Assignment 6: Lighting

CS Spring 2018

Due Date: 19/04/2018

Follow the instructions carefully. If you encounter any problems in the setup, please do not hesitate to reach out to TA.

Introduction

Lighting in the real world is extremely complicated and depends on way too many factors, something we can't afford to calculate on the limited processing power we have. Lighting in OpenGL is therefore based on approximations of reality using simplified models that are much easier to process and look relatively similar. These lighting models are based on the physics of light as we understand it. One of those models is called the Phong lighting model. The major building blocks of the Phong model consist of 3 components: ambient, diffuse and specular lighting. We will start with the simplest *ambient lighting*.

Ambient Lighting

Light usually does not come from a single light source, but from many light sources scattered all around us, even when they're not immediately visible. One of the properties of light is that it can scatter and bounce in many directions reaching spots that aren't in its direct vicinity; light can thus reflect on other surfaces and have an indirect impact on the lighting of an object. Algorithms that take this into consideration are called global illumination algorithms, but these are expensive and/or complicated.

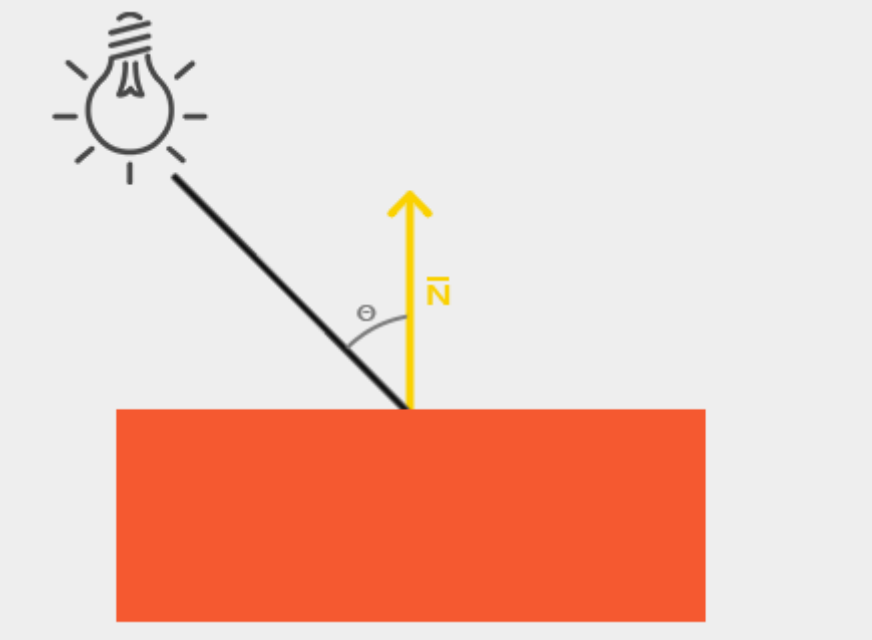
Since we're not big fans of complicated and expensive algorithms we'll start by using a very simplistic model of global illumination, namely ambient lighting. As you've seen in the previous section we use a small constant (light) color that we add to the final resulting color of the object's fragments, thus making it look like there is always some scattered light even when there's not a direct light source.

```
float ambientStrength = 0.1;
vec3 ambient = ambientStrength * lightColor;

vec3 result = ambient * objectColor;
FragColor = vec4(result, 1.0);
```

Diffuse Lighting

Ambient lighting by itself does not produce the most interesting results, but diffuse lighting will start to give a significant visual impact on the object. Diffuse lighting gives the object more brightness the closer its fragments are aligned to the light rays from a light source.



To the left we find a light source with a light ray targeted at a single fragment of our object. We then need to measure at what angle the light ray touches the fragment.

To calculate diffuse lighting, we need: *normal vector* and *the directed light ray*.

Normal Vectors

A normal vector is a (unit) vector that is perpendicular to the surface of a vertex. Since a vertex by itself has no surface (it's just a single point in space) we retrieve a normal vector by using its surrounding vertices to figure out the surface of the vertex. We can use a little trick to calculate the normal vectors for all the cube's vertices by using the cross product, but since a 3D cube is not a complicated shape we can simply manually add them to the vertex data.

