Graphics Programming Document

**CMP301**

|  |  |
| --- | --- |
| Student Name | Keiran Millar |
| Student Number | 1502338 |

Table of Contents

[User Controls 3](#_Toc500173807)

[Shaders 3](#_Toc500173808)

[Blur (Horizontal and Vertical) 3](#_Toc500173809)

[Depth 4](#_Toc500173810)

[Shadow 4](#_Toc500173811)

[Tessellation 4](#_Toc500173812)

[Vertex 5](#_Toc500173813)

[Reflection 5](#_Toc500173814)

# User Controls

On runtime for my program you will be met with a scene of a cube being lit by a light, to move around the scene you can use WASD, Q and E will move you up and down in the Y-Axis. To look around you only need to press SPACE and it will allow the mouse to look around the scene, pressing SPACE again will free to mouse to allow you to interact with the GUI. In the first scene’s GUI you can move a slider which will change the scale of the texture used to make the blur effect, increasing the number to 10 will make it 10 times. The higher the number used will drastically increase performance, but it will also look considerably worse. On the other hand, you can move the slider down to 0.25 which will cause the post processing to be calculated on a texture which is 4 times larger than the screen dimensions, allowing for a better looking scene but is very process intensive (caution at 0.25 scale is advised due to low fps). There are also two other sliders, named blur angle and blur intensity, moving these will affect the blur post processing. Changing the intensity will make the screen blurrier or even turn the blur off completely. The blur angle slider will change which direction the blur is applied, 0, 180 and 360 degrees will be a horizontal blur, 90 and 270 degrees will be a vertical blur.

Checking the Switch Scenes box will take you to another scene to show off other shaders I have worked on but was unable to get them all working together. Here is where you will see tessellation and vertex manipulation. The controls are identical to before but in the GUI, you will see a checkbox that will toggle wireframe mode. Using this box allows you to see the tessellation at work, tessellation is based on distance so the closer to the centre you are, the more tessellated the quad will be.

Pressing ESC will close the program.

# Shaders

## Blur (Horizontal and Vertical)

For my variable blur shader, there are two important variables in this. Those being the angle and intensity variables, they allow for the blur to be customised during run time. This is achieved in the vertex shader by multiplying the texel size by the cos or sine of the angle given (cos and sine used depending on whether horizontal or vertical blur is being used) as shown below.





The texel size is then added to the input texture coordinates and is given to the pixel shader. These variables are used to add the surrounding colours of a pixel to achieve this blur effect. Due to it being sine and cos, at points where the calculation comes to be 0 then the associated direction is cancelled out.

Another part to this which isn’t inside the shader (but is still important) is the ability to dynamically change the size of the texture being processed on. This causes the image to look better or worse dependant on the size used. It also has the obvious consequence of rendering the blur useless if the image is upscaled. This is counter-acted by increasing the blur intensity

## Depth

The depth shade is a simple shader that allows the depth map to be created, which is required for the shadows to be accurately portrayed. This is achieved by passing in the view and projection matrices from the light, instead of using the matrices from the camera. This means that everything that the light can see is set to be white (a shade of white that gets darker over distance), whereas anything that it can’t see is set to black so you know where to light in the shadow shaders.

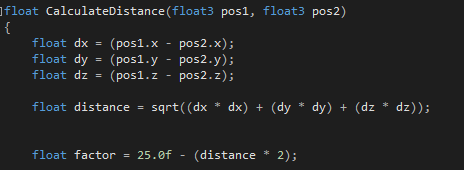
## Shadow

The shadow shader uses the previously calculate depth map to consider whether to light a pixel or not. The important variables that are used are, the depth map texture, the ambient and diffuse colours from the light and the lights position.

The pixel shader then uses this depth map to calculate whether a pixel will only receive ambient light or whether it will receive diffuse light. If it does receive diffuse light it will be given the required amount based on how close it is to the light.

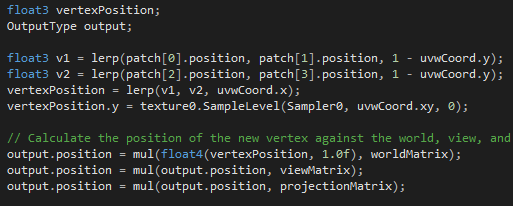
## Tessellation

The hull and domain shader is where the magic happens for this shader, the hull shader will take in the position of the camera and uses that to calculate the distance from the camera to the centre of the mesh that is being tessellated. This is achieved by taking the position of each corner, adding them together then dividing the position by 4. The camera position and the centre point of mesh is then given to my function called Calculate distance.



In the above shown function, a vector is created by taking away the coordinates of each point, the magnitude of this vector is then calculated and used to determine the tessellation factor of the mesh.

Onto the domain shader, this is where the shader creates new vertex’s using the current patches from the mesh as shown below.



## Vertex

Inside of the vertex shader is a basic version of some tessellation code. There is also some commented out code which will make the shape it is given be manipulated in the form of a sine wave. Most of the variables passed through this shader is standard variables, like positions, texture coordinates and normal. Some variables passed into the vertex shader however, is the Time buffer that I made to allow for the height, frequency and time variables needed for the vertex manipulation. Time and frequency were both used for the sine wave. There is an if statements that decides what way the manipulation should be applied, this allows a cube to have all of its sides manipulated in different directions.

# Reflection

One aspect of the application I would like to fix is that there are two different “scenes”. This is because I was unable to get the tessellation working with the blur and lighting shaders. Also, due to time restraints I was unable to implement a functioning method of calculating a normal map for my tessellated mesh. Some ways I would like to extend my application is to simply add more things into it, for example I had the idea of using the vertex manipulation shader to add a plane that appeared to look like water. I would also like to have implemented a way to use an array of tessellated mesh’s but have them all share a texture, so it would appear like a large environment if used with a height map.