ADVANCED LIGHTING

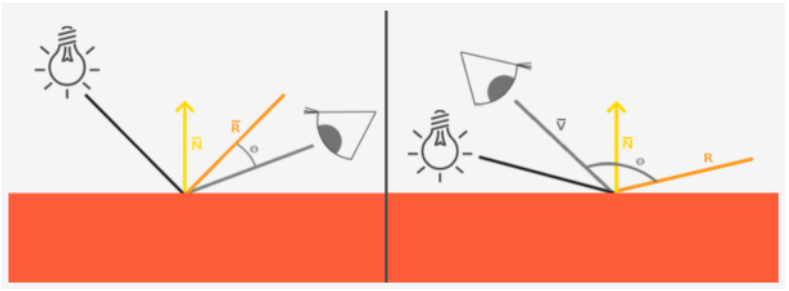
In the lighting chapters, we briefly introduced the Phong lighting model. While it looks nice, there are some nuances to that technique that we discuss in this chapter.

Blinn-Phong Model

The content we cover in this chapter most closely relates to what we learned in the past lighting chapters, however, all the shaders and important code for setting up the scene are provided here.

Phong lighting is a very efficient approximation of lighting, but its specular reflections become rough when the shininess of a material is low.

- As seen in the image to the right, the edges of the specular area (the whiter area) is immediately cut off.
  - This happens because the angle between the view and reflection vector doesn't go over 90 degrees, so if the angle is larger than 90 degrees, the resulting dot product becomes negative and this results in a specular exponent of `0.0`.

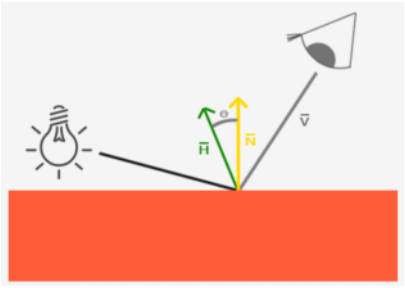


You may think that we don't need to get any light when the angle between the view and direction vector are over 90 degrees, but that's incorrect.

- This only applies to the diffuse component where an angle higher than 90 degrees between the normal and light source means the light source is below the lit surface and, thus, the light's diffuse contribution should equal 0.
- However, with specular lighting, we're not measuring the angle between the light source and the normal, but between the view and reflection vector.

The issue should become apparent after viewing the two images to the right.

- The left image shows Phong reflections with Θ less than 90 degrees.
- The right image shows Phong reflections with Θ greater than 90 degrees.
  - This nullifies the specular contribution.
  - This generally isn't a problem since the view direction is far from the reflection direction, but if we use a low specular exponent, the specular radius is larger enough to have a contribution under these conditions.
  - Since we're nullifying this contribution at angles larger than 90 degrees, we get the artifact as seen in the first image.



The Blinn-Phong shading model was introduced as an extension to the Phong shading model (that we've been using).

- The Blinn-Phong model is largely similar, but addresses the specular model slightly differently, resolving the visual artifact described above (reference the image to the right).
  - Instead of relying on a reflection vector, we use a **halfway vector**, which is a unit vector exactly halfway between the view direction and the light direction. The closer the halfway vector aligns with the surface's normal vector, the higher the specular contribution.
  - When the view direction is perfectly aligned with the (now imaginary) reflection vector, the halfway vector aligns perfectly with the normal vector, emitting the full specular highlight.
  - With this model, no matter what direction the viewer looks from, the angle between the halfway vector and the surface normal never exceeds 90 degrees (unless the light is far below the surface, of course). The results are slightly different from Phong reflections, but generally more visually plausible, especially with low specular exponents.

Getting the halfway vector is as simple as adding the light's direction vector and the view vector together and normalizing the result, like so:

- $\vec{H} = \frac{\vec{L} + \vec{V}}{||\vec{L} + \vec{V}||}$

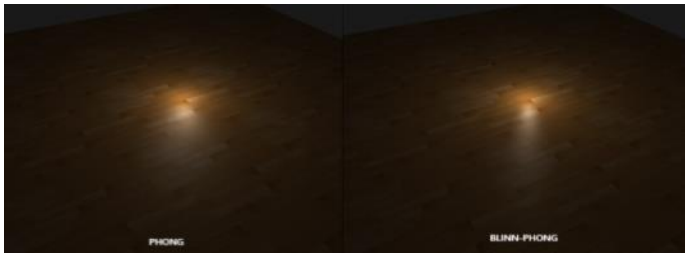
Implement Blinn-Phong specular shading.

