**Setting Local Rotation**

**Objective**: We're going to we're going to learn something really important about rotations. We're going to learn that the order in which we do things with rotations matters and we're going to learn how to set a rotation from code.



1. Lets me sure that we’re in **local mode**
2. Using the rotation tool have your ship mirror this position.



It should be 30 degrees to the right (yaw) and 30 up (pitch). Now does the order in which you do this matter? (It does). If you try to pitch first then yaw all of your rotation degrees will change. So you try and fix the x but now the y has gone off and the z has gone off.



That is not because of anything to do with local or global, this is much deeper than that. This is because the order of rotations matter.

1. Lets create a ProcessRotation method inside the Update method. Inside of this new method lets type transform.localRotation. Now, you might think you just go dot x, don't try and do that. It says the x component of the core turning arm, don't modify this directly unless you know quaternions inside out. What are quaternions I here you ask? They are a different internal representation of rotation, is all we need to know about them for now.

The point is, you can't just set local rotation directly like that. What we need is to create a rotation.



Now Euler returns a rotation that rotates z degrees around the z axis, x degrees around the x axis, and y degrees around the y axis; applied in that order. so lets give it the following



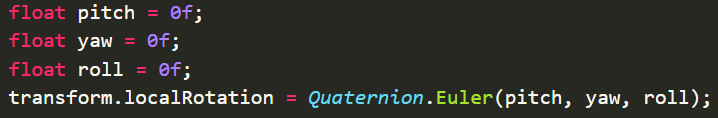
1. Lets go to Unity and press play!



Now lets recreate our earlier experiment.



1. Lets create pitch, yaw, and roll variables and replace the literals in the Euler method call in order to use these variables.



What I want to do is have some sort of parameter that I can tweak. So that when the ship is up at say, imagine it's up at y equals 2, what we want the thing to be rotated, pitched down by about 10. Okay so, it looks like about a -5 to 1 factor, or if the ship was down at -2, then we would want it pitched to +10.

Okay, so we've got a position pitch factor, lets call it. A pitch that depends on the position on the screen. A position pitch factor of about -5.

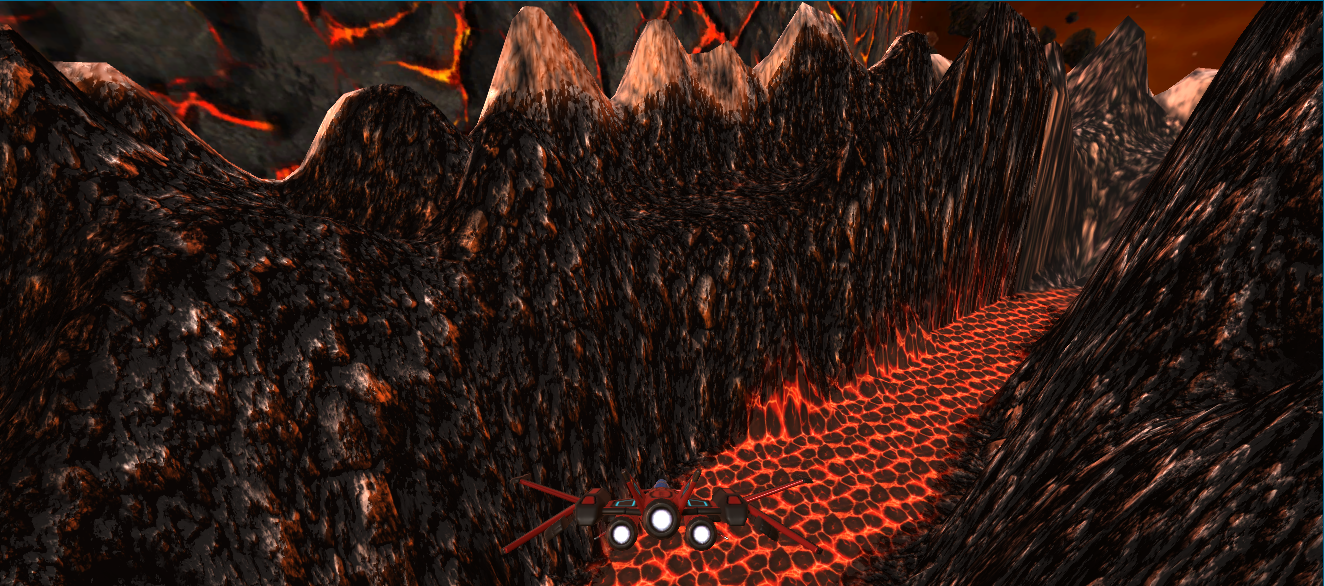
1. Lets create a new SerializeField variable called **positionPitchFactor**

