Homework6

作业要求

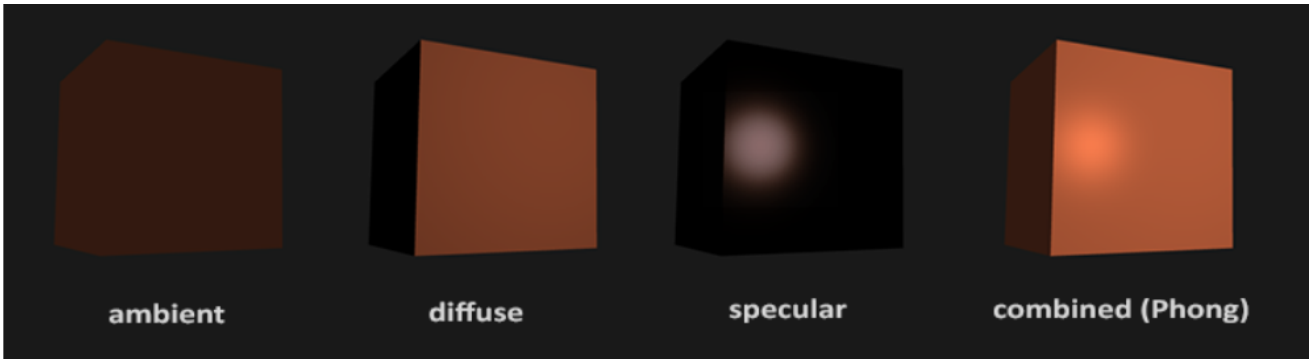
1. 实现Phong光照模型： 场景中绘制一个cube 自己写shader实现两种shading: Phong Shading 和 Gouraud Shading，并解释两种shading的实现原理 合理设置视点、光照位置、光照颜色等参数，使光照效果明显显示
2. 使用GUI，使参数可调节，效果实时更改： GUI里可以切换两种shading 使用如进度条这样的控件，使ambient因子、diffuse因子、specular因子、反光度等参数可调节，光照效 果实时更改
3. Bonus: 当前光源为静止状态，尝试使光源在场景中来回移动，光照效果实时更改。

冯氏光照模型(Phong Lighting Model)

照明和物体的材料属性共同决定了它的外观。基于光的特性和对象材质的特性，照明模型描述了光与对象的交互方式。其中一个模型被称为冯氏光照模型(Phong Lighting Model)。冯氏光照模型的主要结构由3个分量组成：环境(Ambient)、漫反射(Diffuse)和镜面(Specular)光照。

- 环境光照(Ambient Lighting)：即使在黑暗的情况下，世界上通常也仍然有一些光亮（月亮、远处的光），所以物体几乎永远不会是完全黑暗的。为了模拟这个，我们会使用一个环境光照常量，它永远会给物体一些颜色。

- 漫反射光照(Diffuse Lighting)：模拟光源对物体的方向性影响(Directional Impact)。它是冯氏光照模型中视觉上最显著的分量。物体的某一部分越是正对着光源，它就会越亮。
- 镜面光照(Specular Lighting)：模拟有光泽物体上面出现的亮点。镜面光照的颜色相比于物体的颜色会更倾向于光的颜色。



Phong Shading和Gouraud Shading

Phong Shading是指三维计算机图形中用于表面着色的插值技术。它也被称为Phong插值或法向量插值着色。具体来说，它在栅格化多边形中插入曲面法线，并基于插值法线和反射模型计算像素颜色。

Gouraud Shading，以Henri Gouraud命名，是计算机图形学中用来产生多边形网格表示的曲面连续着色的一种插值方法。在实践中，Gouraud着色最常用于通过计算每个三角形角处的照明，并对三角形所覆盖的每个像素的结果颜色进行线性插值，从而在三角形表面上实现连续照明。

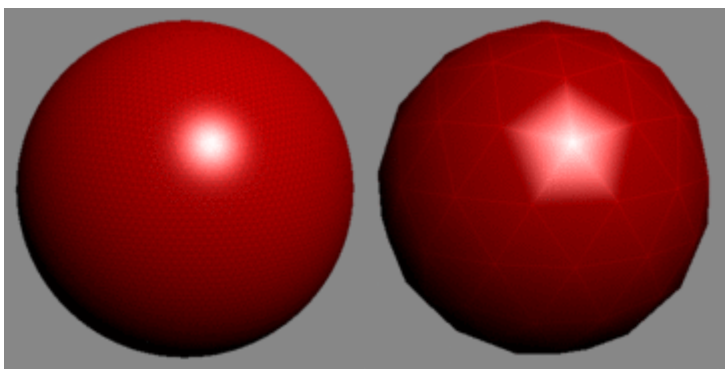
与Phong Shading相比，Gouraud Shading的强弱在于插值。如果一个网格在屏幕空间中覆盖的像素多于它的顶点，那么从顶点处昂贵的照明计算样本中插入颜色值比为每个像素（如Phong Shading）执行照明计算的处理器强度要小。但是，局部的照明效果（例如镜面高光，例如苹果表面反射光的闪烁）将无法正确渲染，并且如果高光位于多边形的中间，但不扩散到多边形的顶点，则在Gouraud Shading中不会明显；相反，如果高光发生在多边形的顶点，它将在这个顶点上被正确渲染。

Gouraud明暗处理只在多边形顶点处采用Phong局部反射模型计算光强，而在多边形内的其他点采用双向线性插值，这样做的优点是高效，但是无法很好的处理镜面高光问题，依赖于其所在多面形的相对位置；

而Phong明暗处理，通过差值计算每个顶点的法向量（3次差值，在x，y，z三个方向分别进行差值计算），然后计算每个点上的光强值，这样效果好，但计算复杂，需要付出比Gouraud 4-5 倍的时间。

通常，在一个比较复杂的场景中，当物体镜面发射很微弱时，我们对其采用Gouraud明暗处理，而对于一些镜面高光的物体，采用Phong明暗处理，这样既保证质量，又保证速度。

Phong Shading和Gouraud Shading的对比如下图：



代码

光源

光源的着色器代码较为简单，只需将光源设置为白色即可

```
// 片段着色器
#version 330 core
out vec4 FragColor;

void main()
{
    FragColor = vec4(1.0f); // white light
}
```

```
// 顶点着色器
#version 330 core
layout (location = 0) in vec3 aPos;

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

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

GUI

根据实验要求，GUI里可以切换两种shading，因此用radiobutton作为两种shading的选择按钮，然后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，最后用checkbox，实现光源移动和摄像机移动的选择。

```
ImGui::Begin("Lights and Shading\n");
ImGui::SetWindowSize(ImVec2(300, 220));

if (ImGui::RadioButton("Phong Shading", isPhong)) {
    isPhong = true;
    isGouraud = false;
};
if (ImGui::RadioButton("Gouraud Shading", isGouraud)) {
    isPhong = false;
    isGouraud = true;
};

ImGui::SliderFloat("ambient", &ambient, 0.0f, 1.0f);
ImGui::SliderFloat("diffuse", &diffuse, 0.0f, 3.0f);
```

```
ImGui::SliderFloat("specular", &specular, 0.0f, 3.0f);
ImGui::SliderInt("reflecting", &reflectance, 0, 256);
ImGui::Checkbox("move", &isMove);
ImGui::Checkbox("enable camera", &isEnableCamera);

