Math’s & Tech Implementation report

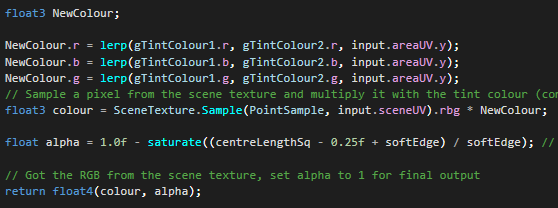
# Introduction

In this report I will discuss my implementation of the math’s and tech assignment, I will talk about the post processers I have added which were required by the brief and some other post processers I have added.

At the bottom of this report there is a check list of everything I have implemented and a list of additional features I have added to the assignment for more marks.

# Basic Requirements

Vertical Gradient Tint

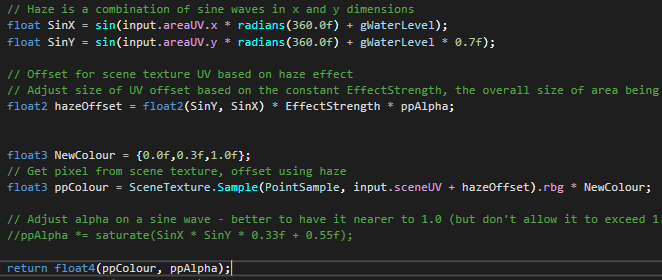
The tint post process requirements me to pass two vector3 which will contain the rgb values of the colors I intended to show, once in the shader I create a new colour and set the rgb values of that new colour to the lerp values of the two colours rgb values at the current y of the UV.

After I have lerp the two colours I then get the scene texture and times that by the new colour I created and return that scene texture.

Blur

The blur gets updated to a two-pass garrison blur, read extension to basic requirements for details on two pass garrison blur.

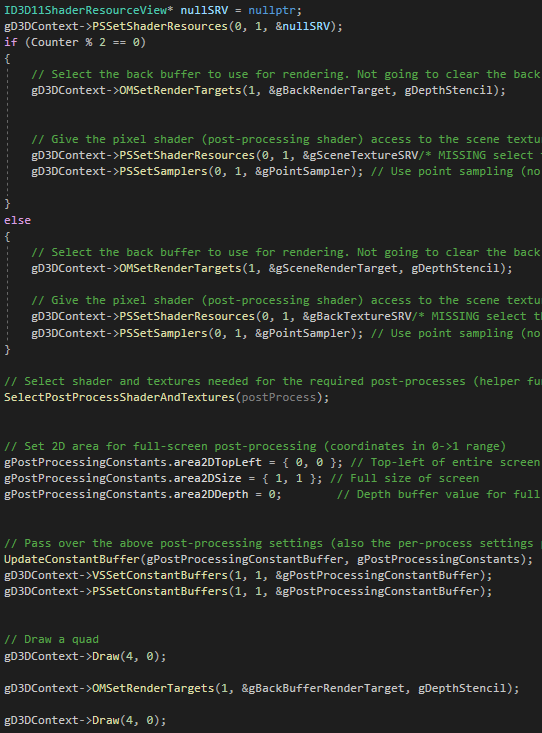
Underwater

The underwater shaders are very similar to the heat haze shader so the code is very similar, in underwater I create and store a sin wave for the X & Y, this is so I can create an offset for the wiggle effect.

After I create the offset, I create a colour for the water, which is default to blue, I then sample the texture but added the UV and offset together to added the wiggle then times the texture by the colour I created then return.

# Extension To Basic Requirements

Allowing multiple post processers

When it came to have two post processers on screen, I had to create another set of Textures, SRVs and render targets. This was so that I could swap between the render targets and srv’s for allowing multiple post processers.

When a post process gets activated I added to a vectors of post processing data, after it gets added in render scene it will go into a for loop to loop through each point in the vector, depending on what mode the post process is will depend on what will happen to it but I will use full screen for my example.

So, when it enters the “FullScreenPostProcess” I set the Vertex and GS shaders as well as setting the blend states, stencil state and the culling. But I also decouple the current SRV.

