Adam Stafford

Graphics PRogramming Coursework

***S1512672***

*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*. Adam Stafford

Contents

[1.1 MainGame() 2](#_Toc514250800)

[1.1.1 Game Controls 2](#_Toc514250801)

[1.1.2 MainGame() Structure 2](#_Toc514250802)

[1.1.3 drawGame() structure 3](#_Toc514250803)

[1.2 Shader Class 3](#_Toc514250804)

[1.2.1 Shader Uniforms 3](#_Toc514250805)

[1.2.2 Initialization 4](#_Toc514250806)

[2.1 Rim Lighting Shader 4](#_Toc514250807)

[2.2 ToonRim Lighting Shader 6](#_Toc514250808)

[2.3 ToonRim Fog Lighting Shader 7](#_Toc514250809)

[2.4 Phong Lighting Shader 8](#_Toc514250810)

[2.5 Blinn-Phong Lighting Shader 10](#_Toc514250811)

[Appendix 12](#_Toc514250812)

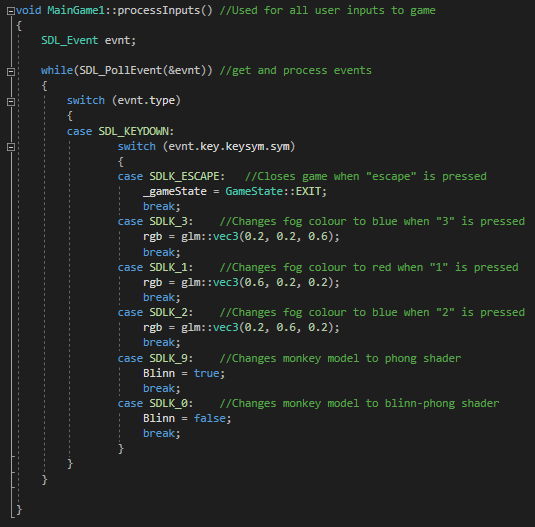
[Models 12](#_Toc514250813)

[Tutorials 12](#_Toc514250814)

# MainGame()

The MainGame class is a vital feature of the script that is used in several different ways in the running of the game. Majority of these features are in the drawing of the games meshes and shaders. It is also where the player controls are initialized and used while the game is running.

## 1.1.1 Game Controls

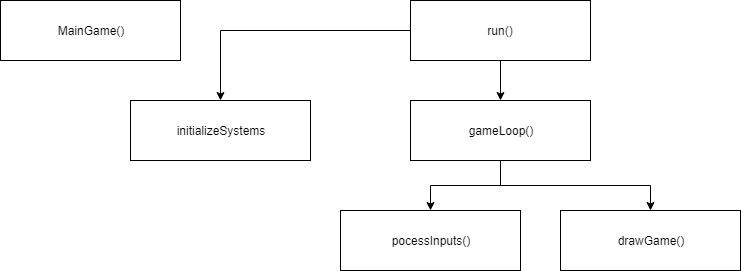
This script is used to inputs from the keyboard to change a number of different aspects with the games graphics. The controls are as follows:

Figure

|  |  |
| --- | --- |
| **Button** | **Action** |
| Esc | Exits Game |
| 1 | Changes fog colour to red |
| 2 | Changes fog colour to green |
| 3 | Changes fog colour to blue |
| 9 | Changes monkey model to phong lighting shader |
| 0 | Changes monkey model to blinn-phong lighting shader |

Table 1

## 1.1.2 MainGame() Structure

From this we can see the MainGame() method which is where all of the pointers for my game are created. In the run() method we can see that it branches to the initializesystems() method and the gameLoop() method. initializesystems() is where all of the models and shaders are loaded for later use. While gameLoop() has two more methods called processInputs() and drawGame(). Process inputs is shown in the control section and is just the inputs for the game where draw game is where each model and shader is drawn.

