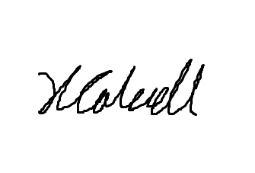


**Graphics Programming Documentation**

|  |  |
| --- | --- |
| **Module Name:** | Graphics Programming |
| **Module Code:** | M3I625657 |
| **Module Leader:** | Bryan Young |
|  |  |
| **Student ID:** | S1920423 |
| **Student Name: Year:** | Harry Calwell 2022 |

**By submitting this assignment, I agree to following statement:**

“Except where stated explicitly, all work in this report, project and accompanying source code, is my own original work and has not been submitted elsewhere in the fulfilment of the requirement of this or any other award”

****  
**Signed:**

**Date:** 08/05/2022

**Video Demo Link:** <https://www.youtube.com/watch?v=9iL769cSIYM>

**GitHub link:** <https://github.com/maryqueenofpox/graphics>

Contents

[2 Introduction 3](#_Toc103016907)

[3 Code Explanation and Implementation 4](#_Toc103016908)

[4 References 9](#_Toc103016909)

**Video Demo Link:** <https://www.youtube.com/watch?v=9iL769cSIYM>

**GitHub link:** <https://github.com/maryqueenofpox/graphics>

# Introduction

The additional graphical technique for this project is the “customShader” class within the build; an inheritance of the Shader.h script which calls the “customShader.frag” and “customShader.vert” scripts to act as the fragment shading and vertex shading respectively. The goal of the shader is to give a textured object a lighting effect that will darken the object and its hue when it is far away from a light source but then increase hue, brightness, and ambient lighting as it gets closer to the light source then at a set distance the object will begin to restore its texture’s original colours. This is to give game objects a readable perspective of distance from light. Listed below are three figures that show how an object will look when it is far from light source, middle distance to light source, and close to light source. The light source for this scene is set to the camera’s position but can be mapped to any game object’s position.

  
*Figure 1*: Banana mesh far from light source. The texture’s colour is completely removed as the model’s hue is shadowed to a dark-orange hue.

A picture containing snack food, close, blurry

Description automatically generated  
*Figure 2*: Banana mesh is middle-distance from light source. The texture’s colour is beginning to reappear as the rest of the model’s hue is gradually increased to a brighter orange colour.

A picture containing indoor, snack food

Description automatically generated  
*Figure 3*: Banana mesh is close to light source. The texture’s colour is restored.

# Code Explanation and Implementation

The code for this shader effect utilises two steps to achieve the outcome. First is calculating light intensity and determining the colour of the model and texture. The second is using ADS lighting techniques to make the light diffuse along the object to produce shadows as the object moves away from the light.

First step is utilising a toon-shading technique that will be responsible for determining light intensity and loading the model and texture colours based on that intensity (Young, 2018) and then mapping a texture to the object and applying the ADS lighting.

In the vertex shader, set up the variables.

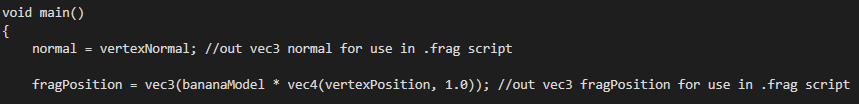
Text

Description automatically generated  
*Figure 4:* Variables set in customShader.vert

The “uniform mat4” variables are set in the MainGame.cpp script. Set up a linkCustomShader() method in the MainGame.cpp script to set these variables.

Text

Description automatically generated  
*Figure 5:* linkCustomShader() method; responsible for mapping specific variables in the vertex and fragment shader scripts.

Next, in the void main() method in the customShader.vert script is used to calculate the “out vec3” variables.  
  
  
*Figure 6:* Out variables set in the main() method in the customShader.vert script

“normal” is the vec3 holding the normal of the vertex. “fragPosition” holds the vertex positions associated with the model.

The final step in the customShader.vert script is calculating the gl\_Position variable. This is a built-in GLSL variable that is used to store the position of the current vertex (WDN Web Docs, 2022). This is calculated using all the “mat4” variables and multiplying them together with the vertex position.  
  
  
*Figure 7:* Calculation for gl\_Position in customShader.vert script

Next step is setting up the customShader.frag, which is the fragment shading part of the custom shader effect. First the variables are set in the script.