ImGui::End();
```

环境光照

光通常都不是来自于同一个光源，而是来自于我们周围分散的很多光源，即使它们可能并不是那么显而易见。光的一个属性是，它可以向很多方向发散并反弹，从而能够到达不是非常直接临近的点。所以，光能够在其它的表面上**反射**，对一个物体产生间接的影响。

把环境光照添加到场景里非常简单。我们用光的颜色乘以一个很小的常量环境因子，再乘以物体的颜色，然后将最终结果作为片段的颜色。

漫反射

环境光照本身不能提供最有趣的结果，但是漫反射光照就能开始对物体产生显著的视觉影响了。漫反射光照使物体上与光线方向越接近的片段能从光源处获得更多的亮度。计算漫反射光照需要：

- 法向量：一个垂直于顶点表面的向量。
- 定向的光线：作为光源的位置与片段的位置之间向量差的方向向量。为了计算这个光线，我们需要光的位置向量和片段的位置向量

法向量是一个垂直于顶点表面的（单位）向量。由于顶点本身并没有表面（它只是空间中一个独立的点），我们利用它周围的顶点来计算出这个顶点的表面。我们能够使用一个小技巧，使用叉乘对立方体所有的顶点计算法向量，但是由于3D立方体不是一个复杂的形状，所以我们可以简单地把法线数据手工添加到顶点数据中。

我们现在对每个顶点都有了法向量，但是我们仍然需要光源的位置向量和片段的位置向量。由于光源的位置是一个静态变量，我们可以简单地在片段着色器中把它声明为uniform。然后在渲染循环中（渲染循环的外面也可以，因为它不会改变）更新uniform。

最后，我们还需要片段的位置。我们会在世界空间中进行所有的光照计算，因此我们需要一个在世界空间中的顶点位置。我们可以通过把顶点位置属性乘以模型矩阵（不是观察和投影矩阵）来把它变换到世界空间坐标。这个在顶点着色器中很容易完成，所以我们声明一个输出变量，并计算它的世界空间坐标。

镜面反射

和漫反射光照一样，镜面光照也是依据光的方向向量和物体的法向量来决定的，但是它也依赖于观察方向。我们通过反射法向量周围光的方向来计算反射向量。然后我们计算反射向量和视线方向的角度差，如果夹角越小，那么镜面光的影响就会越大。它的作用效果就是，当我们去看光被物体所反射的那个方向的时候，我们会看到一个高光。

观察向量是镜面光照附加的一个变量，我们可以使用观察者世界空间位置和片段的位置来计算它。之后，我们计算镜面光强度，用它乘以光源的颜色，再将它加上环境光和漫反射分量。

Phong Shading

综上，我们可以得到着色器代码：

```
// 片段着色器
uniform float ambientStrength;
uniform float specularStrength;
uniform float diffuseStrength;
uniform int reflectance;

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

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

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

```
// 顶点着色器
#version 330 core
layout (location = 0) in vec3 aPos;
layout (location = 1) in vec3 aNormal;

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

out vec3 Normal;
out vec3 FragPos;

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

Gouraud Shading

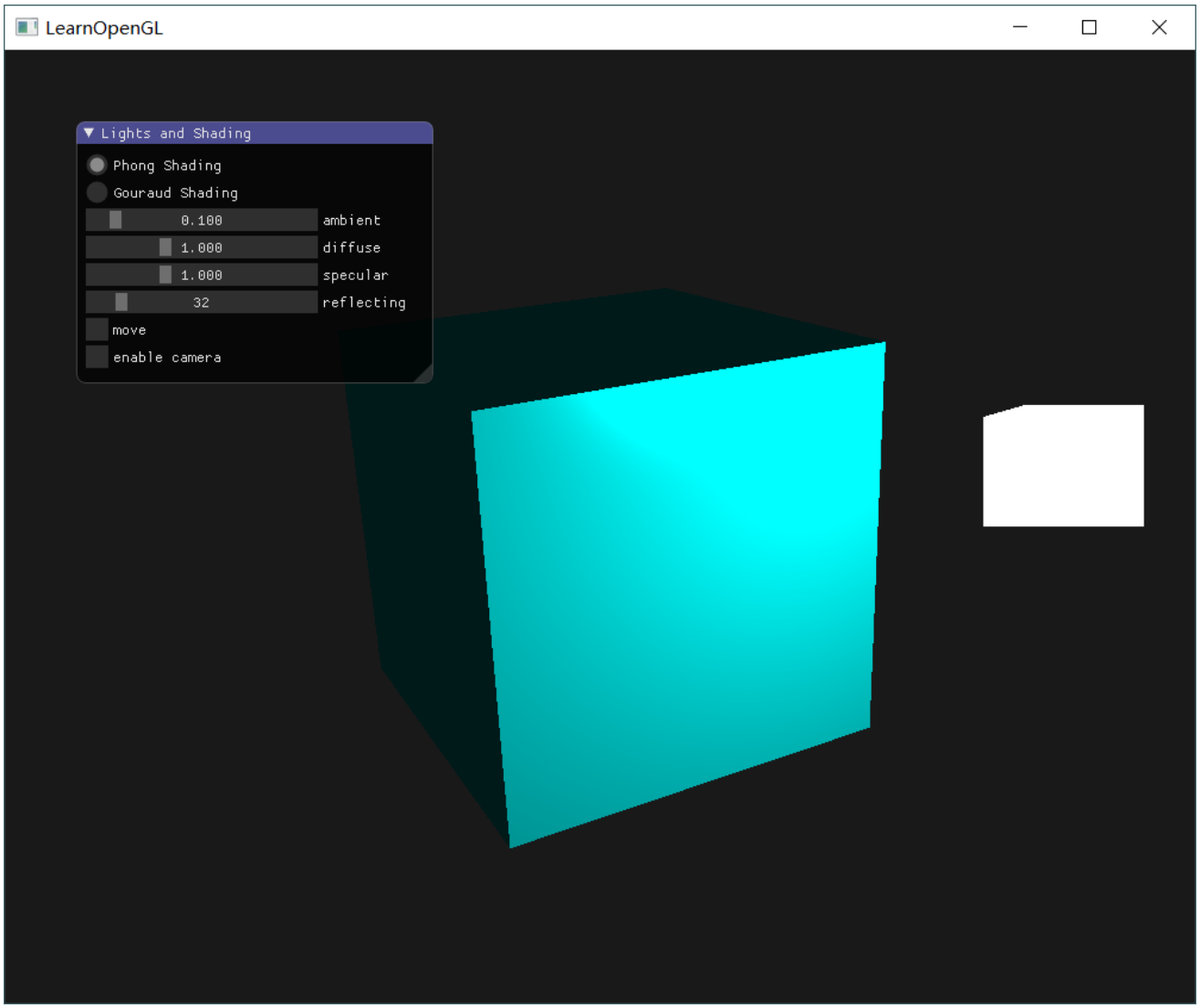
```
// 片段着色器
#version 330 core

in vec3 lightingColor;
out vec4 FragColor;
uniform vec3 objectColor;

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

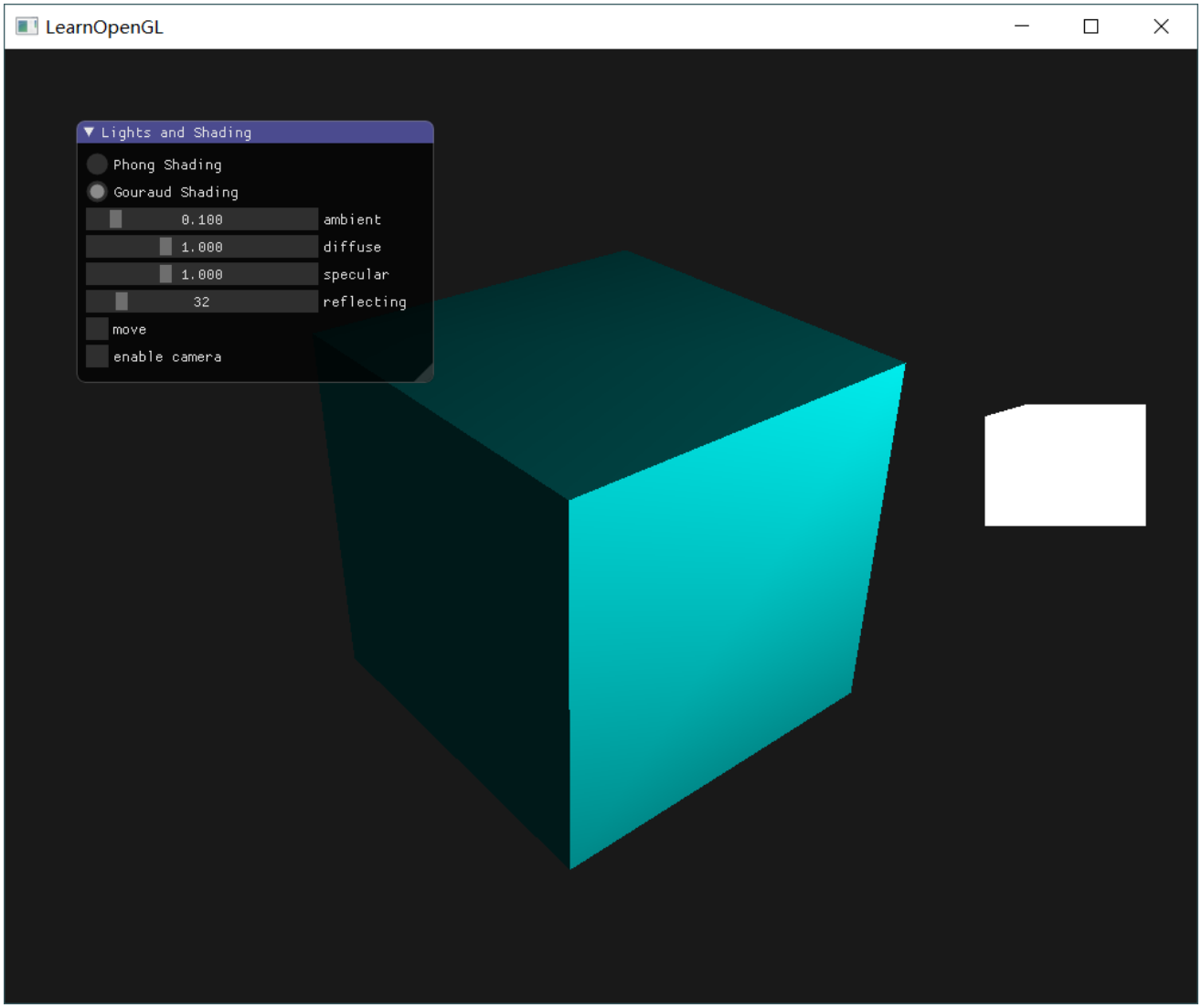
```
// 顶点着色器
#version 330 core
layout (location = 0) in vec3 aPos;
layout (location = 1) in vec3 aNormal;

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

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

uniform float ambientStrength;
uniform float diffuseStrength;
uniform float specularStrength;
uniform int reflectance;

out vec3 lightingColor;

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

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

  vec3 ambient = ambientStrength * lightColor;

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

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

  lightingColor = ambient + diffuse + specular;
```

```
}
```

