Lighting in OpenGL

By Lewis Arnold

# My method

Lighting an OpenGL scene starts off very simple but can very quickly become quite complicated and very expensive. To start, a representation of a Light is needed. This is the simplest part of lighting and is just a class containing a transform, a color for the light and the falloff amount of the light. As far as the effectiveness of this implementation of my Light class, it is as small as it could be done. Other features could be added like light radius to completely cut off a light effecting anything past a certain radius, even if the light has no falloff.

Once the Light class is set up, a lighting method needs to be decided. I chose Phong Lighting as it meets a balance of simplicity and effectiveness. Either end of this scale lies Gourad lighting which is Phong shading implemented in the vertex shader and Blinn-Phong lighting. Since Gourad lighting is calculated in the vertex shader, this results in a per-vertex shading which shows a characteristic flashing (Sellers, Wright, & Haemel) of the specular highlight. Whilst Gourad is the more efficient method (Sellers, Wright, & Haemel), Phong lighting gets around the issues of specular highlights by being written in the Fragment shader. (Sellers, Wright, & Haemel) say that Blinn-Phong lighting is considered an extension on Phong lighting. Blinn-Phong can be more efficient than regular Phong as some values can be calculated per-vertex and still give the same result as Phong lighting.

The result of this was impressive, especially when paired with Normal Mapping. However, the light was missing a sense of where it was in the scene. Whilst the Phong Lighting does consider the angle of the light to the normal, there is no mention of how far the light is. This means that the light will affect an object next to it as much as it will affect an object far, far away. Therefore, I added a light falloff. This works by effecting the strength of the diffuse and specular elements of the Phong Lighting Model inversely proportional to the distance the fragment is away from the light. This results in the fragment getting darker as the light moves further away. I chose to make the relationship an inversely proportional one because I thought it would more intuitive to increase the falloff value and increase the range of the light since there is no value for the range of a light in my implementation. Should I do this again, I would design the light class with this in mind from the beginning and use terms for their proper use.

With light falloff implemented, it was now possible to see the effect of multiple lights in a scene. I started by creating a second light object and changed the colour of one to blue and one to red to see the full effect of having multiple lights. What comes next is to change my uniform in the fragment shader for GL\_LightData to be an array so that I can pass in multiple lights. Passing the data into the shader was very simple but very inefficient. Since an OpenGL uniform has one location, meaning one location per variable, per GL\_LightData struct in the array that we are using, we need 6 uniform locations for the two GL\_LightData structs. This resulted in a lot of hardcoding for each index in the array. A massive improvement over this would be to somehow have a single uniform location for the start of the array, and fill in the data in the array from this one uniform and a list of lights, rather than using 3 different uniform locations for each light. As far as my current light setup is, this is not a massive inconvenience but the expandability of this system is next to null.

# Improvements

I think the most significant improvement to the lighting in my scene would be the introduction of Shadow mapping. This is a technique where rays are cast out from the camera at the fragment we are trying to light, and if there is an object blocking this ray, then we tint the color of the fragment to be darker. (JoeyDeVries, ShadowMapping, n.d.) defines a shadow as the absence of light due to occlusion and goes into great depth at how shadows can be generated. Their implementation generates a shadow map for each object, which is a texture that is applied to the mesh on top of its regular textures. The importance of shadows is mainly to see how the objects relate to each other (JoeyDeVries, ShadowMapping, n.d.). However, a good shadow implementation can be used in design to portray the mood or good/evil alignment of an area or person. (Marta, Simon, & Manuel, 2013) state that lighting is linked with a knowledge of goodness, while shadows have commonly been the place of evil. This can be backed up by the portrayal of certain video game protagonists and antagonists. For example, the antagonist of The Legend of Zelda: Breath of The Wild, Ganon, is shrouded in shadow and darkness whereas antagonists such as Link are often bathed in light; reinforcing the need for both lighting and shadows in a game.

Another great improvement would be the introduction of different types of lights. An overview of the different types I would add would be a directional light and a spotlight. At the moment, the point lights contribute to the ambient colour of the scene and I think it would be much better if the ambient lighting be controlled by a separate light source. This way, the other light types could play their part a lot better, without tinting the whole scene. The importance of lighting in games is seen in a study from (Seif El-Nasr, Niedenthal, Kenz, Almeida, & Zupko, 2007) where dynamic lighting was used to convey danger in a game, and some feedback described “the lighting as a method of portraying game state information”. Because of the effect that lighting can have on a game, creating different types of lights which are highly customizable is a necessity.

# Alternative methods

The Phong Lighting Model is a great model to easily render some lit objects in a scene. However, the air of realism can be lost and with games moving more towards photorealistic graphics, a different lighting model needs to be considered. (Collins, 2020) talks about the use of photogrammetry in games being leveraged heavily. However, without a lighting model that can keep up with modern techniques, it is not worth creating such detailed assets . Therefore, PBR lighting is the best alternative I have found for Phong Lighting. PBR stands for Physically Based Rendering and is a collection of render techniques that are more or less based on the same underlying theory that more closely matches that of the physical world; this is also the technique adopted on most popular engines (JoeyDeVries, PBR Theory, n.d.). The PRB model comprises of a culmination of the microfacet surface model, energy conservation, and physically based BRDF which ultimately calculates how much each ray should contribute to each fragment. Using a PBR lighting model, the effects of the lights would be made much more substantial, with the extra tweaking on how rough a surface is and its metallic properties.

# References

Collins, M. (2020, April). Retrieved from GDC Vault: https://www.gdcvault.com/play/1026692/Art-Direct-for-100-MPH

JoeyDeVries. (n.d.). *PBR Theory*. Retrieved from LearnOpenGL: https://learnopengl.com/PBR/Theory

JoeyDeVries. (n.d.). *ShadowMapping*. Retrieved from LearnOpenGL: https://learnopengl.com/Advanced-Lighting/Shadows/Shadow-Mapping

Marta, F., Simon, N., & Manuel, A. (2013). Context Matters! In M. Knonstantin, H. Simon, J. Wimmer, M. G. Wagner, & H. Rosenstingl, *The sense of lighting inside game worlds. Myth and meaning in gameplay and game mechanics* (pp. 57-69).

Seif El-Nasr, M., Niedenthal, S., Kenz, I., Almeida, P., & Zupko, J. (2007, August). *Dynamic Lighting for Tension in Games.* Retrieved from Game Studies: http://gamestudies.org/0701/articles/elnasr\_niedenthal\_knez\_almeida\_zupko

Sellers, G., Wright, R. S., & Haemel, N. (n.d.). OpenGL SuperBible.