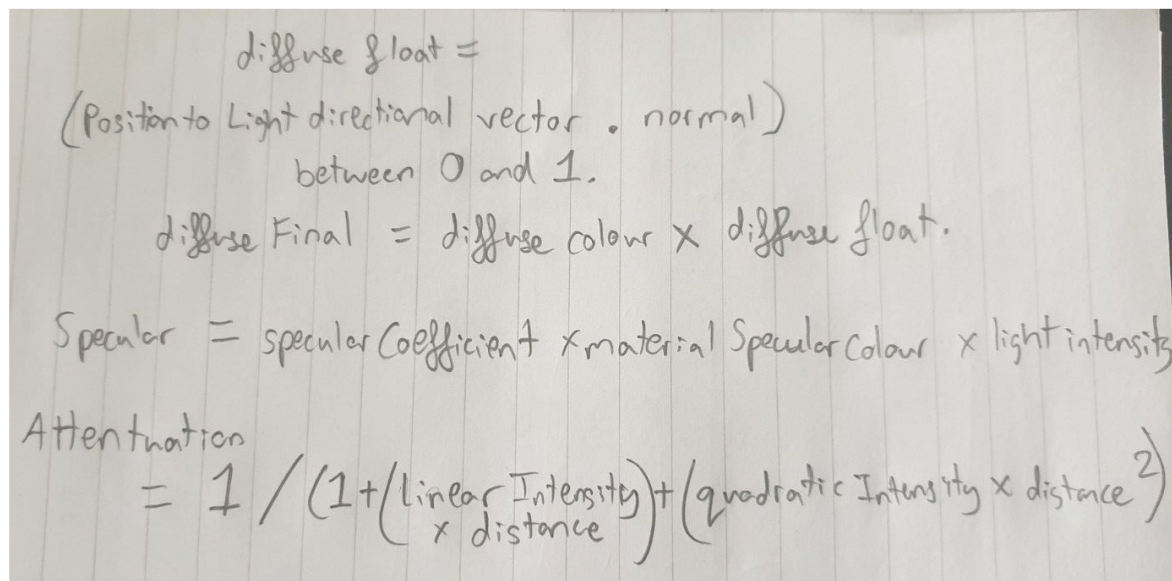


In my project, I aim to learn and increase my understanding of 3D graphics in video games by using the c++ and the OpenGL API to create a 3D game engine. I aim to do this by investigating matrices and shaders for displaying a 3D object, rendering textures and also implementing basic lighting. I also aim to implement a camera system to move space around the world.

### Research

To learn how to create a 3d graphics game engine, I refer mainly to the "LearnOpenGL.com" website as the in- depth tutorials and explanations made it easier to understand openGL. I also looked at the matrices and how to use a camera with the 3dgep.com website. The website helped me understand the trigonometry behind the rotation of the camera and i implemented equations that i found from the site within my code. I also used youtube tutorials and more equations online to figure out how to implement lighting into my graphics engine.

These were the lighting equations i have found to calculate the diffuse and specular lighting.



diffuse float =  
(Position to Light directional vector . normal)  
between 0 and 1.

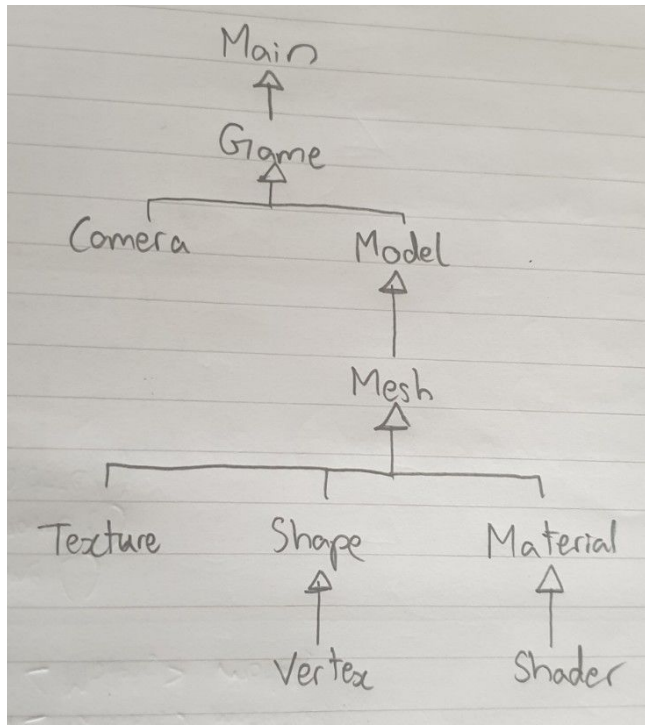
diffuse Final = diffuse colour x diffuse float.

Specular = specularCoefficient x material Specular colour x light intensity

Attenuation  
=  $1 / (1 + (\text{Linear Intensity} \times \text{distance}) + (\text{quadratic Intensity} \times \text{distance}^2))$

## Design

My program will be split into multiple classes to make readability a lot easier especially during testing and also as i will be incorporating a lot of different object types. Here is a UML diagram, split up into multiple classes, that i have designed to help me program my graphics engine.



