Homework 6 - Lights and Shading

Phong Shading

Phong Shading 是对顶点的**法向量**进行插值，在OpenGL中的实现思路是：在Vertex Shader中求顶点的法向量，然后传递给Fragment Shader，然后再应用Phong Lighting模型计算颜色。

Phong Vertex Shader

```
#version 330 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 proj;

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

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

顶点着色器传入了顶点的位置和法向量，随后将位置左乘`Model`矩阵转换到世界坐标系，将法向量左乘$(Model^{-1})^T$算出世界坐标系中顶点的法向量，最后将世界坐标系中的位置和法向量传递给片段着色器。

其中，法向量转换矩阵推导如下：

$P_w = M_p P_l$，其中P_w, P_l分别是顶点在世界坐标系和本地坐标系中的位置，M_p是Model矩阵

$N_w = M_n N_l$，假设M_n是法向量的Model矩阵，N_w, N_l分别是顶点在世界坐标系和本地坐标系中的法向量

$P_w^T N_w = P_l^T M_p^T M_n N_l = 0$，世界坐标系中顶点的位置向量和法向量相乘依旧为0，且$P_l^T N_l = 0$，故只需让$M_p^T M_n = I$即可，所以$M_n = (M_p^{-1})^T$

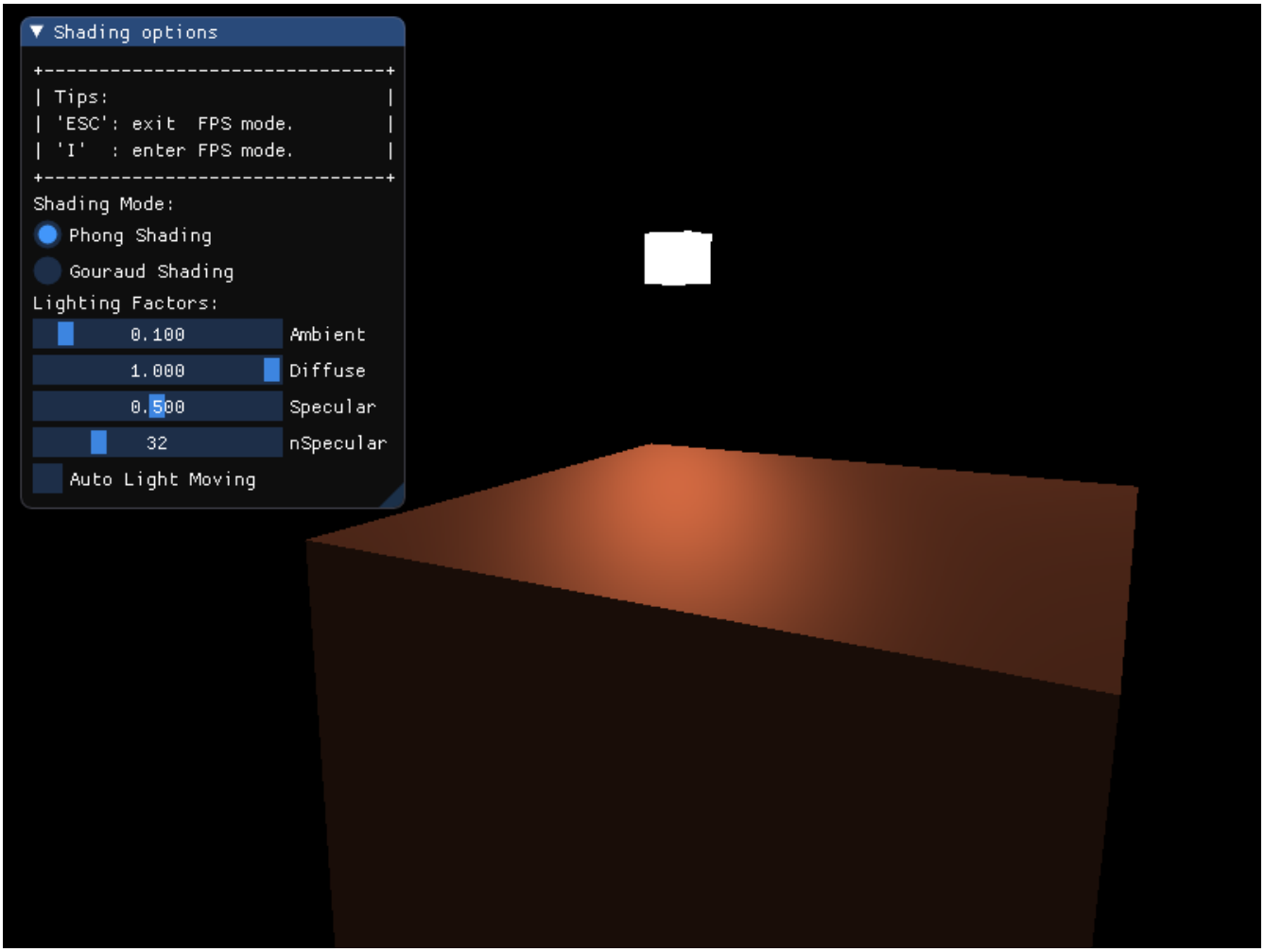
