Godot

- Getting Started with Godot
- Simple XR Rig
- VR Controller Input in Godot

Getting Started with Godot

Before jumping into VR, you should get yourself acquainted with the engine and how things work. Godot utilizes a node-based system, similar to Unity except everything is its own node.

Documentation (4.2 branch)

https://docs.godotengine.org/en/stable/index.html

Overview of Godot's key concepts

https://docs.godotengine.org/en/stable/getting_started/introduction/key_concepts_overview.html#scenes

Your first 3D game

https://docs.godotengine.org/en/stable/getting_started/first_3d_game/index.html

Additional resources

Below is a tutorial on how to orient yourself in Godot. It's recommended for beginner's to follow this guide to get familiar with the engine.

https://kidscancode.org/blog/2019/03/godot_31_3d_intro/

Simple XR Rig

A lot has changed from the original version of this page. Below, I have linked the first article but you must follow all the chapters. These include:

- Introduction to the XR system in Godot
- Prerequisites for XR in Godot 4
- OpenXR
- Setting up the XR scene

Setting up XR (4.2 branch)

https://docs.godotengine.org/en/stable/tutorials/xr/setting_up_xr.html

> You may find the following two articles: **Introduction to XR Tools** and **Basic XR Locomotion**. These articles are valid but I would caution against using them and consider designing your own implementation and use these articles for reference only. If you have completed work in Unreal for things like teleportation, you will find there is a cross-over on making this work in Godot/Unity/etc. by translating the nodes into the APIs in the newer engine. I (Wes) am happy to assist.

VR Controller Input in Godot

Preface:

I use C# for coding in Godot due to the speed and resources (we need all the resources we can get when building with VR). Below is an example using C# but can be converted to GDScript as well.

For more information about C# in Godot:

https://docs.godotengine.org/en/stable/tutorials/scripting/c_sharp/c_sharp_basics.html

Getting Started

Assuming you have followed the previous tutorial and confirmed VR is working with controller visuals (even cubes) and OpenXR, you can get started.

By default, after installing OpenXR within Godot enables a default **OpenXR Action Map**. Typically, I leave these defaults alone. There is a good article in the documentation that reviews how to create your own custom inputs, but these work great.

The event structure on how you process this input is up to you! Below is an EXAMPLE but should be structured!

1. Create subscription events for ButtonPressed in Released.

Create a script on your XRController3D node (the VR controller) and the script should inherit from XRController3D.

Inside the script, create two override functions: _EnterTree() and _Exit Tree(). Add ButtonPressed and ButtonReleased and subscribe to a method. Here is the layout:

```
public override void _EnterTree()
{
  ButtonPressed += OnButtonPressed;
  ButtonReleased += OnButtonReleased;
}

public override void _ExitTree()
{
  ButtonPressed -= OnButtonPressed;
  ButtonReleased -= OnButtonReleased;
```

```
}
```

Now, create two functions, OnButtonPressed and OnButtonReleased:

```
void OnButtonPressed(string name)
{

}

void OnButtonReleased(string name)
{

}
```

2. Process events

You will notice on the methods created above that there is a parameter called string name. This string is what is received from your **OpenXR Action Map** (or anything else with a button). For me, I have added a switch statement to process these inputs and invoke methods:

```
    void OnButtonPressed(string name)
    {
        switch (name)
        {
            case "grip_click":
                OnGripPressed();
                break;
            case "trigger_click":
                OnTriggerPressed();
                break;
            case "ax_button":
                OnAXPressed();
                break;
            case "by_button":
                OnBYPressed();
                break;
        }
    }

    void OnButtonReleased(string name)
    {
```

```
        switch (name)
        {
            case "grip_click":
                OnGripReleased();
                break;
            case "trigger_click":
                OnTriggerReleased();
                break;
            case "ax_button":
                OnAXReleased();
                break;
            case "by_button":
                OnBYReleased();
                break;
        }
    }
```

I'm not including every single method, but you can see that I now have a system to process input!

3. Expand (with some Object Oriented Programming)!!

Obviously, this script would be a pain to copy for both hands and alter the names of the methods slightly. You can do this if you wish to keep it simple.

Object Oriented Programming (OOP):

For me, I created a parent script called something like XRHand.cs. Then, I created two scripts called LeftHand.cs and RightHand.cs that are children of XRHand. Using the methods above, I made them virtual so in my LeftHand.cs script for example, I can configure special implementation.

Here is the final script for XRHand.cs

```
using Godot;
using System;

public partial class XRHand : XRController3D
{
    public override void _EnterTree()
    {
        ButtonPressed += OnButtonPressed;
```

```
    ButtonReleased += OnButtonReleased;

    InputFloatChanged += OnInputFloatChanged;

    InputVector2Changed += OnInputVector2Changed;
}

public override void _ExitTree()
{
    ButtonPressed -= OnButtonPressed;
    ButtonReleased -= OnButtonReleased;

    InputFloatChanged -= OnInputFloatChanged;

    InputVector2Changed -= OnInputVector2Changed;
}

void OnButtonPressed(string name)
{
    switch (name)
    {
        case "grip_click":
            OnGripPressed();
            break;
        case "trigger_click":
            OnTriggerPressed();
            break;
        case "ax_button":
            OnAXPressed();
            break;
        case "by_button":
            OnBYPressed();
            break;
    }
}

void OnButtonReleased(string name)
{
    switch (name)
    {
```

```
        case "grip_click":
            OnGripReleased();
            break;
        case "trigger_click":
            OnTriggerReleased();
            break;
        case "ax_button":
            OnAXReleased();
            break;
        case "by_button":
            OnBYReleased();
            break;
    }
}

void OnInputFloatChanged(string name, double value)
{

}

void OnInputVector2Changed(string name, Vector2 value)
{
    switch (name)
    {
        case "primary":
            OnThumbstickMoved(value);
            break;
    }
}



// Digital Input
public virtual void OnGripPressed() { }
public virtual void OnGripReleased() { }
public virtual void OnTriggerPressed() { }
public virtual void OnTriggerReleased() { }
public virtual void OnAXPressed() { }
public virtual void OnAXReleased() { }
public virtual void OnBYPressed() { }
public virtual void OnBYReleased() { }
```

```
    // Axial-1D Input

    // Axial-2D Input
    public virtual void OnThumbstickMoved(Vector2 value) { }



    // Signals

}
```

And as for LeftHand.cs or RightHand.cs, I would just override a button that I would need:

```
using Godot;
using System;

public partial class LeftHand : XRHand
{
    public override void OnGripPressed()
    {
        GD.Print("Right grip released!");
    }

    public override void OnGripReleased()
    {
       GD.Print("Left grip released!");
    }
}
```

The LeftHand.cs and RightHand.cs would now go on their respective XRController3D nodes, in the scene.