**Final Project Report**

Lighting Model:

Within the project we try to re-create the simulation of lighting within the real world and the components that are used create such effects. With OpenGL, we can create an approximation of outcome as to an object based in the real-life lighting. A lighting model “Phong” consists of 3 lighting techniques which are foundational, these are ambient lighting, diffuse lighting, and specular lighting.

Directional Lighting:

This is when the far of light source becomes parallel with light rays when they get close to each other. Giving the effect that the light rays coming in from the same direction of light, this direction does not change matter the position of the object or viewer.

Chart

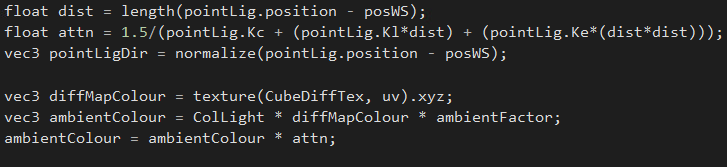
Description automatically generated with low confidenceText

Description automatically generated

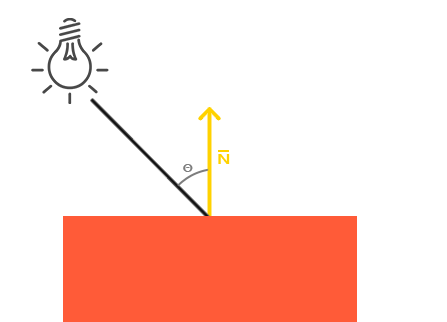
To input directional light, we would need 2 types of direction the first being the direction in which the light is traveling at and the direction in which the surface is facing, taking the value of both direction, and putting them in a dot product gives us the cosine between both angles. *I0= Ld\*Md\*cos(θ).*

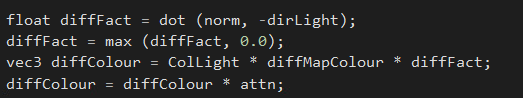
Ambient Lighting:

This allows us to see the model even when it is in dark surroundings the model in real life would often be illuminated by some other factor i.e., the moon or streetlight in the distance. Ambient lighting allows us to recreate that effect. To implement Ambient lighting the user would need the following equation: Ambient (the end product) = lightColour \* ObjectColour (these are the colour of the objects within the scene) \* AmbientFactor (the variable, is typically [0,1]).



Diffuse Lighting:

Diffuse lighting gives a brighter look to the overall fragments since the fragments are aligned closer together. The ray of light is targeted at a single point to a single fragment. To calculate the angel at which the light touches a single fragment, to measure this angle the use of a normal vector which is an angle perpendicular to the fragment’s surface. To implement Diffuse Lighting the user would need the following equation: Diffuse = lightColour \* objectColour \* AmbientFactor(use of dot product a . b = |a| |b| cos 0).



Shape, arrow

Description automatically generatedSpecular Lighting:

Similarly, to diffused lighting the vector for the lighting direction and the object’s normal vector produce the specular lighting, allowing objects to appear with a bright spotlight on them, in terms of the object being highlighted the spot of light is more inclined to produce the colour of the light rather than the colour of the object. The direction in which the player looks at the fragment also make a difference this is known as the view direction. To attain the specular lighting Specular Component = lightColour \* specularFactor (calculated through dot product of view direction and reflection direction)\*specularStrength (initialised by float to 0.2).

Text

Description automatically generated

Blinn Phong:

This is an advance application of phong. This was created by a man called Jim Blinn that was created in 1977 to improve the speed of the reflection around the surface normal. Instead of using the refection vector we remove it, then create a vector called Halfway which is halfway between the vector of view direction and the light direction. To achieve a higher specular contribution the halfway vector would need to be closer aligned to the normal vector. To calculate the Blinn phong H = I+V/|I+V| = H = I + V.

