ASSIGNMENT 4: SHADING AND LIGHTING

CS 148 Autumn 2018-2019

Due Date: Monday, 22 October 2018 by 7pm

- Follow the instructions carefully. If you encounter any problems in the setup, please don’t hesitate to reach out to CAs on Piazza or come to office hours.
- Start early! This assignment is more coding intensive than the others.
- Please SEARCH Piazza before asking questions. Chances are someone already asked the same question.
- READ THIS DOCUMENT AND THE PROVIDED LINKS TO DOCUMENTATION THOROUGHLY. Links in the PDF are blue.

Figure 1: Screenshot from the Unreal Engine 4 Infiltrator Demo (from Epic’s SIGGRAPH 2013 presentation).

1 Introduction

This week, we will explore more advanced BRDFs, materials, and lighting types. As you might imagine, Blinn-Phong and simple point lights are generally unable able to represent a wide range of shading and lighting conditions. As a result, many companies (i.e. Epic, Crytek, EA DICE, Disney, etc.) have been moving towards physically based shading and lighting. We will not fully explore everything they do in this assignment, but we will give you a brief overview! Should you be interested in this topic, you are more than welcome to explore it more yourself. There are three parts towards achieving more realistic shading:

- BRDF
- Material
- Lighting

In this assignment, we will explore what Epic Games does for their BRDF and Material in their Unreal Engine 4 as well as some new types of lights. Additionally, you may find it very useful to actually look at Unreal Engine's
source code. Follow the instructions here to get access to it. Namely the file: Engine/Shaders/Private/BRDF.ush as well as Engine/Shaders/Private/ShadingModels.ush. The most relevant functions to look at are D_GGX, Vis_Schlick, F_Schlick in BRDF.ush. You can trace this function back further if desired. Note that these files do not implement the exact equations we provide in the homework, so please don’t let these files confuse you! However, the files are similar enough that if you understand what’s going on in this homework, you should feel confident in understanding how this important component of a real-world game engine works! (Note: it is not required to look at these files; it’s just if you’re curious, and provides more context for this assignment.)

2 Background

The bulk of the information here can be found in Epic Games’s 2013 SIGGRAPH presentation which you can see here. You may find it to be a useful resource. The way that the Unreal Engine performs its rendering causes it to be very inefficient to use more than one material type/shading type (they used a deferred renderer). As a result, it is extremely beneficial for them to have a “one-size-fits-all” BRDF which can represent all types of materials well.

There are a couple variables that we will be using throughout this section:

- \( N \) - the normal vector of a vertex
- \( V \) - the normalized vector from the vertex to the camera
- \( L \) - the normalized vector from the vertex to the light
- \( H = \frac{L + V}{\|L + V\|} \)
- \( c_{\text{diff}} \) - the diffuse color of the material
- \( c_{\text{spec}} \) - the specular color of the material
- \( c_{\text{light}} \) - the color of the light
- \( x_{\text{vert}} \) - vertex position

2.1 Material

Epic Games uses three tunable parameters:

- \( r \) - roughness
- \( s_c \) - specular
- \( m \) - metallic

Additionally, throughout their equations, they use \( \alpha = r^2 \). You may wonder, then, how \( c_{\text{diff}} \) and \( c_{\text{spec}} \) are calculated. Their material has another parameter, the base color which is just some color passed to the material (in our case, for this week, that will just be some vertex color). Let’s call it \( c_{\text{base}} \). Then you can compute \( c_{\text{diff}} \) and \( c_{\text{spec}} \) from \( c_{\text{base}} \) using

\[
\begin{align*}
    c_{\text{diff}} &= (1 - m)c_{\text{base}} \quad (12) \\
    c_{\text{spec}} &= \text{lerp}(0.08s_c, c_{\text{base}}, m) \quad (13)
\end{align*}
\]
where lerp is a linear interpolation from the first term to the second term using the third term as the interpolation value (in GLSL, this function is called mix). If you pass in a vector as the first and second parameters to the function, you will get a vector of the same type in return. If you have access to Unreal Engine 4’s source code, you can view Unreal Engine’s computation of these values in their GitHub at Engine/Shaders/Private/BasePassPixelShader.usf.