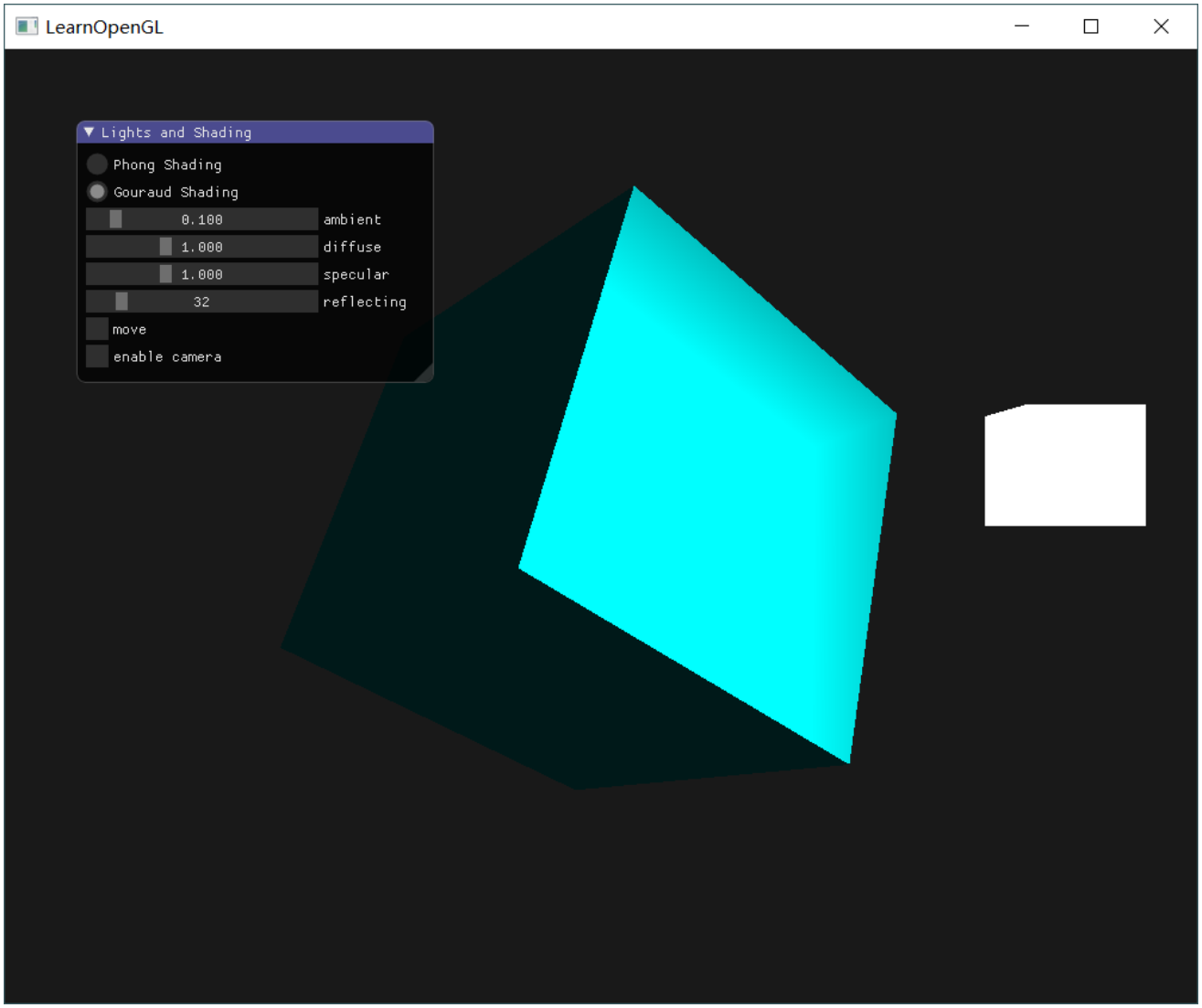
实验结果

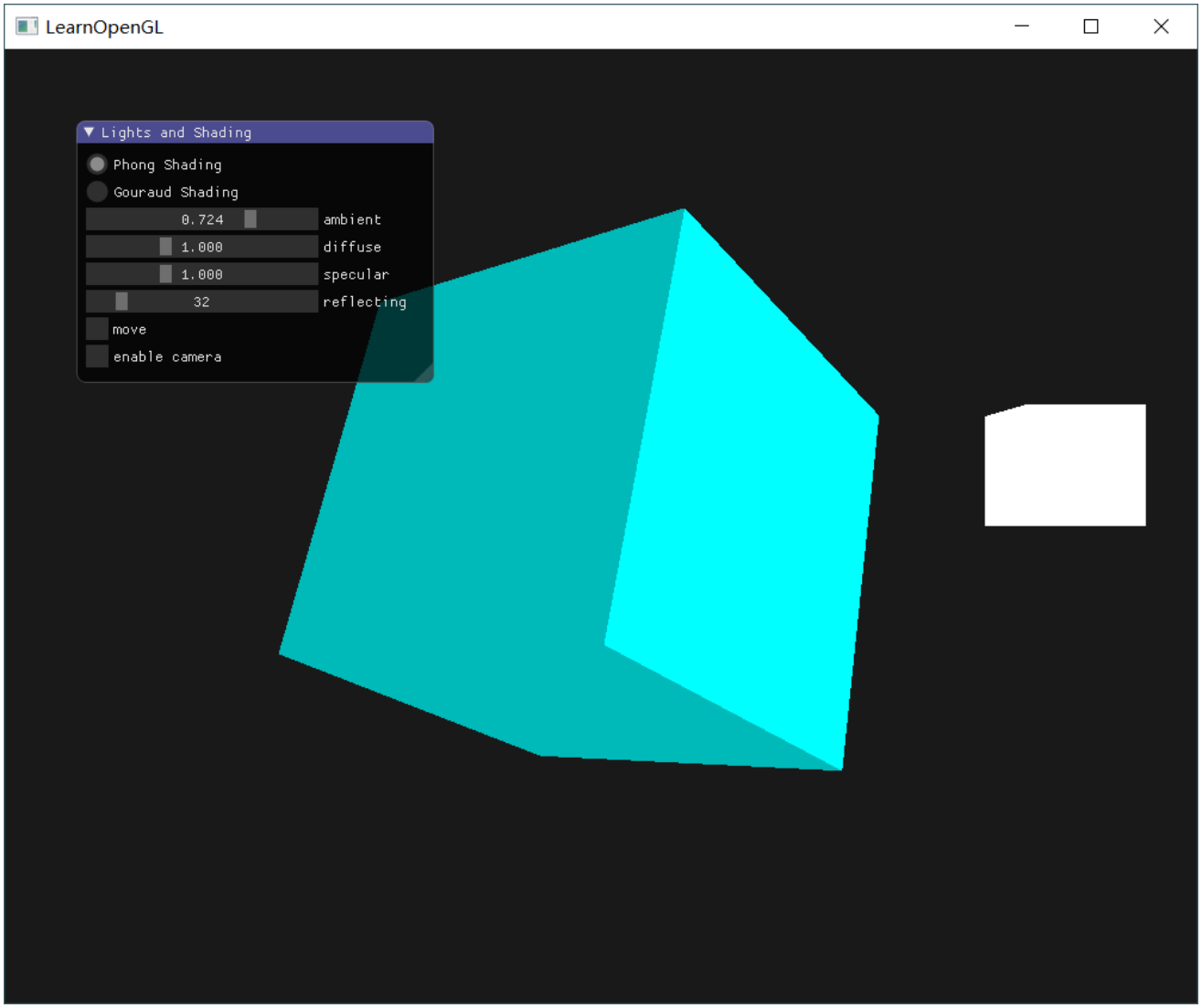
Shading选择



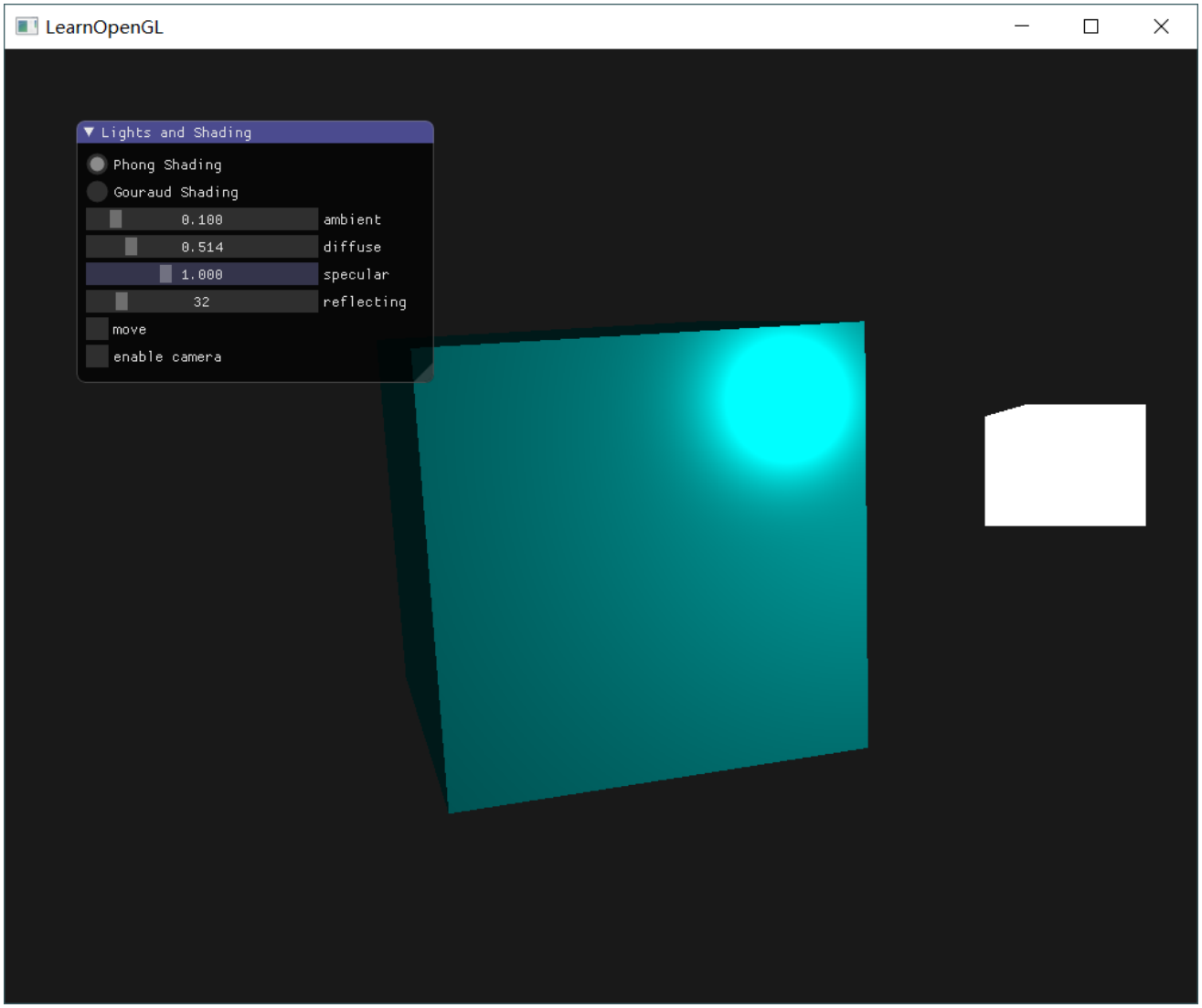


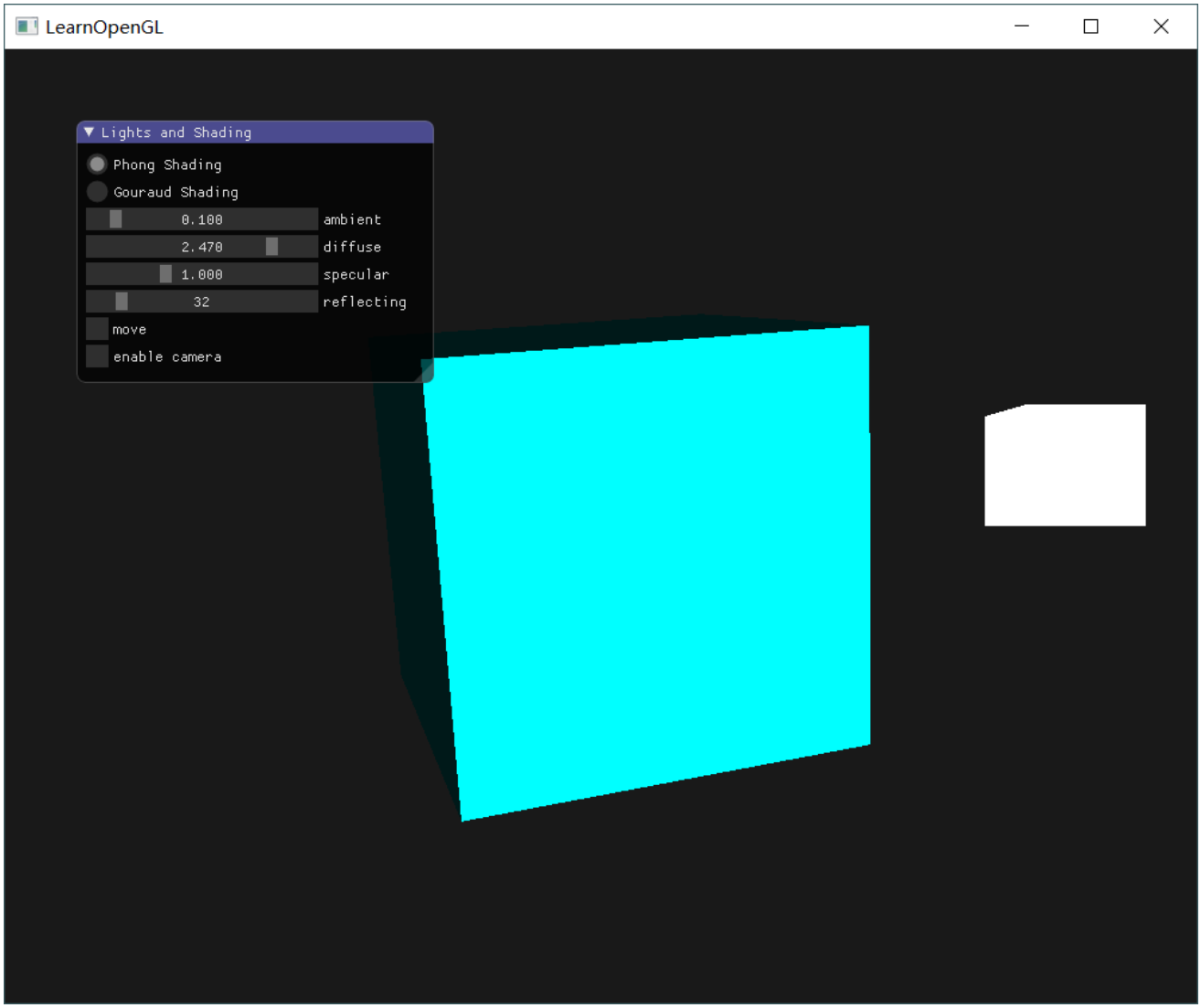
ambient因子的影响


LearnOpenGL
Lights and Shading
Phong Shading
Gouraud Shading
0.100
ambient
1.000
diffuse
1.000
specular
32
reflecting
move
enable camera

