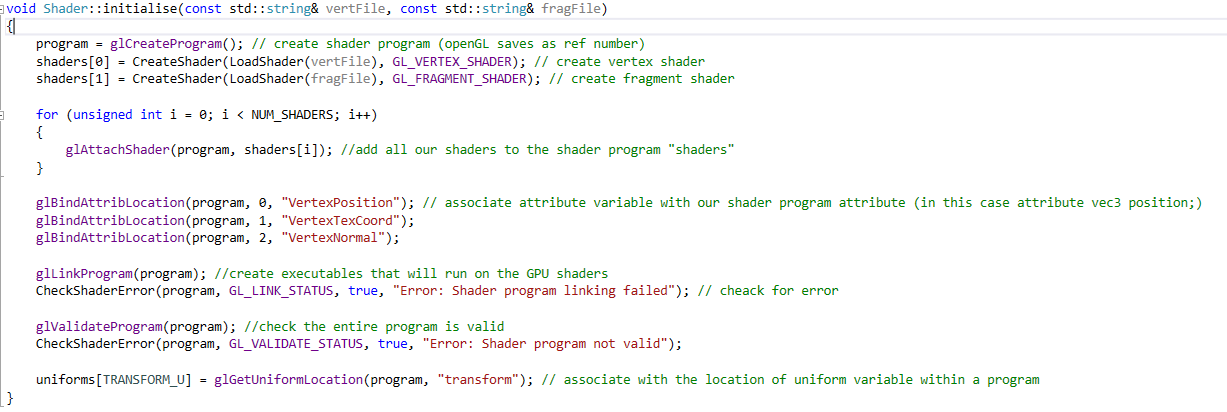
John Reid S1511280

[Company name]  [Company address]

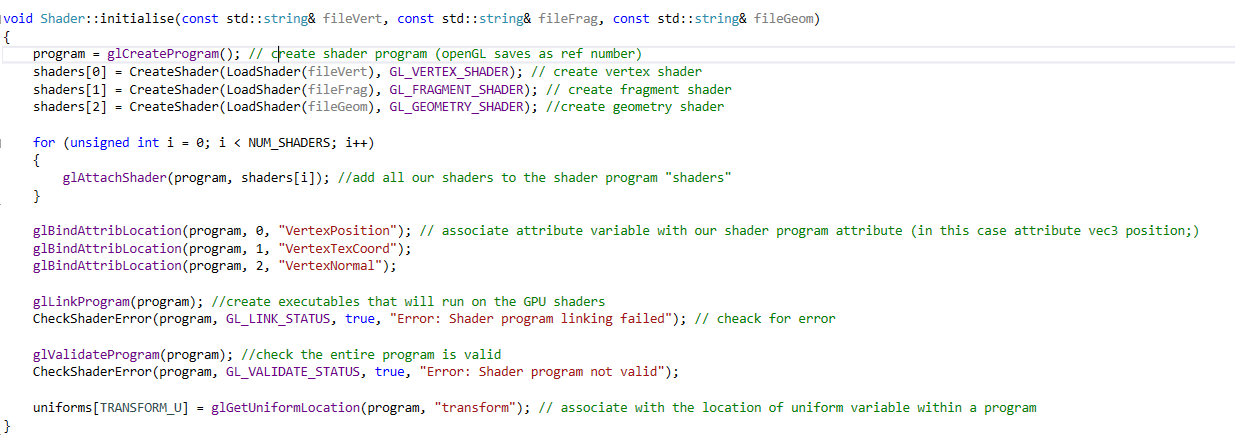
Graphics Programming document

# Shader Header and .Cpp files

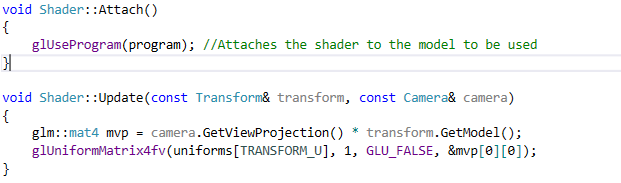
Both the header and cpp shader files are important to the project as they are what is used to create and load the shaders, so the GPU can use them.



