

# 计算机图形学第六次作业

16340028 陈思航

## 题目

- 实现Phong光照模型：
  - 场景中绘制一个cube
  - 自己写shader实现两种shading: Phong Shading 和 Gouraud Shading, 并解释两种shading的实现原理合理设置视点、光照位置、光照颜色等参数, 使光照效果明显显示
- 使用GUI, 使参数可调节, 效果实时更改:
  - GUI里可以切换两种shading
  - 使用如进度条这样的控件, 使ambient因子、diffuse因子、specular因子、反光度等参数可调节, 光照效果实时更改
- Bonus:
  - 当前光源为静止状态, 尝试使光源在场景中来回移动, 光照效果实时更改。

## 光影的理解

我们现实中看到物体的颜色其实是其反射的颜色。物体可以吸收（该颜色我们看不到）和反射（我们看到的颜色）不同颜色的光。在OpenGL中, 我们创建一个白色的光源（RGB值为255, 255, 255）, 把光源的颜色与物体的颜色值相乘, 得到的就是物体所反射的颜色。

## 在原来的基础上添加新的VAO

在立方体的基础上, 我们还需要有表示光源的立方体, 为了方便后面属性的修改, 需要添加新的VAO。

```
unsigned int lightVAO;  
glGenVertexArrays(1, &lightVAO);  
glBindVertexArray(lightVAO);  
glBindBuffer(GL_ARRAY_BUFFER, VBO);  
  
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(float), (void*)0);  
glEnableVertexAttribArray(0);
```

## phong光照模型

phong光照模型由环境（Ambient）、漫反射（Diffuse）以及镜面（Specular）光照组成。

- 环境光照会改变光照的强度, 实现物体的明暗效果
- 漫反射以及镜面光照则改变物体受光照的影响
  - 漫反射分量越大, 物体对着光源的那部分就会越亮
  - 镜面光照分量越大则物体反光能力越强（越容易在物体表面上出现亮点）

其中phong模型的段着色器如下：

```
#version 450 core
out vec4 FragColor;

in vec3 Normal;
in vec3 FragPos;

uniform vec3 lightPos;
uniform vec3 viewPos;
uniform vec3 lightColor;
uniform vec3 objectColor;

uniform float ambientStrength;
uniform float specularStrength;
uniform float shininess;
uniform float diffuseFactor;

void main()
{
    // ambient
    vec3 ambient = ambientStrength * lightColor;

    // diffuse
    vec3 norm = normalize(Normal);
    vec3 lightDir = normalize(lightPos - FragPos);
    float diff = max(dot(norm, lightDir), 0.0);
    vec3 diffuse = diff * lightColor * diffuseFactor;

    // specular
    vec3 viewDir = normalize(viewPos - FragPos);
    vec3 reflectDir = reflect(-lightDir, norm);
    float spec = pow(max(dot(viewDir, reflectDir), 0.0), shininess);
    vec3 specular = specularStrength * spec * lightColor;

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

## 环境光照

环境光照中，对于输入的环境因子 `ambientStrength`，乘以输入的光照的颜色，可以得到环境光照 `ambient`。该 `ambient` 变量乘以光照颜色可以得到片段的颜色（在计算结果中得到的仅仅是片段颜色的一部分）。

```
// ambient
vec3 ambient = ambientStrength * lightColor;
```

## 漫反射

### 坐标转换

首先，我们需要在顶点着色器中将输入的顶点位置坐标乘以模型矩阵，将其转变为世界空间坐标。顶点着色器定义如下：

```
#version 450 core
layout (location = 0) in vec3 aPos;
layout (location = 1) in vec3 aNormal;

out vec3 FragPos;
out vec3 Normal;

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

void main()
{
    FragPos = vec3(model * vec4(aPos, 1.0));
    Normal = mat3(transpose(inverse(model))) * aNormal;

    gl_Position = projection * view * vec4(FragPos, 1.0);
}
```

因为法向量是一个方向向量而不能表达空间中的特定位置（坐标），所以我们通过法线矩阵进行法向量向世界坐标的变换。在运用中，我们使用 `inverse` 以及 `transpose` 函数生成法线矩阵。