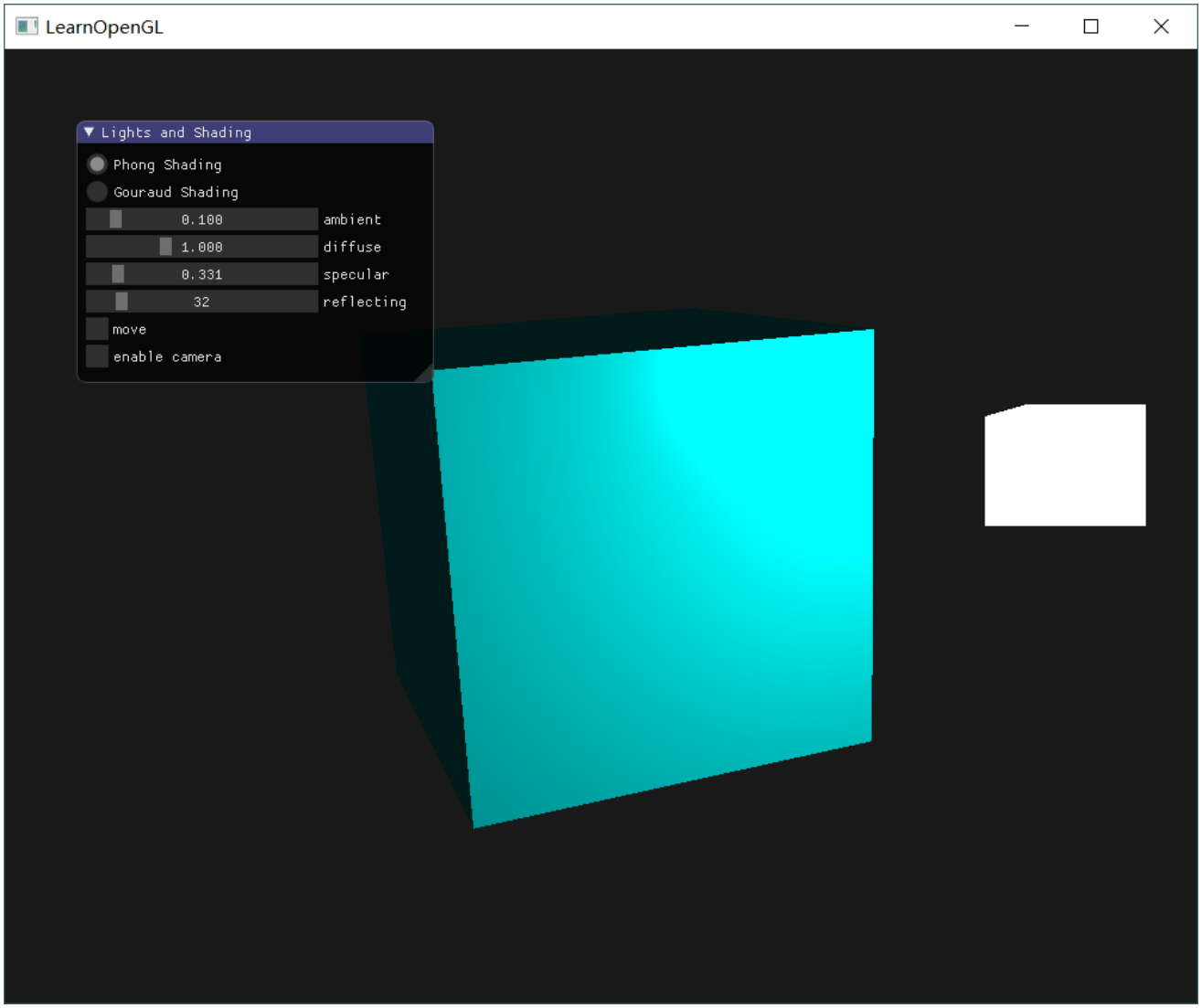

LearnOpenGL
Lights and Shading
Phong Shading
Gouraud Shading
0.724
ambient
1.000
diffuse
1.000
specular
32
reflecting
move
enable camera

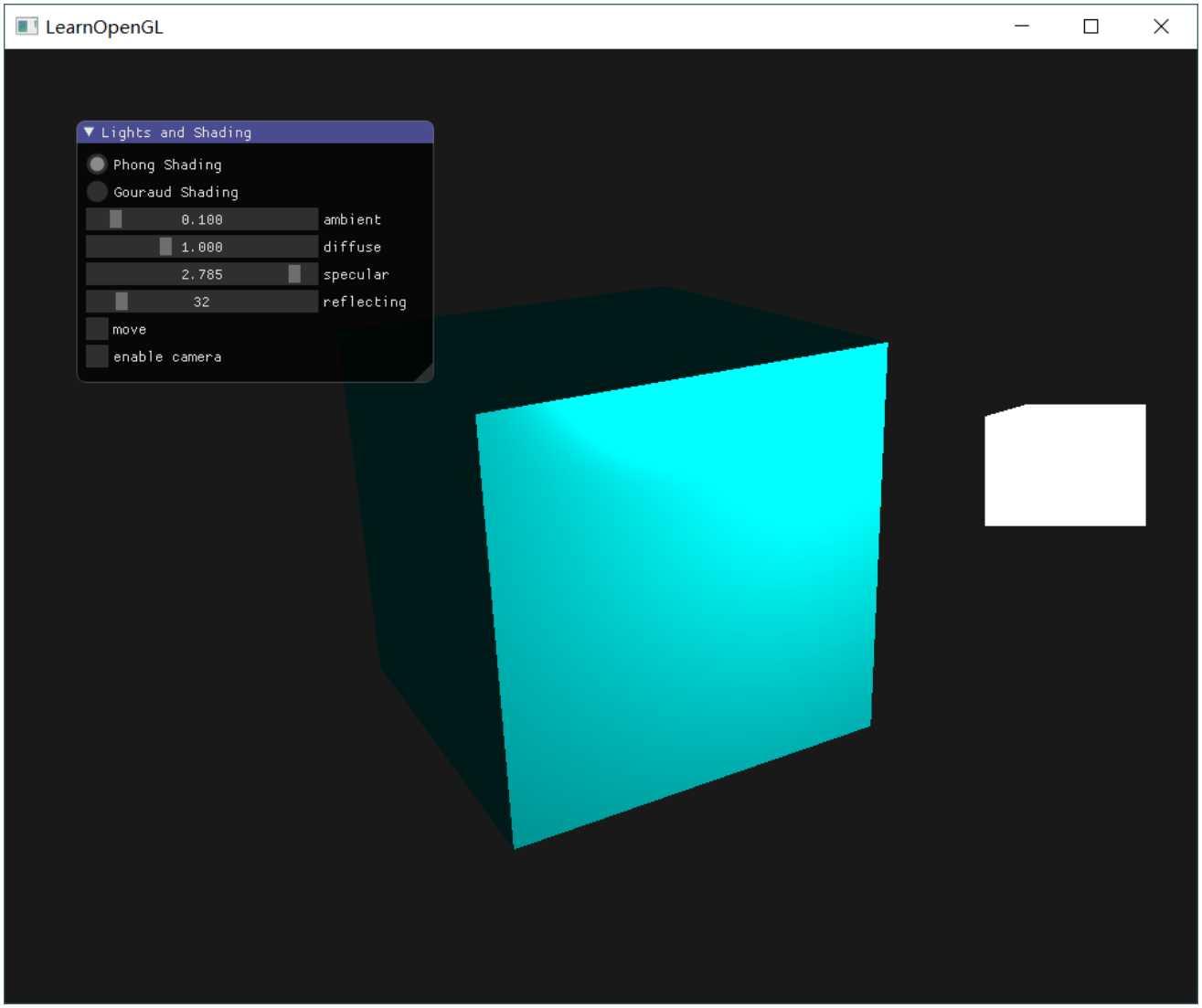
diffuse的影响


LearnOpenGL
Lights and Shading
Phong Shading
Gouraud Shading
0.100
ambient
0.514
diffuse
1.000
specular
32
reflecting
move
enable camera