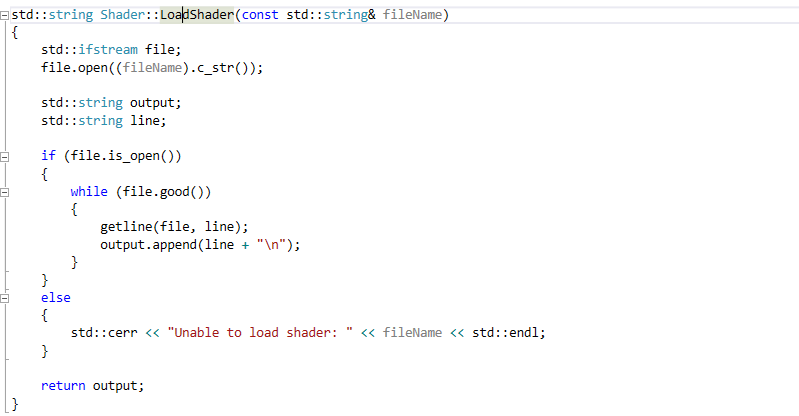
The initialise method is essentially used to load the shader file that is to be used. It takes in two strings that will be a file path to the vertex and fragment shader. The method begins by creating a new shader program and then creating the vertex and fragment shaders using the strings that were passed in. The shaders are then added to the shaders program. The position, texture coordinates and normal for the model are then added to the shader program attribute. Executables are then created that will run on the GPU. Error checks are then ran to ensure that everything is valid.



The following initialise method does the exact same thing but takes in a geometry shader as well as the vertex and fragment shader.



The attach method simply tells the GPU which shader should be used when processing. The Update shader is used to update the shaders that are attached to the models using the transform of the model and the cameras position.



The LoadShader method takes in a string which is a file path to the shader, if the path is valid it will return the file path and will return an error if the file path is not valid.

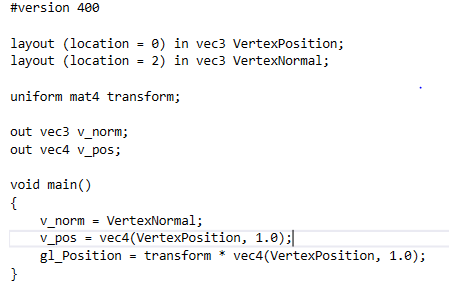
# Shaders

## Phong Lighting Model

The Phong lighting model is a lighting model that uses three lighting components, ambient, diffuse and specular lighting and adds the results of them together to achieve a much more advanced lighting model.

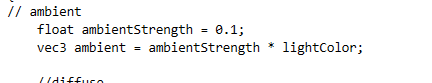
### Vertex Shader

The vertex shader used in this shader is fairly simplistic:



Firstly, we need the Position and Normal of the vertices for calculating specific lighting components that will me mentioned later. Because we need these we must pass them to the fragment shader as that is where all the calculations take place.

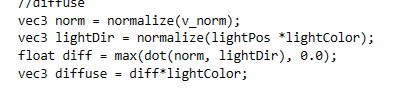
### Ambient Lighting

The ambient lighting is the simplest to implement as it is directionless, so light direction does not matter as it will be evenly lit on all surfaces regardless of position. 

When creating the ambient light we take the take the objects colour and multiply it by a small number is treated as the strength of the light.

### Diffuse Lighting

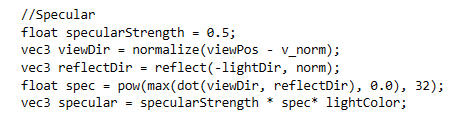
Diffuse lighting is the second component needed to create the Phong lighting model:



Diffuse is different from ambient as the position of the light matters when creating the light so a vector3 has to be created that will represent the position of the light. Next, the normal of the vertices that we passed in from the vertex shader and normalise it. It is normalised it to ensure that we are working with unit vectors. To calculate the light direction, the light position is multiplied by the light colour and normalise it. Once all this has been done the dot product of the normal of the vertices and the light direction are needed to determine the lighting impact on each of the vertices which the is multiplied by the objects colour to give us the diffuse lighting.

### Specular Lighting

Specular lighting is the third and final component of the Phong lighting model:



For this lighting component the camera position is needed and a float that will determine the strength of the specular light is needed for calculations later. Firstly, the view direction is calculated by taking the normal away from the view position and the reflect direction is calculated by reflecting the negative of the light direction away from the normal. The specular lighting is then completed by calculating the lighting component with a shininess factor and is completed by multiplying the result with the specular Strength float with the object colour

### Creating the Phong Model

To create the Phong lighting model the three lighting components need to be combined together:



The final result is the three finished components of each lighting component added together multiplied by the objects colour.

The result is then outputted as the fragment colour which will then create the Phong lighting model on the object the shader is attached to.

### Set-Up

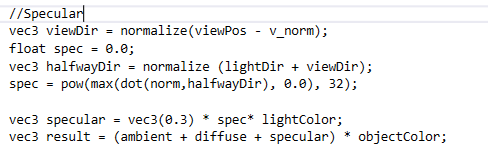
When linking the shaders some set-up is required:



The object colour is set to a vector3, the light colour is also set to a vector3, the light position is set to the position of the light source in the scene and the view position is set to the position of the camera.