```
Normal = mat3(transpose(inverse(model))) * aNormal;
```

## 利用法向量进行计算

漫反射能够对物体产生显著的视觉影响。对于在程序中手动输入并且传到片段着色器中的法向量 `Normal` `Factor`（法向量为垂直于顶点表面的单位向量），我们还需要有如下的操作：

- 计算光源位置与片段位置之间的方向向量，即光的方向向量。
- 将光的方向向量进行标准化，因为我们只关注方向。
- 将标准化后的法向量 `norm` 与光的方向向量 `lightDir` 进行点乘，计算出光源对当前片段的漫反射影响。
  - 如果两个向量的角度越大，点乘的结果越小，则漫反射分量越小。
- 进行合法性判断，如果点乘结果小于零则置为0（角度大于90度时为负数）。
- 计算结果乘以物体颜色以及漫反射参数 `diffuseFactor`，得到结果。

```
// diffuse
vec3 norm = normalize(Normal);
vec3 lightDir = normalize(lightPos - FragPos);
float diff = max(dot(norm, lightDir), 0.0);
vec3 diffuse = diff * lightColor * diffuseFactor;
```

## 镜面光照

镜面光照与光的方向向量以及物体的法向量决定，它与观察向量也是有关的。我们通过法向量计算其反射向量，再计算反射向量与视线方向的角度差。夹角越小则镜面光的影响越大（产生高光）。

在段着色器中，我们需要更改其镜面因素 `specularStrength`。同时，我们还需对 `lightDir` 进行取反，而 `reflect` 函数需要第一个向量是从光源指向片段位置的向量，第二个则是标准化后的法向量。计算的时候，需要注意视线方向与反射方向向量的点乘需要确保大于或等于0，之后取幂值（幂值为输入的 `shininess` 变量，称反光度。如果反光度越高，反射光能力越强，散射越小，同时高光的点越小。）

```
// specular
vec3 viewDir = normalize(viewPos - FragPos);
vec3 reflectDir = reflect(-lightDir, norm);
float spec = pow(max(dot(viewDir, reflectDir), 0.0), shininess);
vec3 specular = specularStrength * spec * lightColor;
```

最后，算出了全部分量。

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

## Gouraud

在顶点着色器中实现phong模型的方法叫做Gouraud着色。

与phong模型类似，需要有环境光照、漫反射光照、镜面光照三个分量。不过在计算反射方向向量的时候，是计算光源到顶点之间的距离，而不是计算顶点到片段之间的距离。

Gouraud模型中的点着色器如下：

```
#version 450 core
layout (location = 0) in vec3 aPos;
layout (location = 1) in vec3 aNormal;

out vec3 LightingColor; // resulting color from lighting calculations

uniform vec3 lightPos;
uniform vec3 viewPos;
uniform vec3 lightColor;

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

uniform float ambientStrength;
uniform float specularStrength;
uniform float shininess;
uniform float diffuseFactor;

void main()
{
    gl_Position = projection * view * model * vec4(aPos, 1.0);

    // gouraud shading
    // -----
    vec3 Position = vec3(model * vec4(aPos, 1.0));
    vec3 Normal = mat3(transpose(inverse(model))) * aNormal;
```

```

// ambient
vec3 ambient = ambientStrength * lightColor;

// diffuse
vec3 norm = normalize(Normal);
vec3 lightDir = normalize(lightPos - Position);
float diff = max(dot(norm, lightDir), 0.0);
vec3 diffuse = diff * lightColor * diffuseFactor;

// specular
vec3 viewDir = normalize(viewPos - Position);
vec3 reflectDir = reflect(-lightDir, norm);
float spec = pow(max(dot(viewDir, reflectDir), 0.0), shininess);
vec3 specular = specularStrength * spec * lightColor;

LightingColor = ambient + diffuse + specular;
}

```

片段着色器如下：

```

#version 450 core
out vec4 FragColor;

in vec3 LightingColor;

uniform vec3 objectColor;

void main()
{
    FragColor = vec4(LightingColor * objectColor, 1.0);
}

```

phong模型是在片段着色器中，对于给定的光照位置、光照颜色、视觉位置、物体位置进行片段颜色的确定。在gouraud模型中，是在顶点着色器内，通过上述三个分量进行光照颜色确定，输出给段着色器。而对于环境光照、漫反射光照以及镜面光照的计算过程，两者是相似的（区别是gouraud利用顶点位置插值计算，phong利用片段位置进行插值计算）。

## 两者区别

Gouraud的计算只在顶点处采用phong的局部反射模型进行光照计算，其他的点均采用双线性插值的方法进行光照强度的计算。Gouraud模型计算更加快速，并且用到较少的浮点数。在多边形更小的情况下渲染的小过更好。但是不能再平滑表面使用，因为有非常明显的马赫带（每一面可能渲染为非常明显的两个三角形）。而对于phong，则通过插值的方法获得法向量再逐像素点进行光照计算。这种方法计算的成本更高，因为计算量增大并且需要进行浮点计算。但是在暗部以及光滑平面的效果更好并且更加逼真。所以，如果需要追求渲染速度，则通过Gouraud模型进行计算；如果追求效果则进行Phong模型计算。

## 添加ImGui

GUI包括选择模型、选择光源是否移动、调整各种因素的功能。