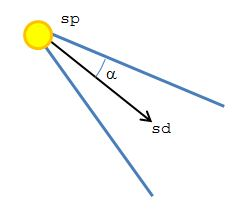
Text

Description automatically generated

**Light Casters**

Point Lighting:

A point Light is a type of lighting within OpenGL that allows the use to set a specific position for the light in the world, with this there is an equal amount of lighting in all directions it also fades with distance, meaning that it attenuates. To find the attenuation the Inverse Square Law is needed, this is the ‘Strength of the light inversely proportional to square of distance from light source.’ To get attenuation in world view you need the following method: Luminosity = 1/Attenuation & Attenuation = Kc (Constant factor) + Kl (Linear factor) \* d (Distance from fragment to the point) + Ke (Exponential factor) \* d2.. By the linear factor and the exponential factor, we can control the range of the light.

Text

Description automatically generated

Spot Lighting:

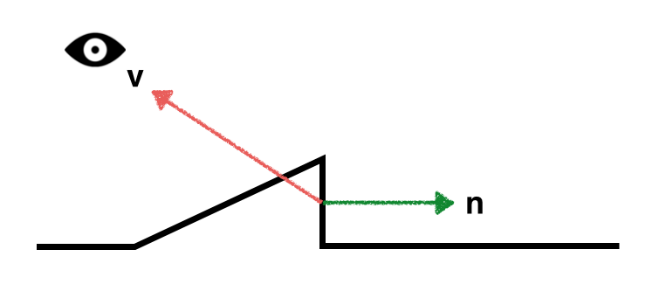
Unlike a point light, a spotlight does not cast light rays in different direction, it only illuminates the fragments that are within the radius of the spotlight. It has equivalent properties to other types of lighting like point light and directional light, but the only difference is its cut-off angle.

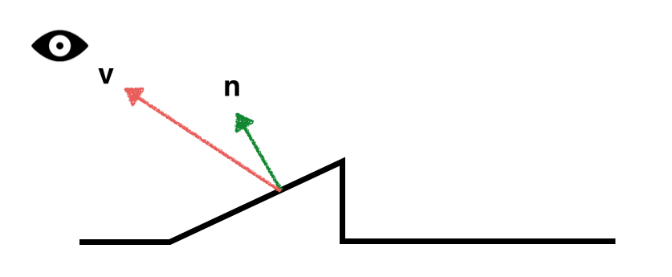
How to define the cut off angle would be, if define the cone, we do this be specifying uniform values to a radius, we also have a light direction that comes in the middle of the two outer radius, to find the cosine of the angel between the directional light and the outer radius we can do this using the dot product, resulting in . To get the fragment/object into the light we would need to use the dot product, and get the vector form the fragment to the light source this would then become the cosine angle between the fragment vector and the directional light. Leading to the result of θc1. To find out if the fragment is inside the uniform radius, we take the values and compare them if then we know that the fragment is outside of the cut off angle.

To create the effect of the light gradually dimming we add an outer radius this way we can define if the fragment is in the inner or outer radius and give it the right amount of lighting. If the fragment is between the inner and outer radius we had to interpolate to find the result. This is done through then .

Text

Description automatically generated

Rim Lighting:

This is often known as back lighting, to implement rim lighting there is only use of two function: surface normal and view direction. This works by having the view direction and the surface normal at a perpendicular angle and becomes more distinguishable. Two other variables that would need to be implemented would be the brightness (B) and sharpness (S) and to calculate the dot product of the view direction and the surface normal, use the equation .

Text

Description automatically generated

**Textures:**

First thing to note is that a texture is a type of storage and is not an image but can store an image. There are three components needed to create texture: it’s type/target (GL\_TEXTURE\_2D), size, this can be width and height and lastly format could be RED, RGB or RGBA.

Texture coordinates, first need to understand that a texel = is a pixel in a texture, these texel have their own coordinates these are associated with a vertex. These positions a show UV, with U being on the y-axis and V being on the x-axis. The texture space always ranges from 0 -1, e.g.(0,0),(1,1).

Call the texture from the file.

Text

Description automatically generated

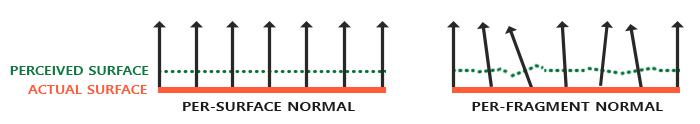
Bind texture to object.