Phong Fragment Shader

```
#version 330 core

out vec4 FragColor;

in vec3 Normal;
in vec3 FragPos;
```

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

uniform float Ka;
uniform float Kd;
uniform float Ks;
uniform int nSpec;

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

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

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

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

片段着色器传入的是每个像素插值得到的位置向量和法向量，然后应用Phone Lighting模型对每个像素计算颜色。

效果



Gouraud Shading

Gouraud Shading 直接在顶点着色器中就计算了顶点颜色，然后对**颜色**进行插值。在OpenGL中的实现思路是：在Vertex Shader中利用Phong Lighting模型计算顶点的颜色，然后传递给Fragment Shader进行插值。

Vertex Shader

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

out vec3 vLightColor;

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

uniform float Ka;
uniform float Kd;
uniform float Ks;
uniform int nSpec;

uniform mat4 model;
uniform mat4 view;
```

```
uniform mat4 proj;

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

    vec3 Position = vec3(model * vec4(aPos, 1.0));
    vec3 Normal = mat3(transpose(inverse(model))) * aNormal;

    // ambient
    vec3 ambient = Ka * lightColor;

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

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

    vLightColor = ambient + Kd * diffuse + specular;
}
```

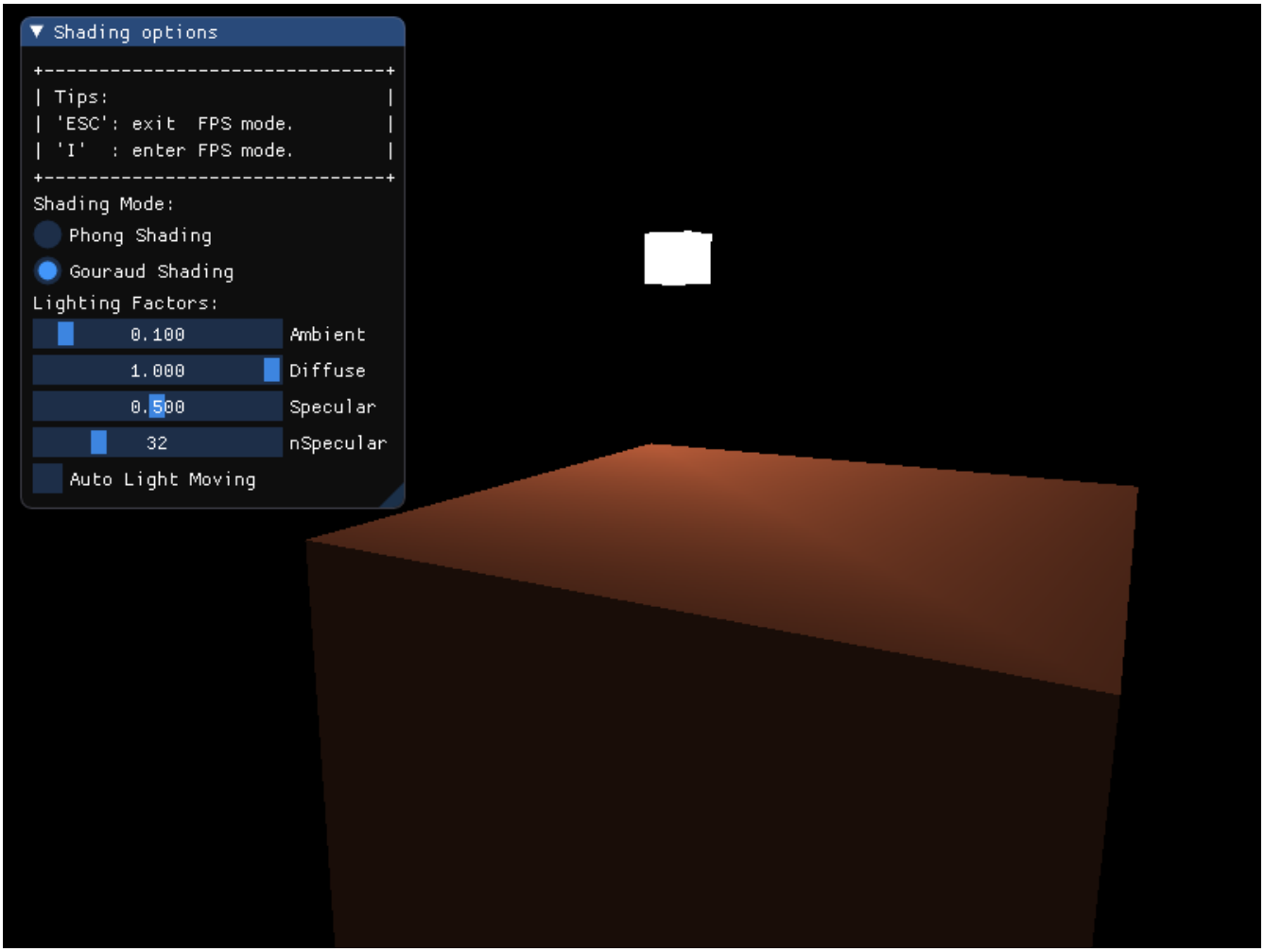
顶点着色器传入顶点的位置向量和法向量，利用M_p和M_n分别计算它们在世界坐标系中的向量表示，然后利用Phong Shading模型计算顶点的颜色，然后传递给片段着色器。

Fragment Shader

```
#version 330 core
out vec4 FragColor;

in vec3 vLightColor;

