Homework 6 - Lights and Shading

陈曦 16340036

Basic:

1.实现Phong光照模型：

- **场景中绘制一个cube**

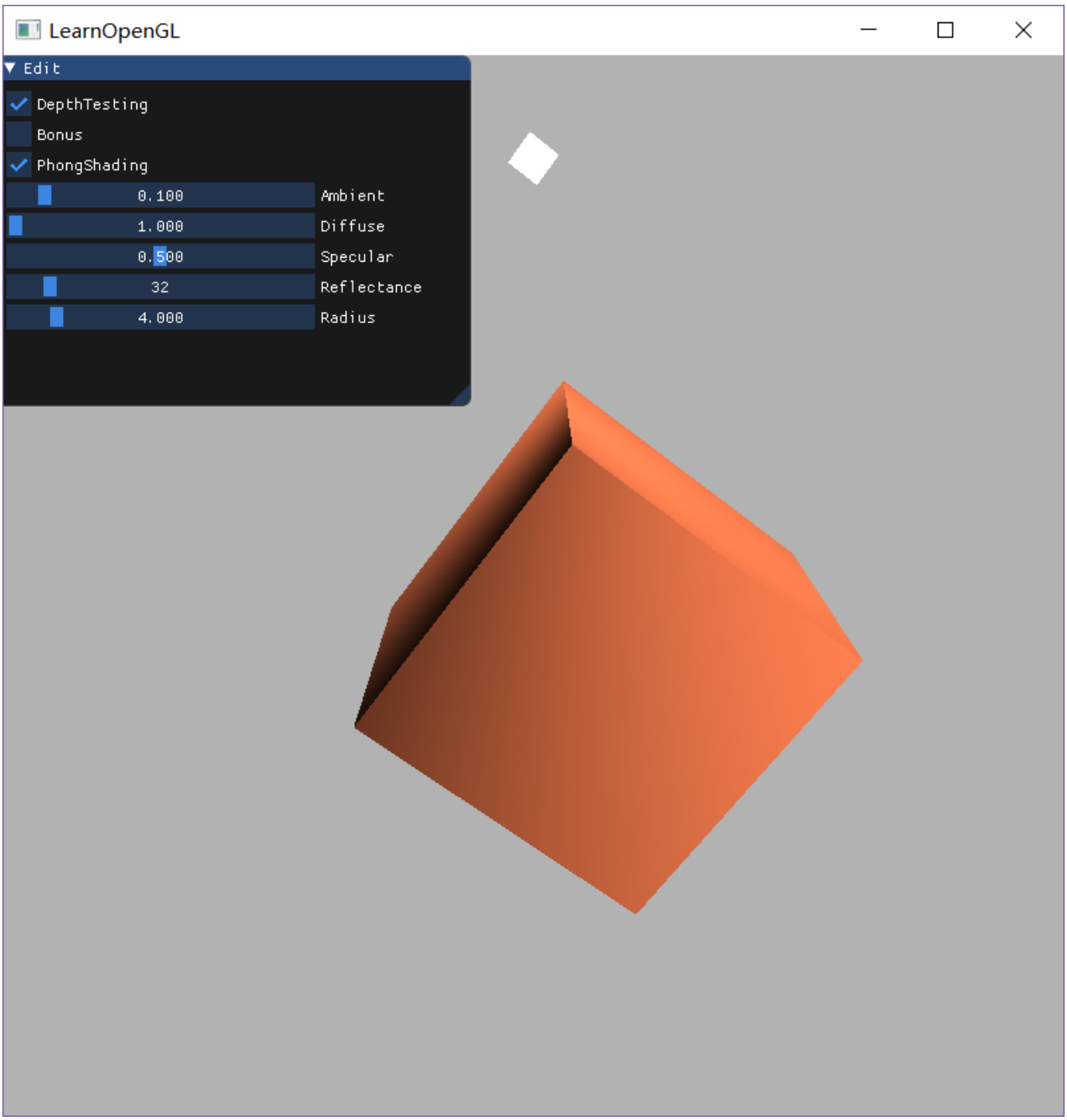
绘制6*2个三角形最终拼接成一个立方体，作为被投光的对象

```
//多个三角形最终拼接成一个立方体（6*2）
float vertices[] = {
   -2.0f, -2.0f, -2.0f, 1.0f, 0.0f, 0.0f, //triangle 1 : begin
    2.0f, -2.0f, -2.0f, 1.0f, 0.0f, 0.0f,
    2.0f,  2.0f, -2.0f, 1.0f, 0.0f, 0.0f, //triangle 1 : end
    2.0f,  2.0f, -2.0f, 1.0f, 0.0f, 0.0f, // triangle 2 : begin
   -2.0f,  2.0f, -2.0f, 1.0f, 0.0f, 0.0f,
   -2.0f, -2.0f, -2.0f, 1.0f, 0.0f, 0.0f, // triangle 2 : end

   -2.0f, -2.0f,  2.0f, 0.0f, 1.0f, 0.0f,
    2.0f, -2.0f,  2.0f, 0.0f, 1.0f, 0.0f,
    2.0f,  2.0f,  2.0f, 0.0f, 1.0f, 0.0f,
    2.0f,  2.0f,  2.0f, 0.0f, 1.0f, 0.0f,
   -2.0f,  2.0f,  2.0f, 0.0f, 1.0f, 0.0f,
   -2.0f, -2.0f,  2.0f, 0.0f, 1.0f, 0.0f,

   -2.0f,  2.0f,  2.0f, 0.0f, 0.0f, 1.0f,
   -2.0f,  2.0f, -2.0f, 0.0f, 0.0f, 1.0f,
   -2.0f, -2.0f, -2.0f, 0.0f, 0.0f, 1.0f,
   -2.0f, -2.0f, -2.0f, 0.0f, 0.0f, 1.0f,
   -2.0f, -2.0f,  2.0f, 0.0f, 0.0f, 1.0f,
   -2.0f,  2.0f,  2.0f, 0.0f, 0.0f, 1.0f,

    2.0f,  2.0f,  2.0f, 1.0f, 1.0f, 0.0f,
    2.0f,  2.0f, -2.0f, 1.0f, 1.0f, 0.0f,
    2.0f, -2.0f, -2.0f, 1.0f, 1.0f, 0.0f,
    2.0f, -2.0f, -2.0f, 1.0f, 1.0f, 0.0f,
    2.0f, -2.0f,  2.0f, 1.0f, 1.0f, 0.0f,
    2.0f,  2.0f,  2.0f, 1.0f, 1.0f, 0.0f,

   -2.0f, -2.0f, -2.0f, 1.0f, 0.0f, 1.0f,
    2.0f, -2.0f, -2.0f, 1.0f, 0.0f, 1.0f,
    2.0f, -2.0f,  2.0f, 1.0f, 0.0f, 1.0f,
    2.0f, -2.0f,  2.0f, 1.0f, 0.0f, 1.0f,
   -2.0f, -2.0f,  2.0f, 1.0f, 0.0f, 1.0f,
   -2.0f, -2.0f, -2.0f, 1.0f, 0.0f, 1.0f,

   -2.0f,  2.0f, -2.0f, 0.0f, 1.0f, 1.0f,
    2.0f,  2.0f, -2.0f, 0.0f, 1.0f, 1.0f,
    2.0f,  2.0f,  2.0f, 0.0f, 1.0f, 1.0f,
    2.0f,  2.0f,  2.0f, 0.0f, 1.0f, 1.0f,
   -2.0f,  2.0f,  2.0f, 0.0f, 1.0f, 1.0f,
   -2.0f,  2.0f, -2.0f, 0.0f, 1.0f, 1.0f
};
```