2.2 BRDF

As in Blinn-Phong, there are two parts to a BRDF: the diffuse component and the specular component.

2.2.1 Diffuse

In Epic’s BRDF, they use Lambertian diffuse. In otherwords, given the diffuse color of the material, the final diffuse color is:

\[ d = \frac{c_{diff}}{\pi} \]  

(14)

Note that the dot product between the normal and the light direction will be accounted for later.

2.2.2 Specular

Where the BRDF starts to differentiate itself, however, is in the specular term. Epic Games uses the Cook-Torrance Microfacet model:

\[ s = \frac{DFG}{4(N \cdot L)(N \cdot V)} \]  

(15)

Note that unless you are told otherwise, this document assumes dot products (i.e. \(N \cdot L\)) are clamped to be non-negative. To clamp a value you should either use the clamp or max function. You can read more about these terms in the presentation, but for this homework, you just have to implement them using the following formulas

\[ D = \frac{\alpha^2}{\pi((N \cdot H)^2(\alpha^2 - 1) + 1)^2} \]  

(16)

\[ k = \frac{(r + 1)^2}{8} \]  

(17)

\[ G_1(v) = \frac{N \cdot v}{(N \cdot v)(1 - k) + k} \]  

(18)

\[ G = G_1(L)G_3(V) \]  

(19)

\[ F = c_{spec} + (1 - c_{spec})^2(-5.55473(V \cdot H) - 6.98316)(V \cdot H), \]  

(20)

Note that in \(G_1(v)\), \(v\) is the input to the function and is not the same as \(V\) which is the normalized vector from the vertex to the camera.
2.2.3 Final Color

Then to put it all together, the final color is:

\[ c_{\text{final}} = c_{\text{light}}(N \cdot L)(d + s) \]  

\[ (21) \]

Note that while the notation provided treats each color channel separately. You will have to perform the identical computations for each color channel (note: you should use vector operations). You can also split out the light color into a “diffuse color” \( c_{\text{light, diffuse}} \) and “specular color” \( c_{\text{light, specular}} \) as is done in the provided Blinn Phong shader in which case you would compute the final color as

\[ c_{\text{final}} = (N \cdot L)(d_{\text{light, diffuse}} + s_{\text{light, specular}}). \]  

\[ (22) \]

2.3 Lighting

In this assignment, we will also be implementing four types of lights: point lights, directional lights, hemisphere lights, and spot lights. These lights are fairly simple. If you decide to go a step further and implement area lights or image-based lighting, you would have to implement sampling which is something we will get to once we start learning about ray-tracing.

2.4 Point Light

A point light a source of light that we say exists at only one point in space. Point lights are fairly simple and you can already see an example of them in action in the Blinn Phong shader in “source/shaders/brdf/blinnphong/frag.” For a point light, the only thing you need to do is pass the light position: \( l_{\text{pos}} \) and the input light color \( c_{\text{light}} \) to the light shader. Then, you can calculate

\[ L = \frac{l_{\text{pos}} - x_{\text{vert}}}{\|l_{\text{pos}} - x_{\text{vert}}\|} \]  

\[ (23) \]

and then shade using the BRDF equations as usual.

2.5 Directional Light

Directional lights can be thought of like a point light that is infinitely far away from you, as a result, all the lights are pointing in the same direction. Directional lights are commonly used to emulate lights that are infinitely far away like the sun. Instead of passing in the light position, you would pass in the direction the light is facing in so you would pass in the light direction \( l_{\text{dir}} \) and the light color \( c_{\text{light}} \). Then you would calculate \( L \) as

\[ L = -l_{\text{dir}} \]  

\[ (24) \]

which is negated because the light direction is in the opposite direction of the vector from the vertex to the light.
2.6 Hemisphere Light

Without the use of image based lighting, it is hard to get the “lighting” that comes from the scene (aka global illumination). One way to fake this is to use hemisphere lights. You can imagine that there is an imaginary sphere around every object and when the normal points upwards, it would see one color and when it points downwards, it sees another. And when it points towards the side, it takes a little bit of both. See Figure 8 for a nice visualization.