Text

Description automatically generated

**Normal mapping:**

The purpose of normal mapping is to create a sense of depth in the texture when light is reflected of the surface allowing the texture to reveal a more realistic look. To get the colour vector of 2D texture we get the r, g, b components, however when working with normal vectors we replace the colour vector components with x, y, z components ranging from -1 – 1 and mapped [-1,1].



The normal are given to us through the normal map texture this can be identified as the texture being blue tint, this allows us to sample the normal from the texture making the normal per-fragment in opposed to per-vertex.

The normal that are worked with are active within their coordinated local space, which is Tangent space, this space which local to the primitive we transform the local tangent space to model space we do this using a matrix, we use the Tangent Bitangent and Normal Matrix (TBN Matrix) this form a space. We calculate this by having Bitangent = cross (normal, tangent). The T and B are seen equally to the UV values with the position of the vertices accumulated we can calculate the tangent and bitangent.

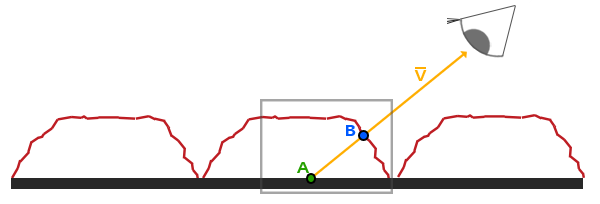
Text

Description automatically generated

**Parallax Mapping:**

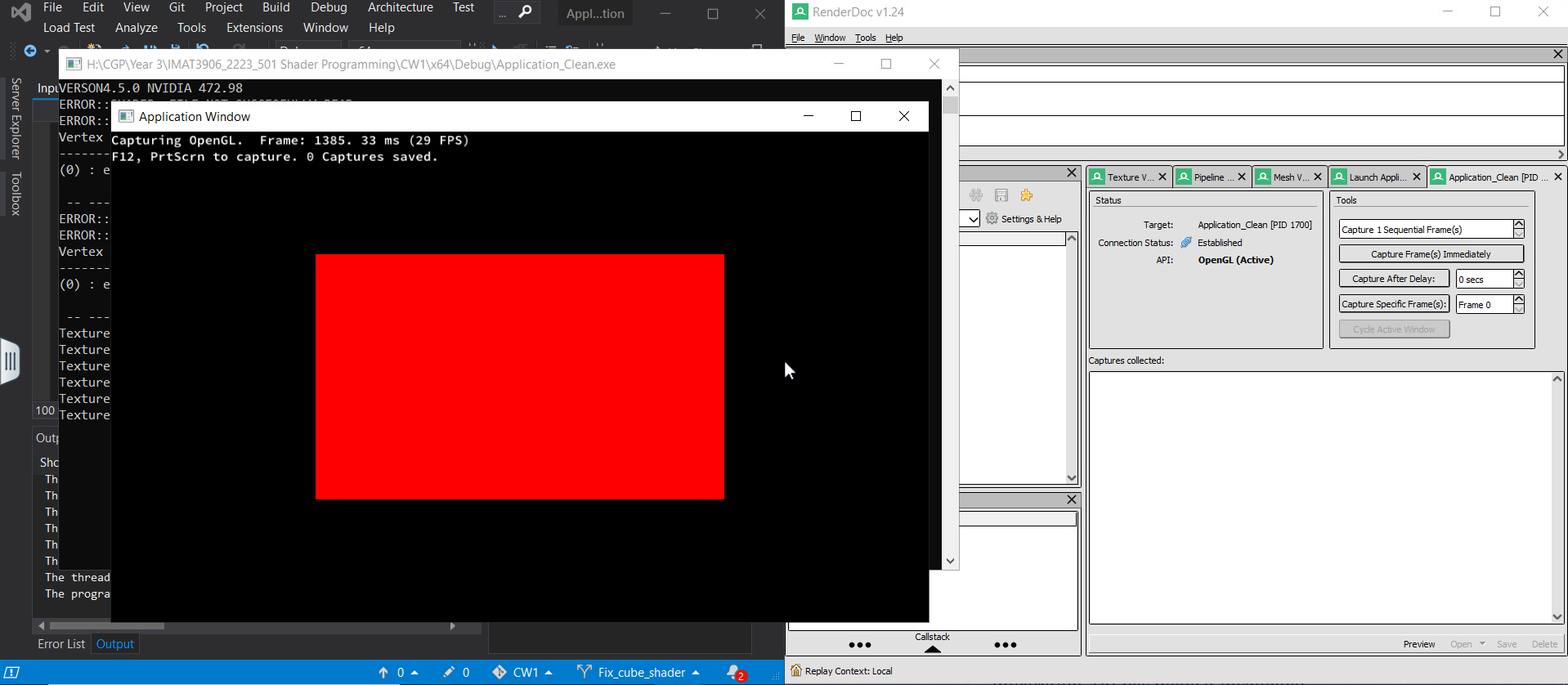
This technique is like that of normal mapping and correlate with similar features but different principles. The difference between them is that parallax mapping conveys a higher sense of depth together with normal mapping making the texture a higher level of realism compared to that of just normal mapping. They are typically used for floors and walls in games. Parallax mapping can be related to its familiar Displacement mapping, which displaces the vertices of the geometrical information given to us inside of a texture.