As you can see from the image, i have used a lot of inheritance to make my code alot more modular and it will help alot when i have issues while testing my graphics engine. The main class will just setup and run the game class, in which all of the code and engine will be managed. Game will create the window and call every other function to initialise the engine and also run the update function every frame.

```

class Game {
    GLFWwindow window
    Camera camera
    vector<Shader> shaders
    vector<Texture> textures
    vector<Materials> materials
    vector<Models> models

    Game()
    ~Game()
    update()
}
  
```

Game will hold vectors that hold every single shader, texture, material and model used in the program.

```

class Camera {
    GLfloat movement speed
    GLfloat sensitivity
    vec3 position
    GLfloat pitch, yaw, roll

    Camera()
    ~Camera()
    void move()
    void update MouseInput()
}
  
```

The camera class will get the user input to allow the user to move around the world space. I have made it into its own class to separate it from all the classes that handle the graphics and object rendering.

The model class takes in the material and textures of an object and uses the values to render and display the object.

```

class Model {
    Material material
    Texture diffuse
    Texture specular
    vector<Mesh> meshes
    vec3 position

    Model()
    ~Model()
    void render(Shader)
}
  
```

Mesh

```

Vertex  vertex Array
GLuint  index Array
GLuint  VAO, VBO, EBO
mat4    Model Matrix

void initVAO()
void updateUniforms()
void updateModelMatrix()
Mesh()
~Mesh()
void render()

```

This mesh class stores the vertex array and vertex buffer object. It will initialise the VAO and then stores the values of an object such as position, origin, rotation and scale. These values will be sent into mesh class every update where it can use them to render and draw the objects.

Texture

```

GLuint id
int height, width
int type

Texture()
~Texture()
void bind()
void unbind()
void LoadFromFile(char FileName)

```

The texture class handles reading the image and object files and send the information to the other classes to be used higher up the UML diagram to create the object using the texture that the class has read.

Shape

```

vector<Vertex> vertices
vector<GLuint> indices

Shape()
~Shape()
void set(vertices, indices)

class Triangle
class Square
class Pyramid
class Cube

```

The shape class stores all the vertices and indices to create a simple triangle, square, pyramid and cube object. The class can be called in to create the object and using textures that were read from the texture file, the mesh and model classes will create the object with the texture applied on it.

Vertex

```

struct Vertex
{
    vec3 position, color, normal
    vec2 texCoord
}

Shader Program
std::string LoadShaderSource
GLuint LoadShader
void LinkProgram

void use()
void unuse()

```

Vertex class stores basic information about a vertex in a struct.

The shaderprogram class will handle reading the vertex and fragment shader files.

Material

```

vec3 ambient, diffuse, specular
vec3 diffuseTex, specularTex

void SendToShader(shader program)

```

Material class holds information about how the light will affect the object and how much and where the specular light reflects from.

Analysis and conclusion.

Looking at the strengths of my program, i feel like i have written it quite efficiently. I have successfully managed to render multiple 3D objects with a texture. I am also happy with my shaders and lighting as i have managed to implement basic lighting, lighting maps, where the specular lighting will reflect off certain materials more than others, and also added attenuation where the light will fade exponentially the further an object is from the light source. I have also utilized matrix transformations to handle camera movement and rotations. I had also previously used them to rotate, move and scale objects on screen.

However, while testing my program, i have discovered many weaknesses. I tried to implement 3D model and object loading however i had a problem with the object loading library ASSIMP so i was forced to remove it from my code. This meant i had to change up my model class completely to accept both 2D images and 3D obj files and did not manage to read 3D obj files. I would've liked to implement more complexity in my lighting as i really enjoyed looking into the maths behind it and if i had more time i would have researched more about bloom, glow and HDR lighting and implemented a better program with aspects that would make it more similar to a game rather than a simulation.

References

<https://learnopengl.com/>

<https://www.tomdalling.com/blog/modern-opengl/07-more-lighting-ambient-specular-attenuation-gamma/>

[https://learnopengl.com/code\\_viewer.php?code=getting-started/cube\\_vertices](https://learnopengl.com/code_viewer.php?code=getting-started/cube_vertices)

Vertices i used to create my cube. The tex coordinates were not correct though so i had to change them up and make it fit how i rendered my triangles.

I used this thread to help me understand how texture coordinates work and fix my issues.

<https://stackoverflow.com/questions/5532595/how-do-opengl-texture-coordinates-work>

<https://www.youtube.com/watch?v=I3LeSKTD1p4&list=PL6xSOsbVA1eYSZTKBxnoXYboy7wc4yg-Z&index=36>

Youtube tutorial series that helped me figure out and implement lighting

I had a few other issues as well and used these websites to help me understand the problem and fix them.

<http://www.cplusplus.com/forum/windows/58523/>

<http://www.cplusplus.com/forum/windows/58523/>

OpenGL

GLSL

GLFW

SOIL2