- Blinn-Phong Shading

The image to the right shows a noticeable difference in the two shading methods when the shininess of the material is low (`0.5` in the image).

Another difference between Phong and Blinn-Phong shading is that the angle between the halfway vector and the surface normal is often shorter than the angle between the view and reflection vector. As a result, to get visuals similar to Phong shading, the specular shininess exponent has to be set higher (2 or 4 times higher is the general rule of thumb).

- In the image to the right, the Phong specular component is set to `8.0` and the Blinn-Phong specular component is set to `32.0`. Their similarity emphasizes the above rule of thumb.
  - You can see that the Blinn-Phong specular component is a bit sharper compared to Phong, so you will have to do a bit of tweaking to get similar results.



Scene Setup

Tuesday, May 24, 2022 11:59 PM

Everything here is named some variant of blinn-phong rather than phong, even though the implementation below is for the phong model. This is just because the two are very similar and I didn't want to make two separate implementations for basically the same thing, though you may want to separate phong and blinn-phong shading models into two separate shaders.

blinn-phong.vert

```
#version 330 core

layout (location = 0) in vec3 aPos;
layout (location = 1) in vec3 aNormal;
layout (location = 2) in vec2 aTexCoords;

out vec3 fragPos;
out vec3 normal;
out vec2 texCoords;

uniform mat4 projection;
uniform mat4 view;
uniform mat4 model;

void main() {
   gl_Position = projection * view * model * vec4(aPos, 1.0);
   texCoords = aTexCoords;
   fragPos = vec3(model * vec4(aPos, 1.0));
   normal = normalize(mat3(transpose(inverse(model)))* aNormal);
}
```

blinn-phong.frag

```
#version 330 core

struct Material {
   sampler2D diffuse;
   sampler2D specular;

   float shininess; // scattering/radius of the specular highlight
};
uniform Material material;

struct PointLight {
   vec3 position;

   float constant;
   float linear;
   float quadratic;

   vec3 ambient;
   vec3 diffuse;
   vec3 specular;
};
uniform PointLight pointLight;

in vec2 texCoords;
in vec3 normal;
in vec3 fragPos;

out vec4 FragColor;

uniform vec3 viewPos;

vec3 CalcPointLight(PointLight light, vec3 viewDir);

void main() {
   vec3 viewDir = normalize(viewPos - fragPos);

   vec3 color = CalcPointLight(pointLight, viewDir);

   FragColor = vec4(color, 1.0);
}

vec3 CalcPointLight(PointLight light, vec3 viewDir) {
   // Calculating Attenuation
   float distance = length(light.position - fragPos);
   float attenuation = 1.0 / (light.constant + light.linear * distance + light.quadratic * (distance * distance));
```

```
    // Ambient Lighting
    vec3 ambient = light.ambient * vec3(texture(material.diffuse, texCoords));

    // Diffuse Lighting
    vec3 lightDir = normalize(light.position - fragPos);
    float diff = max(dot(normal, lightDir), 0.0);
    vec3 diffuse = light.diffuse  * diff * vec3(texture(material.diffuse, texCoords)) * attenuation;

    // Phong Specular Lighting
    vec3 reflectDir = reflect(-lightDir, normal);
    float spec = pow(max(dot(viewDir, reflectDir), 0.0), material.shininess);
    vec3 specular = light.specular * spec * vec3(texture(material.specular, texCoords)) * attenuation;

    return (ambient + diffuse + specular);
}
```

main.cpp

```
float planeVertices[] = {
    // Positions            // Normals         // Tex Coords
     10.0f, -0.5f,  10.0f,  0.0f, 1.0f, 0.0f,  10.0f,  0.0f,
    -10.0f, -0.5f,  10.0f,  0.0f, 1.0f, 0.0f,   0.0f,  0.0f,
    -10.0f, -0.5f, -10.0f,  0.0f, 1.0f, 0.0f,   0.0f, 10.0f,

     10.0f, -0.5f,  10.0f,  0.0f, 1.0f, 0.0f,  10.0f,  0.0f,
    -10.0f, -0.5f, -10.0f,  0.0f, 1.0f, 0.0f,   0.0f, 10.0f,
     10.0f, -0.5f, -10.0f,  0.0f, 1.0f, 0.0f,  10.0f, 10.0f
};
```

