

Coursework

Graphics Programming

|  |  |
| --- | --- |
| **Module Name:** | Graphics Programming |
| **Module Leader:** | Bryan Young |
|  |  |
| **Student ID:** | S1918159 |
| **Student Name:** | George Gordon Guthrie |

*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*.

*Signature*. G.Guthrie

Contents

[1. Shader Explanation 3](#_Toc102936773)

[2. Code Explanation 4](#_Toc102936774)

[3. References 8](#_Toc102936775)

Github link: https://github.com/Georgeguthrie/Graphics-Programming

**Figures**

Figure 1 (Page 3): ADS Shader Example (Wolff, 2012)

Figure 2 (Page 3): Ambient math Equation (Wolff, 2012)

Figure 3 (Page 3): Diffuse math Equation (Wolff, 2012)

Figure 4 (Page 4): Vector of pure reflection equation (Wolff, 2012)

Figure 5 (Page 4): Specular math equation (Wolff, 2012)

Figure 6 (Page 4): Final ADS shading equation (Wolff, 2012)

Figure 7 (Page 4): ADS vert main code

Figure 8 (Page 5): Variables of the vert

Figure 9 (Page 5): Ambient component code

Figure 10 (Page 5): Diffuse component code

Figure 11 (Page 5): Bugged ADS shader

Figure 12 (Page 6): Specular component code

Figure 13 (Page 6): Creation of the ADS shader

Figure 14 (Page 6): Variables of the frag

Figure 15 (Page 7): Function used to write to uniform variables

Figure 16 (Page 7): Final ADS Result

# 1. Shader Explanation

The shader added to the project was an ADS (Ambient, Diffuse, and Specular) shader. This shader works as a combination of three different parts. The ambient will simulate a light that has been reflected so much that it gives the feeling of coming from all sides. The diffuse handles the reflection on the object from multiple directions and finally the specular which will model how shiny the surface of the model is as well as handling reflections on the object from a specific direction (Wolff, 2012). All these parts are then combined to create the final ADS shader, an example of which can be seen below.

Logo

Description automatically generated

Figure 1: ADS shader example (Wolff, 2012)

To go into further detail. The ambient part of the shader will light up all surfaces of the object equally and reflect the light equally in every direction and can be created using the following mathematical equation.

A picture containing text, furniture, seat, table

Description automatically generated

Figure 2: Ambient math equation (Wolff, 2012)

Essentially, the ambient can be found by multiplying the light source intensity by the reflectivity of the surface (Wolff, 2012). The equation for finding the diffuse is very similar to the ambient equation with the addition of the dot product of the surface normal and vector towards the light source (Wolff, 2012).



Figure 3: Diffuse math equation (Wolff, 2012)

The specular is more complicated to find. Firstly, the vector of pure reflection must be found which can be calculated with the below equation. Where s is the light vector and n is the surface normal (Wolff, 2012).



Figure 4: Vector of pure reflection equation (Wolff, 2012)

After that a similar equation is again used with the light source intensity and reflectivity being multiplied by the dot product of the pure reflection and the vector towards the viewer raised by a power that represents the shininess of the object (Wolff, 2012).



Figure 5: Specular math equation (Wolff, 2012)

Finally, the result of the ambient equation, diffuse equation and specular equation are added together to provide the final ADS shading equation (Wolff, 2012).

Text, letter

Description automatically generated

Figure 6: Final ADS shading equation (Wolff, 2012)

# 2. Code Explanation

The code for the frag and the vert of the ADS shader was created following the tutorial found at <https://learnopengl.com/Lighting/Basic-Lighting>. To begin with, the vert is used to find the position of the object the shader will be attached to in relation to the camera using the transform of the model as well as the view and projection of the camera. The vert is also used to find the fragment’s position vector for use in the diffuse and specular code as well as the location of the object in the world space for the diffuse code. A change was made to the code from the tutorial. Instead of finding the location of the object, the tutorial code originally found the normal vectors. This was changed to fix a bug that was encountered that will be explained later.

A screenshot of a computer

Description automatically generated with medium confidence

Figure 7: ADS vert main code

For the variables of the vert. The fragment’s position vector and the location of the model are pushed out to be used in the frag of the shader. The projection, view and model variables used in finding the position of the object in relation to the camera view are set to uniform variables that are written to in the link function located in MainGame.cpp.

Text

Description automatically generated

Figure 8: Variables of the vert

The code of the frag is separated into sections for calculating the ambient, diffuse and specular then a final section for combining them. To start with the ambient code, a float is declared for setting the strength of the ambient component. This strength is then multiplied by the colour of the light source to create the ambient component.

Text

Description automatically generated

Figure 9: Ambient component code

For the diffuse component, a new vec3 variable is declared that holds the normalization of the position variable that was calculated in the vert of the shader. Next the light direction is found by the normalization of the position of the light source – the position of the fragments that was calculated in the vert. The dot product of the normalization of the position and the light direction is then calculated and multiplied by the colour of the light source to create the diffuse component.

Text

Description automatically generated

Figure 10: Diffuse component code

Another change was made here, as mentioned previously the tutorial code found the normalization of the normal vectors instead of the position variable causing the bug seen below.

A picture containing colorful

Description automatically generated

Figure 11: Bugged ADS shader

A similar bug was found in one of the other shaders in the project and was fixed using the same method used for the other shader, changing the code to find the normalization of the position of the model instead of the normalization of the normal vectors.

Lastly, for the specular component, a new float is declared for setting the strength of the specular component. The viewing direction is then found by the normalization of the position of the camera – the fragment position vector. Next, the direction of the reflection is found by reflecting the negative of the direction of the light by the normalization of the position. The dot product of the viewing direction and the reflection direction is calculated and raised by a power of 32 representing the shininess of the object. Finally, this dot product is multiplied by the specular strength and light colour to create the specular component.

Text

Description automatically generated

Figure 12: Specular component code

After these three components are created, they are finally added together and multiplied by the colour of the object to create the final ADS shader.



Figure 13: Creation of the ADS shader

For the variables of the frag. The position and fragment position variables are read in from the vert code. The colour of the light and object as well as the positions of the light source and the camera are set to uniform variables that are written to in the link function located in MainGame.cpp.

Text

Description automatically generated

Figure 14: Variables of the frag

In the MainGame.cpp script, a new function is then created called “linkADS” this function will handle writing all the required uniform variables that are used in the frag and vert of the shader. These include setting the colour of the lighting and the object, the position of various objects in the scene such as the light source, camera and object and the projection and view of the camera.

Text

Description automatically generated

Figure 15: Function used to write to uniform variables

This function will be called in the drawGame function in the same way as the other two shaders to result in the final product shown below.

A picture containing grass, tree, outdoor, field

Description automatically generated

Figure 16: Final ADS result

# 3. References

WOLFF, David., 2012.  *The Basics of GLSL 4.0 Shaders* [online]. GameDev. 27 January. [viewed 30 April 2022]. Available from: <https://www.gamedev.net/tutorials/_/technical/opengl/the-basics-of-glsl-40-shaders-r2861/>

DE VRIES, Joey., 2014. *Basic Lighting* [online]. Learn OpenGL. [viewed 2 May 2022]. Available from: <https://learnopengl.com/Lighting/Basic-Lighting>