```

/**
 * 使用IMGUI
 */
ImGui_ImplOpenGL3_NewFrame();
ImGui_ImplGlfw_NewFrame();
ImGui::NewFrame();
ImGui::Begin("Attributes");
ImGui::Text("Choose the model");
ImGui::RadioButton("phong", &currentModel, phongType);
ImGui::RadioButton("gouraud", &currentModel, gouraudType);
ImGui::Separator();
ImGui::Checkbox("Move the light", &isMove);

ImGui::SliderFloat("Ambient Strength", &ambientStrength, 0.0f, 3.0f);
ImGui::SliderFloat("Specular Strength", &specularStrength, 0.0f, 5.0f);
ImGui::SliderFloat("Shininess", &shininess, 1.0f, 4000.0f);
ImGui::SliderFloat("Diffuse Factor", &diffuseFactor, 0.0f, 1.0f);
ImGui::End();

```

## 光源旋转

该功能与前两次的地球公转类似，只需要通过参数方程利用函数 `glfwGetTime()` 获得时间从而确定其位置即可。

```

// 移动光源位置
if (isMove) {
    float camPosX = sin(glfwGetTime() * 0.5) * (float)radius;
    float camPosZ = cos(glfwGetTime() * 0.5) * (float)radius;
    lightPos = glm::vec3(camPosX, lightPos.y, camPosZ);
}

```

## 光源立方体的渲染

光源立方体的渲染过程与前面作业的立方体渲染过程类似，利用 `projection`、`model`、`view` 变量进行投影以及视角调整，这里不多赘述。