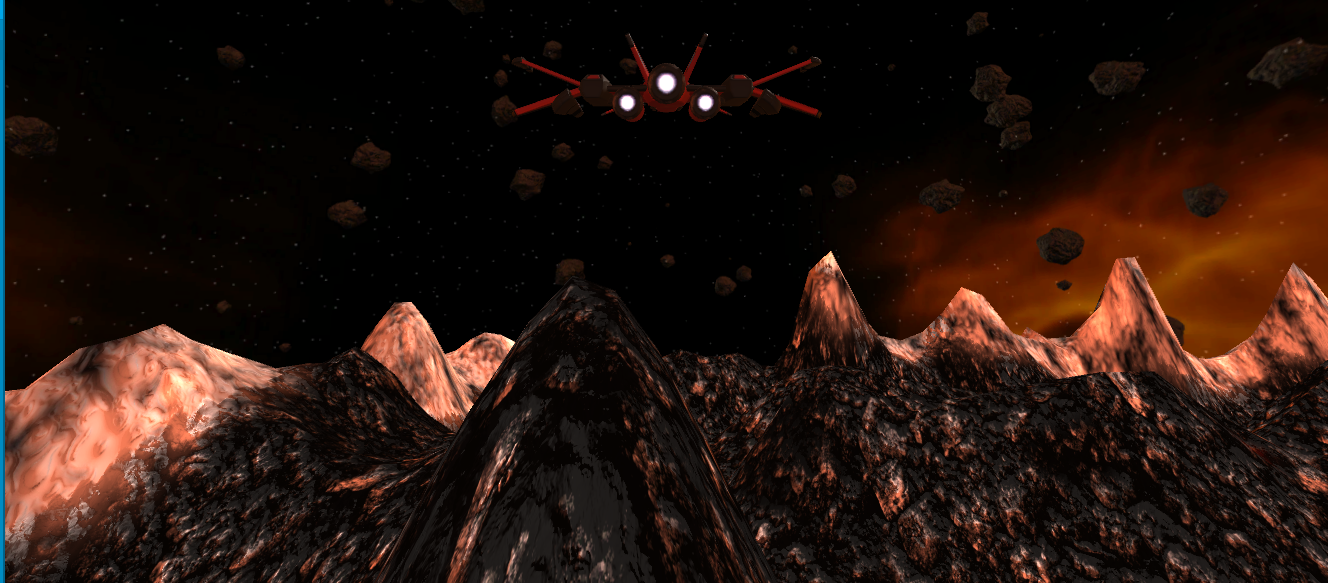


1. In our **ProcessRotation** method we’re going to set our Pitch factor to this.



1. Lets go to Unity and test this out.





Ship looks good!!

Now we need to continue making some adjustments to our ship.

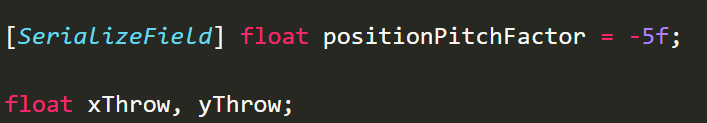
|  |  |  |
| --- | --- | --- |
|  | Position on Screen | Control Throw |
| Pitch | Coupled | Coupled |
| Yaw | Coupled | - |
| Roll | - | Coupled |

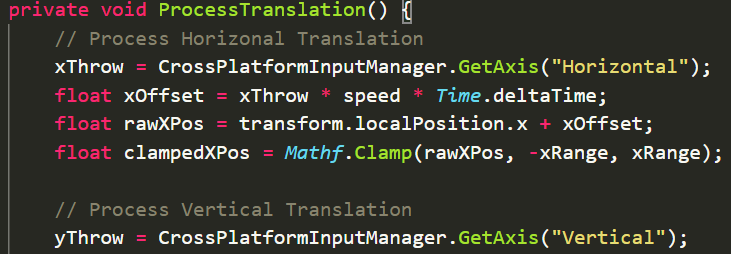
What I mean by that is that when I move up and down I want it to, kind of, nudge its nose up and down, temporarily, while I'm holding the key down. to do that lets all yThrow to the pitch.



Now why am I getting this error??

Its because the **yThrow** variable only exist inside of the **ProcessTranslation** method. To fix this we’re going to make **yThrow** a member variable so it can be accessed across all of the class. SO lets declare **yThrow** and **xThrow** after the **positionPitchFactor**.



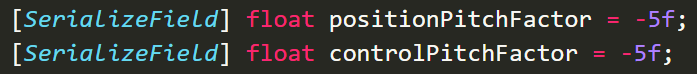


Lets go test this out.

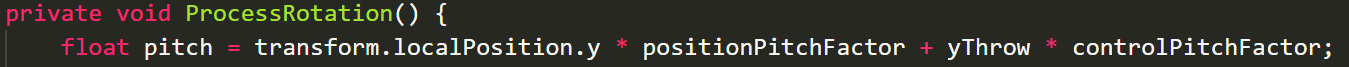
So when we're not pushing the key, we get the same effect. When I am pushing the key, well, I probably am getting an extra. If you watch the rotation and I push w at the top of the screen, it goes from -15 to -14, it's tilting down slightly further.

