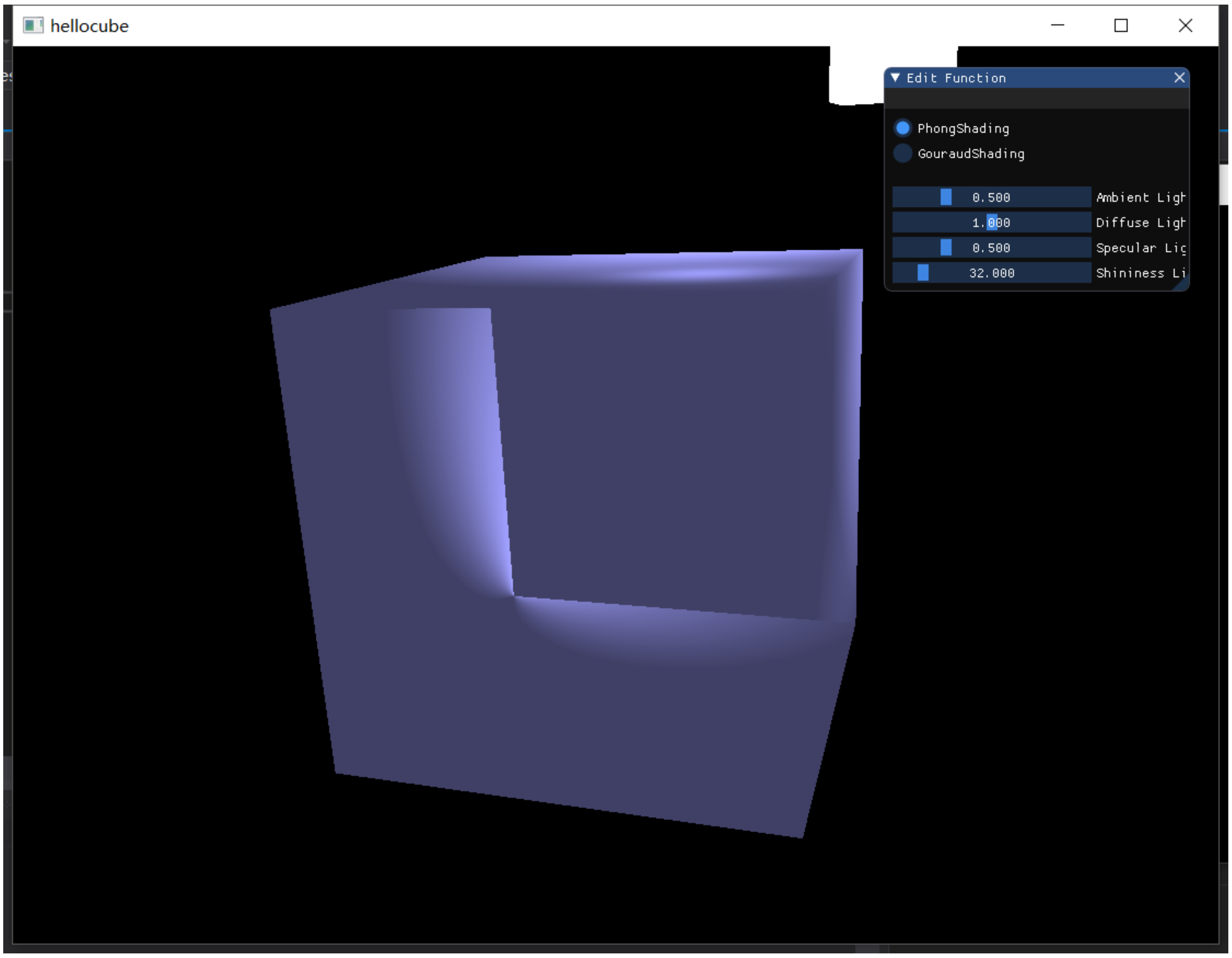
Homework 6 - Lights and Shading

运行结果

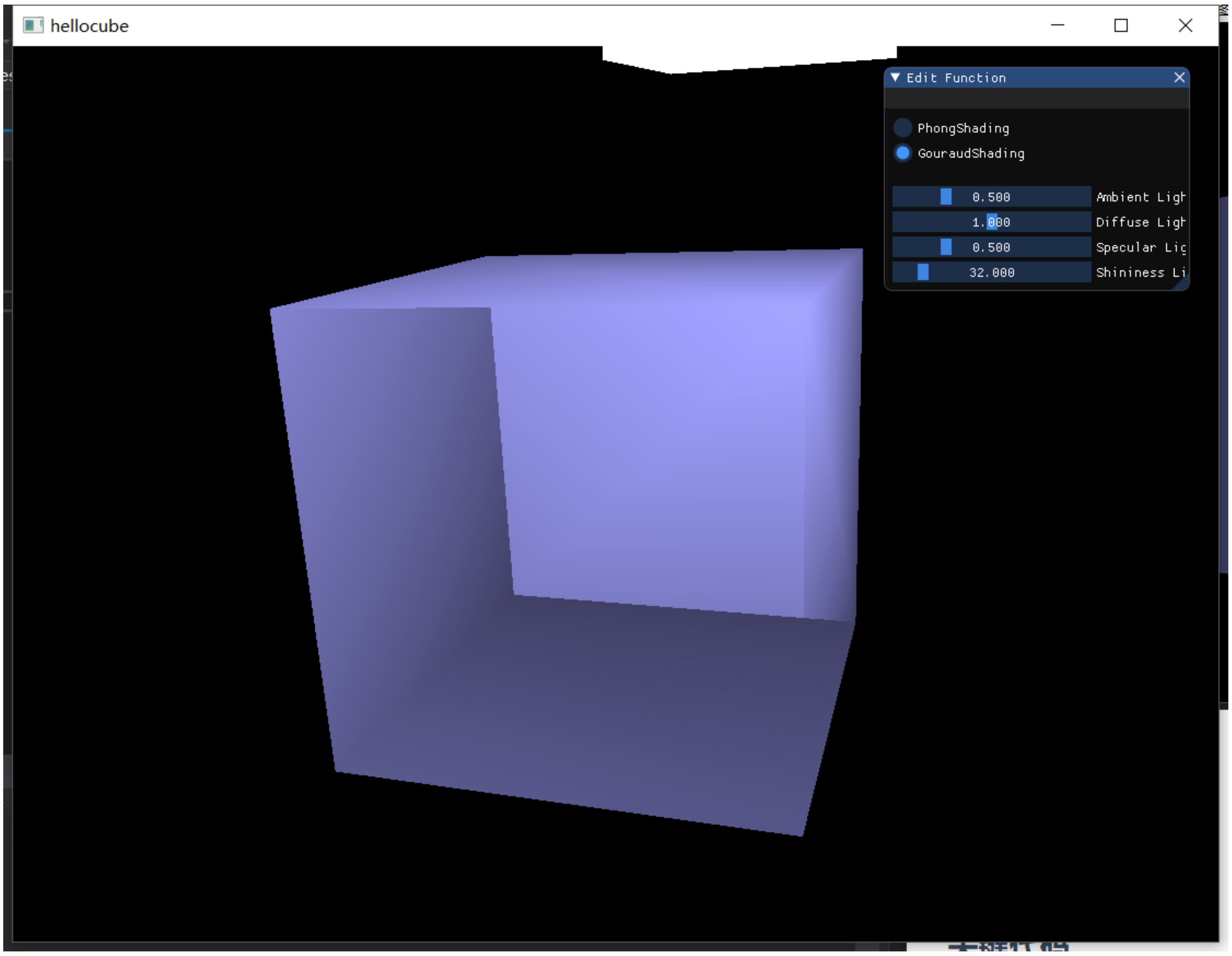
Phong Shading



PhongShading [✓]
GouraudShading

```
0.500    Ambient Ligh
1.000    Diffuse Ligh
0.500    Specular Lig
32.000   Shininess Li
```