```

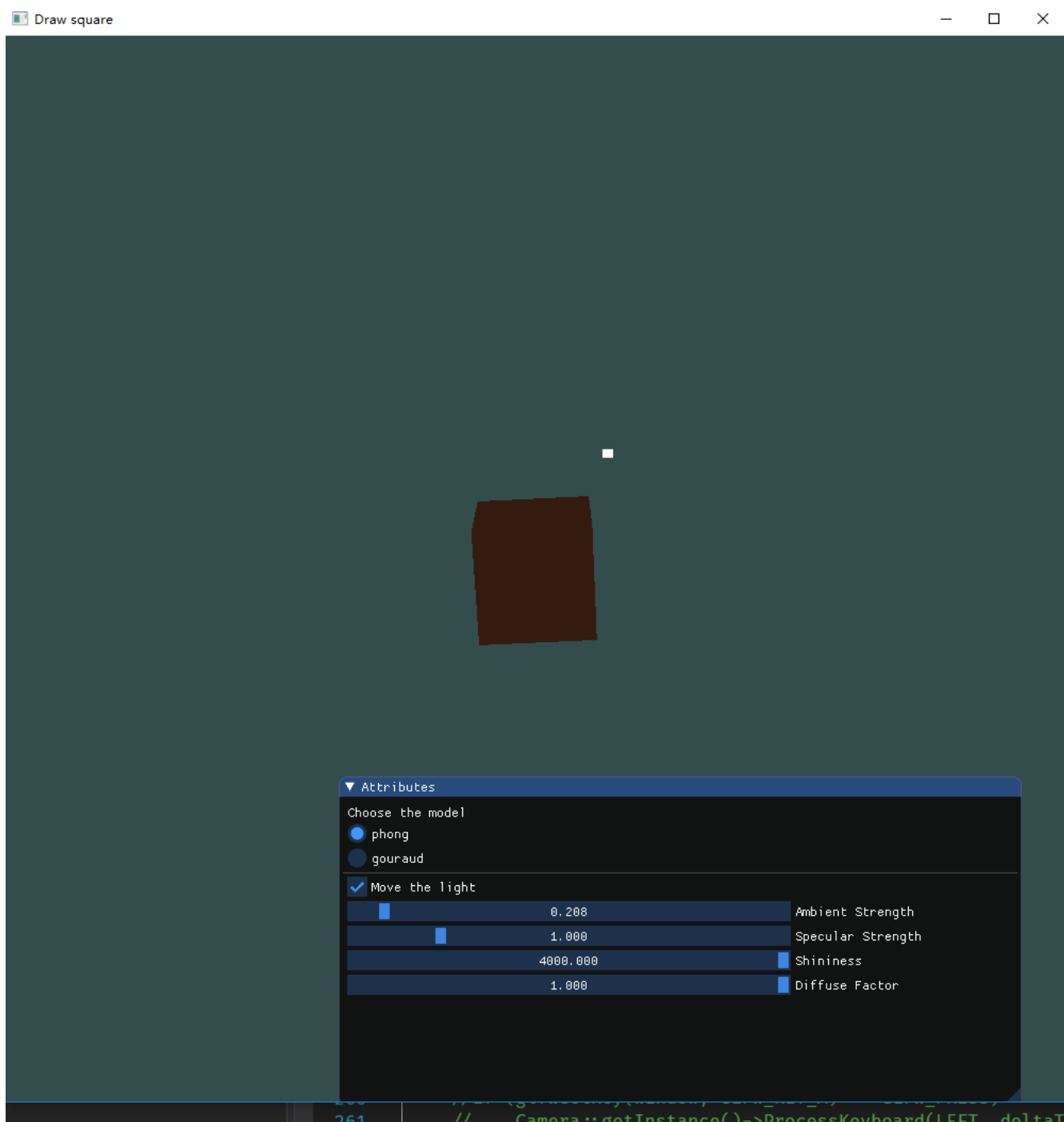
lampShader.use();
model = glm::mat4(1.0f);
model = glm::translate(model, lightPos);
model = glm::scale(model, glm::vec3(0.1f, 0.1f, 0.1f));