So, the updated vertex shader becomes

```
#version 330 core
layout (location = 0) in vec3 aPos;
layout (location = 1) in vec3 aNormal;
```

Now that we added a normal vector to each of the vertices and updated the vertex shader we should update the vertex attribute pointers as well.

```
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 6 *
sizeof(float), (void*)0);
glEnableVertexAttribArray(0);
```

All the lighting calculations are done in the fragment shader, so we need to forward the normal vectors from the vertex shader to the fragment shader.

```
gl_Position = projection * view * model * vec4(aPos, 1.0);
Normal = aNormal;
```

Calculating the Diffuse Color

We now have the normal vector for each vertex, but we still need the light's position vector and the fragment's position vector. Since the light's position is just a single static variable we can simply declare it as a uniform in the fragment shader:

```
uniform vec3 lightPos;
```

And then update the uniform in the game loop (or outside since it doesn't change). We use the lightPos vector declared in the previous tutorial as the location of the light source:

```
lightingShader.setVec3("lightPos", lightPos);
```

Then the last thing we need is the actual fragment's position. We're going to do all the lighting calculations in world space, so we want a vertex position that is in world space.

```
gl_Position = projection * view * model * vec4(aPos, 1.0);
FragPos = vec3(model * vec4(aPos, 1.0));
Normal = aNormal;
```

Now that all the required variables are set we can start with the lighting calculations in the fragment shader.

The first thing we need to calculate is the direction vector between the light source and the fragment's position. We mentioned that the light's direction vector is the difference vector between the light's position vector and the fragment's position vector.

```
vec3 norm = normalize(Normal);
vec3 lightDir = normalize(lightPos - FragPos);
```

Next, we want to calculate the actual diffuse impact the light has on the current fragment by taking the dot product of the norm and lightDir vector.

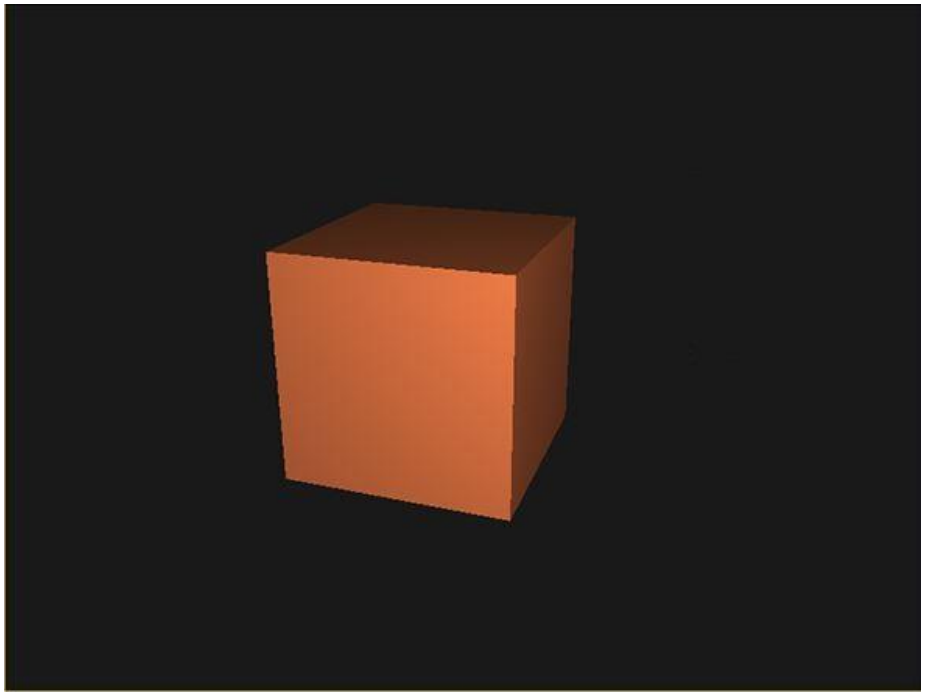
```
float diff = max(dot(norm, lightDir), 0.0);
vec3 diffuse = diff * lightColor;
```

Now that we have both an ambient and a diffuse component we add both colors to each other and then multiply the result with the color of the object to get the resulting fragment's

output color:

```
vec3 result = (ambient + diffuse) * objectColor;
FragColor = vec4(result, 1.0);
```

If your application (and shaders) compiled successfully you should see something like this:



Exercise

1. Try to get a similar result as the tutorial's. Since most of you already know where to find the source code, the requirements are:
   - Only one cube with (sustech green) in the scene;
   - your viewing angle must be different from the example;
   - try different diffuse lighting parameter to the cube.