Figure

## 1.1.3 drawGame() structure

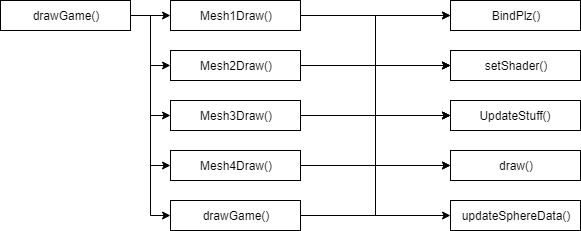
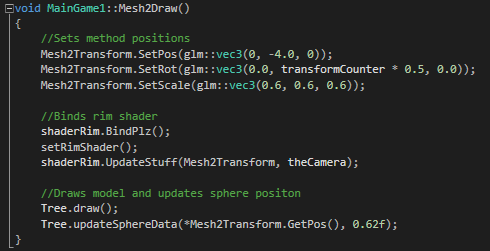
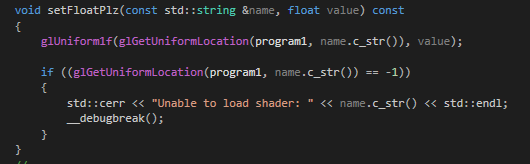
In the drawGame() method we can see that there is a method to draw each individual model in the scene. We can also see that in each method are the methods to bind a shader to a model, set the shader values, update the shader, draw the model and update the sphere data in relation to the model.

Figure 4

Figure 3

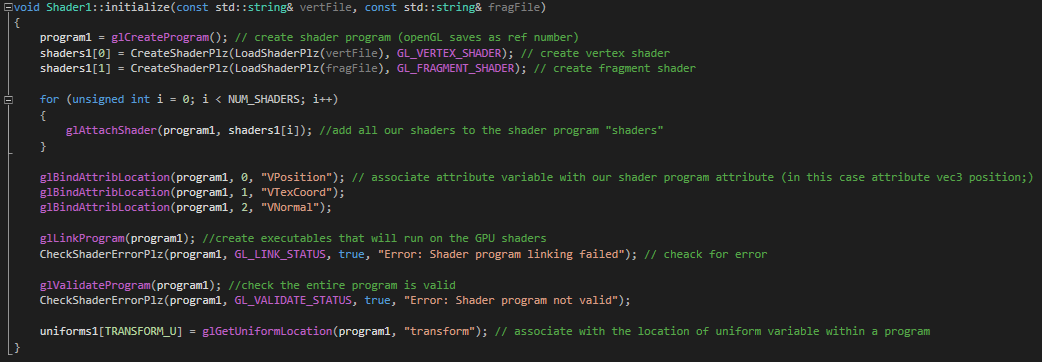
# Shader Class

## 1.2.1 Shader Uniforms

For shaders to render they need to be linked using glUniform. The glUniform method works by setting the values of uniforms within a shader. This is an example of a version of this method used to set a float. There are a multitude of different versions of this ranging from, Booleans and integers to vector 3’s and matrix 3’s.

Figure

## 1.2.2 Initialization

Another method required is the initialize method. This method is used to load shaders that utilize vertex and fragment files. For a shader to be loaded the vertexes and fragments need to be attached to the programme by binding them to attribute locations so that can later be used in the vertex shaders.

Figure

# 2.1 Rim Lighting Shader

The first shader featured in my game that I am going to explain is the “Rim Lighting Shader”. Rim lighting, also known as back-lighting is a lighting effect that creates a light which bleeds around the object from a light source, placed behind it, having no influence of the shading on the surface of the model. Due to this bleeding effect the name rim lighting was given due to the rim of light that goes around the outline of an object.

