

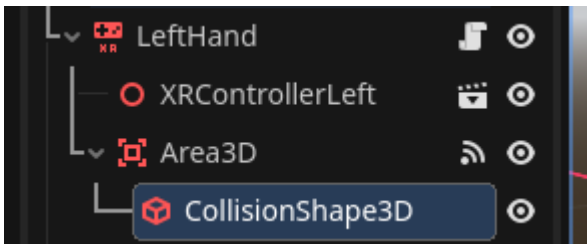
# Basic Grab

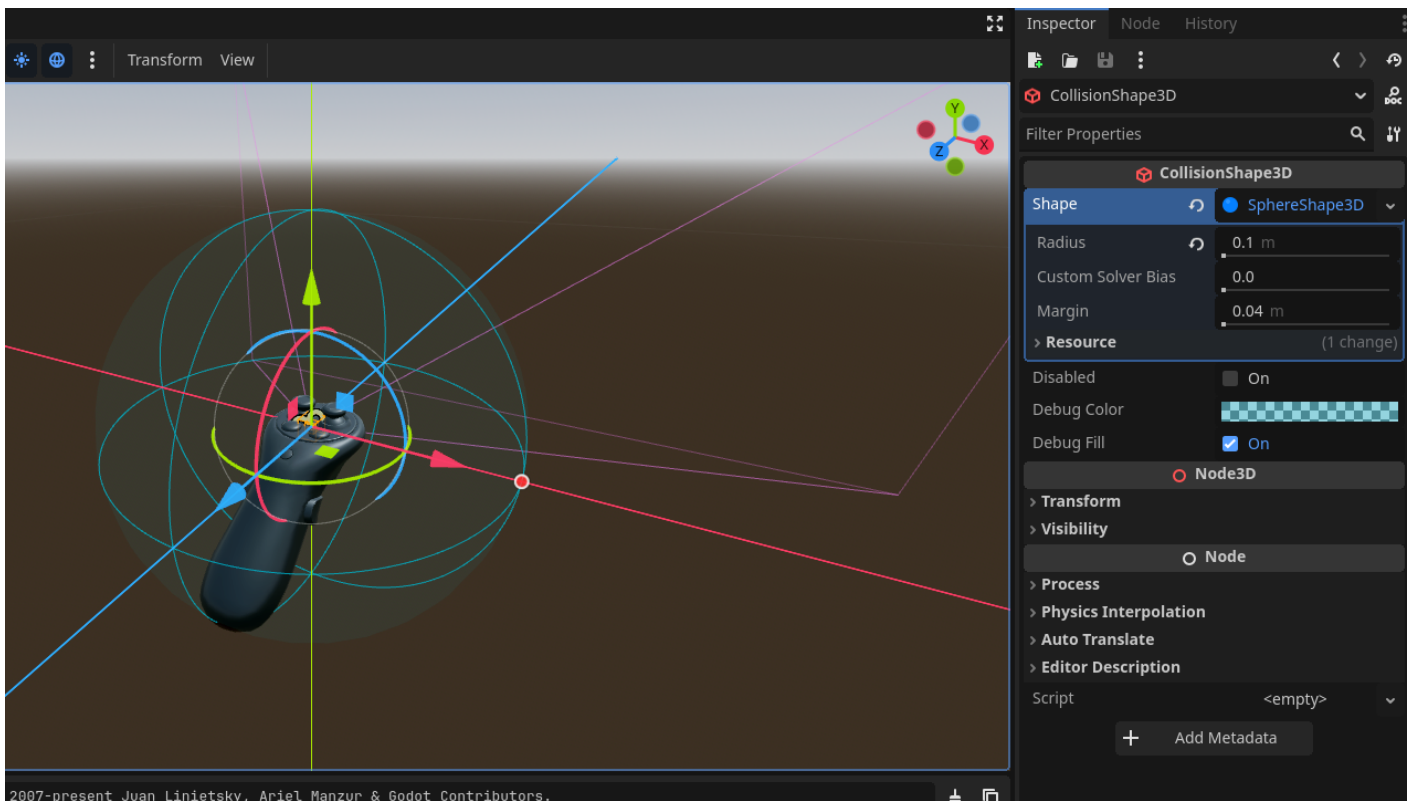
## Prerequisites:

- Basic [familiarity](#) with nodes in Godot
- [XRRig](#)
- [Controller Input](#)
- Ensure you have a ready LeftHand and RightHand script for input.