还需要一个立方体作为光源，并为这个立方体创建一个专门的VAO

```
unsigned int lightVAO;
glGenVertexArrays(1, &lightVAO);
glBindVertexArray(lightVAO);
// 只需要绑定VBO不用再次设置VBO的数据，因为箱子的VBO数据中已经包含了正确的立方体顶点数据
glBindBuffer(GL_ARRAY_BUFFER, VBO);
// 设置灯立方体的顶点属性（对我们的灯来说仅仅只有位置数据）
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(float), (void*)0);
glEnableVertexAttribArray(0);
```

光源和立方体截图如下，初始为Phong Shading：



- **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**

Phong Shading:

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。

顶点着色器

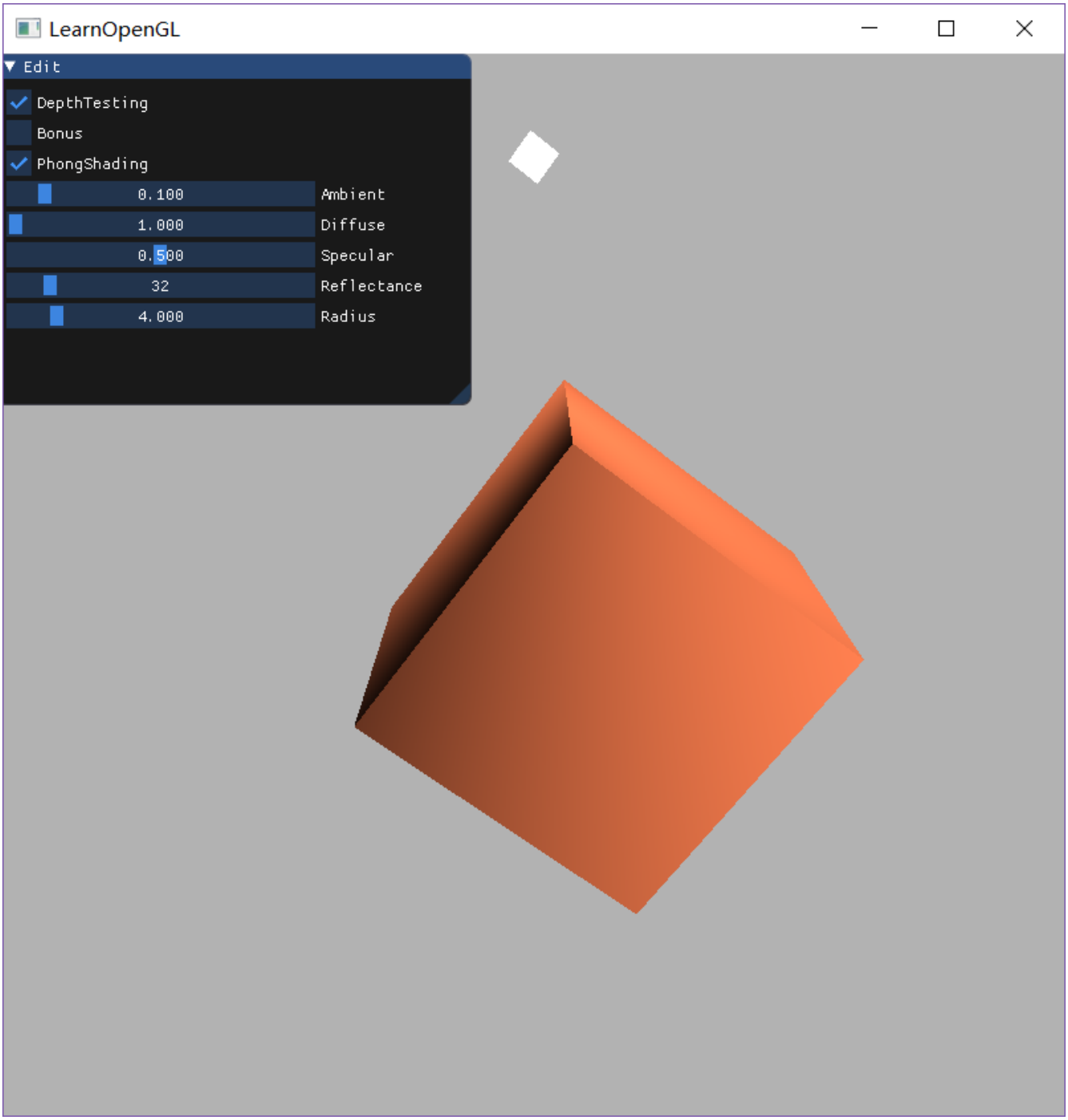
```
const char *PhongVertexShaderSource = "#version 330 core\n"
"layout(location = 0) in vec3 aPos;\n"
"layout(location = 1) in vec3 aNormal;\n"

"out vec3 FragPos;\n" //片段位置，世界空间中的顶点位置
"out vec3 Normal;\n" //法向量

"uniform mat4 model;\n"
"uniform mat4 view;\n"
"uniform mat4 projection;\n"

"void main()\n"
"{\n"
"    gl_Position = projection * view * model * vec4(aPos, 1.0);\n"
"    FragPos = vec3(model * vec4(aPos, 1.0));\n" //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
"    Normal = mat3(transpose(inverse(model))) * aNormal;\n" //生成法线矩阵，把被处理过的矩阵强制转换为3×3矩阵
"}\0";
```