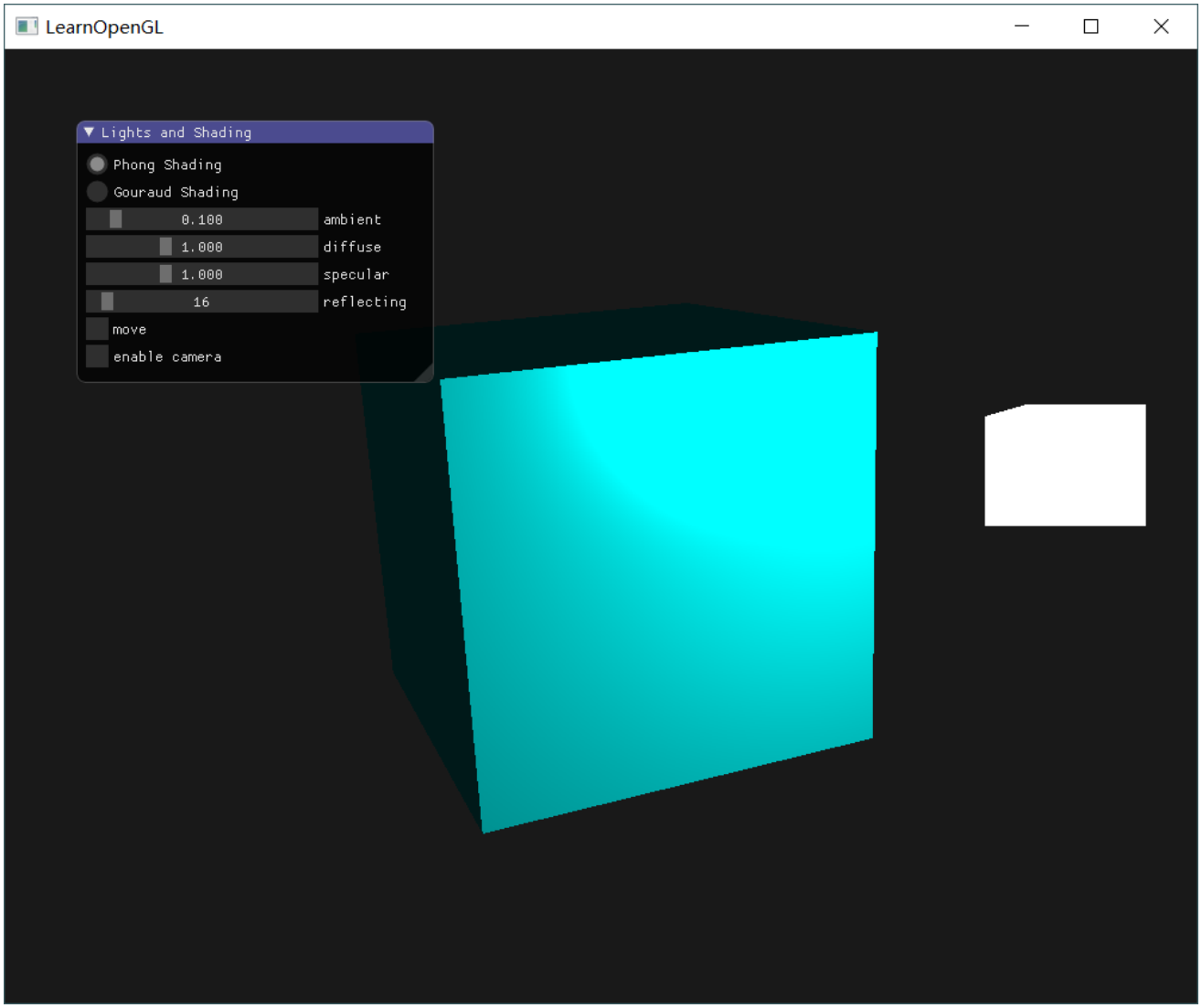


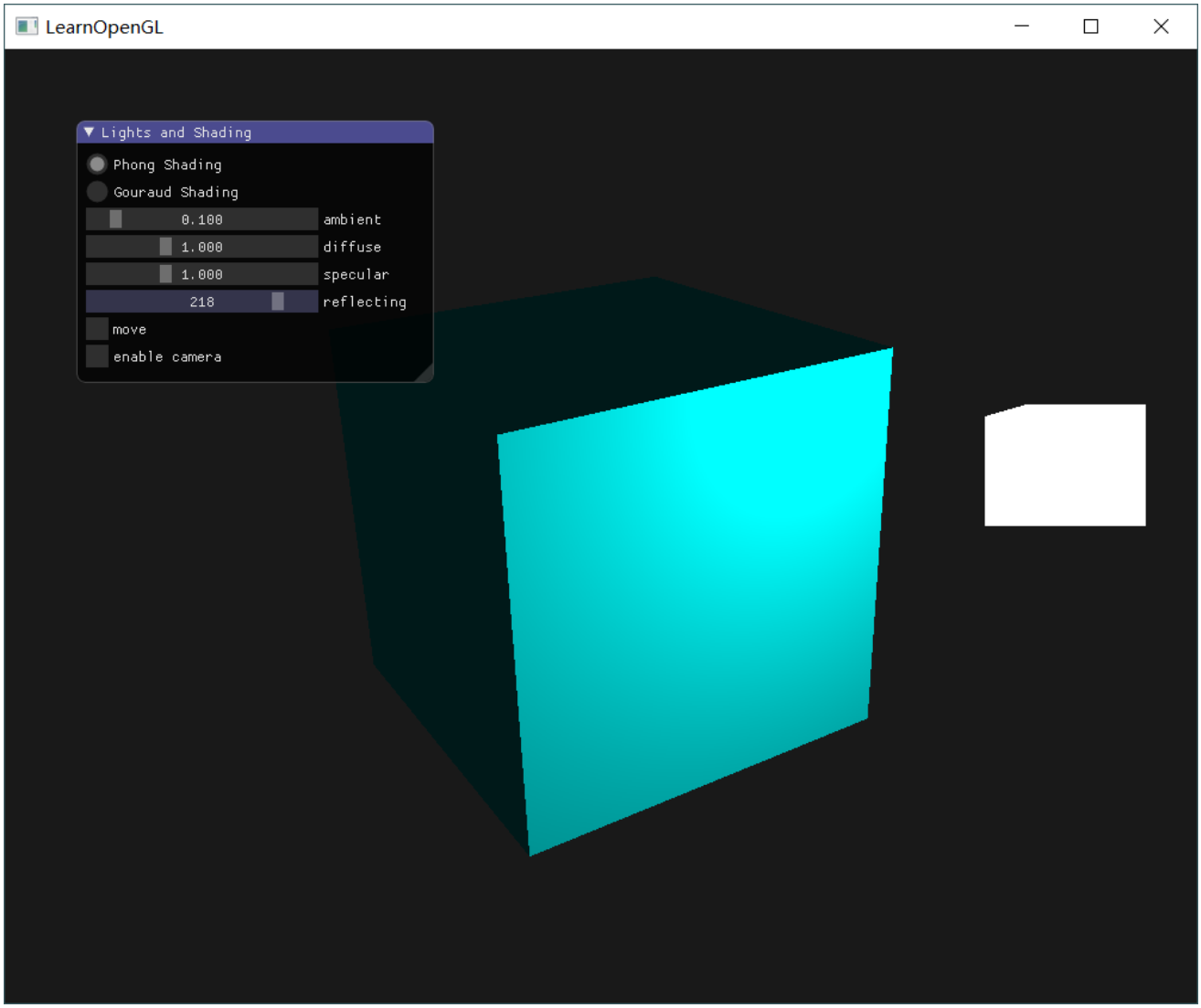
specular的影响


LearnOpenGL
Lights and Shading
Phong Shading
Gouraud Shading
0.100
ambient
1.000
diffuse
0.331
specular
32
reflecting
move
enable camera