lampShader.setMat4("projection", projection);
lampShader.setMat4("model", model);
lampShader.setMat4("view", view);

```

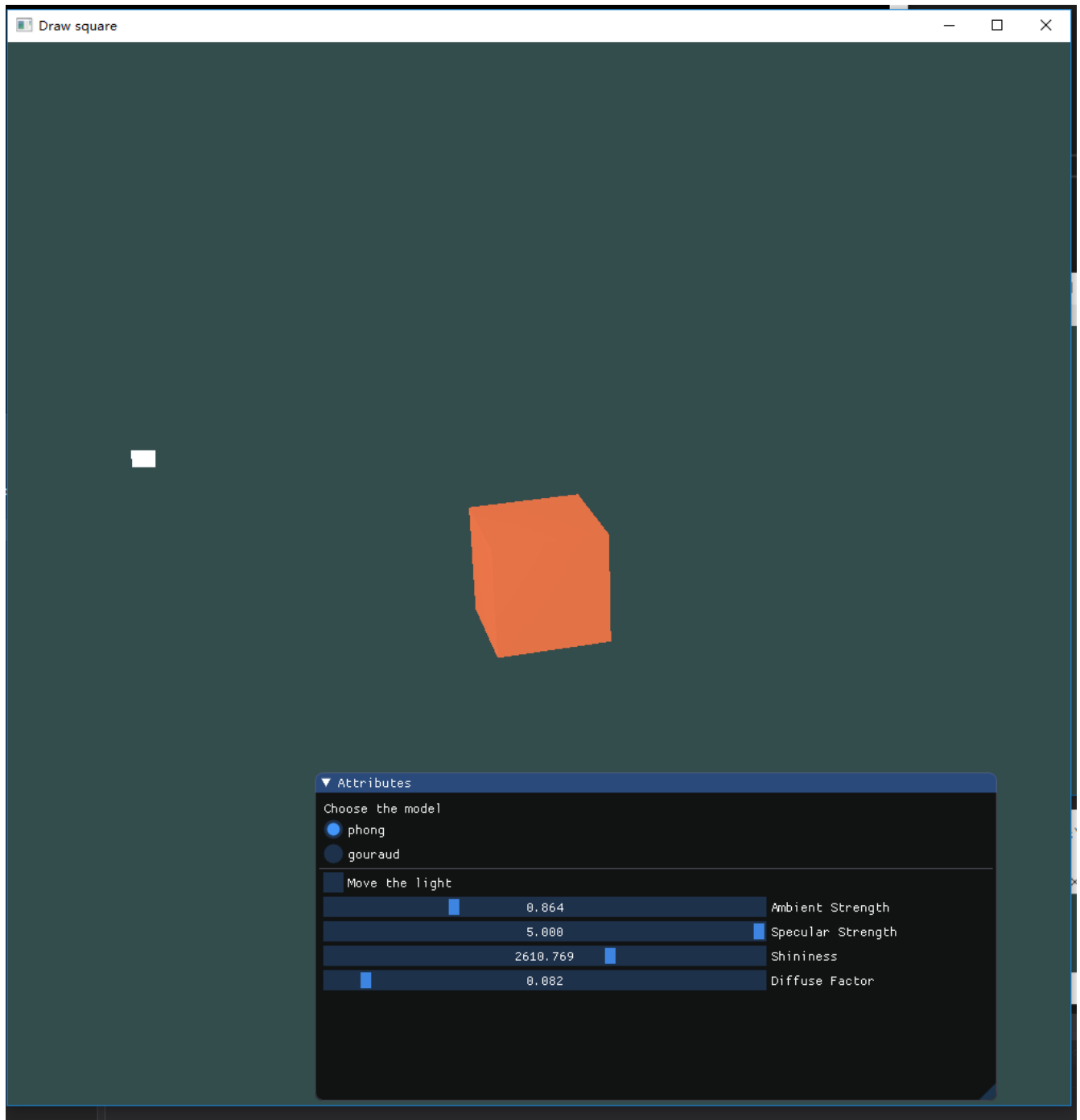