片段着色器

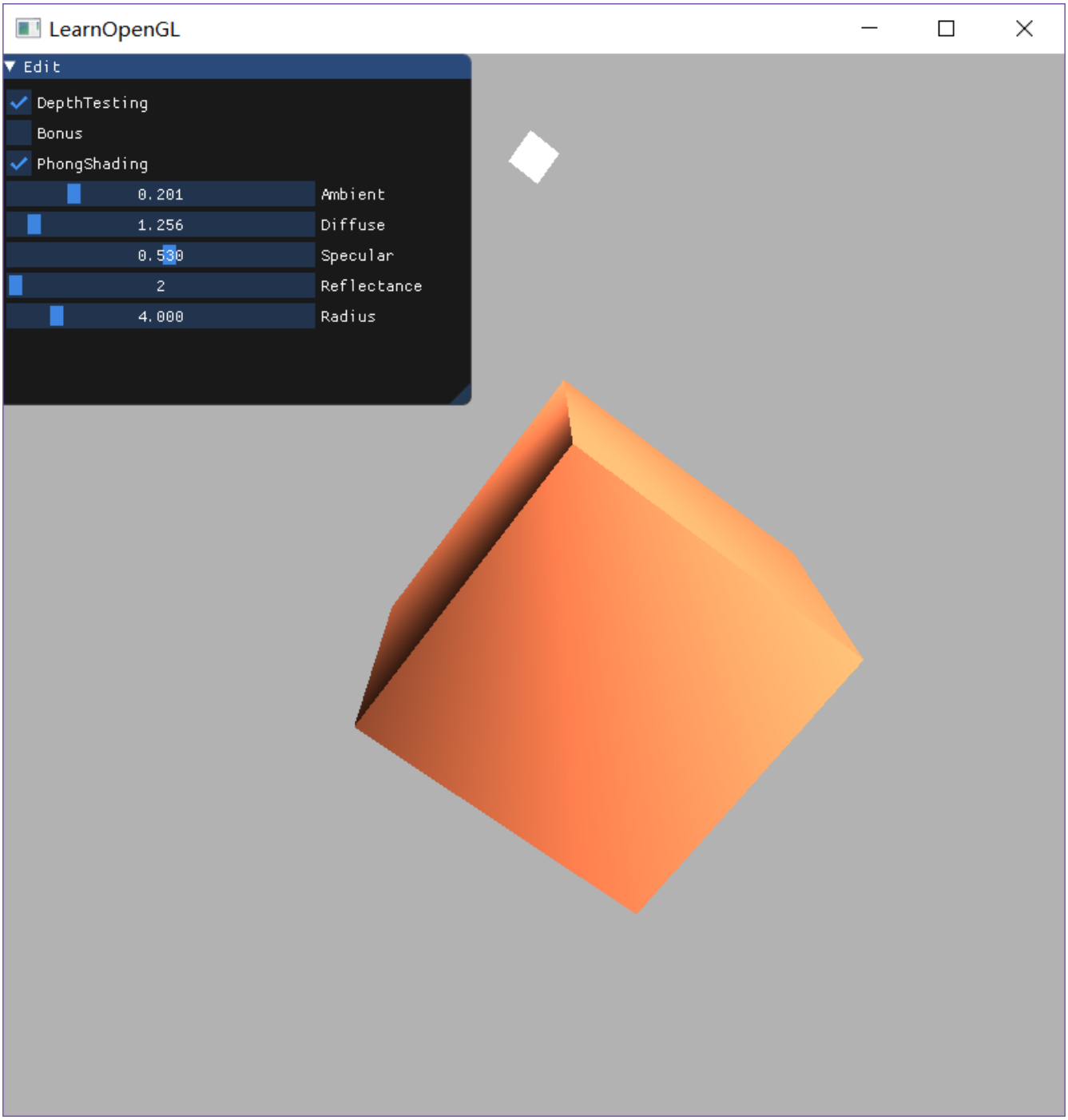
```
const char *PhongFragmentShaderSource = "#version 330 core\n"
"in vec3 FragPos;\n"
"in vec3 Normal;\n"
"out vec4 FragColor;\n"

"uniform vec3 lightPos;\n" //光源位置
"uniform vec3 viewPos;\n"  //摄像机位置坐标
"uniform vec3 lightColor;\n"
"uniform vec3 objectColor;\n"

"uniform float ambientStrength;\n" //ambient因子
"uniform float diffuseStrength;\n" //diffuse因子
"uniform float specularStrength;\n" //specular因子
"uniform int reflectance;\n" //反光度

"void main()\n"
"{\n"
"    vec3 ambient = ambientStrength * lightColor;\n" //光的颜色乘以一个很小的常量环境因子
"    vec3 norm = normalize(Normal);\n" //标准化
"    vec3 lightDir = normalize(lightPos - FragPos);\n" //光源和片段位置之间的方向向量
"    float diff = max(dot(norm, lightDir), 0.0);\n" //对norm和lightDir向量进行点乘，计算光源对当前片段实际的漫发射影响
"    vec3 diffuse = diffuseStrength * diff * lightColor;\n" //得到漫反射分量
```

```
"    vec3 viewDir = normalize(viewPos - FragPos);\n" //视线方向向量
"    vec3 reflectDir = reflect(-lightDir, norm);\n"  //对应的沿着法线轴的反射向量
"    float spec = pow(max(dot(viewDir, reflectDir), 0.0), reflectance);\n" //计算镜面分量
"    vec3 specular = specularStrength * spec * lightColor;\n" //计算镜面分量
"    vec3 result = (ambient + diffuse + specular) * objectColor;\n"
"    FragColor = vec4(result, 1.0);\n"
"}\n\0";
```

截图如下：



改变参数截图如下：



Gouraud Shading:

在顶点着色器中实现的冯氏光照模型叫做Gouraud着色(Gouraud Shading)，片段的颜色值是由插值光照颜色所得来的。由于插值，这种光照看起来有点逊色。冯氏着色能产生更平滑的光照效果。

