Basic Grab

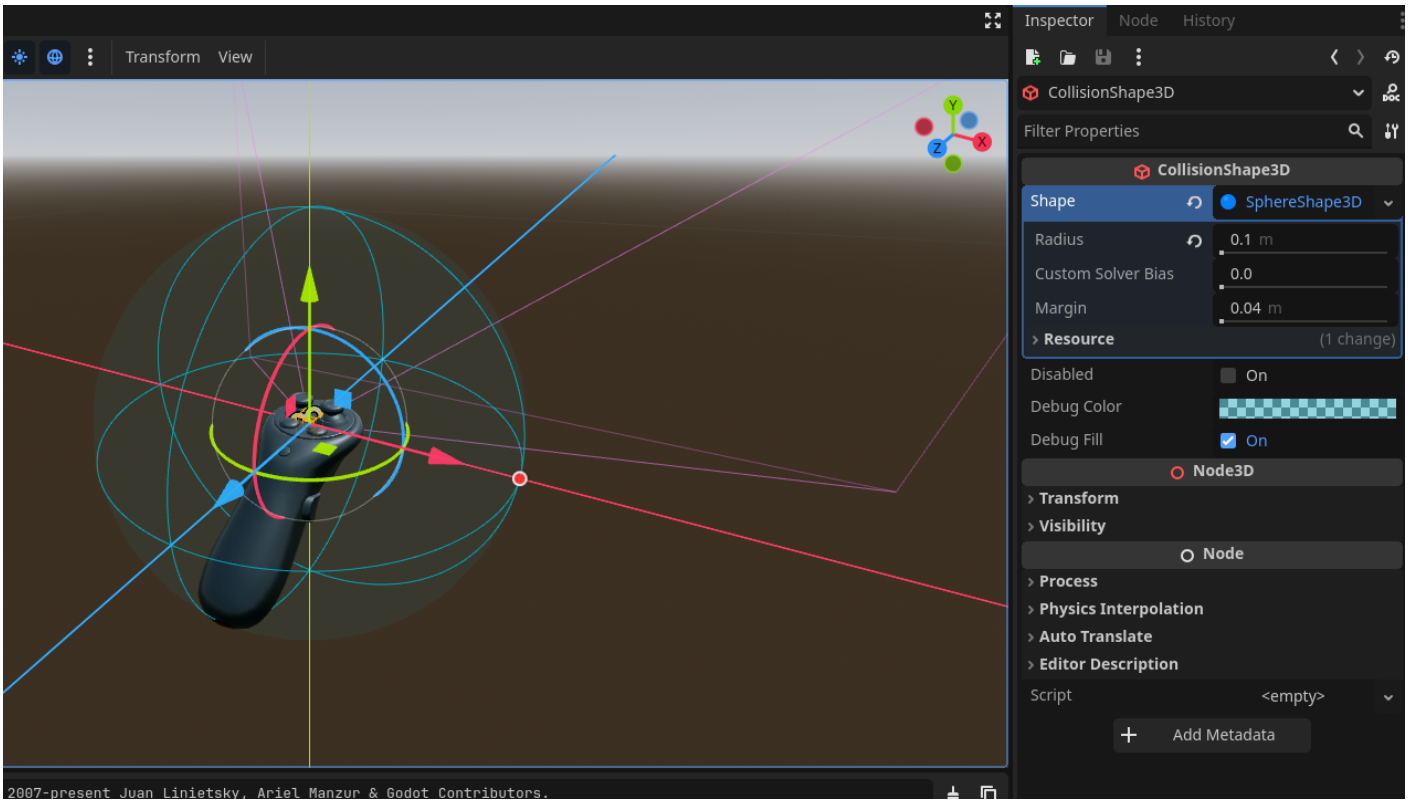
Prerequisites:

- 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** and ensure the script is attached. Now, let's add some code:

C#

```
using Godot;
using System;

public partial class Grippable : RigidBody3D
{
	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)
{
	Reparent(parentNode);

	CallDeferred("RestorePhysics", receivedVelocity);
}

void RestorePhysics(Vector3 receivedVelocity)
{
	Freeze = false;

	SetAxisVelocity(receivedVelocity);

	LinearVelocity = receivedVelocity;
}
}
```

GDScript

```
class_name  Grippable

extends RigidBody3D

var parent_node: Node3D

func _ready() -> void:
	parent_node = get_parent()

func pick_up(received_controller: Node3D):
	freeze = true
	reparent(received_controller, true)

func drop(received_velocity: Vector3):
```

```
reparent(parent_node)

call_deferred("restore_physics", received_velocity)

func restore_physics(received_velocity: Vector3):
freeze = false

set_axis_velocity(received_velocity)

linear_velocity = received_velocity
```

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 will reparent the object back to the original parent. For Godot, we are going to restore physics using **CallDeferred**, which is used to delay the execution of a function until the current frame finishes and the engine is in a safe, idle state. This calls **RestorePhysics**.
- **RestorePhysics** is the method called after **CallDeferred** is completed. Physics is restored by setting freeze to false and applying velocity so the object leaves our hand naturally.

Add functionality to our hand (each)

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

C#

```
    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;
    }
```

GDScript

```
extends XRHand

var velocity: Vector3
var previous_position: Vector3
var grippable: Grippable

@onready var area = $Area3D

func _process(delta: float) -> void:
	velocity = (position - previous_position) / delta
	previous_position = position

func on_grip_pressed():
	var bodies = area.get_overlapping_bodies()
	for body in bodies:
		if body is Grippable:
			grippable = body as Grippable
			grippable.pick_up(self)
			return

func on_grip_released():
	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, you may wish to keep these methods for grabbing types, but add new classes that track the controller's movement instead of parenting when picked up or dropped, possibly a refactor.

Revision #13
Created 2025-04-28 14:16:02 UTC by Wes
Updated 2026-08-05 14:25:47 UTC by Wes