## 实验效果

### phong

正常效果如下：



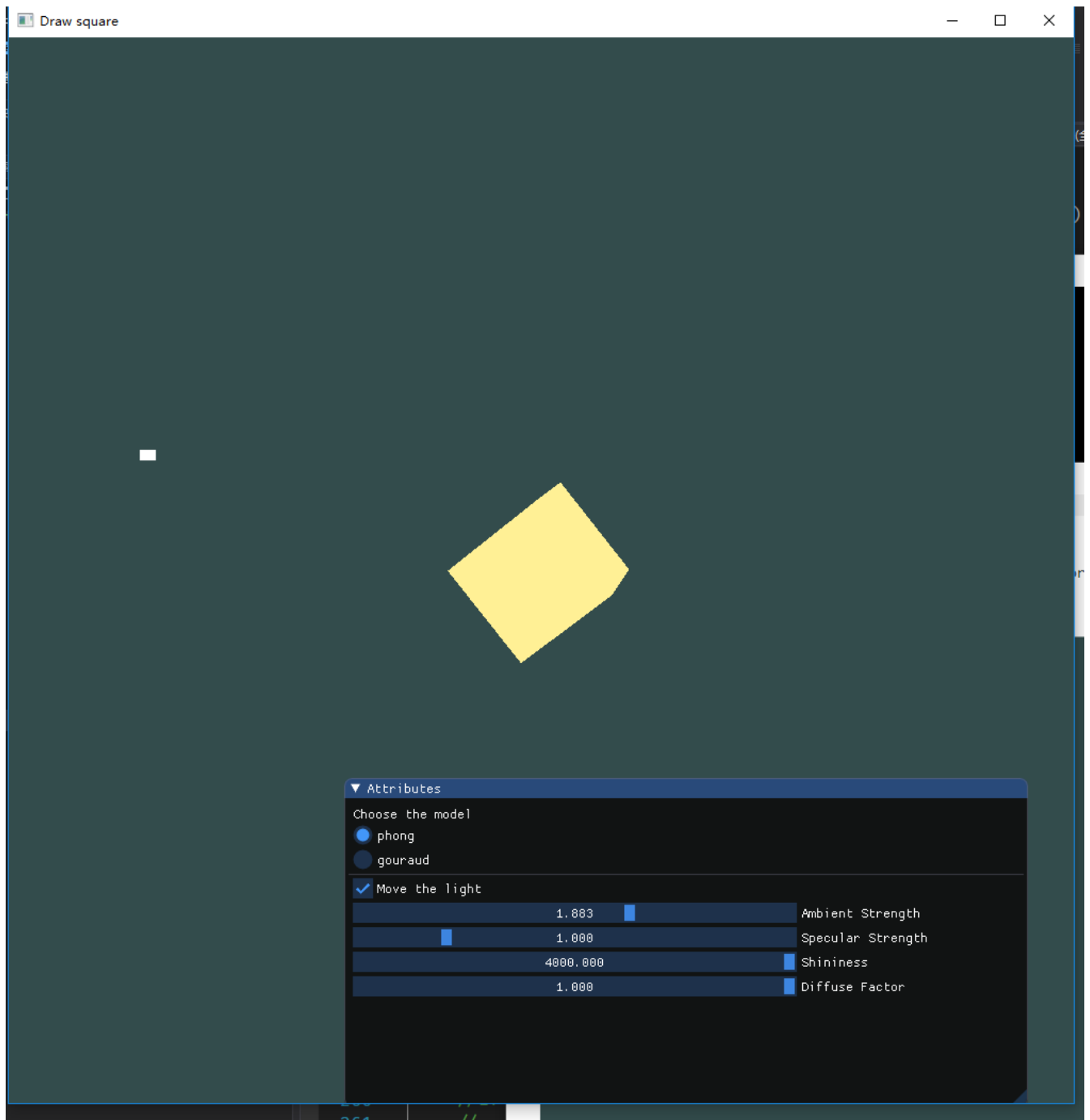
在光源静止时候进行截图：



物体本身不发光，是通过反射光源的光产生颜色，所以光源照射的地方被遮挡时，视角内物体较暗。

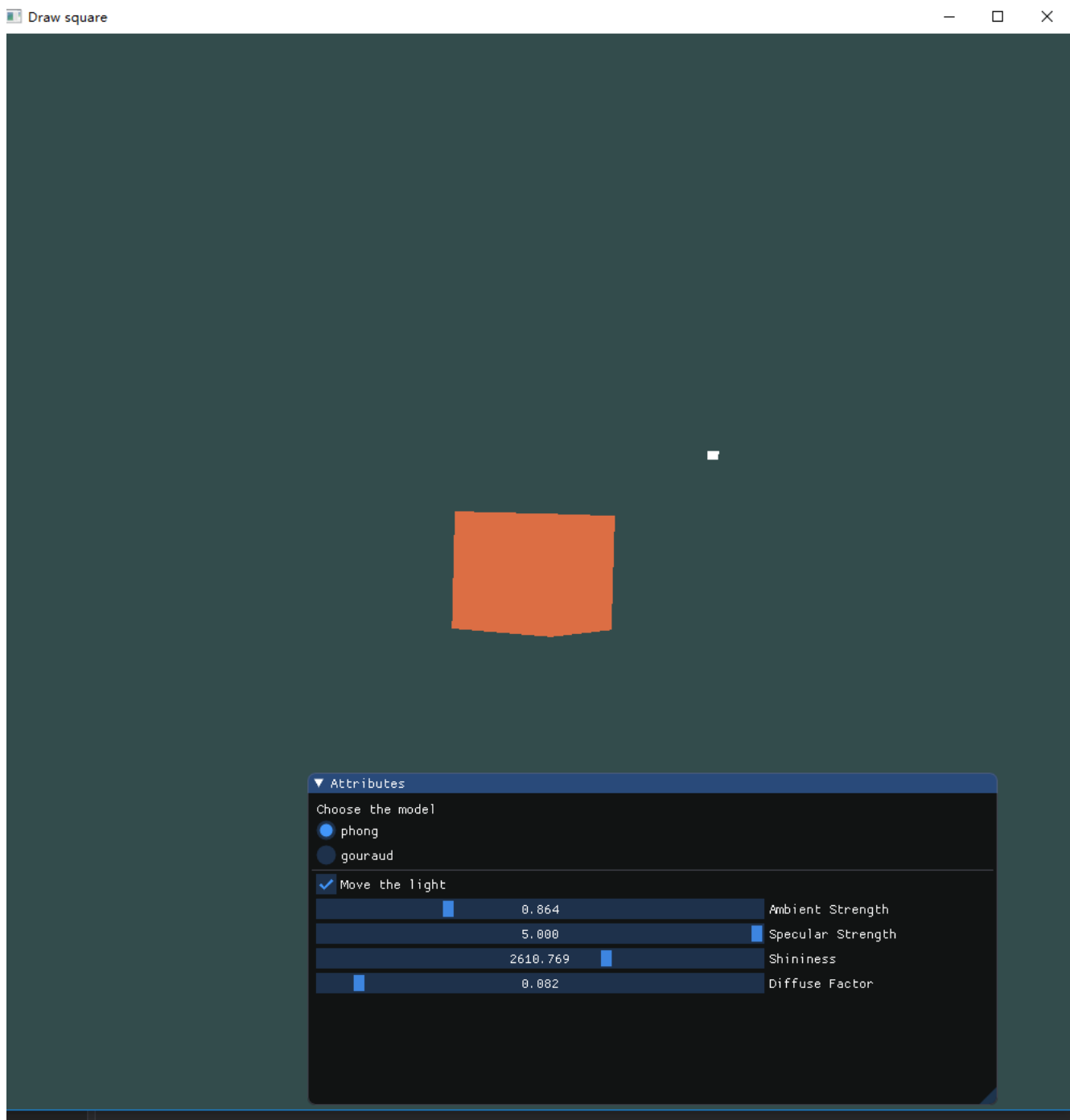
