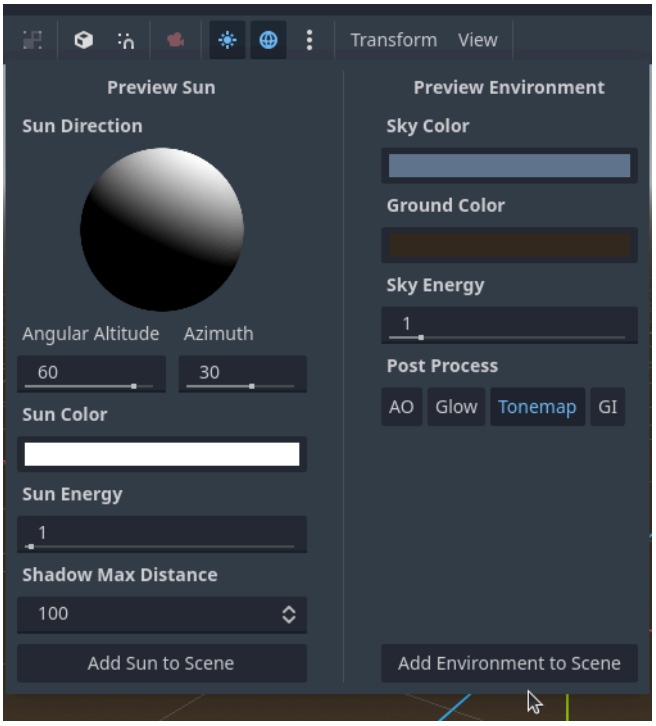


Then select "OpenXR" and set SteamVR as OpenXR Runtime.

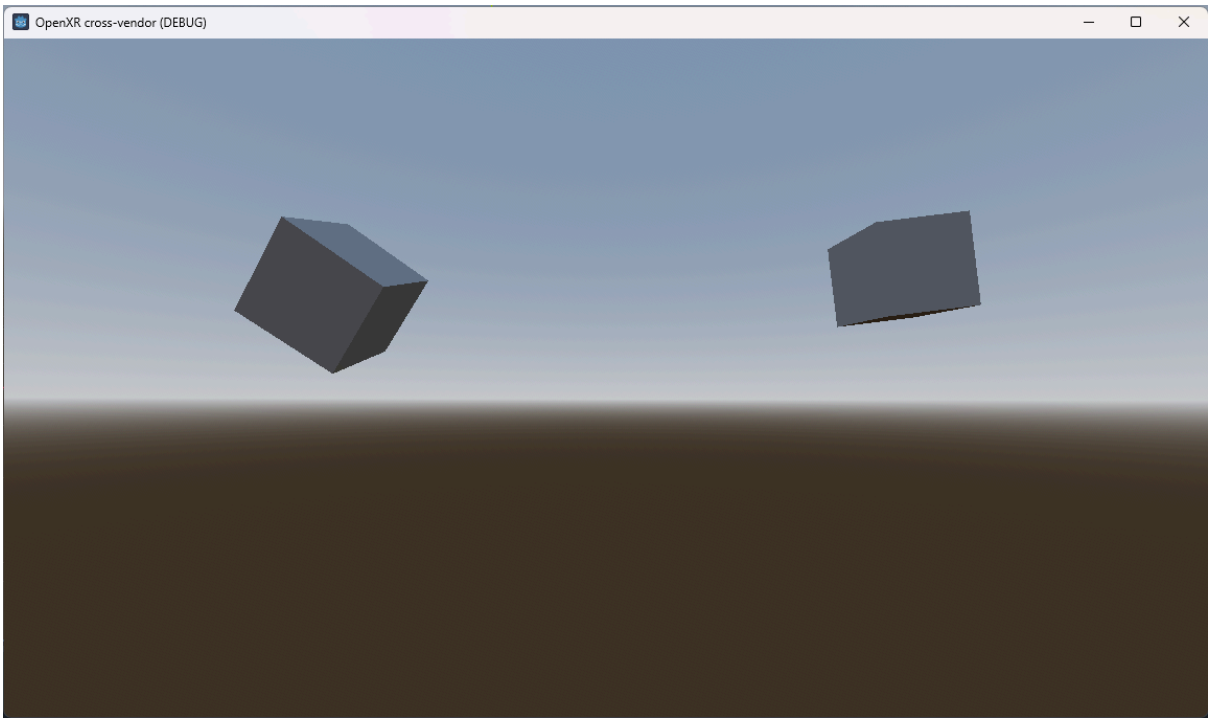


Note: Make sure the "Meta Plugin Compatibility" is "Off" to prevent SteamVR from simulating the behaviour of a Meta OpenXR runtime. You could switch it to "On" later if needed.

Finally, while we see an environment in our editor, this environment and its lightning are added by our editor so we can see the content of our scene. If we run our scene now, we wouldn't see anything. Use the three dots at the top of the scene window to open up the environment panel and click on "Add Environment to Scene"



Now run your project, you should be floating somewhere in space and be able to look around.



You can look around and move your two controller-tracked hands (cubes).