Text

Description automatically generated  
*Figure 8:* Variables set in customShader.frag

The uniform vec3 variables of “lightPosition” and “viewPosition” are both set in the MainGame.cpp script. They are the position of the light source and the position of the camera respectively. For this project, it is set to the light source is coming from the camera position so is set in the “linkCustomShader()” method from the MainGame script as follows.

  
*Figure 8:* linkCustomShader() method; responsible for mapping specific variables in the vertex and fragment shader scripts.

The other variables are used in calculating the final colour values of the model. “specularStrength” is the float value of how strong the specular light is along the model and the “sampler2D texture” is the texture of the object. The “vec3 fragPosition” and “vec4 normal” are two values set in the customShader.vert script that are used in the .frag for calculations. The final lighting effect is set as the “out vec4 fragColor”.

The next calculations are all used in the void main() method within the customShader.frag script, they are used to calculate the ADS shading effect which takes a sum of three calculations to achieve the effect: diffuse lighting, ambient lighting, and specular lighting. Diffuse lighting is how much light dims the further away it is from the origin of the specular light. Specular light is like a spotlight effect, it’s the bright centre that represents the origin at which the light source hits the object. Finally, ambient light is the amount of brightness that the object has at parts of the model that are not in the direct location of the specular light (Vries, 2014).

Inside the void main() function of the customShader.frag script, set up calculations to determine the light direction and the normal of the fragPosition. These will be used to calculate the diffuse lighting as well as the light intensity that will spawn the texture colour.

  
*Figure 9:* Light direction and normal position calculations in void main()

Then calculate the diffuse light for the model.

Text

Description automatically generated  
*Figure 10:* Diffuse calculation for ADS lighting

The float for the diffuse amount is the strength of the diffuse light. It is calculated through the product of the normal position and light direction, the value set to “0.2” is how strong you want the diffuse lighting to be, a higher value will produce more brightness throughout the model. The diffuse lighting is then converted to its vec3 value by multiplying by the value of white light (1.0, 1.0, 1.0.)

Next set the value for ambient light. This is how bright you wish the non-luminated part of the model to be. For this example, it is set to a value of (0.1, 0.1, 0.1). This can be increased if the shadowed parts of the model are to be brighter.

Text

Description automatically generated  
*Figure 11:* Ambient calculation for ADS lighting

Then Calculate the specular light on the model.

Text

Description automatically generated  
*Figure 12:* Specular calculation for ADS lighting

Specular light is calucalted through the product of the normal of the view of the player to the fragment position with the reflection of the light direction to the normal positions. The final float value of “specularAmount” determines the strength of the point light that hits the object. The final value of “vec3 specularLight” is calculated by multiplying the two previously calculated values with the vec3 value of white light.

To get the final ADS result, the sum of the three lighting types is calculated. The diffuse light, the ambient light, and specular light are added together.

Text

Description automatically generated  
*Figure 13:* Final calculation for ADS lighting effect

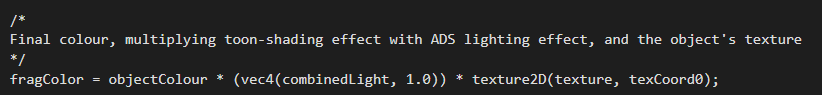
Next step is to calculate the amount of shading based on the light intensity to achieve the toon-shading aspect of the model.

Text

Description automatically generated  
*Figure 14:* Ambient calculation for ADS lighting

Light intensity is the scaling distance of the light source to the normal position. The “objectColour” is how bright the object will be based on this intensity. For this example, at a high intensity the light will be a bright yellow colour to match the banana model; at a low intensity the lighting will be dark orange.

The final step is then to multiply the ADS effect with the “objectColour” toon-shaded effect as well as the texture of the object.

  
*Figure 15:* Final fragColor achieved by the product of both effects with the texture of the model

# References

MDN WEB DOCS, 2022. GLSL Shaders [online]. MDN Web Docs. [viewed 1 May 2022]. Available from: <https://developer.mozilla.org/en-US/docs/Games/Techniques/3D_on_the_web/GLSL_Shaders>

VRIES, J., 2014. Basic Lighting [online]. Learn OpenGL. [viewed 1 May 2022]. Available from: <https://learnopengl.com/Lighting/Basic-Lighting>  
  
YOUNG, B., 2018. LAB2: Toon Shading. 26 February 2018. [viewed 10 February 2022].