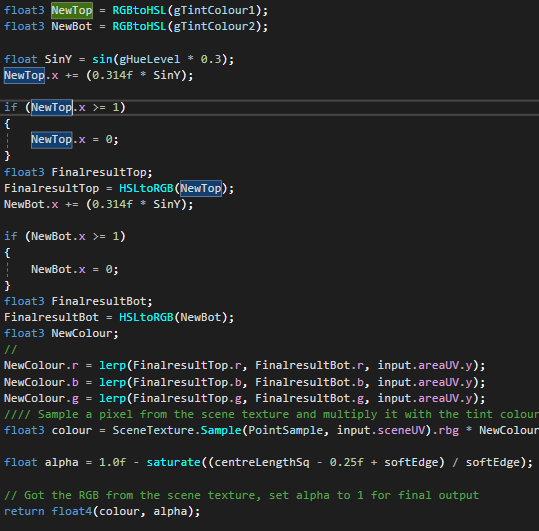
After I have done all this, I have a variable to keep track of where I am currently in the vector and I check the modules of that variable, this will allow me to determine where I should be rendering and setting the srv to. If it’s even, then I will render to the first target but set the srv to the second one.

Once I have determined which one to render to, I then do which post process has been pasted, then I update the buffers and constants. I then draw that screen and set the render target to be the back-buffer render target and draw again.

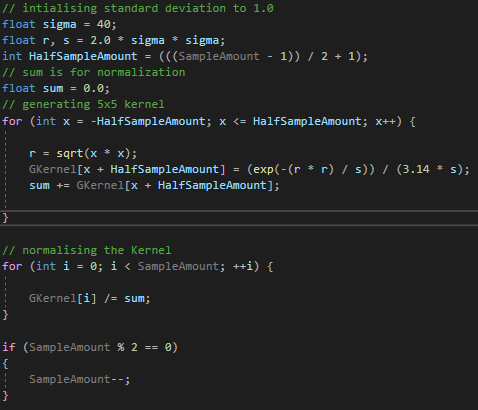
This will allow the user to add multiple post processers at once.

HSL Colour Gradient

The HSL colour gradient requires two colours to be passed to it, once in the shader I create two new colours for the top and bot of the screen and assign them the two colours pasted but before assignment I convert the colours to HSL so I can access the hue value.

Then I create a sin from and timer and add it to the HSL value of the colour, if the hue is above 1 then I reset it to 0. I then convert back to RGB and assign it to a new colour, I repeat this process for the other colour too.

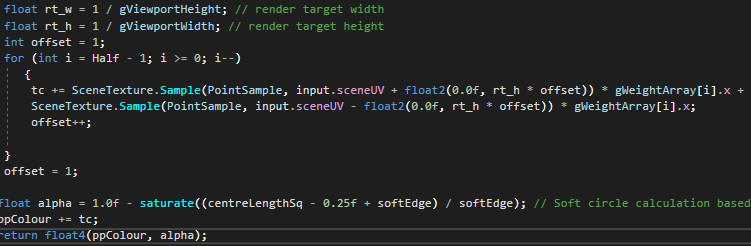
I create a new colour and set the rgb values of that new colour to the lerp values of the two colours rgb values at the current y of the UV.

After I have lerp the two colours I then get the scene texture and times that by the new colour I created and return that scene texture.

Two Pass garrison blur

When it came to doing the two-pass garrison blur I need to make the weights properly, I found a function online at [https://www. geeksforgeeks.org/gaussian-filter-generation-c/](https://www.geeksforgeeks.org/gaussian-filter-generation-c/) which generated the weights for me. Although this function split the weight disputation in the middle of the array so I had to work around that.

Since the weight were already done all I had to do was pass them back into the shader. Once in the shader I create an empty float3 and find what the middle point of the array was, once I had that I get the scene texture and times that by the value at the half point in the array since is the start of the disputation.

I then run through a for loop for all the weights, in this loop I get the empty float3 and add the scene texture times the current weight, after that I add the scene texture to another scene texture times the current weight again. Look at the code below for a more clearer explanation.

I then add the scene texture times the middle weight and the float3 together and return.

# Polygon Post Processing

# Advanced Post Processing

# Additional Implementations

# IMGUI

# Brief Check List

* Vertical Colour Gradient.
* 2 Pass Garrison Blur.
* Underwater.
* Multiple post process functionality (Can handle Fullscreen, Polygon, Model Polygon and Area all at once).
* HSL Colour space gradient.
* Feedback to Blur (Maybe).
* Wall model with one window (Has a post processers).
* Wall model with four windows (all have post processers).
* Retro Game Mode Shader.
* Full screen Bloom

Additional Features.

* Black and white post process.
* Night vision post process.
* Inverse post process.
* Predator Vision post process.
* Seeing Worlds post process
* Scanlines post process
* Added a new post processing mode called “ModelPolygon”

IMGUI:

* Model Selection
* Post processing mode selection
* Menu to add post processes at runtime.
* Button to allow the user to clean the screen.
* “X” button to remove a post process.
* “Up” button to move the post process up in the vector.
* “Down” button to move the post process down in the vector.