**Graphics Programming Coursework**



Connor Andrew McSheffery

S1714052

2020

*I confirm that the code contained in this file (other than that provided or authorised) is all my own work and has not been submitted elsewhere in fulfilment of this or any other award*.

Connor McSheffery

Contents

[Shader Overview 3](#_Toc39938056)

[Vertex File 4](#_Toc39938057)

[Variables 4](#_Toc39938058)

[Methods 4](#_Toc39938059)

[Fragment File 6](#_Toc39938060)

[Variables 6](#_Toc39938061)

[Methods 6](#_Toc39938062)

[Process 8](#_Toc39938063)

# Shader Overview

The shader that I have created is a double textured shader, the shader uses the position of each vertex and will display the shader for the correct area of the screen. Only two shaders are used within this, however the code can be manipulated to work with many different shaders.

The two shaders used within this conjoined shader are the Toon shader that was given through labs and a glass texture shader which is created using the Refract() method. Together this gives one model that will change texture as it passes along the centre of the screen.

This is incredibly useful for games that feature a change in scenery as the user crosses a threshold. The shader takes the position of each vertex so the model can be seen transferring between textures as if to cross a threshold.

# Vertex File

## Variables

**layout (location = 0) in vec3 VertexPosition;**

**layout (location = 2) in vec3 VertexNormal;**

The layouts define the position and the Normal vector of the each vertex, the VertexTexCoord is not used within the file so it has been left out to allow for clean code.

**uniform mat4 transform;**

The transform variable holds the modelViewProjection matrix. The three separate matrices are also included to show that both variants work.

**varying vec3 normal;**

The normal variable is used to store the VertexNormal data which is then sent to the fragment file for it to be used in finalising the shader.

**out vec3 Position;**

The Position variable is used to store the position of the vertex which is then delivered to the fragment shader for it to be used in finalising the shader.

**uniform mat4 model;**

**uniform mat4 view;**

**uniform mat4 projection;**

These matrices are the same as the transform variable, however, these are added to show that both variants work.

## Methods

**if(VertexPosition.x < 500)**

This if statement checks the position of the vertex at the moment it is called and if it is on the left side of the centre point of the screen (determined by the x position being less than 500) then the code that will create the Toon shader will be called. Otherwise, if the vertex is on the right side of the centre point of the screen (determined by the x position being more or equal to 500) then the code that will display the glass effect will be called and run.

**normal = VertexNormal;**

**gl\_Position = transform \* vec4(VertexPosition, 1.0);**

These lines of code create the basis of the toon shader by storing the normal vector of the vertex and setting the position of the vertex using the transform variable and the original vertex’s position.

**normal = mat3(transpose(inverse(model))) \* VertexNormal;**

**Position = vec3(model \* vec4(VertexPosition, 1.0));**

**gl\_Position = projection \* view \* model \* vec4(VertexPosition, 1.0);**

These lines of code create the basis of the glass shader texture by storing the multiplication of the normal vector of the vertex and the transposed inverse of the model matrix, the position of the original vertex’s position multiplied by the model matrix, and then setting the vertex’s position using the projection, view, model matrices and the original vertex’s position.

# Fragment File

## Variables

**uniform vec3 lightDir;**

The lightDir variable stores the direction that the light source is projected for use within the Toon shader.

**varying vec3 normal;**

The normal variable is taken from the Vertex file in order to be used to finalise the shader.

**in vec3 Position;**

The position variable is also taken from the Vertex file to be used to finalise the shader.

**uniform vec3 cameraPos;**

The camera’s position is needed to allow the glass texture to run smoothly and correctly.

**uniform samplerCube skybox;**

The skybox is taken to be projected onto the model when creating the glass effect so that it shows the opaqueness of the glass.

**float intensity;**

**intensity = dot(lightDir,normal);**

The intensity variable is created and data is stored to develop the Toon shader .

**float ratio = 1.00 / 1.52;**

**vec3 I = normalize(Position - cameraPos);**

**vec3 R = refract(I, normalize(normal), ratio);**

These variables are used to create the glass texture and project the skybox texture onto the model.

## Methods

**if(gl\_FragCoord.x < 500)**

This if statement checks the position of the fragment in the same way as the if statement in the Vertex file.

**gl\_FragColor = intensityCheck(intensity);**

If the fragment is on the left side of the screen, then the fragment’s colour is changed by receiving a vec4 value from the intensityCheck() method depending on the light’s intensity on the model.

**gl\_FragColor = vec4(texture(skybox, R).rgb, 1);**

If the fragment is on the right side of the screen, then the fragment’s colour will be changed into a glass texture with the skybox texture also applied to show the opaqueness of the glass.

**vec4 intensityCheck(float intensity)**

The intensityCheck() method finds the lights intensity on the model and will return the colour that will match the intensity of the light on the model. The method also takes in the intensity float value created within the main() method to be used for calculations.

**if (intensity > 0.95)**

**return vec4(1.0,0.5,0.5,1.0);**

**else if (intensity > 0.5)**

**return vec4(0.6,0.3,0.3,1.0);**

**else if (intensity > 0.25)**

**return vec4(0.4,0.2,0.2,1.0);**

**else**

**return vec4(0.2,0.1,0.1,1.0);**

The if statement takes the intensity float value and returns the rgba value that will suit the texture the best.

# Process

To create the shader, I initially searched through lots of different types of shaders and looked at what was possible for me to complete. After searching, I found the gl\_FragCoord variable that is present in glsl and decided that a combination of two different shaders within the same model that changes as it surpasses a threshold seemed doable and could help in lots of scenarios.

Once I decided that I wanted to work with the gl\_FragCoord, I then had to decide on the two shaders that would be used on the model. From the beginning, I wanted to create a glass texture as it seemed doable, a challenge and it would be very beneficial to know how to create a glass texture for future projects.

To create the glass texture, I took across the reflection shader and changed the reflection for refraction, I then found online the correct ratio for refraction to give a glass effect and implemented it in.

Once I had the glass texture created, I wanted to find a strong contrast to the shiny glass texture and remembering from the labs, the Toon shader contrasted highly against it, so I decided to take the toon shader from the lab and implement it into the model.

I ran into a slight issue when creating the shader as I would only be able to get one shader textured as both shaders require different methodologies to texture the model also. To combat this issue, I took the vertexPosition variable as it holds the vertex’s position on the screen in the same way the gl\_FragCoord holds the fragment’s position. So, I created an if statement much in the same way as within the fragment shader which allowed for both texturing methodologies to be used to texture both shaders featured on the model.

As the shaders are changed due to their position on the screen, how many shaders can be implemented onto a single model is only limited by the screen space. A slight variation in the if statements can allow for many more shaders to be applied to the same model.