Laboratory work 8

1 Objectives

The objective of this laboratory is to briefly describe several advanced lighting techniques in OpenGL 4.

2 The Blinn-Phong Lighting Model

The Blinn-Phong lighting model is a slightly modified version of the Phong model and was developed by Jim Blinn. Its aim is to improve the specular reflections in certain conditions like a very small shininess coefficient (thus resulting a large specular area). Specular reflections in the Phong model tend to be cut-off immediately when the angle between the viewing direction and the light reflection increases above 90.0 degrees (the cosine becomes negative and thus is clamped to 0.0).

The solution proposed by Blinn is to use a so called **half vector** instead of the reflection vector to compute the specular component. The half vector (**H**) is a unit vector halfway between the viewing direction and the light direction (Figure 1).

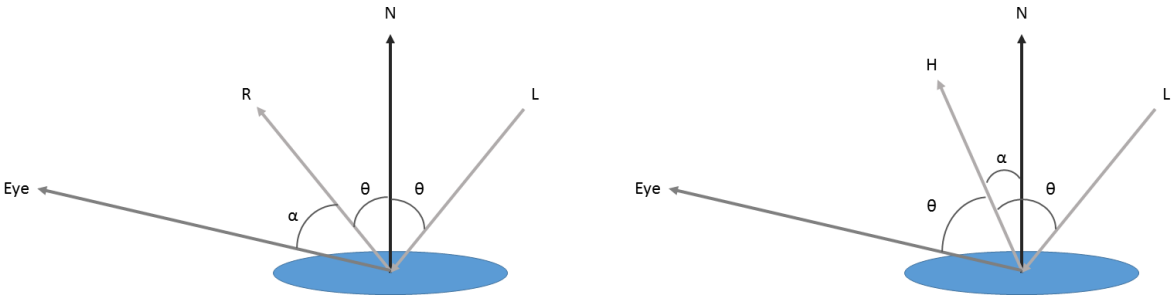

Figure 1 - Phong specular model (left) versus Blinn specular model (right)

The half vector is computed as:

$$\boldsymbol{H} = \frac{\boldsymbol{L} + \boldsymbol{Eye}}{\|\boldsymbol{L} + \boldsymbol{Eye}\|}$$

This translates into GLSL code as follows:

```
//normalize light direction
vec3 lightDirN = normalize(lightDir);

//compute view direction
vec3 viewDirN = normalize(viewPosEye - fragPosEye.xyz);

//compute half vector
vec3 halfVector = normalize(lightDirN + viewDirN);
```

The specular component is then computed as the cosine of the angle between the normal and the half vector:

```
//compute specular light
float specCoeff = pow(max(dot(normalEye, halfVector), 0.0f), shininess);
specular = specularStrength * specCoeff * lightColor;
```

The Blinn-Phong model also improves performance by avoiding the expensive computation of the reflection vector.

3 Point lights

Laboratory work 8 introduced **directional lights**, which are light sources infinitely far away. All light rays coming from a directional light source are parallel and thus the direction of the light involved in lighting computation is constant across objects and all their fragments, making them ideal for global illumination in our scenes.

Point lights are light sources with a given position that illuminate radially and uniformly in all directions. Unlike directional lights, rays generated by point lights fade out over distance, thus making objects closer to the source appear brighter than objects situated farther away.

When using point lights, the direction of the light is not constant across fragments and must be computed for each different fragment:

```
//compute light direction
vec3 lightDirN = normalize(lightPosEye - fragPosEye.xyz);
```

To reduce the intensity of the point light over the distance we should apply an attenuation coefficient. Attenuation can be computed as a linear or quadratic function over the distance. Linear attenuation requires less complex computation but offers less realism since, in the real world, light sources are bright when standing close to them but quickly diminish in brightness the farther we stand from them. Realism is considerably increased when incorporating quadratic attenuation over the distance. Thus, attenuation can be computed using:

$$Att = \frac{1.0}{K_c + K_l * d + K_q * d^2}$$

where K_c is a constant term, usually kept at 1.0, K_l is a linear term, K_q is a quadratic term and d is the distance.

Choosing the right values for these constants depends on many factors such as the environment, the distance required for the light to travel, the type of light etc. You can use Table 1 as a guide for specifying the attenuation coefficients based on the distance the light is required to travel.