![Figure 2: Sourced from Sun and Black Cat](image)

This may seem a little complex, but the math behind it is fairly simple. Instead of passing one light color to the shader, you must instead pass two colors $c_{sky}$ and $c_{ground}$. Then, the resultant light color is

$$c_{light} = \text{lerp}(c_{ground}, c_{sky}, \text{clamp}(N \cdot (0, 1, 0) * 0.5 + 0.5, 0, 1))$$ (25)

$$L = N$$ (26)

The only problem with hemisphere lighting is that it produces inaccurate specular reflections. In other words, we use hemisphere lighting to very simply approximate diffuse interreflection. As a result, you want to disable specular highlights for hemisphere lighting only.

2.7 Spotlight

Spotlights are almost exactly like point lights; however, instead of lighting objects in all directions around the light location, spotlights only light objects that lie within a cone in front of the light location. Therefore, you can compute

$$L = \frac{l_{pos} - x_{vert}}{||l_{pos} - x_{vert}||}$$ (27)
and now we want to use \(-L\) and \(l_{dir}\) to determine whether or not the point is visible to the spotlight. We say that the light’s color \((c_{l,\text{diffuse}} \text{ and } c_{l,\text{spec}})\) is attenuated by a variable \(\gamma\) such that the final diffuse and specular color of this light are \(\gamma c_{l,\text{diffuse}}\) and \(\gamma c_{l,\text{spec}}\). How do we find this \(\gamma\) then?

This is accomplished using two new variables, \(\theta_1\) and \(\theta_2\) which are the inner and outer cone angles respectively (note that \(\theta_2 \geq \theta_1\)). If the angle between \(-L\) and \(l_{dir}\) is \(\leq \theta_1\), then the point at \(l_{pos}\) is fully lit. If the angle between \(-L\) and \(l_{dir}\) is \(> \theta_2\), then the point at \(l_{pos}\) is not lit. If the angle between \(-L\) and \(l_{dir}\) is in between the two cone angles, then the point is partially lit. Convince yourself that this will light objects within a cone in front of the spotlight.

Let us define the angle between \(-L\) and \(l_{dir}\) as \(\theta\); however, computing \(\theta\) using the cosine rule is expensive (who wants to do an acos!). Instead, we can just leverage dot products since we know that \(\cos \theta = -L \cdot l_{dir}\). Furthermore, we also know that we can ignore when the dot product is negative because that would indicate that \(l_{pos}\) is behind the light. Thus, we are only concerned for when \(\cos \theta \geq 0\). In that case, we know that if \(\theta_a > \theta_b\) then \(\cos \theta_a < \cos \theta_b\). Therefore, all we need to do is compute

\[
\cos \theta = -L \cdot l_{dir}
\]

as well as \(\cos \theta_1\) and \(\cos \theta_2\) where \(\cos \theta_2 < \cos \theta_1\). We can thus compute

\[
\gamma = \begin{cases} 
1.0 & \text{if } \cos \theta > \cos \theta_1 \\
0.0 & \text{if } \cos \theta < \cos \theta_2 \\
\cos \theta - \cos \theta_2 & \frac{\cos \theta_1 - \cos \theta_2}{\cos \theta_1 - \cos \theta_2} & \text{otherwise}
\end{cases}
\]  

(30)

Note that this will linearly interpolate the value of \(\gamma\) from 1 to 0 as \(\cos \theta\) goes from \(\cos \theta_1\) to \(\cos \theta_2\).

3 Instructions

At this point, if you have not yet read Section 2, do yourself a favor and go back and read through it carefully. Note that while you are given a barebones structure with functions to fill in, you are welcome to (and encouraged to) modify the function signatures or the way in which they are called in order to complete the homework.