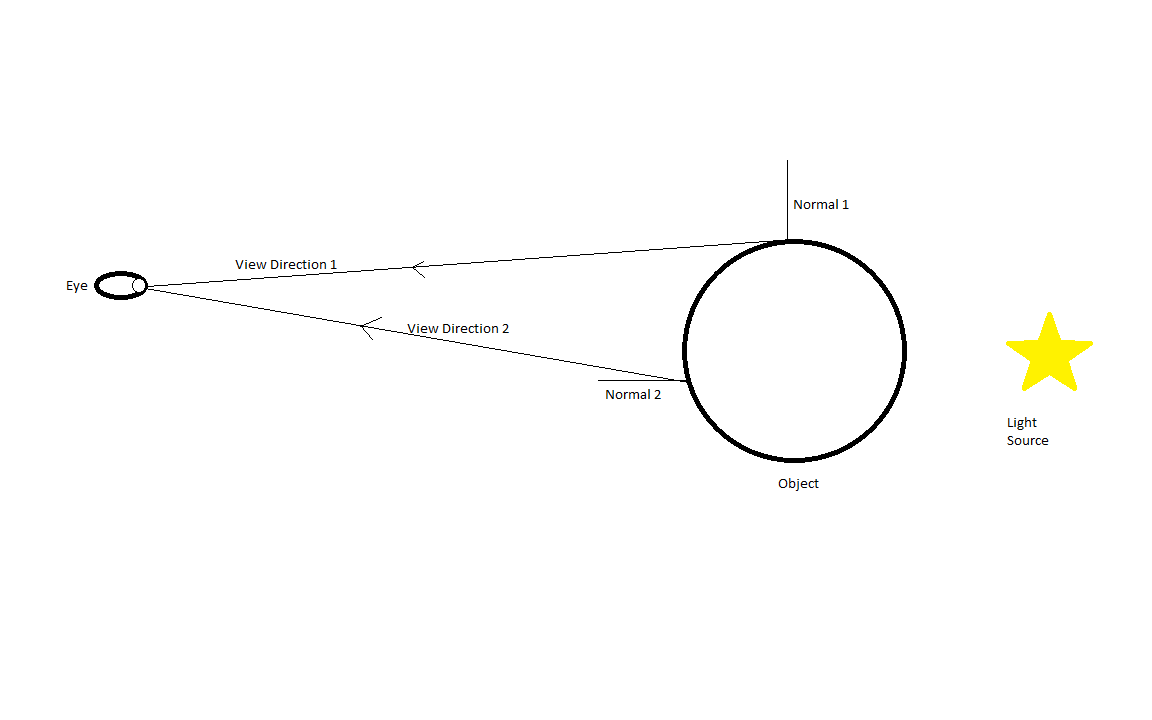
Rim shading takes in a surface normal and a view direction, when the view direction is face on to the surface model the view vector from this will be collinear, this is when Rim lighting will be shown least. However, when the view direction just glances the surface the view vector will almost be perpendicular, this is when rim lighting is at its greatest. We can then take the dot product between the two vectors. A value that

Figure 7

should be one providing the two vectors are collinear. Then as the two vectors become orthographical the dot product’s value will decrease becoming closer to zero. Thus rim light can be created by taking this dot product and making the intensity of the rim light inversely proportional to it.

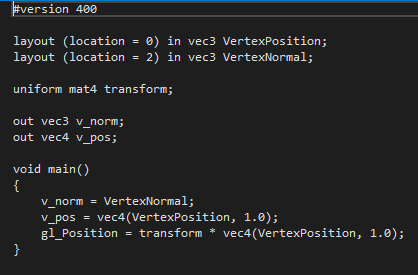
For Rim we can see here in the vertex shader that we are taking in the vertex positions and vertex normal. These are then set to new values and the vertex position is set as a vector 4. These are then both outputted for the fragment shader to later use.

Figure 8

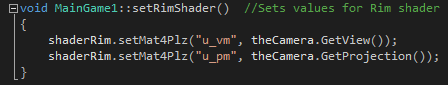
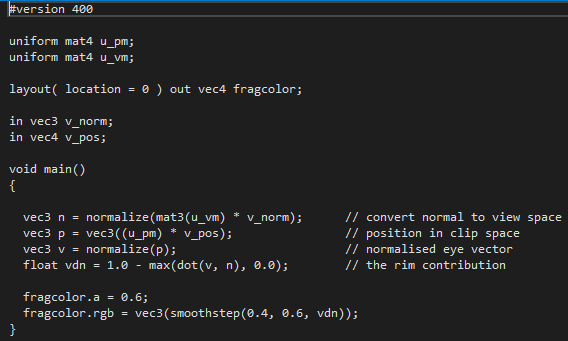
In the aforementioned main game class, we can see where we set the shader uniforms for the fragment shader. These are then set to the direction the camera is facing and the projection matrix of the camera. These are now ready to be used when the shader is being called in the game loop. In the fragment shader we can see that the shader is calling for the matrixes set in the main game class and it is inputting the vectors from the vertex shader. The main method then normalizes the projection matrix after it has been multiplied with the vertex normal from the vertex shader to convert the normal to view space. This is then added to the vector 3 “n” for future use. Then the vector 3 “p” is then set to the projection matrix multiplied by the vertex normal, this is the position of the object in the clip space. Then “v” is set to the normalized “p”, to give us a normalized eye vector. Then a float called “vdn” is set to 1 minus the previously discussed dot product of “v” and “n”, this acting as the rim contribution. Then mesh shader is ready to be drawn to the game.