To implement attenuation, we should define the attenuation coefficients in the shader or send them from the application via uniform variables:

```
float constant = 1.0f;
float linear = 0.0045f;
float quadratic = 0.0075f;
```

Distance	Constant	Linear	Quadratic
7	1.0	0.7	1.8
13	1.0	0.35	0.44
20	1.0	0.22	0.20
32	1.0	0.14	0.07
50	1.0	0.09	0.032
65	1.0	0.07	0.017
100	1.0	0.045	0.0075
160	1.0	0.027	0.0028
200	1.0	0.022	0.0019
325	1.0	0.014	0.0007
600	1.0	0.007	0.0002
3250	1.0	0.0014	0.000007

Table 1 – Common values for attenuation coefficients (source: Ogre3D wiki)

We should then compute the attenuation using the equation provided above:

```
//compute distance to light
float dist = length(lightPosEye - fragPosEye.xyz);
//compute attenuation
float att = 1.0f / (constant + linear * dist + quadratic * (dist * dist));
```

Last, we should apply the attenuation to the ambient, diffuse and specular components:

```
//compute ambient light
ambient = att * ambientStrength * lightColor;

//compute diffuse light
diffuse = att * max(dot(normalEye, lightDirN), 0.0f) * lightColor;

specular = att * specularStrength * specCoeff * lightColor;
```

4 Lighting Maps

Up until now, when it comes to lighting, we have considered objects as being composed of a single material (having a single base color) (Figure 2). However, this approach does not offer too much flexibility on the visual output of an object. We will now extend the lighting model to incorporate **lighting maps**.

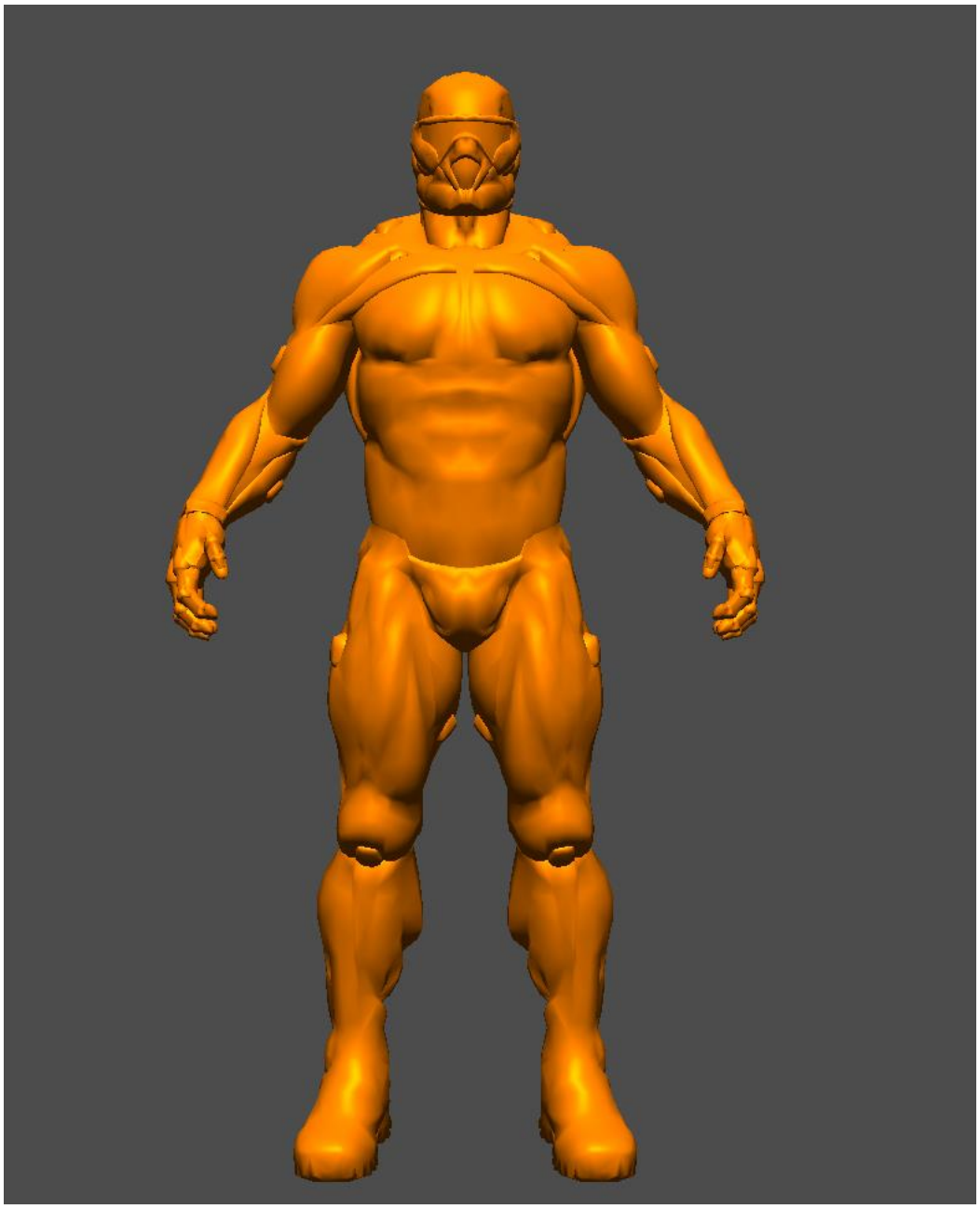

Figure 2 – Crysis 2 nanosuit rendered with point light and orange base color

Lighting maps provide us with means to sample the base color for each fragment from a texture for each lighting component (ambient, diffuse and specular). They are incorporated into the material properties of OBJ models (the MTL file).