main()

```
Shader blinnPhongShader{ "Shaders/blinn-phong.vert", "Shaders/blinn-phong.frag" };

Texture2D planksDiffuseTexture{ "Textures/planks.png", 0, GL_REPEAT, GL_REPEAT };
Texture2D fullSpecularTexture{ "Textures/full_specular.png", 1, GL_REPEAT, GL_REPEAT };

// Plane VAO setup
unsigned int planeVAO, planeVBO;
glGenVertexArrays(1, &planeVAO);
glGenBuffers(1, &planeVBO);

glBindVertexArray(planeVAO);
glBindBuffer(GL_ARRAY_BUFFER, planeVBO);

glBufferData(GL_ARRAY_BUFFER, sizeof(planeVertices), &planeVertices, GL_STATIC_DRAW);

glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(float), (void*)0);
glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(float), (void*)(sizeof(float) * 3));
glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 8 * sizeof(float), (void*)(sizeof(float) * 6));

glEnableVertexAttribArray(0);
glEnableVertexAttribArray(1);
glEnableVertexAttribArray(2);

glBindVertexArray(0);
glBindBuffer(GL_ARRAY_BUFFER, 0);
```

Render Loop

```
// RENDERING
glClearColor(0.2f, 0.2f, 0.2f, 1.0f);
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

glm::mat4 objModel{ 1.0f };
glm::mat4 view = camera.GetViewMatrix();
glm::mat4 projection = glm::perspective(glm::radians(camera.fov), (float)WIDTH / (float)HEIGHT, 0.1f, 100.0f);

blinnPhongShader.use();

planksDiffuseTexture.bind();
fullSpecularTexture.bind();

// Sets the transformation matrices
glUniformMatrix4fv(glGetUniformLocation(blinnPhongShader.id, "model"), 1, GL_FALSE, glm::value_ptr(objModel));
glUniformMatrix4fv(glGetUniformLocation(blinnPhongShader.id, "view"), 1, GL_FALSE, glm::value_ptr(view));
glUniformMatrix4fv(glGetUniformLocation(blinnPhongShader.id, "projection"), 1, GL_FALSE, glm::value_ptr(projection));
```

```
glUniform3fv(glGetUniformLocation(blinnPhongShader.id, "viewPos"), 1, glm::value_ptr(camera.position));

// Sets the material properties
glUniform1i(glGetUniformLocation(blinnPhongShader.id, "material.diffuse"), planksDiffuseTexture.getTexUnit());
glUniform1i(glGetUniformLocation(blinnPhongShader.id, "material.specular"), fullSpecularTexture.getTexUnit());
glUniform1f(glGetUniformLocation(blinnPhongShader.id, "material.shininess"), 0.5f); // Set to 0.5f to view visual artifact

// Sets the point light properties
glm::vec3 lightColor{ 1.0f, 1.0f, 1.0f };
glm::vec3 lightPos{ 0.0f, 3.0f, 0.0f };
glUniform3fv(glGetUniformLocation(blinnPhongShader.id, "pointLight.position"), 1,
    glm::value_ptr(lightPos));
glUniform3fv(glGetUniformLocation(blinnPhongShader.id, "pointLight.ambient"), 1,
    glm::value_ptr(glm::vec3(0.05f) * lightColor));
glUniform3fv(glGetUniformLocation(blinnPhongShader.id, "pointLight.diffuse"), 1,
    glm::value_ptr(glm::vec3(0.8f) * lightColor));
glUniform3fv(glGetUniformLocation(blinnPhongShader.id, "pointLight.specular"), 1,
    glm::value_ptr(glm::vec3(1.0f) * lightColor));
glUniform1f(glGetUniformLocation(blinnPhongShader.id, "pointLight.constant"), 1.0f);
glUniform1f(glGetUniformLocation(blinnPhongShader.id, "pointLight.linear"), 0.09f);
glUniform1f(glGetUniformLocation(blinnPhongShader.id, "pointLight.quadratic"), 0.032f);

glBindVertexArray(planeVAO);
glDrawArrays(GL_TRIANGLES, 0, 6);
```

Blinn-Phong Shading

Wednesday, May 25, 2022 12:25 AM

```
vec3 CalcPointLight(PointLight light, vec3 viewDir) {
   ...

   // Blinn-Phong Specular Lighting
   vec3 halfwayDir = normalize(viewDir + lightDir);
   // Calculates cosine angle between halfway and normal vector, clamped to at least 0.0, and exponentiates by the shininess of the material
   float spec = pow(max(dot(normal, halfwayDir), 0.0), material.shininess);
   vec3 specular = light.specular * spec * vec3(texture(material.specular, texCoords)) * attenuation; // Same as always

   return (ambient + diffuse + specular);
}
```