3D Graphics Programming
Lab 8: Lighting

Karsten Pedersen
Department of Creative Technology

November 13, 2018

In this lab we are going to look into implement some basic lighting. The first step is to have a think about the uniform values we need to send to our shader. The simplest values are for the *Emissive* and *Ambient* lighting. For this we simply set a vec3 containing these lighting colors.

```
shaderProgram->setUniform("in_Emissive", glm::vec3(0, 0, 0));
shaderProgram->setUniform("in_Ambient", glm::vec3(0.2, 0.2, 0.2));
```

Here you can see that I use no emissive lighting because the cat doesn't need to glow. I also then add 0.2 in the rgb values for ambient lighting because very rarely is anything pitch black.
In your fragment shader, you simply multiply the output texture pixel by these values.

```
vec3 lighting = in_Emissive + in_Ambient;
gl_FragColor = tex * lighting;
```

You should now see similar to the image shown in figure **Figure 1**.

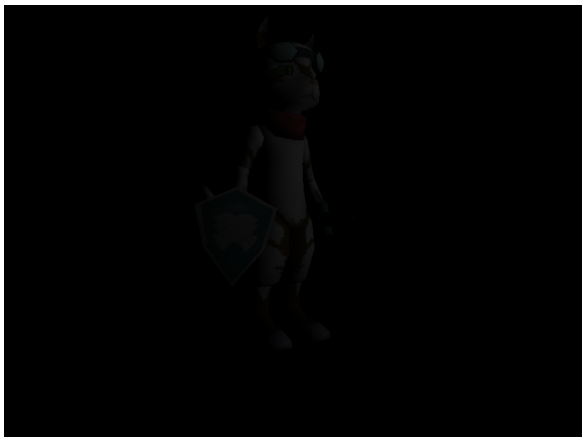

Figure 1: *The model with ambient and emissive lighting. Remember that the cat model has some "fake" lighting baked into his texture.*

Note:
Your scene will likely end up very dark at this point. This is because we have not yet added the diffuse lighting yet.

Next for the diffuse lighting we need to add the idea of a light position.

```
shader->setUniform("in_LightPos", glm::vec3(50, 10, 10));
```

We now need to pass a few things from our vertex shader into the fragment shader. This is because our light calculation is going to be done per pixel in the fragment shader. We need to pass through the normals (in a similar way to the texture coordinates) and the vertex position. Traditionally we have set the vertex position in *gl_Position* and forgotten about it. However in this case we need it as part of the lighting calculation.
We need to convert both the normals and the position to world space as we pass it into the fragment shader. This is because we do our calculation in world space and we do not want to mix coordinate spaces.

Note:
You will notice that we use a *mat3* for transforming the normals. This is because we only care about direction with them and do not want to apply translation or scale. You could achieve a similar result by setting the w component to 0 and use a *mat4*.

```
varying vec3 ex_FragPos;
varying vec3 ex_Normal;
...
ex_Normal = mat3(in_Model) * in_Normal;
ex_FragPos = vec3(in_Model * vec4(in_Position, 1.0));
```

Once we have these passed into the fragment shader, we then normalize the passed in normals and calculate the light direction. We are now ready to perform our diffuse lighting equation. Remember to look through the lecture slides for reference on this.

```
vec3 norm = normalize(ex_Normal);
vec3 lightDir = normalize(in_LightPos - ex_FragPos);

float diff = max(dot(norm, lightDir), 0.0);
vec3 diffuse = diff * vec3(tex);

gl_FragColor = vec4(diffuse, 1);
```

You should now see similar to the image shown in figure **Figure 2**.

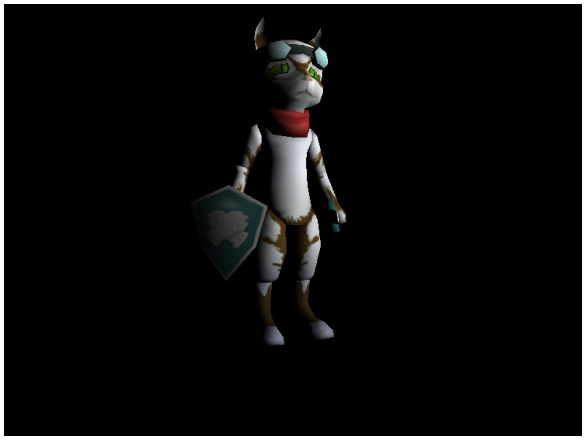

Figure 2: *You should now see your object with directional lighting*

In this example I simply multiply diffuse amount by the texture pixel but remember that ideally we

want to merge the diffuse lighting with our ambient and emissive lighting. You should ideally end up with something which looks similar to:

```
vec3 diffuse = diffuseColor * diff;
vec3 light = emissive + ambient + diffuse;
gl_FragColor = tex * light;
```

You should now see similar to the improved image shown in figure **Figure 3**.

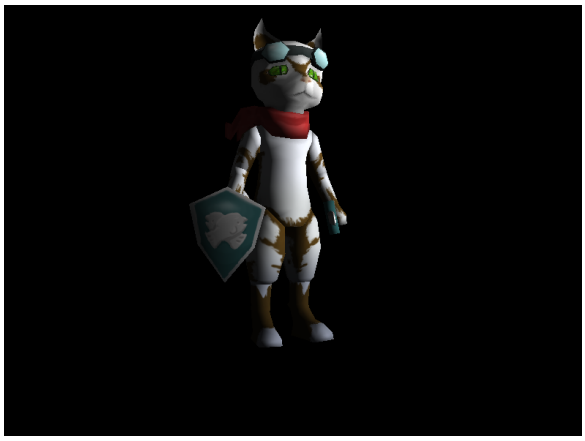

Figure 3: *Notice that even when in full shadow, you can still see the model and texture*

Note:
This shows you how to set up one light source. You might want to look into supporting multiple lights and passing them through into your shader as an array. Potentially even an array of structures. However, note that in general, I find that I never need no more than two lights because much of the more complex lighting should be baked into the texture or shadow map.