uniform vec3 objectColor;

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

片段着色器传入的是每个像素插值后光照的颜色，将其与物体顶点的颜色相乘即可得到最终我们看到的颜色。

效果

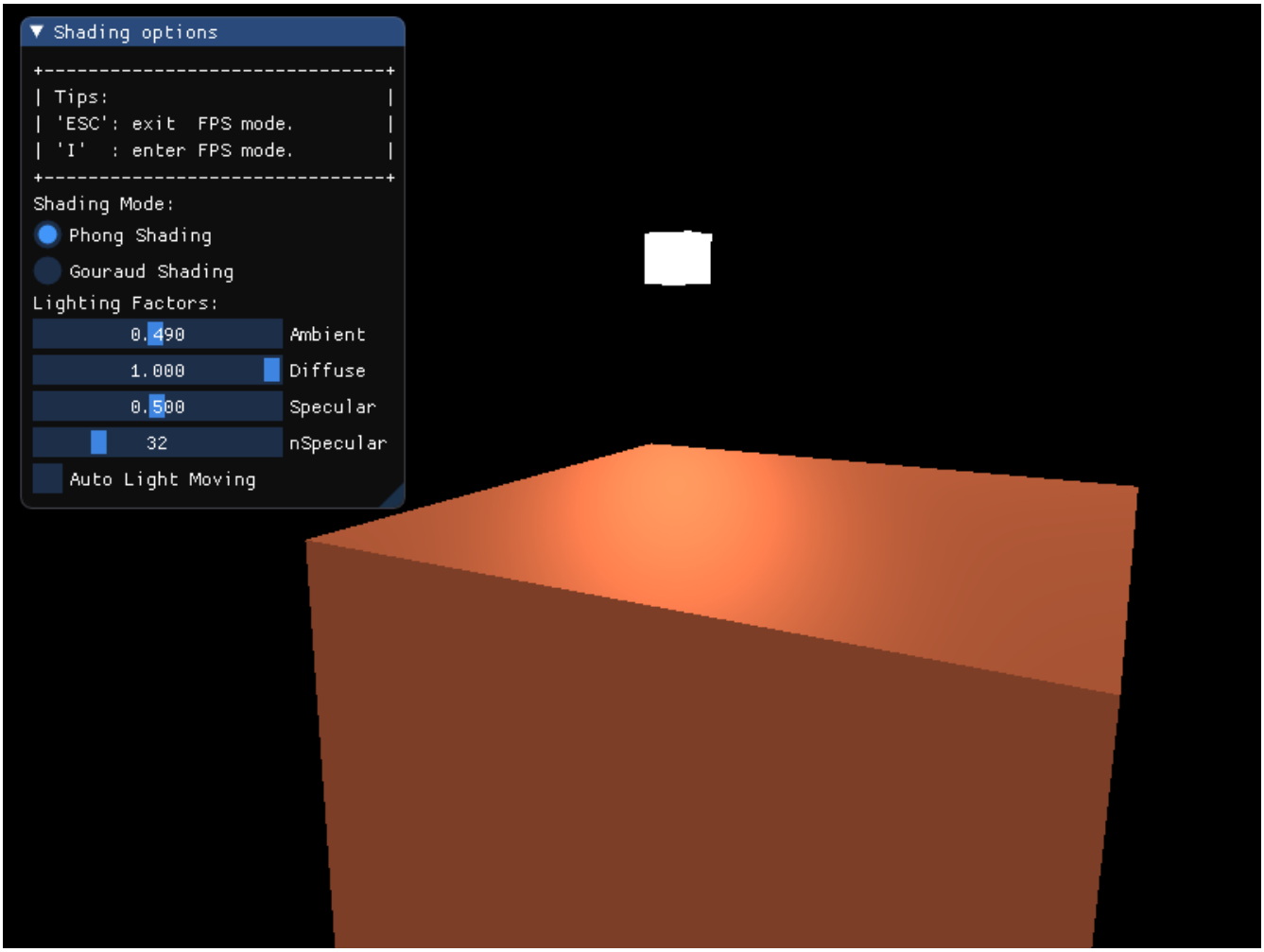


光照模型因子

Ambient

Ambient 表示环境光的强度，越大的话物体总体颜色越亮。

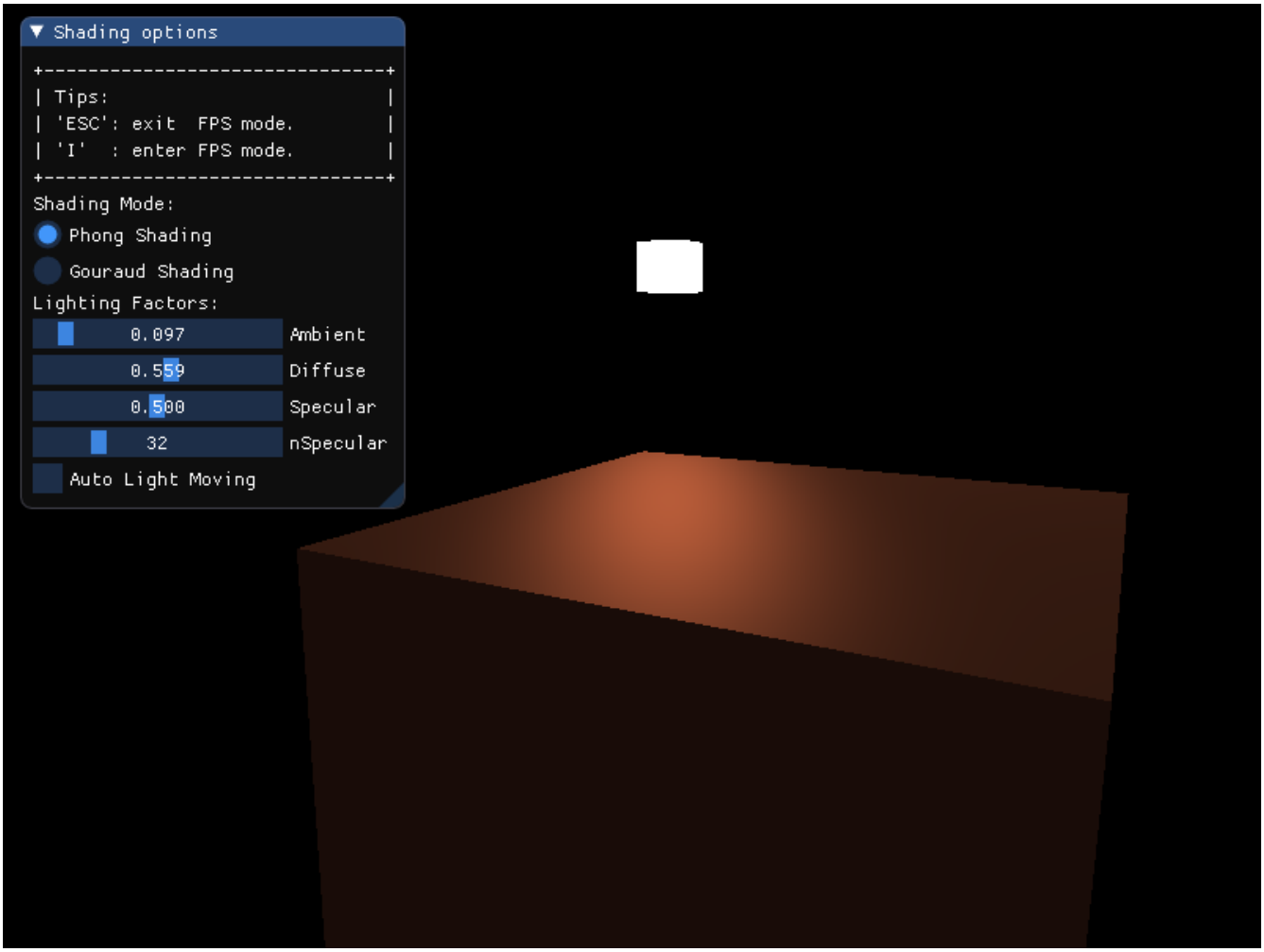
调节Ambient至0.49，得到以下效果，与前面的图对比可以看到立方体的颜色整体变亮了。



Diffuse

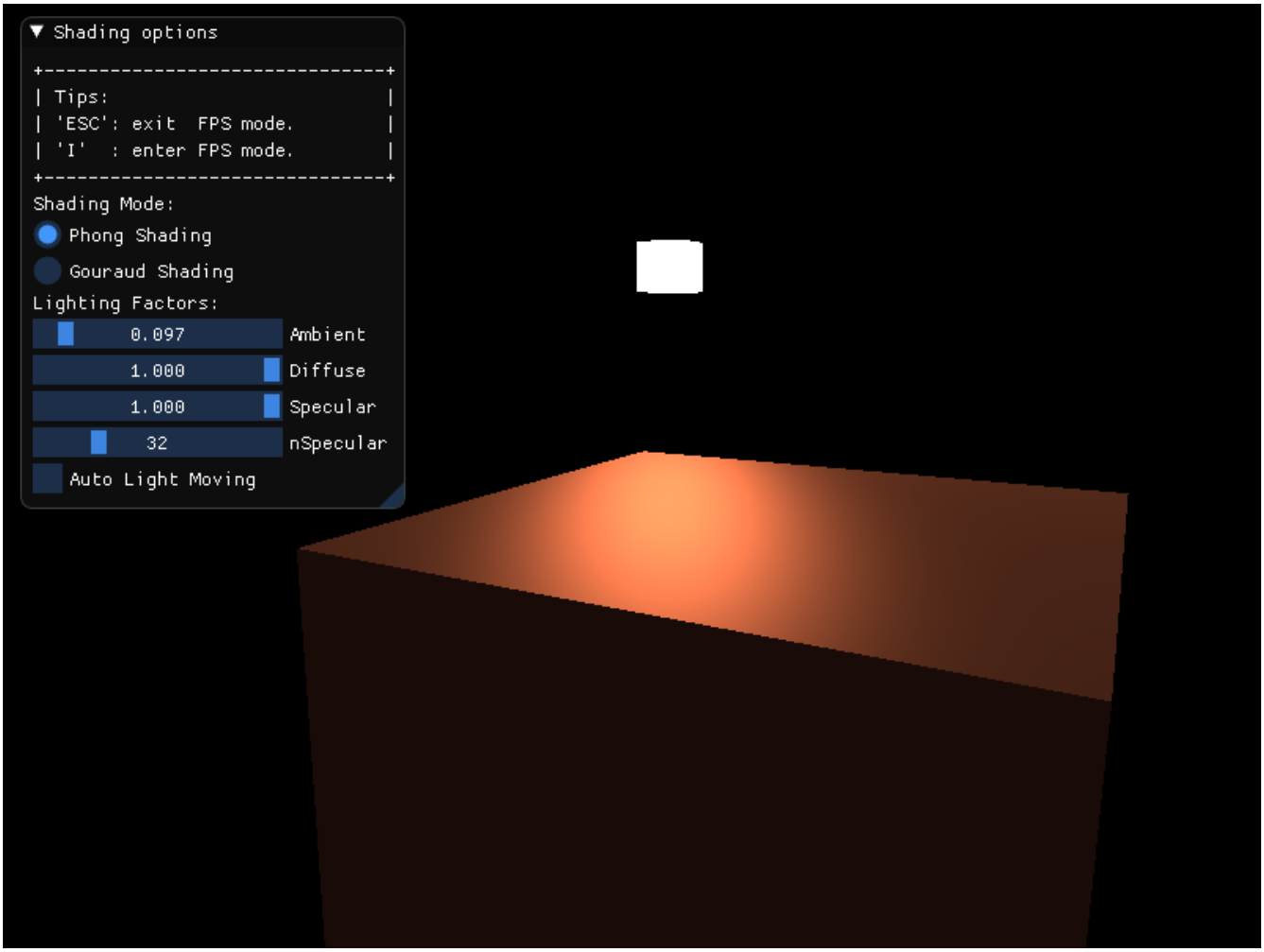