The bases of parallax mapping is to alter the coordinates of a texture based on the directional view of a fragment, giving the impression that a fragments surface is higher or lower than its meant to be.

The red line represents the height given to us from the displacement map giving the illusion of a brick V¯ representing the view direction. The surface would be visible at point B if actual displacement was added, however the surface at point A is a flat plane which is what viewers would see. Through parallax mapping we can change the position of point A to point B by using the UV coordinates, the offsetting of this can only be done by view direction and we use the UV coordinates to sample diffuse, normal maps and specular. To find the offset we, by reading the value from the displacement map with UV cords it returns a float, which is the height/displacement we take that distance along the view direction. PA – D(PA).

Testing:

The testing is done through Render Doc, this is a graphic debugger this advocates for easier debugging through quick, easy single frame capture with detailed introspection, often non as a default framebuffer.



Here I have launched the execution of the program but there is an issue with the code not allowing it to run and to capture the graphic content.

Project profile:

The overall success of the project went well, there were many thing that could’ve been done to improve the code clean the code up or optimise better use of the application. The goal was to create a scene in which there were 3d models, different types of light models and caster, textures and light maps, framebuffers and processing. The majority of each of these targets were hit for example the lighting models compared to meeting the target of the framebuffers, there some milestone within the textures and maps that I was able to meet for example adding to texture to objects and creating normal maps for the cubes. Time scale of the creation of the scene was not taken into account until heading to the deadline, I had spent too much time on a specific part rather than trying to cover all bases of the goal.

Bibliography:

(no date) *Tutorial 13 : Normal mapping*. Available at: http://www.opengl- tutorial.org/intermediate-tutorials/tutorial-13-normal-mapping/ (Accessed: January 2, 2023).

*Advanced lighting* (no date) *LearnOpenGL*. Available at: https://learnopengl.com/Advanced-Lighting/Advanced-Lighting (Accessed: January 2, 2023).

*Multiple lights* (no date) *LearnOpenGL*. Available at: https://learnopengl.com/Lighting/Multiple-lights (Accessed: January 2, 2023).

*Normal mapping* (no date) *LearnOpenGL*. Available at: https://learnopengl.com/Advanced-Lighting/Normal-Mapping (Accessed: January 2, 2023).

*OpenGL rim shader* (no date) *ROXLU*. Available at: https://www.roxlu.com/2014/037/opengl-rim-shader (Accessed: January 2, 2023).

*Parallax mapping* (no date) *LearnOpenGL*. Available at: https://learnopengl.com/Advanced-Lighting/Parallax-Mapping (Accessed: January 2, 2023).

*WebGL 3D - directional lighting* (no date) *WebGL 3D - Directional Lighting*. Available at: https://webglfundamentals.org/webgl/lessons/webgl-3d-lighting-directional.html (Accessed: January 2, 2023). -