But this isn't enough, okay? So we need a factor on here as well, so lets create ourselves a factor.

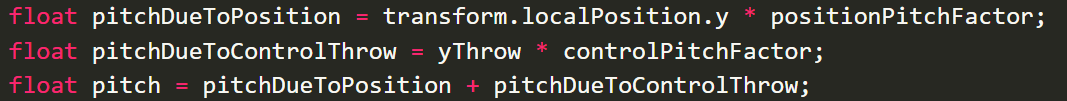
1. In our script lets create a **controlPitchFactor** after **positionPitchFactor.** Now what should I set it to? Lets think about it. If I’m at the bottom of the screen, let's say, if I want to go to the top of the screen I want the ships nose to come up. How much do I want it to come up? Well I want it to come up by about 5 degrees probably



so in our ProcessRotation method lets multiply our yThrow times controlPitchFactor



1. Now that is a lot in one line so lets break it up into **pitchDueToPosition** and **pitchDueToControlThrow**.



Also this works in our favor because our code is now self-documenting we don’t need to comment these lines of code.

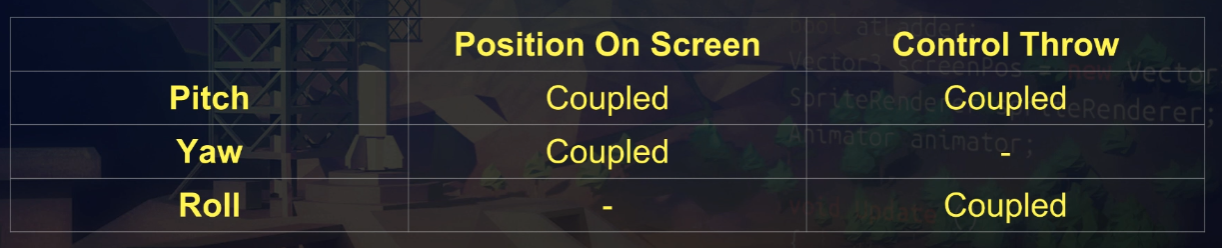
Lets go test our changes. So our ship changes up and down kind of gradually and too slow. Lets increase Control Pitch Factor to -20. Now our ship looks like its pointing up/down kind of steep!



In the end I actually decided to stick with **-30**.



So what we just did was to couple pitch to the Control Throw.

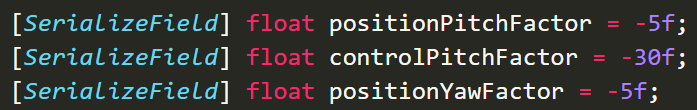


**Challenge:** Now your challenge is to complete the rest of this table. I want you to couple yaw to Position On Screen and roll to Control Throw. All of our intended rotations should work. Your parameters should feel good.

YAW

1. In **ProcessRotation** method lets fix yaw.





I don’t know what the value should be so lets go into Unity and experiment. Lets move to the left and that’s the exact opposite of what we want.



Lets try a +5. It's about right, it's doing something weird now with, I think, this is just camera perspective, in terms of what it's doing here. Because the rotation ease only yawing. It's just that he's looking a little bit odd, from a perspective point of view, with the width of my camera.

1. Go to the Main Camera and lets change the field of view to 45 and move the rocket a little bit forward. That looks better. Lets make sure our Yaw Factor is set to +5 in the inspector and in our script.

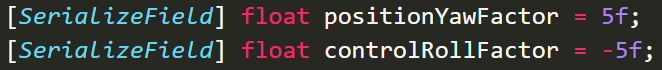
Okay, we're getting a little bit of perceived roll in there. It's not actually rolling if we look at the rotation in Zed here. It's not actually rolling, but it kind of looks a bit like it's rolling.

ROLL (left to right)

1. With roll, we're just going to say the **xThrow** times the **controlRollFactor**.

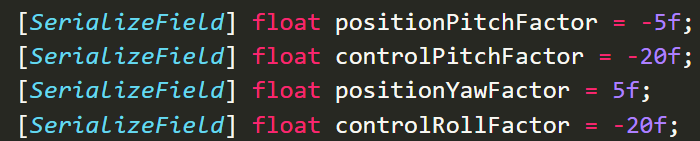


Lets create this variable



Playing around with all of these factors working together I decided to change **Control Pitch Factor** and **Control Roll Factor** to **-20** I also changed the **X Range** to 5 since the cameras field of view made my ship go off screen.

One last thing im going to change the default values of my script to math these values.



So let's just recap the code and we are done. What we've done here in process rotation is, say that the pitch should be due to both position and control throw. The yaw should just be due to position. And the roll should just be due to control throw.