顶点着色器

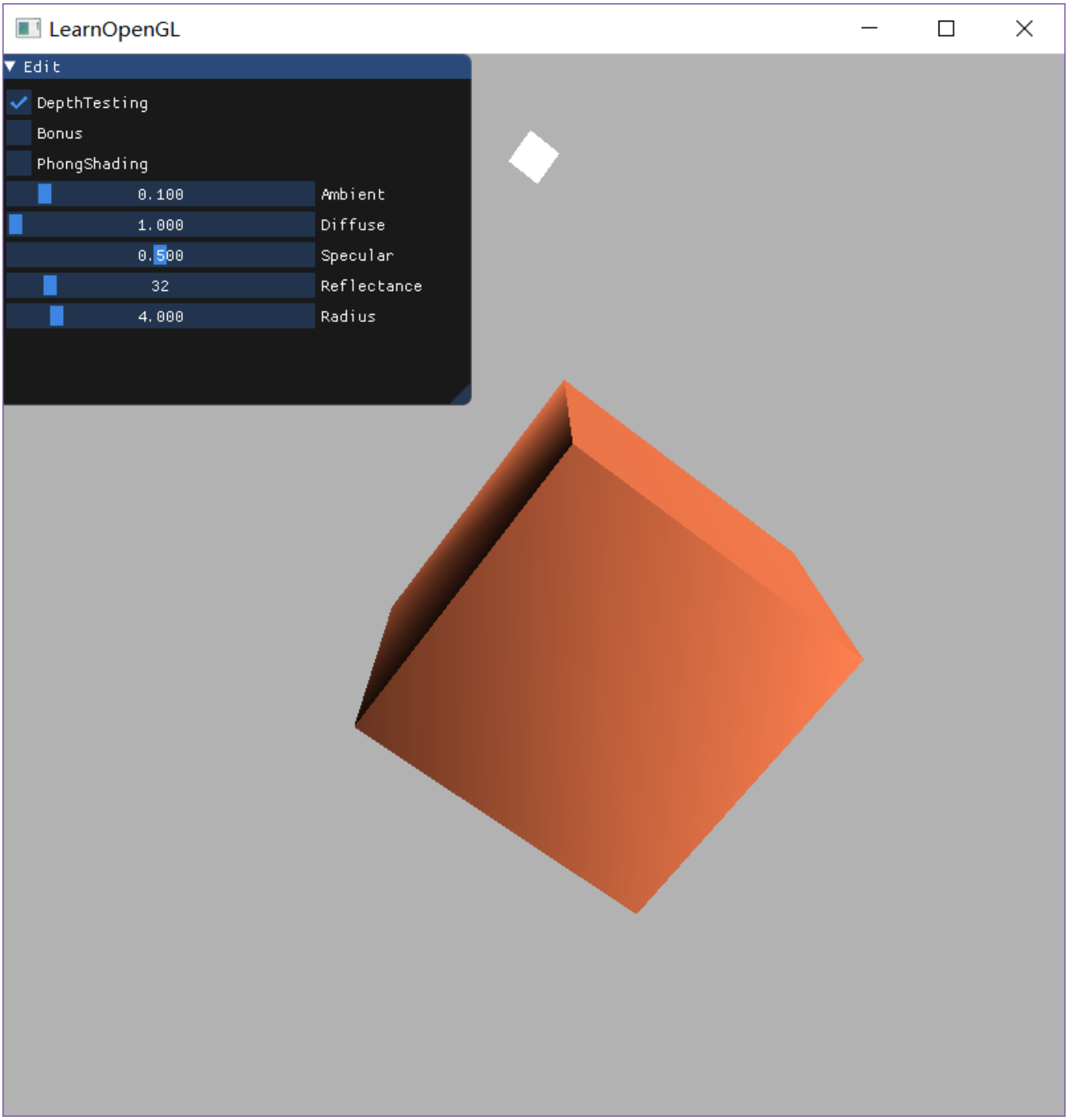
```
const char *GouraudVertexShaderSource = "#version 330 core\n"
"layout(location = 0) in vec3 aPos;\n"
"layout(location = 1) in vec3 aNormal;\n"

"out vec3 LightingColor;\n"

"uniform vec3 lightPos;\n"
```

```
"uniform vec3 viewPos;"
"uniform vec3 lightColor;"

"uniform float ambientStrength;\n"
"uniform float diffuseStrength;\n"
"uniform float specularStrength;\n"
"uniform int reflectance;\n"

"uniform mat4 model;\n"
"uniform mat4 view;\n"
"uniform mat4 projection;\n"
"void main()\n"
"{\n"
"   gl_Position = projection * view * model * vec4(aPos, 1.0);\n"

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

"   vec3 ambient = ambientStrength * lightColor;\n"

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

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

"   LightingColor = ambient + diffuse + specular;\n"
"}\0";
```