## Blinn-Phong Lighting Model

The Blinn Phong lighting model is an extension of the Phong lighting model. The BlinnPhong model is very similar to the Phong model, the only difference is that the Blinn Phong model uses a halfway vector between the view direction and the light direction instead of using a reflection vector when calculating the specular lighting:



The major difference between the two lighting methods is that the Blinn Phong method uses the halfway vector instead of the reflection vector which only affects the angle as with the Phong method the reflection angle can’t go higher than 90 degrees whereas the Blinn Phong method can which creates more realistic lighting.

### Set-Up

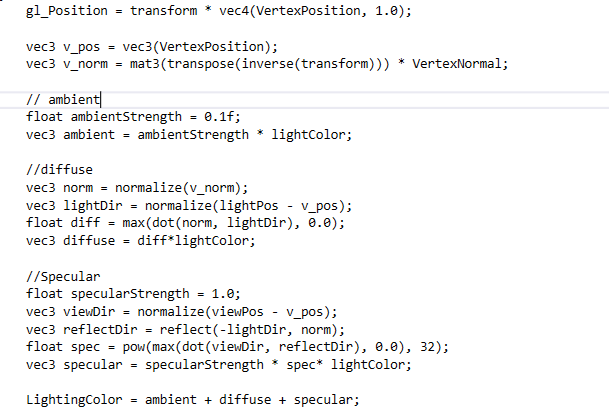
When linking the Blinn-Phong shaders some set-up is required:



Since the blinn-phong shaders are almost exactly the same the set-up for both are the same.

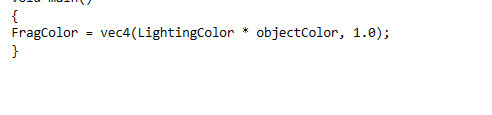
## Gouraud Lighting Model

The Gouraud model again is similar to the Phong lighting model in that it uses the same lighting components to create the lighting model. The major difference between these two models is that the Gouraud technique does all the calculations in the vertex shader whereas the Phong model does everything in the fragment shader:



From the above code it shows that the two methods are very similar in that they both use ambient, diffuse and specular lighting to create their lighting models. They are all calculated the same way as they would be in the Phong model but the Gouraud technique does this in the vertex shader instead.

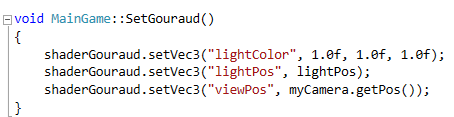
Once the LightingColor variable has been calculated it is then passed into the fragment shader:



Once it is passed in it is set as the FragColor and outputted.

### Set-Up

When linking the shaders some set-up is required:

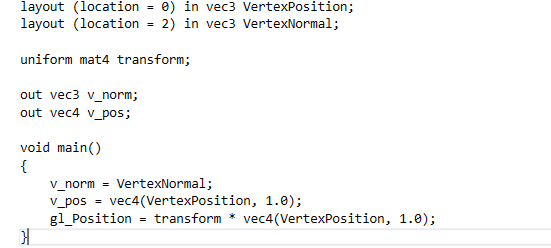


This again is similar to the phong set up only some things are removed as they are not needed.

## Exploding Shader

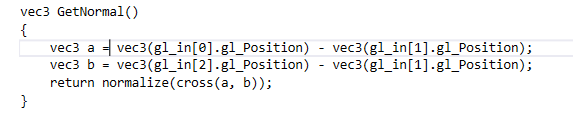
The exploding shader makes use of the geometry shader in which is splits the model into many triangles and moves them simulating something exploding into many pieces.

### Vertex Shader



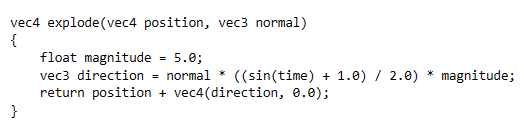
The vertex shader for this shader is exactly the same as the Phong vertex shader in that the vertices have a position and a normal and the gl\_position is calculated the same way.

### Geometry Shader

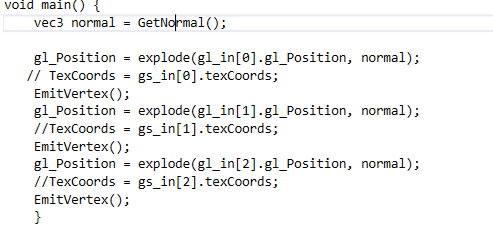
To create this shader the triangles normal vector is needed: 

To calculate the normal a vector that is perpendicular to the surface of the triangle is needed. This is done by using the cross product of a and b which are parallel to the surface of the triangle so the cross product will give is the perpendicular vector that is needed.