3.1 Setup

Assuming you followed the instructions to download the code from Assignment 1, to get the code for the updated assignment 4 run

```
git fetch source
git merge source/master
```

Go to `main.cpp` and change the line that says:

```
#define APPLICATION Assignment3
```
to

```
#define APPLICATION Assignment4
```
The scene will be constructed inside “Assignment4.cpp” instead of “Assignment3.cpp.” At this point, if you compile and run the assignment framework, you will see some spheres moving up down with what seems to be a singular point light.

![Image](image.png)

Figure 3: What you should see when you first start running the program.

### 3.2 Point Lights

In this section, you will modify the `pointLightSubroutine` function in `shaders/hw4/epic.frag` to properly perform the shading described in Section 2 for the point light described in Section 2.4. You do not have to modify the vertex shader for this assignment. This function should return the output of $(N \cdot L)(\text{dc}_l,\text{diffuse} + \text{sc}_l,\text{spec})$ where $d$ and $s$ are computed using 14 and 15. **This is also true for the other functions you will implement in future sections.** Also remember that a vec4 color represents RGBA color. For the purposes of this assignment, you can assume that the alpha channel should always be set to 1.0. Note that the vectors $N$, $L$, and $N \cdot L$ are already computed for you. Furthermore, the parameters $r$, $s_c$ and $m$ are passed in for you as uniform variables `material.matRoughness`, `material.matSpecular`, and `material.matMetallic`. You should also assume that `fragmentColor` corresponds to $c_{\text{base}}$, `genericLight.diffuseColor` corresponds to $c_{l,\text{diffuse}}$, and `genericLight.specularColor` corresponds to $c_{l,\text{spec}}$.

First, you should compute $c_{\text{diff}}$ and $c_{\text{spec}}$ using Equations 12 and 13. You can then use $c_{\text{diff}}$ to compute $d$ using Equation 14. Multiply this result with $c_{l,\text{diffuse}}$ and return the final result from `pointLightSubroutine` and you will have implemented Lambertian diffuse. See Figure 4.

Now, you can move on to computing $s$ using Equation 15. First, compute $V$ (see Equation 2). Note that you are given the position of the vertex in `vertexWorldPosition` and the position of the camera in `cameraPosition`. Do not forget to normalize this vector. At this point you should be able to compute $4(N \cdot L)(N \cdot V)$ from Equation 15. In the case $4(N \cdot L)(N \cdot V)$ is equal to 0, you should set $s = 0$ as well. Now you should be able to use Equations 16, 17, 18, 19, and 20 to compute the rest of the terms needed to compute $s$. You will need to use the `pow` function to compute $F$. Finally, add the result of $sc_{l,\text{spec}}$ to the result of $dc_{l,\text{diffuse}}$ and return this value from `pointLightSubroutine` and you should now see your red diffuse spheres with white specular highlights! See Figure 5.
3.3 Directional Lights

Now that the hard part is out of the way, let’s start experimenting with different lights! The first new light we will add is the directional light as described in Section 2.5. You can implement this functionality inside the \texttt{directionalLightSubroutine} function. As mentioned earlier, directional lights are used to imitate lights that are infinitely (or just really) far away like the sun. We give you \( l_{\text{dir}} \) as \texttt{genericLight.directionalLightDir} which you can use to easily compute \( L \) using Equation 24. Then you can do the exact same shading you did in the previous section using the new \( L \) you computed here. The spheres should now look like there is a green light that is rotating around all of them (see Figure 7).
3.4 Hemisphere Lights

Now we will implement hemisphere lights as described in Section 2.6. For hemisphere lights, we will reuse `genericLight.diffuseColor` and `genericLight.specularColor` as $c_{ground}$ and $c_{sky}$. Note that this will not affect the computations you did for point or directional lights because each light is handled separately so `genericLight.diffuseColor` will only be equal to $c_{ground}$ when `hemisphereLightSubroutine` is called (or the same is true for `genericLight.specularColor` and $c_{sky}$). Remember, hemisphere lights do not affect specular! Thus, using the shading equations from before, $c_{l,\text{diffuse}} = c_{light}$ from Equation 25 and $c_{l,\text{spec}} = 0$. Combining this fact with the equation for $L$ from Equation 26 allows you to easily reuse your existing shading computations to compute the hemisphere light color. After implementing this correctly, you should now see
your spheres lit by a blue light from the top and lit with a brown-ish light from the bottom (see Figure 8).