Figure 10

Figure 9

# 2.2 ToonRim Lighting Shader

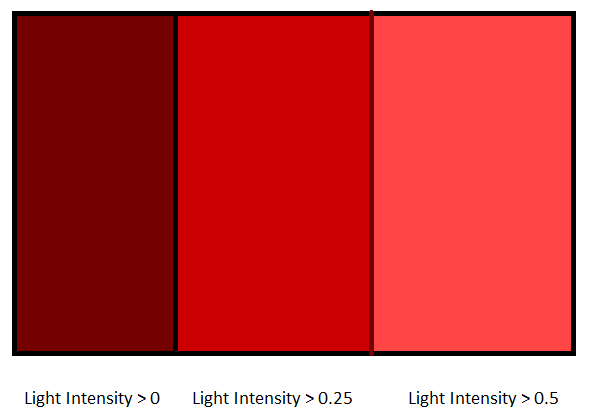
The next shader in the game that I am going to discuss is the “ToonRim Lighting Shader”. Toon Rim lighting is a combination of both “Toon lighting” and “Rim lighting”. Toon lighting is a shader that makes a game look cell shaded, mimicking the style of cartoons and comic books. It does this by using less gradient shading and utilizes more solid colours, thus giving the illusion of being hand drawn.

Figure 11

Toon is achieved by having different parts of the object be different colour depending on the intensity of the light on that part of the object. Thus ToonRim is a combination of toon lighting and rim lighting, creating a toon shading that has the bleeding effect from Rim lighting to create an outline of light around an object.

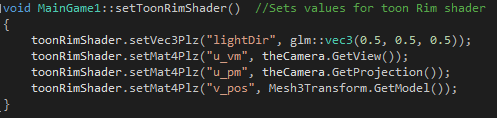
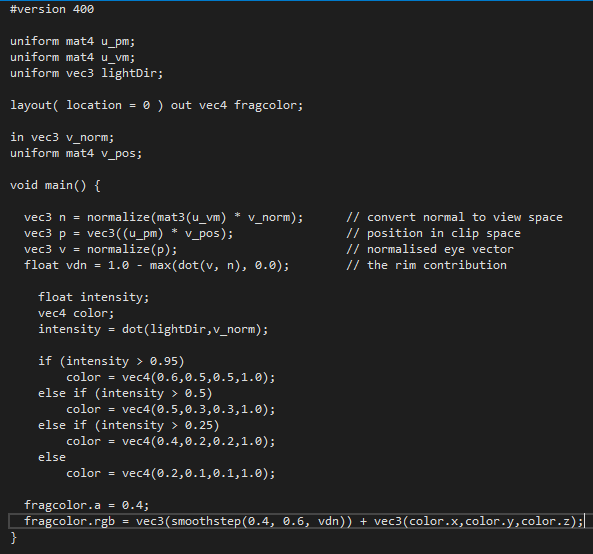
The vertex shader for ToonRim is identical to Rim as it uses the same inputted vectors as rim does, thus the same [vertex code](#_Rim_Lighting_Shader) can be used in relation to ToonRim. In the main game class, the setting of the required uniform values for the fragment shader is also very similar to Rim lighting with some minor additions. As we can see we have to set values for 2 new uniform variables “lightDir” and “v\_pos”. Light direction is quite self-explanatory as it is the direction that the light on the shader object is coming from. The “v\_pos” is get the location of the model so that the light effect is updated as the model moves around the scene.