调整环境光因素 AmbientStrength 如下：





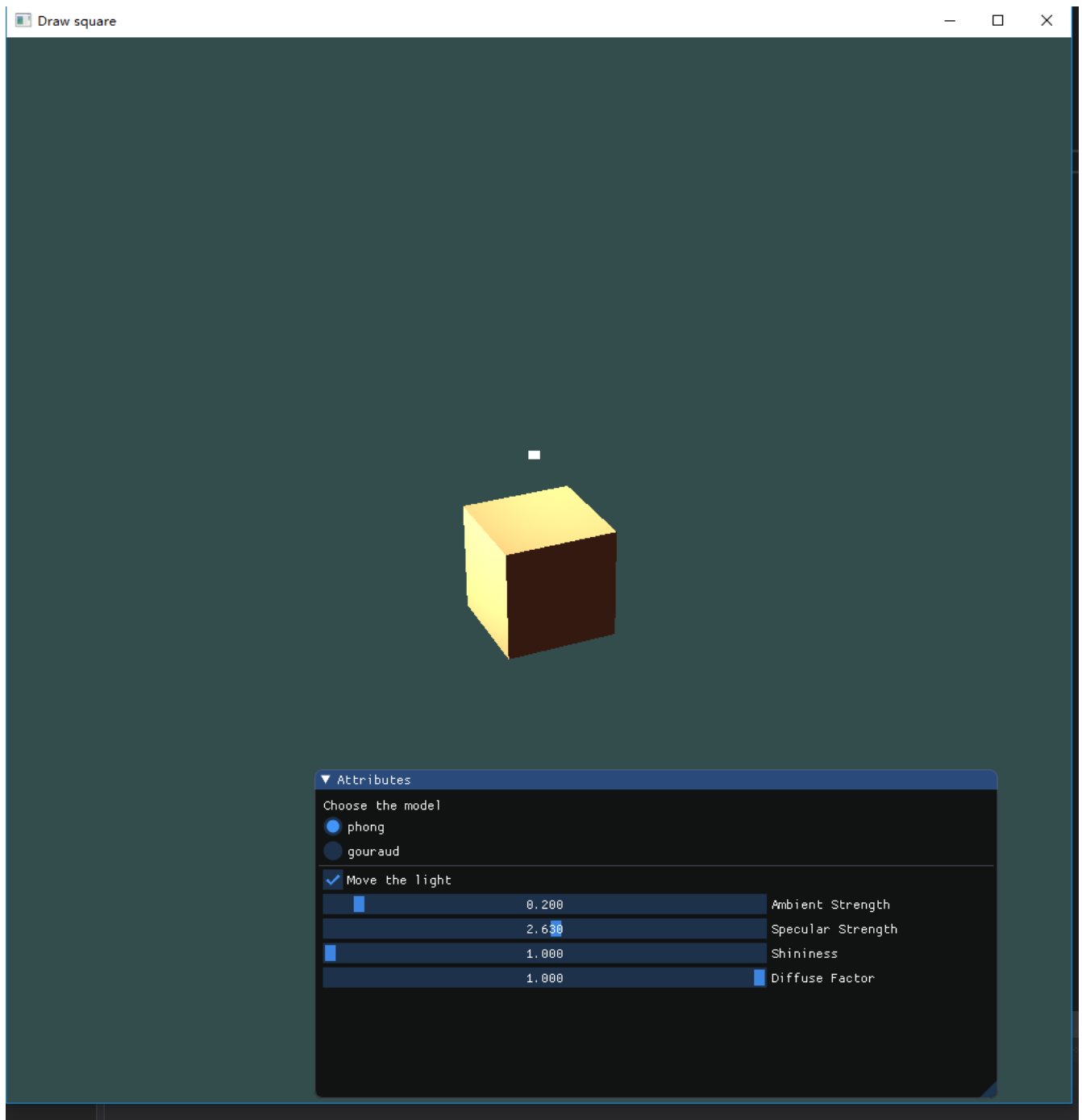
可以发现立方体明显变亮。

调整漫反射因素 `diffuseFactor` 后如下：



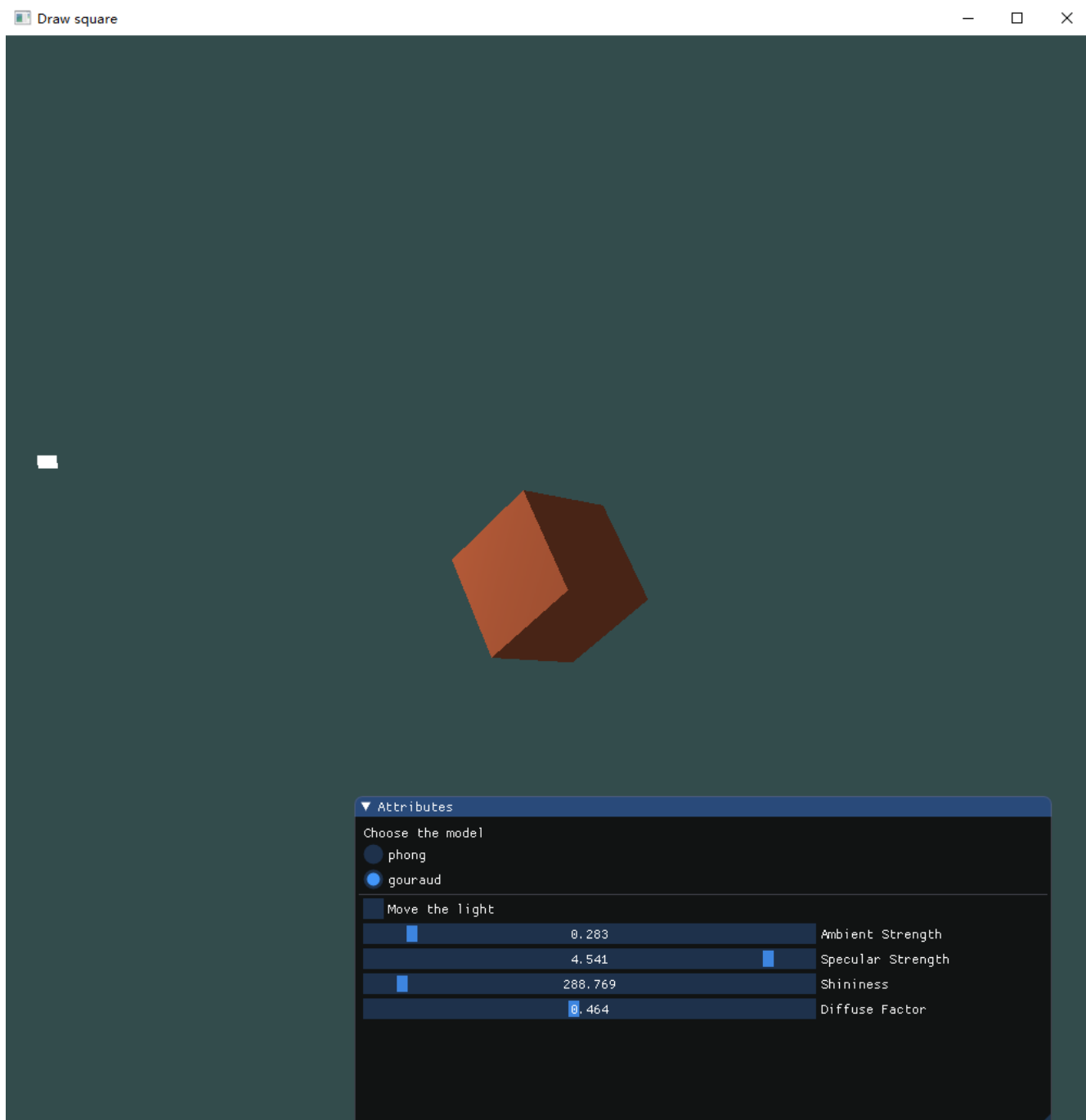
可以发现因素越大，物体对着光源的那部分就会越亮。

调整镜面因素 `specularStrength`，从而改变其光线的反射能力：



## gouraud

更改光照模型：



通过gif图可以发现，在高漫反射因素时渲染效果明显失真，立方体的每一面变成两个明显的三角形（截图看不出）。印证了之前的观点。

运行的两个gif图片：