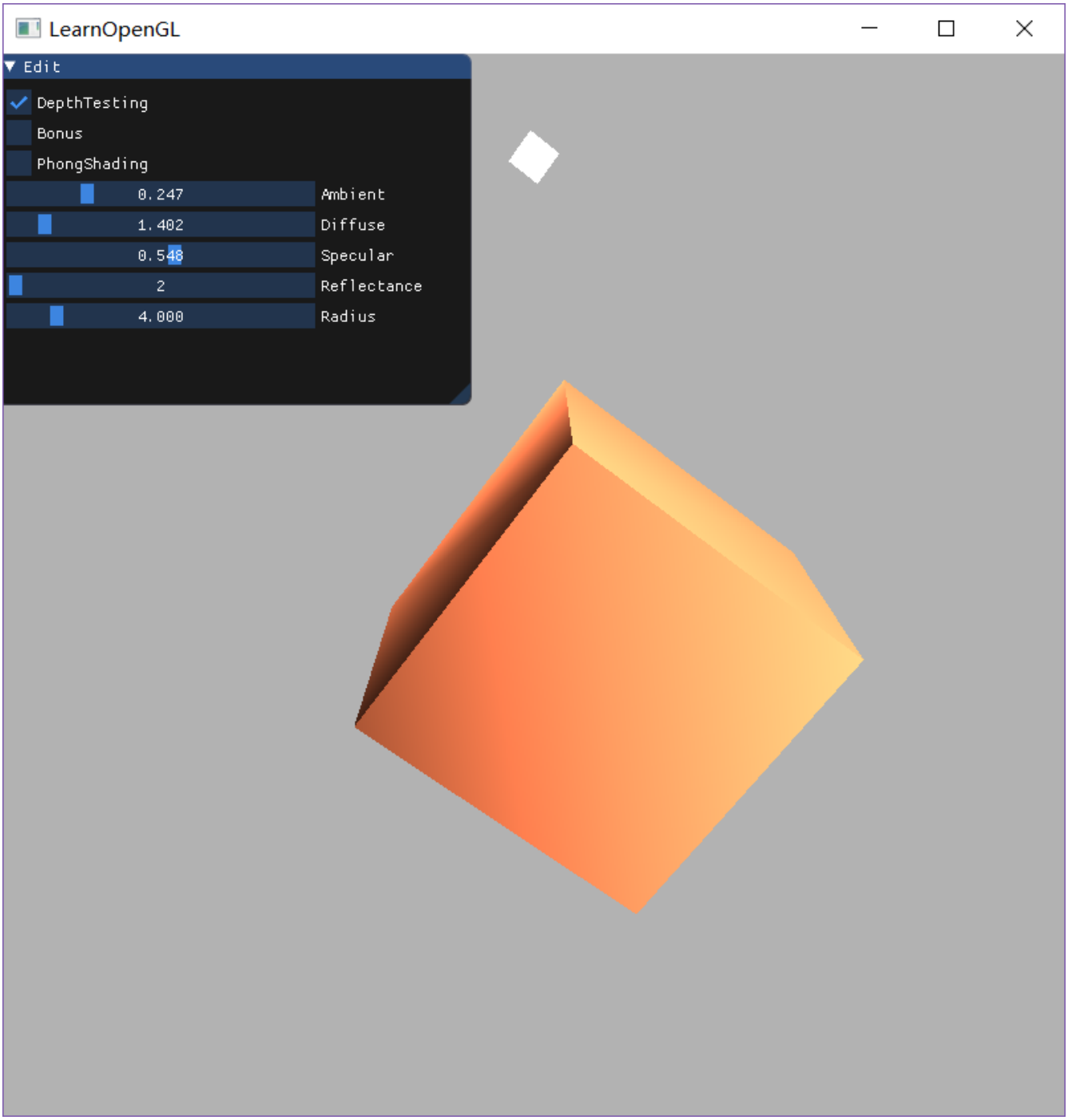
片段着色器

```
const char *GouraudFragmentShaderSource = "#version 330 core\n"
"out vec4 FragColor;\n"

"in vec3 LightingColor;\n"

"uniform vec3 objectColor;\n"

"void main()\n"
"{\n"
"   FragColor = vec4(LightingColor * objectColor, 1.0);\n"
"}\n\0";
```

截图如下:


LearnOpenGL
Edit
DepthTesting
Bonus
PhongShading
0.100
Ambient
1.000
Diffuse
0.500
Specular
32
Reflectance
4.000
Radius

改变参数截图如下：



- **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**

```
bool Phong = true;
bool depth = true;
bool bonus = false;
float ambientStrength = 0.1;
float diffuseStrength = 1.0;
float specularStrength = 0.5;
int reflectance = 32;
float radius = 4.0f;
```