Figure 12

The start of the ToonRim method is nearly the same as [Rim Lighting](#_Rim_Lighting_Shader) as it is the first stage in achieving the required shader. The next stage in the code though is where all of the features of toon are demonstrated. Toon starts by creating a float called intensity that is for the light intensity and a vector 4 called color to set the object colour. The intensity float is then set to the dot product between the light direction from main game and the vertex normal. Then a basic decision tree is created using the intensity float and the color vector to decide what colour a section of the object will be in relation to the level of light intensity. Finally, at the end of the shader the Rim values and the toon values are combined to create the “ToonRim Lighting Shader”.

Figure 13

# 2.3 ToonRim Fog Lighting Shader

The next shader in the game that I’m going to review is the fog shader. The fog shader is quite self-explanatory in what it does in the sense that like foggy weather it restricts how well the user can see an object depending on how far away the object is from the person. This is achieved by mixing the colour of each fragment with a consistent fog colour, with the strength of the fog being determined by its distance from the camera.

The vertex shader for fog is the same as the previous 2 [shaders](#_Rim_Lighting_Shader) with no differences and thus will not be discussed. In the main game class, the setting of the shaders uniforms is once again quite different from the previous ones.

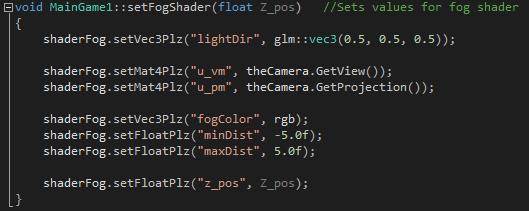
As we can see it is similar to the previous method with some additions. The first of which is the “fogColor” uniform. This uniform determines the colour of the fog in the fragment shader and is set to a vector 3 as [shown previously](#_MainGame). The next uniform is a float called “minDist” this float determines the minimum distance in which the fog will affect the object. The next float “maxDist” is the same as “minDist” except that it determines the maximum distance that the fog will affect the object. Finally, the last float shown is the “z\_pos” float. This float is set to a float that is determined when calling the method and will be used to show the z position of the object and an offset for the fog.

Figure 15

Figure 14

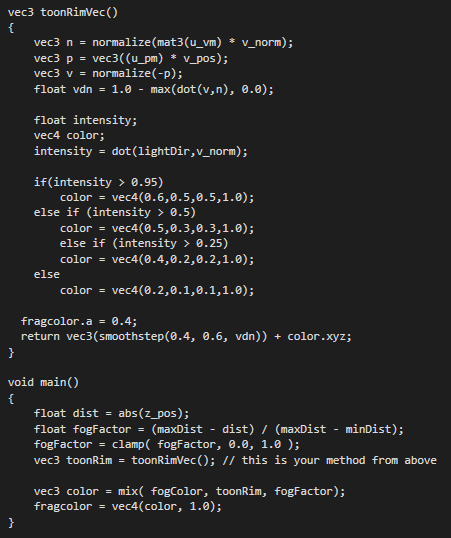
The start of the fog fragment shader is very similar to the previous ones at is just the inputting of values from the vertex and calling uniform values.The main section of the shader starts be returning a vector 3 that is exact same what was outputted from ToonRim expect that it doesn’t change the final fragment colour to anything. The main method of the fragment then creates a float called “dist” which is set to the absolute value of the z position of the object the shader is bound to. A float called fogFactor that is the max distance minus “dist” divided by max distance minus the minimum distance. The value is then constrained between a certain value. After that a vector 3 called “toonRim” is set to the output from the ToonRim vector. A vector 3 called color is then set to a mix of the fog color vector, the toon rim vector and the fogFactor float. This is then set to a vector 4 and outputted to be drawn in the game.