Diffuse maps define the colors of an object. Color information is basically sampled from a texture and combined with the effects of lighting to determine the final fragment color (Figure 3). Diffuse maps usually also serve as the base color for the ambient component. Thus, the computation of the ambient and diffuse components should change accordingly:

```
    ambient *= texture(diffuseTexture, fragTexCoords);
    diffuse *= texture(diffuseTexture, fragTexCoords);
```

Figure 3 – Crysis 2 nanosuit rendered with diffuse maps

Usually not all the materials an object is composed of reflect the same amount of specular highlights. To differentiate between the reflectivity of different materials we can use **specular maps** that assign a higher specular coefficient to highly reflective materials and a lower one to materials that do not

normally present specular components (Figure 4). Specular maps can be sampled exactly as diffuse maps but are instead applied to the computation of the specular components:

```
specular *= texture(specularTexture, fragTexCoords);
```

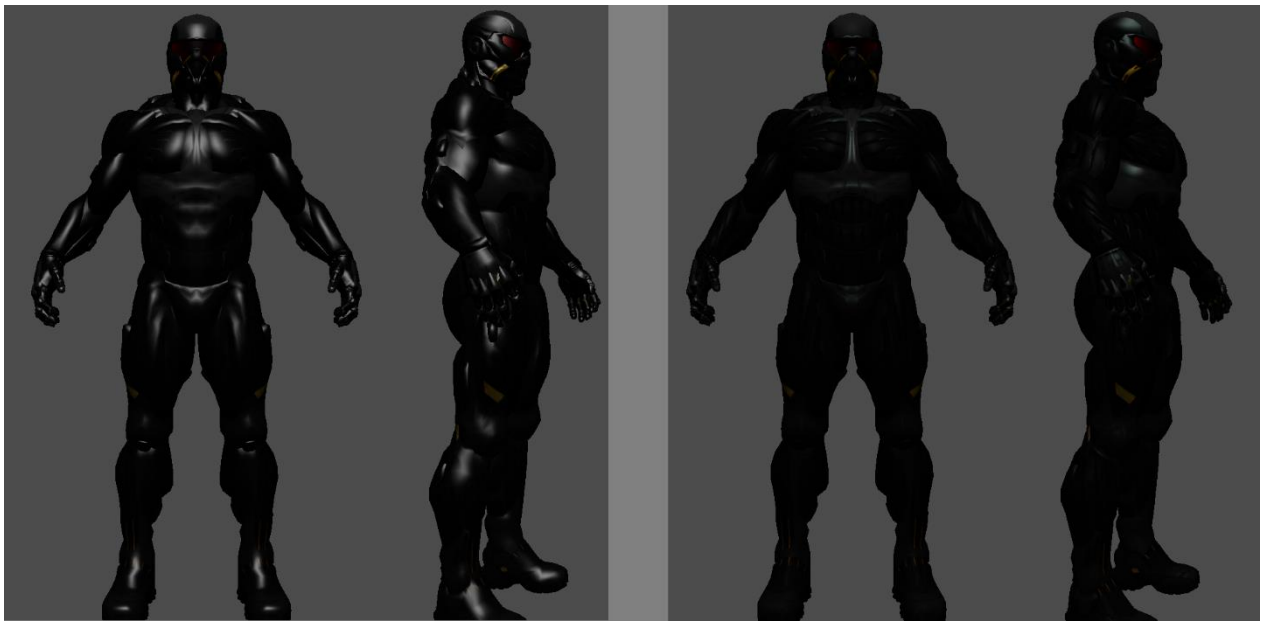

Figure 4 – Crysis 2 nanosuit rendered without specular maps (left) and with specular maps (right)

The OBJ loading library provided already includes loading diffuse and specular maps (textures) and proper binding of these texture objects inside the Model3D and Mesh classes. The uniforms "diffuseTexture" and "specularTexture" must be defined in the shader program to allow sampling of the lighting maps of and object.

5 Further reading

- OpenGL Programming Guide 8th Edition – Chapter 7
- OpenGL tutorials – Light casters, Lighting maps, Multiple lights, Advanced lighting - http://www.learnopengl.com/

6 Assignment

1. Download and explore the code samples provided as resources for this laboratory
2. Implement the Blinn-Phong lighting model for the given demo application
3. Transform the directional light into a point light for the demo application
4. Render several objects scattered throughout the scene to observe the effect of point lights
5. Incorporate lighting maps into the demo application
6. Define a directional light and a second point light in the demo application