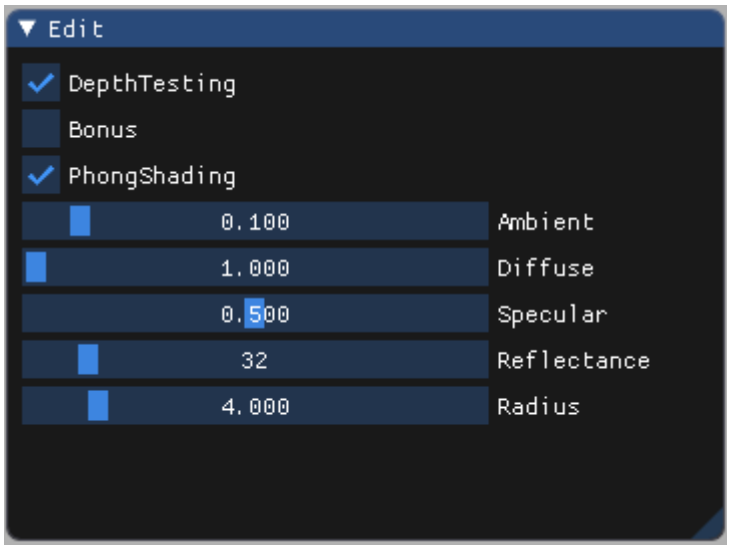
2.使用GUI，使参数可调节，效果实时更改：

- **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**

```
//创建ImGui
ImGui_ImplOpenGL3_NewFrame();
ImGui_ImplGlfw_NewFrame();
NewFrame();

Begin("Edit");
Checkbox("DepthTesting", &depth);
Checkbox("Bonus", &bonus);
Checkbox("Phong", &Phong);
SliderFloat("Ambient", &ambientStrength, 0.0f, 1.0f);
SliderFloat("Diffuse", &diffuseStrength, 1.0f, 5.0f);
SliderFloat("Specular", &specularStrength, 0.0f, 1.0f);
SliderInt("Reflectance", &reflectance, 2, 256);
SliderFloat("Radius", &radius, 3.0f, 10.0f);
End();
```