The next method needed is the one that will move the triangles simulating the explosion:



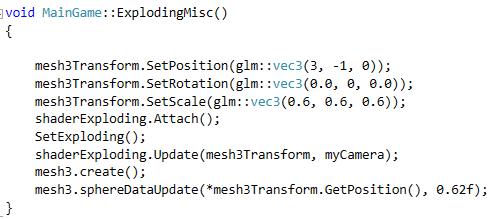
This method returns a vector that is used to translate between the original position and this position to simulate exploding. When calculating the direction we add one and divide as we want a value between 0 and 1 so the model doesn’t implode.

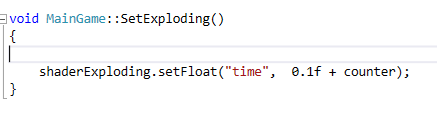


The main method uses the previous two methods discussed to simulate exploding, the normal of each triangle is calculated and is then passed into the explode method to create along with triangle to create vector that the triangle will translate.

### Set-Up

When attaching the shader some things need to be set up:



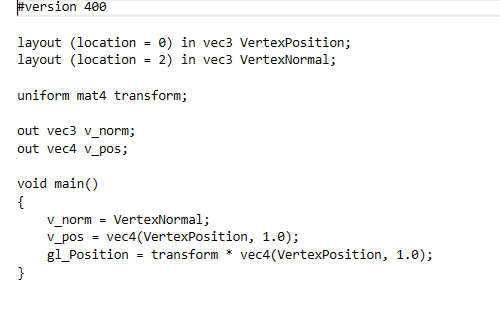


Time needs to be set so the exploding method will work correctly.

## Rim Toon Shader

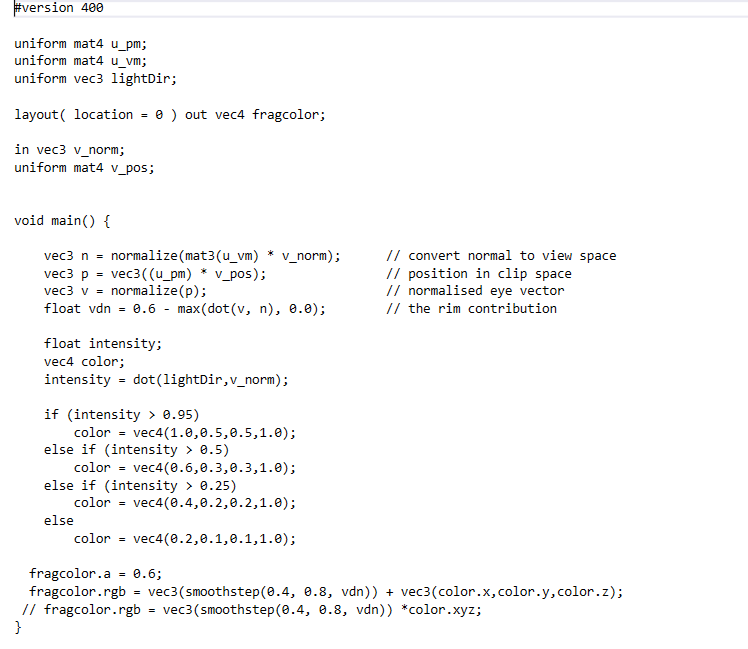
The RimToon shader is used to combine the Rim and Toon effects. Rim shading is a technique that keeps the outside of an object lit and simulates the bleeding of light. Toon shading is a colouring technique that changes colours using full colours.

### Vertex Shader



The vertex Shader is the same as the previous shaders.

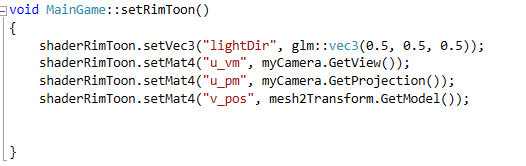
### Fragment Shader



The first 4 lines of code are used to calculate the rim section of technique, firstly, the view matrix is then multiplied by the model matrix then normalised. Next the projection matrix is then multiplied by the model matrix. They are then normalised. The dot product of these two calculations is then calculated and 0.6 is subtracted to ensure that the rim shading is always larger when the angles are larger. The second part of the method is used to calculate the toon section of the shader. The colour of the fragment is determined by the intensity of the light, the intensity is first calculated by using the dot product of the light direction and the vertex normal. The intensity is then ran through the series of if statements to determine the colour. The final lines of code are the combination of the two techniques. The fragcolor is determined by adding the two vectors together. Smoothstep is used in the rim calculation so the colour has a smooth cut-off.