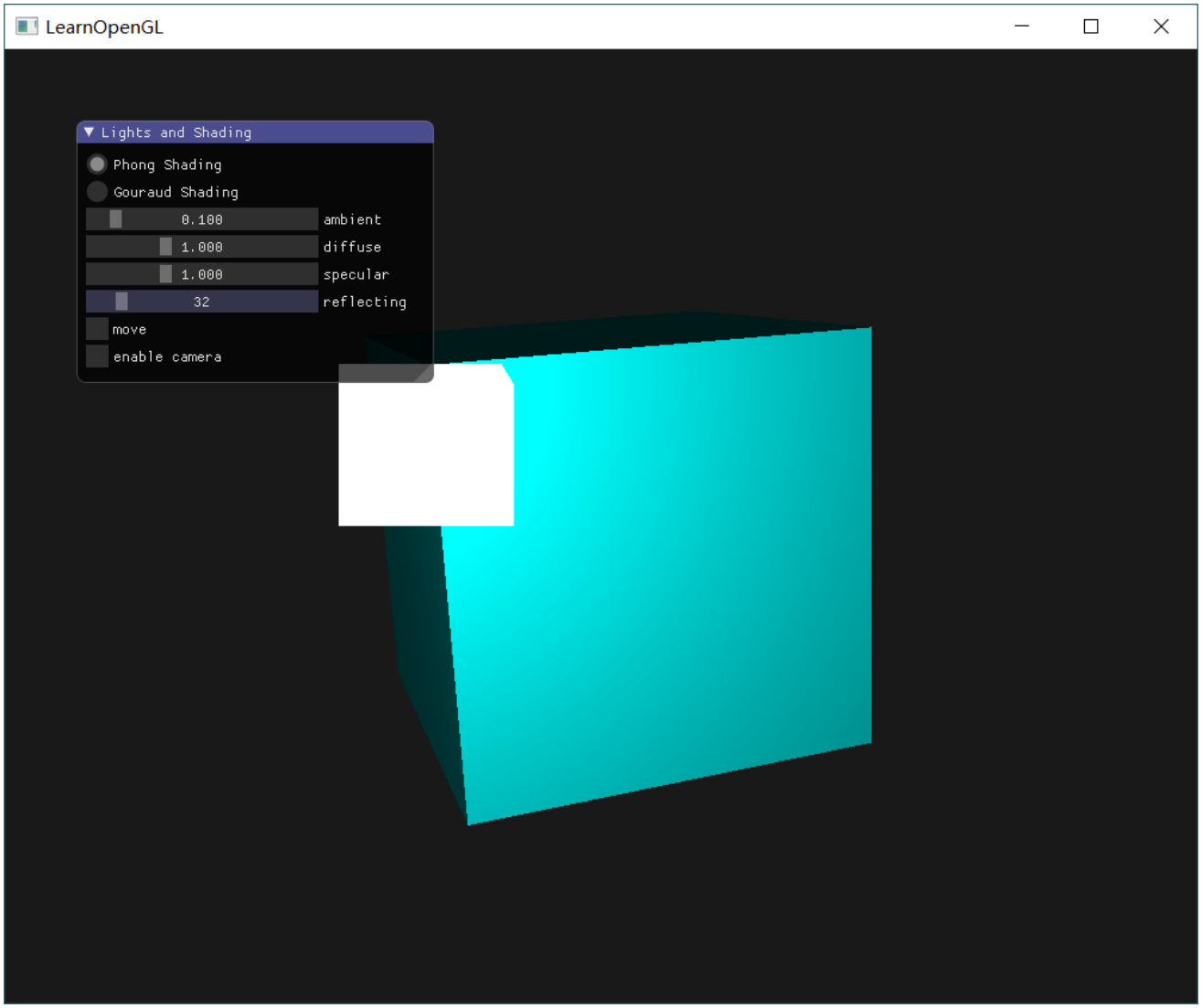

LearnOpenGL
Lights and Shading
Phong Shading
Gouraud Shading
0.100
ambient
1.000
diffuse
2.785
specular
32
reflecting
move
enable camera

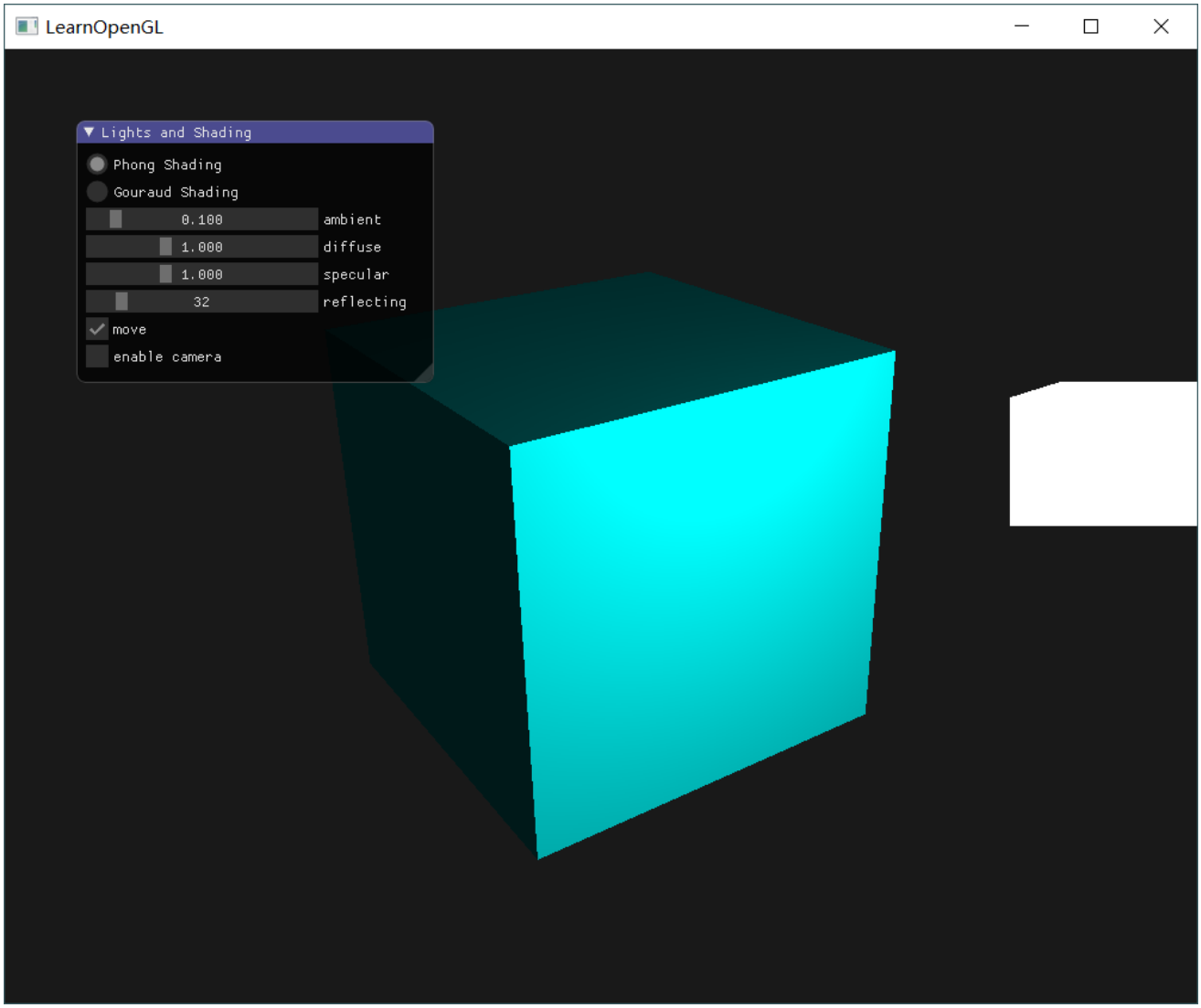
反光度的影响


LearnOpenGL
Lights and Shading
Phong Shading
Gouraud Shading
0.100
ambient
1.000
diffuse
1.000
specular
16
reflecting
move
enable camera