Figure 16

# 2.4 Phong Lighting Shader

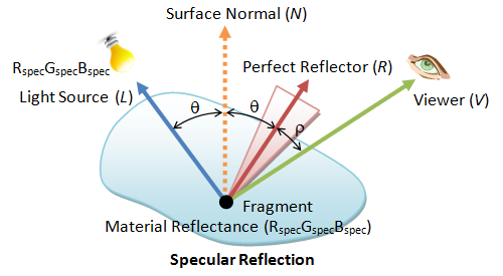
The next shader in my game I am going to discuss is the “Phong Lighting Model”. This model is a common lighting model that has been used in industry for a number of years. Phong is the combination of three different lighting properties known as, Ambient light, Diffuse light and specular light.

Figure 17

Ambient light is light that is directionless as while it came from somewhere it has bounced so much around a room that it has become directionless. Ambient light is also evenly lit on all surfaces on all directions.

Diffuse light is directional light that comes from a distant light source. The light is reflected off of a surface and then dispersed equally in all directions, appearing the same to all people viewing it with no relation to the viewer’s position.

Specular light, like Diffuse light is directional light that interacts more sharply with a surface. Specular light also creates a bright spot on objects called the specular highlight.

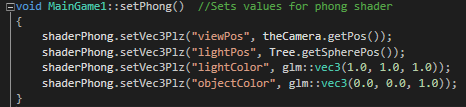
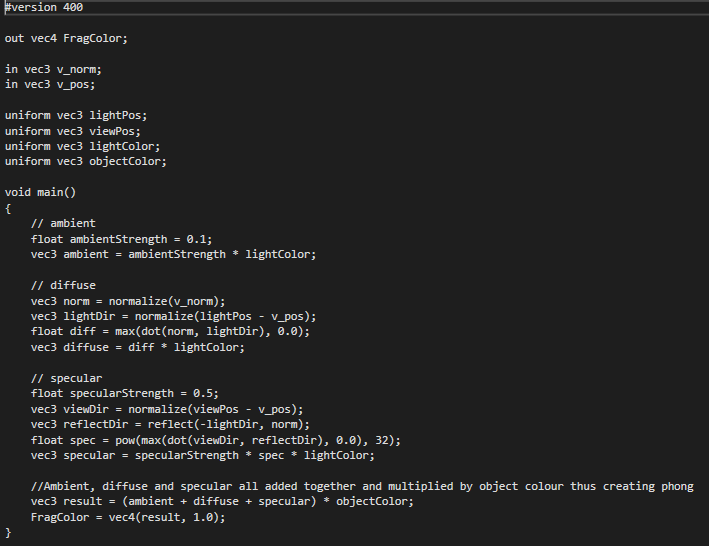
The Phong Shaders vertex fragment is the same as the [previous shaders](#_2.1_Rim_Lighting) and thus needs no further explanation. The setting up of the shaders uniform values is quite different from the previous ones as we are using some previously unseen values. The fist value is called “viewPos”. This is a vector 3 that is set to the position of the camera. The next value is “lightPos”, this value is a vector 3 that I set to the position of a tree in the scene, setting it as the light source. The “lightColor” value is a vector 3 for the colour of the light coming from the light source. The final value “objectColor” is the colour of the object that the shader is going to be affecting.

Figure 18

In the main method of the fog shader we can see that we start by getting the ambient light value. This is done by setting a float called “ambientStrengh” to equal 0.1. We then created a vector3 called “ambient” and set it to equal the ambientStrengh multiplied by the lightColor.

Then we calculate the diffuse lighting value, first by normalizing the vertex normal and setting it to a vector 3 called “norm”. After which we subtracted the vertex position from the light position and normalized it, setting to the vector 3 “lightDir”. We then created a float called “diff” and set it to greater of 2 values that include, the dot product of “norm” and “lightDir” and the zero. Finally, we then create a vector 3 called diffuse and set it to diff multiplied by lightColor. Thus getting the diffuse value.

The next stage involves getting the specular value, the first step of which involves setting a float called “specularStrengh” to 0.5. We then subtract the vertex position from the view position and normalize it to get the vector 3 called “viewDir”. We then calculate the reflection between the inverse of the light direction and the normal and set the vector 3 “reflectDir” to equal the result. After this we set the float “spec” to return the maximum value of the dot product between view direction and reflect direction and 0.0, raising it to equal 32. Finally, we set a vector 3 called “specular” to equal “specularStrengh” multiplied by “spec” multiplied by “lightColor”.