Diffuse 表示漫反射的强度，物体表面向量与光线向量夹角小于90度时，会有漫反射，Diffuse影响物体漫反射的程度。

降低 Diffuse ， 对比之前 Diffuse=1 的情况，漫反射的程度较低了。



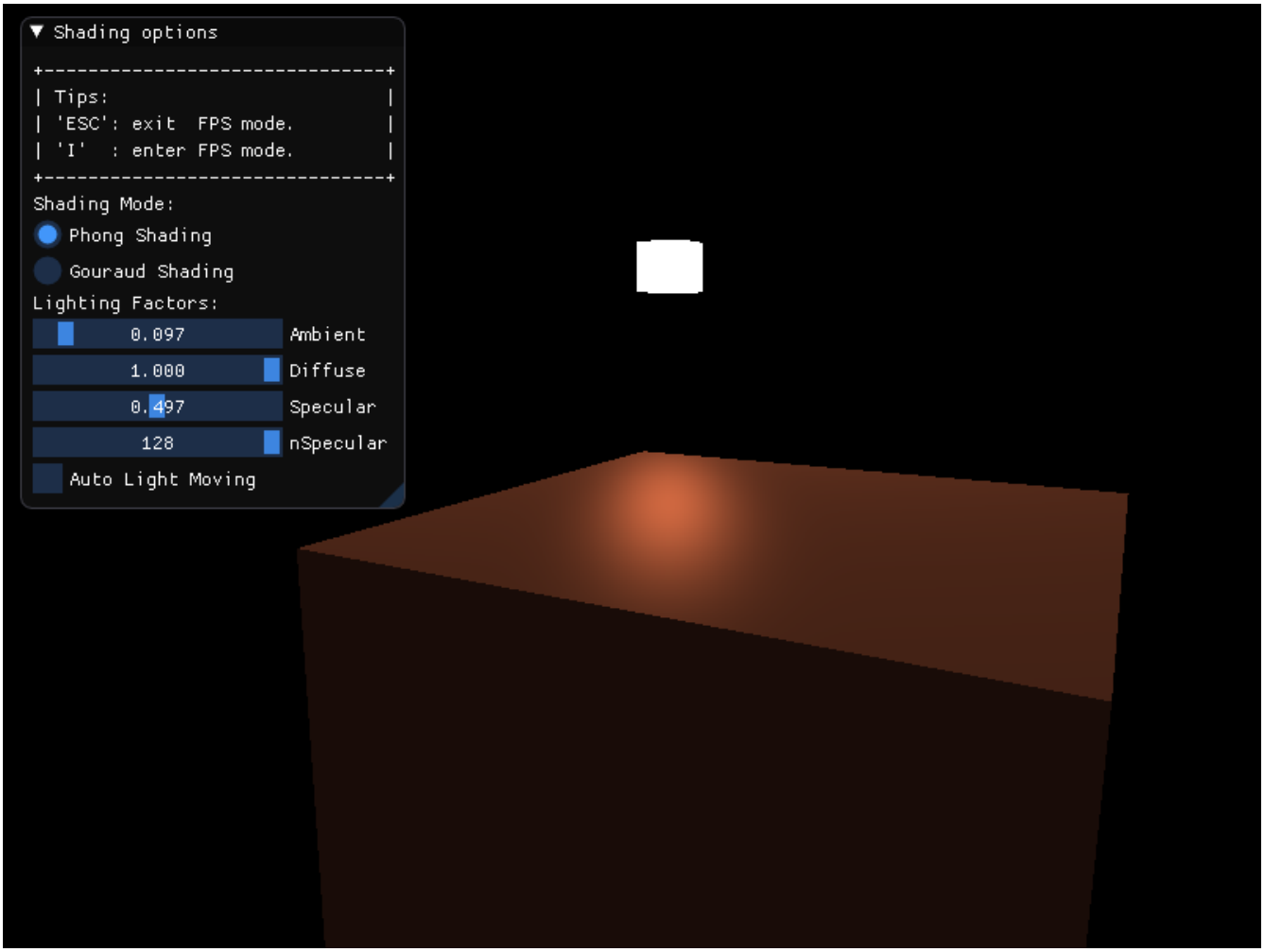
Specular

Specular 表示高光程度，将Specular调至1，可以看到高光效果很强。

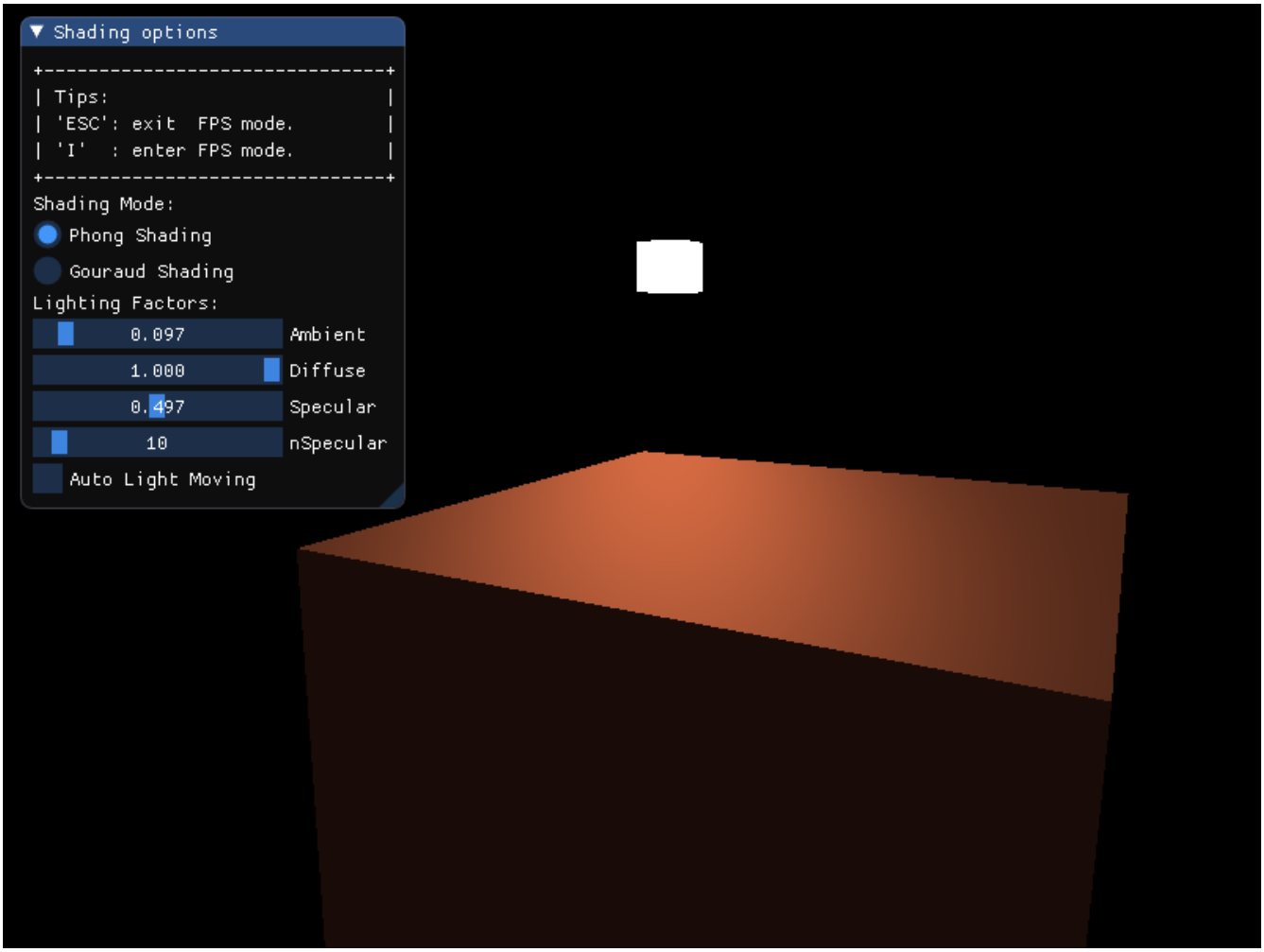


nSpecular

nSpecular 表示物体表面的光滑程度，值越大代表越光滑，这样高光效果的区域就越小。



值越小，高光效果面积越大。



光源移动

```
if (autoLightMoving) {
    lightPos.x = 1.2f + sin(glfwGetTime());
}
```

让光源的位移量随着时间而改变即可。