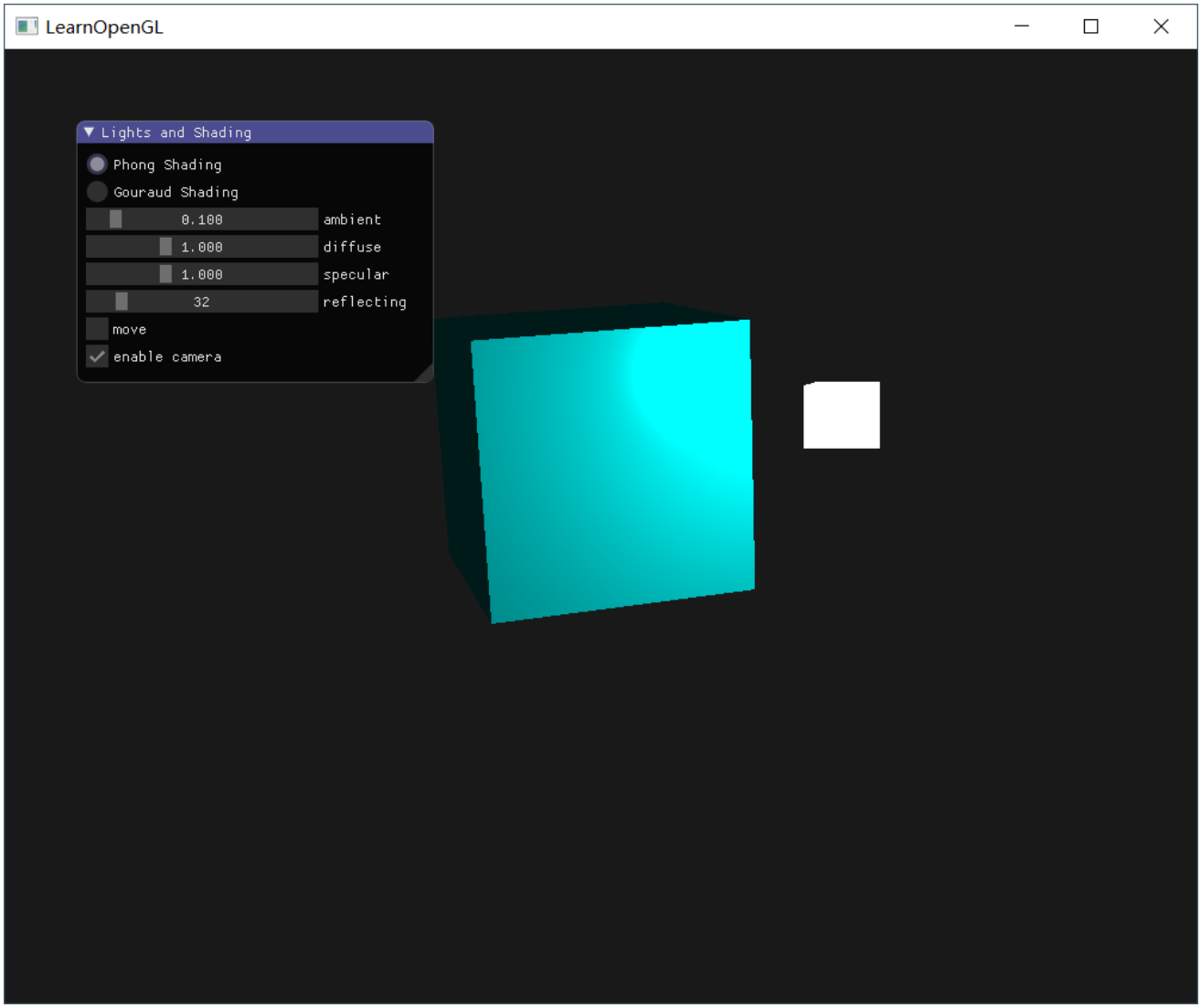

LearnOpenGL
Lights and Shading
Phong Shading
Gouraud Shading
0.100
ambient
1.000
diffuse
1.000
specular
218
reflecting
move
enable camera

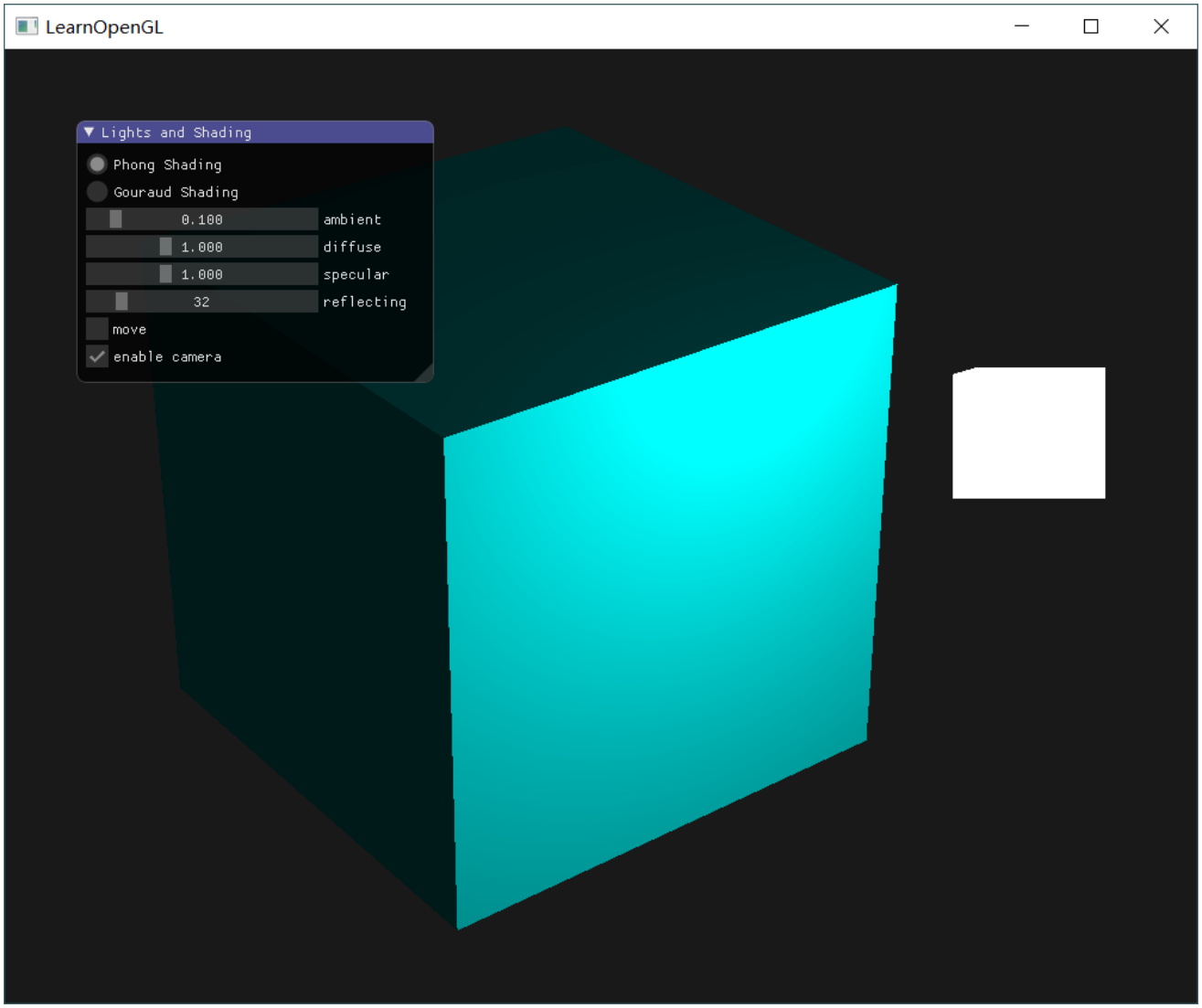
光源移动


LearnOpenGL
Lights and Shading
Phong Shading
Gouraud Shading
0.100
ambient
1.000
diffuse
1.000
specular
32
reflecting
move
enable camera


LearnOpenGL
Lights and Shading
Phong Shading
Gouraud Shading
0.100
ambient
1.000
diffuse
1.000
specular
32
reflecting
move
enable camera

摄影机移动


LearnOpenGL
Lights and Shading
Phong Shading
Gouraud Shading
0.100
ambient
1.000
diffuse
1.000
specular
32
reflecting
move
enable camera


LearnOpenGL
Lights and Shading
Phong Shading
Gouraud Shading
0.100
ambient
1.000
diffuse
1.000
specular
32
reflecting
move
enable camera