### Set-Up

When linking the shader some set-up is required for the shader to work:

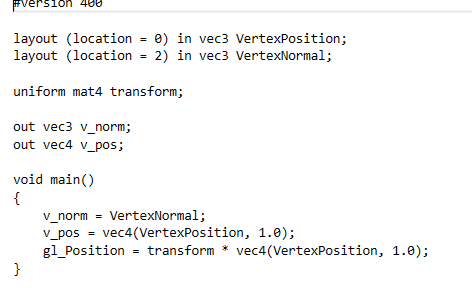


Light direction is set to a vector 3, the view matrix is set to the cameras view matrix, the projection matrix is set to the cameras projection matrix and the position is set to the models transform.

## Fog Shader

The fog shader is used to simulate a fog effect, the further the model is from the camera the darker and more fog like it will look and the closer it is the more details you can see.

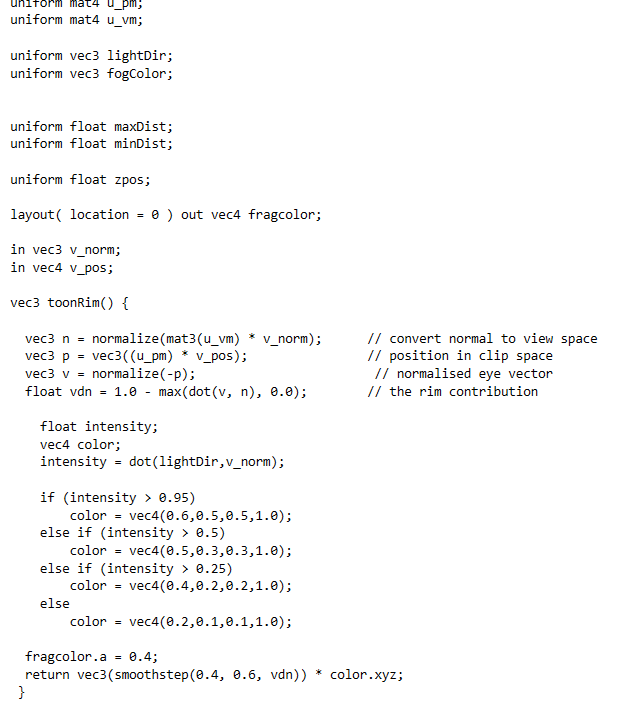
### Vertex Shader



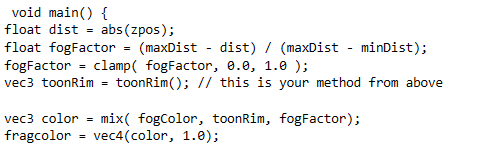
Once again, the vertex shader is exactly the same as the previous vertex shaders.

### Fragment Shader

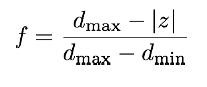
The complete overview of the fragment shader is that it mixes the colour of each fragment with a constant fog colour and the amount of colour that is mixed is determined by the distance the model is from the camera.



The ToonRim method calculates the RimToon method that was discussed earlier in the report.



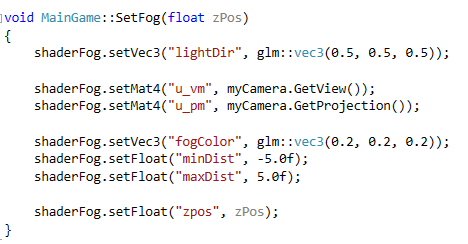
The main method is what is used to actually calculate the fog effect. Firstly, the distance the object must be determined using the models z position, it is worth nothing that x and y positions can also be used but for this project the z position was used. Abs() was used to ensure that a negative distance was not given. Next the fog factor is calculated, this is done by using the following equation:



Using this equation the current models position is taken away from the max distance variable created earlier and then the entire thing is divided by the minimum distance subtracted from the max distance. The fog factor is then clamped to give us a value between 0 and 1. If the calculation was lower than 0 it would be raised to 0 and lowered to 1 if the result was greater than 1. The toon rim vector is determined by the toon rim method above. The finial calculation will determine the colour of the fragment as it mixes the fogColor variable, toon rim vector and fogfactor float. This will determine how much fog is added to the fragment.

### Set-Up

When linking this shader some set-up is required for the shader to work:



This sets up all the variables needed for the shader to work appropriately, the zpos is set to the models current z position in their transform.