CITS3003 – Part 1 Report

Thomas Ankers (Group 47)

21490093

Functionality

The scene editor implemented all the required functionality outlined in the project brief. The finished product works very well and no major bugs or errors were encountered during testing.

Building the scene editor

Work was divided between me and my partner, Jason as evenly as possible. I ended up doing the bulk of the lighting steps, which proved rather interesting albeit difficult at times. Below is each part I worked on and my steps in adding the features:

F – Light Attenuation

Light attenuation was achieved by first calculating the distance of the light from the fragment:

```
float lightDist = length(Lights[i].position.xyz - pos);
```

This distance was then used to calculate the attenuation using the formula:

$$A=\frac{1}{1+kd^2}$$

k = attenuation coefficient (directional lights have less attenuation)

d = distance

The attenuation is then multiplied to the final colour value:

```
colour.rgb += attenuation * brightness * (ambient + diffuse + specular);
```

G – Fragment Lighting

To move lighting into the fragment shader I added two varying variables to the vertex shader, pos and normal. Pos is vPosition*ModelView, normal is simply vNormal. These two variables are then used in the fragment shader to calculate three vectors:

```
vec3 viewDir = normalize(-pos);  // Direction to the eye/camera
vec3 N = normalize((ModelView * normal).xyz);  // Normal vector
vec3 lightDir = normalize(Lights[i].position.xyz - pos); // Direction to
the light source
```

These vectors are then used in the Blinn-Phong model to calculate the halfway vector and subsequently the ambient light and coefficients for diffused and specular light.

```
vec3 halfway = normalize(lightDir + viewDir);  // Halfway vector
vec3 ambient =  AmbientProduct;
float Kd = max(dot(lightDir, N), 0.0);
float Ks = pow(max(dot(N, halfway), 0.0), Shininess);
```

H – White specular highlights

After the lighting calculations are complete, they are multiplied by the colour of the texture. To achieve white specular highlights, I split up this step and simply didn't colour the specular light as seen below.

```
vec4 surfaceColour = texture2D(texture, texCoord * texScale);
vec3 ambient =  AmbientProduct * surfaceColour.rgb;
vec3 diffuse = Kd * DiffuseProduct * surfaceColour.rgb;
vec3 specular = Ks * SpecularProduct;
```

I – Adding second light

Initially I incorporated a second light by hard coding it into the fragment shader. This meant I had two uniform variables LightPosition1 and LightPosition2. I then calculated the required vectors, the ambient, diffuse and specular light. I then added this at the end to gl_FragColor. This worked but the lack of flexibility prompted me to look at alternatives for multiple lights as an extra feature.

Additional functionality

Due to my experience with the lighting system, I decided to implement two extra features involving the lights.

The first feature was to allow for up to 10 (in theory many more) lights to be added to a scene. This was achieved by moving all the lighting calculations to the fragment shader and passing the shader an array of light structs. The basis for this idea was from Tom Dalling's Modern OpenGL series, referenced at the end of the report. Because you cannot pass a struct to a shader easily, I created a function `LightUniform` that sends each member of the struct individually. I then added the following (summarised code) to the fragment shader:

```
vec4 surfaceColour = texture2D(texture, texCoord * texScale);
vec4 colour = vec4(globalAmbient, 1.0) * surfaceColour;
…
…
for(int i =0; i < numLights; i++)
{
        …
        vec3 ambient = AmbientProduct * surfaceColour.rgb * attenuation;
        vec3 diffuse = Kd * DiffuseProduct * surfaceColour.rgb;
        vec3 specular = Ks * SpecularProduct * Lights[i].rgb;
        …
        colour.rgb += attenuation * brightness * (ambient + diffuse + specular);
}
gl_FragColor = colour;
```

Figure 1 - Fragment shader code

In short, it first calculates the global ambient colour then it loops over each light, calculating colour values and adding it to the global ambient. This allows for many lights to be present in

a scene. Jason worked on a menu system for the new lights which can be seen below. The limit of 10 lights is mainly to prevent the menus from becoming too big, a better selection

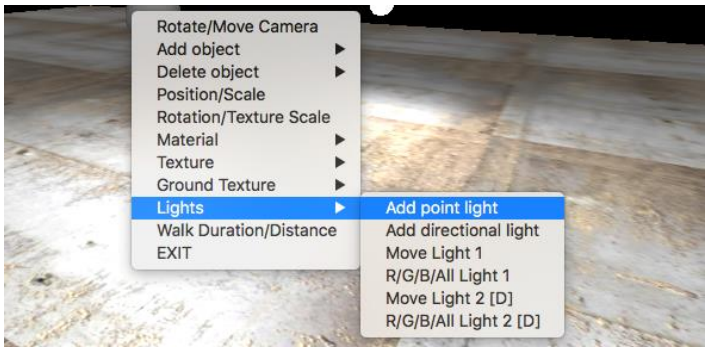

Figure 2 - New lighting menu

The second additional feature was a modification to the directional light. As this light is described to behave like the sun, I felt the light colour should reflect this as well.

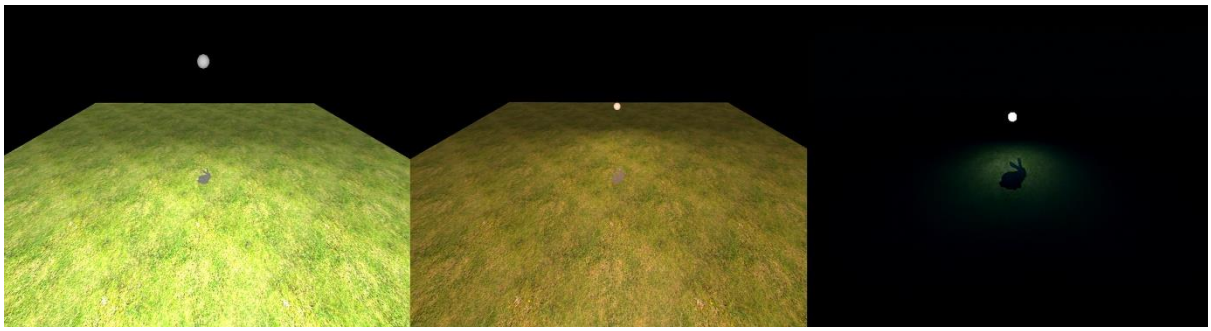

Figure 3 - Midday sun, late evening, night (with point light)

```
//ratio used to replicate sun angle in the sky
float angleModifier = degrees(Lights[i].angle)/90.0;
brightness = brightness * angleModifier;

//simulate sunset colours
lightColour.r = colour.r +(200.0 + 55.0*angleModifier)/255.0;
lightColour.g = colour.g +(125.0 + 130.0*angleModifier)/255.0;
lightColour.b = colour.b +(90.0 + 165.0*angleModifier)/255.0;
```

Figure 4 - Fragment shader code

The code above takes the angle of the sun in the sky (the angle between the x-z plane and the y-axis) and varies the colour of the light based on that angle. Rough RGB values for the sun were found online but tweaking was required.

Reflection

I found this project to very interesting and ultimately rewarding once it all came together. Coming into this unit I had done zero graphics work apart from basic game design while at school. Working on the scene editor allowed me to get a low-level understanding of how a graphics engine works and it's something I want to explore further in the future.

References

- http://www.tomdalling.com/blog/category/modern-opengl/
- https://learnopengl.com/#!Advanced-Lighting/Advanced-Lighting