Figure 19

The final stage of the fragment involves adding all of the previously calculated lighting effects and multiplying them by the object colour, before setting it to FragColor. Making it ready be outputted and drawn to the game. Sadly, I was not able to get Phong fully working as the light effect does not update as the object moves around the scene, due to the fact I was not able to figure out how to update the vertex position and vertex normal for the object.

# 2.5 Blinn-Phong Lighting Shader

The next shader in my game is the “Blinn-Phong Lighting Shader”. Blinn-Phong lighting can be considered an extension or more optimized version of the [Phong Lighting Shader](#_2.4_Phong_Lighting).

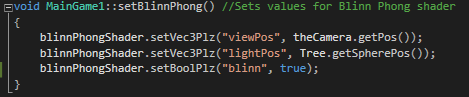
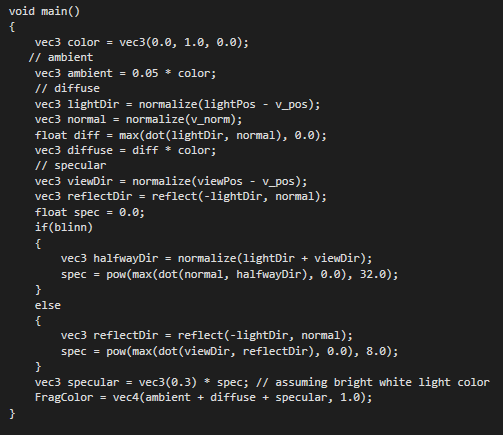
Blinn-Phong uses the same shader as the [previous shaders](#_Rim_Lighting_Shader) and thus needs no more detail. When setting Blinn-Phong in the main game class, there are a few differences from the basic Phong Lighting method. As we can see the “viewPos” and “lightPos” values are unchanged but the rest of the Phong uniforms have been removed and replaced with a Boolean called “blinn” that has been set to true. The fragment shader for Blinn Phong is also extremely similar to Phong, with only a few differences that add to the optimization of the code.

Figure 20

The differences can be first seen with the color been set within the fragment, instead of being a uniform value set in the main game class. The next change is in the calculating of the ambient light as we no longer have a float for the ambient strength and are instead just using numbers. The biggest changes however are in the specular lighting section. The first change being the same as ambient in the sense that we are no longer creating a float to store the specular strength and are instead just setting a number. Another change can be seen with before in normal Phong we would calculate the direction of the reflection between the light direction and the normal we instead are calculating the vector 3 called “halfwayDir” which is the result of the light direction and the view direction been added together and normalized. We then set spec to return the maximum value of the dot product between the normal and “halfwayDir” and 0.0, raising it to equal 8.0. We can also see that we have the original code for normal phong as well in this that can be switched between the 2 by changing the bool. Before resuming to normal and working in the same way as normal Phong.

Figure 21

# Appendix

## Models

3dregenerator, 2011. *Cat Model.* [Online]   
Available at: https://free3d.com/3d-model/cat-95027.html  
[Accessed may 2018].

Andonhh, 2016. *Oragami twitter Bird.* [Online]   
Available at: https://www.cgtrader.com/free-3d-models/animals/bird/free-origami-twitter-bird  
[Accessed May 2018].

aspire, 2017. *Low Poly Tree.* [Online]   
Available at: https://free3d.com/3d-model/low-poly-tree-18385.html  
[Accessed May 2018].

tyrosmith, 2013. *Low Poly Rock.* [Online]   
Available at: https://free3d.com/3d-model/low-poly-rock-4631.html  
[Accessed May 2018].

## Tutorials

Vries, J. d., 2014. *Advanced Lighting.* [Online]   
Available at: https://learnopengl.com/Advanced-Lighting/Advanced-Lighting  
[Accessed May 2018].

Vries, J. d., 2014. *Basic Lighting.* [Online]   
Available at: https://learnopengl.com/Lighting/Basic-Lighting  
[Accessed May 2018].