## Setup for XR Rig

For each *XRController3D* node (VR hands), add an **Area3D** node as a child. You should get a warning about the Area node needing a collision shape, so add a new **CollisionShape3D** node as a child of the **Area3D** node. For my **Shape** type, I chose a new **SphereShape3D** and gave it a radius of **0.1m**. Do this for all *XRController3D* nodes if you are using pickup interactions.





## Setup basic grabbing

In your "world" scene, add a **RigidBody3D** node to your scene and add a **CollisionShape3D** and **MeshInstance3D** node as children, as well as any setup needed for those nodes. In the **RigidBody3D** node, create a new script called **Grippable.cs** and ensure the script is attached. Now, let's add some code:

```
using Godot;
using System;

public partial class SimpleGrab : RigidBody3D, IGrabbable
{
    [Node3D] parentNode;

    public override void _Ready()
    {
        parentNode = (Node3D)this.GetParent();
    }

    public void Pickup(Node3D receivedController)
    {
    }
```

```

    Freeze = true;

    Reparent(receivedController, true);
}

public void Drop(Vector3 receivedVelocity)
{
    Freeze = false;

    Reparent(parentNode);

    //CallDeferred("set_axis_velocity", receivedVelocity);
    SetAxisVelocity(receivedVelocity);
}
}

```

Let's review what's happening here:

- The **parentNode** is a variable used to cache where we need to re-parent this object when it is dropped, which is usually back to the world. In our **\_Ready** method, we just get the current parent.
- The **Pickup** method and its reference to the controller doing the interaction is done here. We **freeze** physics while we are holding this object and parent it to our controller. The **true** parameter in **Reparent** is passed to keep the global transform before parenting so the pickup looks like we are grabbing an intended point.
- The **Drop** method does the reverse of **Pickup**, except an added line that gives the object velocity from the controller when the object is let go. *There is an optional line if you wish to use CallDeferred instead of SetAxisVelocity (for advanced users).*

## Add functionality to our hand (each)

Assuming you have completed all necessary prerequisites, you should have a **LeftHand.cs** and a **RightHand.cs** with some possible functionality. Here is additional code to add to each hand:

```

Vector3 velocity;
Vector3 previousPosition;

Area3D area;
Grippable grippable;

```

```

public override void _Ready()
{
    area = GetNode<Area3D>("Area3D");
}

public override void _Process(double delta)
{
    velocity = (Position - previousPosition) / (float)delta;
    previousPosition = Position;
}

public override void OnGripPressed()
{
    var bodies = area.GetOverlappingBodies();
    foreach (var body in bodies)
    {
        if (body is Grippable _)
        {
            grippable = body as Grippable;

            grippable.PickUp(this);

            return;
        }
    }
}

public override void OnGripReleased()
{
    if (grippable != null)
    {
        grippable.Drop(velocity);
    }
    grippable = null;
}

```

Let's review what this code achieves:

- A variable, **velocity**, is added for when we need to let to drop an object and give it velocity. *The benefit is this calculation does not involve a physics node or other physics*

*calculations that are not needed. You may optionally add a boolean flag in the `_Process` method if you do not wish to calculate velocity every frame except for when you are actually holding something.*

- In **OnGrippedPressed**, we use our **Area3D** node to check all overlapping bodies. If one of these bodies is a Grippable type, then we attempt to pick it up and return out of the method.
- For **OnGrippedReleased**, we do a check to see if we are actually holding a grippable object and, if we are, we drop the object and pass along our hand's velocity. Finally, we indicate we are no longer holding anything in this hand.

## Testing

If you have reached this stage, go ahead and test functionality with both hands.

## Where to go from here?

If you are looking to add advanced interactions like sliders and dials, I may recommend keeping these methods for grabbing types, but add new classes that track the controller's movement instead of parenting when picked up or dropped.

---

Revision #3

Created 28 April 2025 14:16:02 by Wes

Updated 5 May 2025 14:33:37 by Wes