菜单栏截图如下，初始界面默认选中PhongShading，取消选中PhongShading即为Gouraud Shading

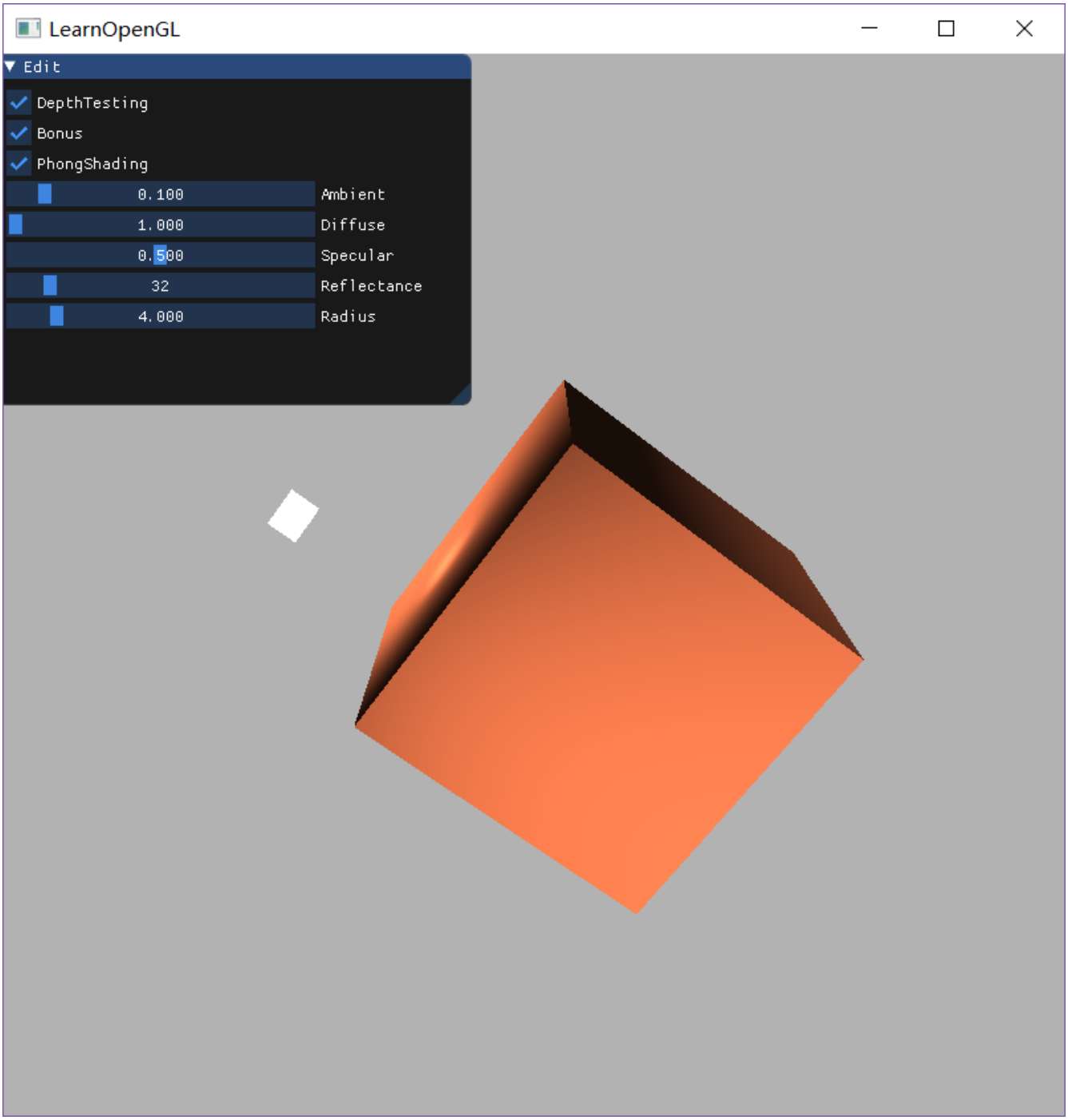


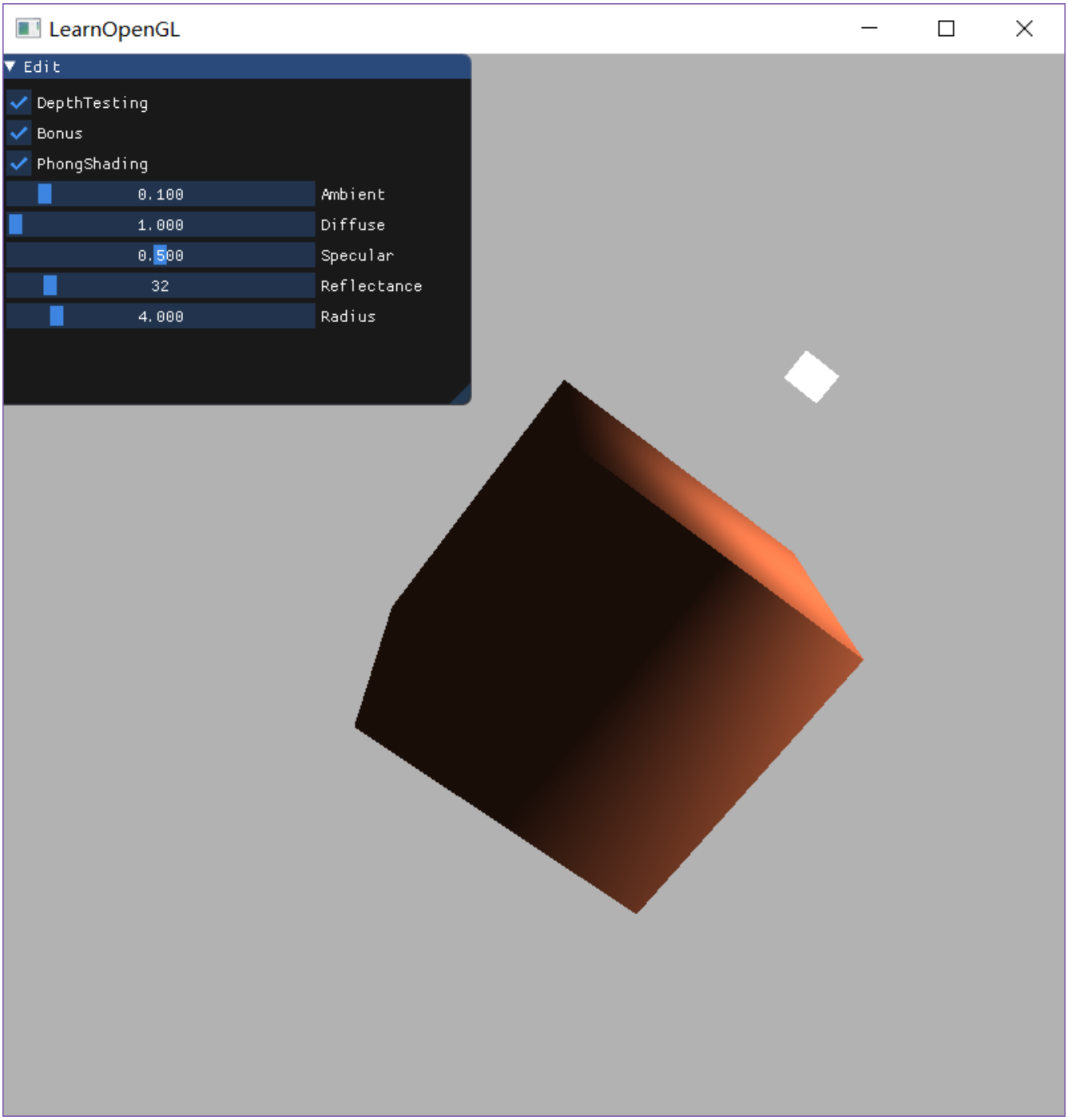
Bonus:

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。

```
    if (bonus) {
        // change the light's position values over time (can be done anywhere in the render
loop actually, but try to do it at least before using the light source positions)
        lightPos.x = cos((float)glfwGetTime()) * radius;
        lightPos.y = sin((float)glfwGetTime()) * radius;
    }
    lightModel = glm::translate(lightModel, lightPos);
    lightModel = glm::scale(lightModel, vec3(0.1f));
```

对光源的处理类似于上一次作业中摄像机的旋转，截图如下：


LearnOpenGL
Edit
DepthTesting
Bonus
PhongShading
0.100
Ambient
1.000
Diffuse
0.500
Specular
32
Reflectance
4.000
Radius


LearnOpenGL
Edit
DepthTesting
Bonus
PhongShading
0.100
Ambient
1.000
Diffuse
0.500
Specular
32
Reflectance
4.000
Radius