![Figure 8](image-url)

Figure 8: A scene lit with a functioning hemisphere light, directional light, and point light.

### 3.5 Spotlights

Now we will implement spotlights as described in Section 2.7. You will write your code in the `spotLightSubroutine` function. Note that you are given the spotlight position $l_{pos}$ as `pointLight.pointPosition` and the direction the spotlight is facing $l_{dir}$ as `genericLight.directionalLightDir`. You are also given $\theta_1$ and $\theta_2$ in degrees, as described in Section 2.7, as `genericLight.spotInnerConeAngleDegrees` and `genericLight.spotOuterConeAngleDegrees` respectively. You must compute $\gamma$ and apply it to $l_{c,diffuse}$ and $l_{c,spec}$ to get credit for implementing spotlights. Upon completion, you should see the spheres in the default scene in the bottom left corner lit with a yellow spotlight (see Figure 9).

### 3.6 Swap your Scene In

At this point, you should be able to take out the sphere scene and do a similar setup to what you did for the scene from Assignment 3. Except that this time you will want to load the shaders you just made, create lights, etc. to get your scene shaded with the Epic BRDF using a point light, directional light, and hemisphere light. Note that the camera is different from Assignment 3 (look at “Assignment4::CreateCamera” and “Assignment4::SetupCamera”). I highly suggest you read the pinned “Camera, Light, and Geometry Position/Rotation/Scale using Autodesk Maya” post on Piazza to help you in placing your lights, objects, and cameras.

### 4 Further Reading (Optional)

These are just a bunch of links to SIGGRAPH presentations by various companies about their move towards physically based rendering. You may find it interesting.

- Unreal Engine SIGGRAPH 2013 Presentation: [here](#).
Figure 9: A scene lit with a functioning spotlight, hemisphere light, directional light, and point light.

- Disney SIGGRAPH 2012 Presentation: here.
- The Order 1886 SIGGRAPH 2013 Presentation: here.

Additionally, there is a ton of freely available source code for you to read online:

- Unreal Engine 4: Instructions for access can be found here.
- Pixar’s Renderman: Get the free and non-commercial version here.

5 Grading

This assignment will be graded on the following requirements:

- Your scene is shaded using the Unreal Engine BRDF and material.
- Your scene contains at least one point light.
- Your scene contains at least one directional light.
- Your scene contains at least one hemisphere light.
- Your scene contains at least one spotlight.

according to the following rubric.

- 4 – All four light types (point/directional/hemisphere/spotlight) using the Epic BRDF are implemented and in your scene at the same time.
- 3 – Two functional light types (point/directional/hemisphere/spotlight) using the Epic BRDF are implemented.
- 2 – One functional light type (point/directional/hemisphere/spotlight) using the Epic BRDF is implemented.
- 1 – Attempted to code up the Epic BRDF but nothing works.
- 0 – Fail to show up for grading.

The contribution to the shading of each light must be clearly demonstrated to receive credit for that light type. If you want, you can toggle each light on/off individually using custom keybindings in
Assignment4::HandleInput to show the effect of each individual light. **Note:** **In the end, all four lights must be in the scene at the same time to receive full points.** The spotlight must exhibit a smooth transition from the inner cone to the outer cone.

### 5.1 Clarifying Points

The term *Your Scene* refers to the scene you made from Assignment 3 (improvements and changes are fine). In other words, the default scene with the spheres is not valid for grading. **If you show up with only the sphere scene, you will receive a 0.** However, you may be asked to demonstrate the lights working on the sphere scene to ensure that you implemented the lights and the Epic BRDF correctly. Note that if you did not do so last week, you will have to make sure your camera, lights, and meshes are placed properly or else your objects will not be shaded or lit correctly.