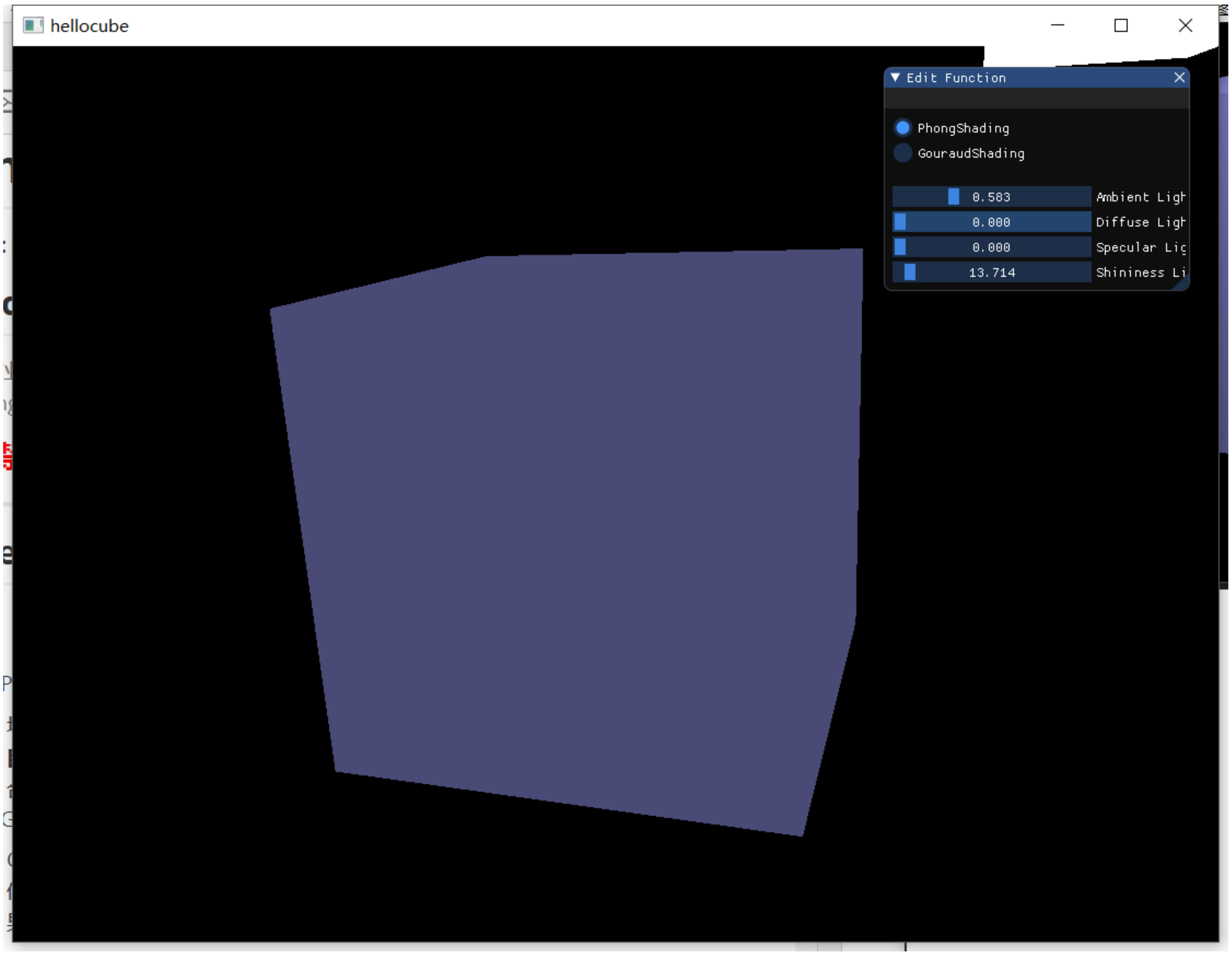
Gouraud Shading



- ○ PhongShading
- ● GouraudShading

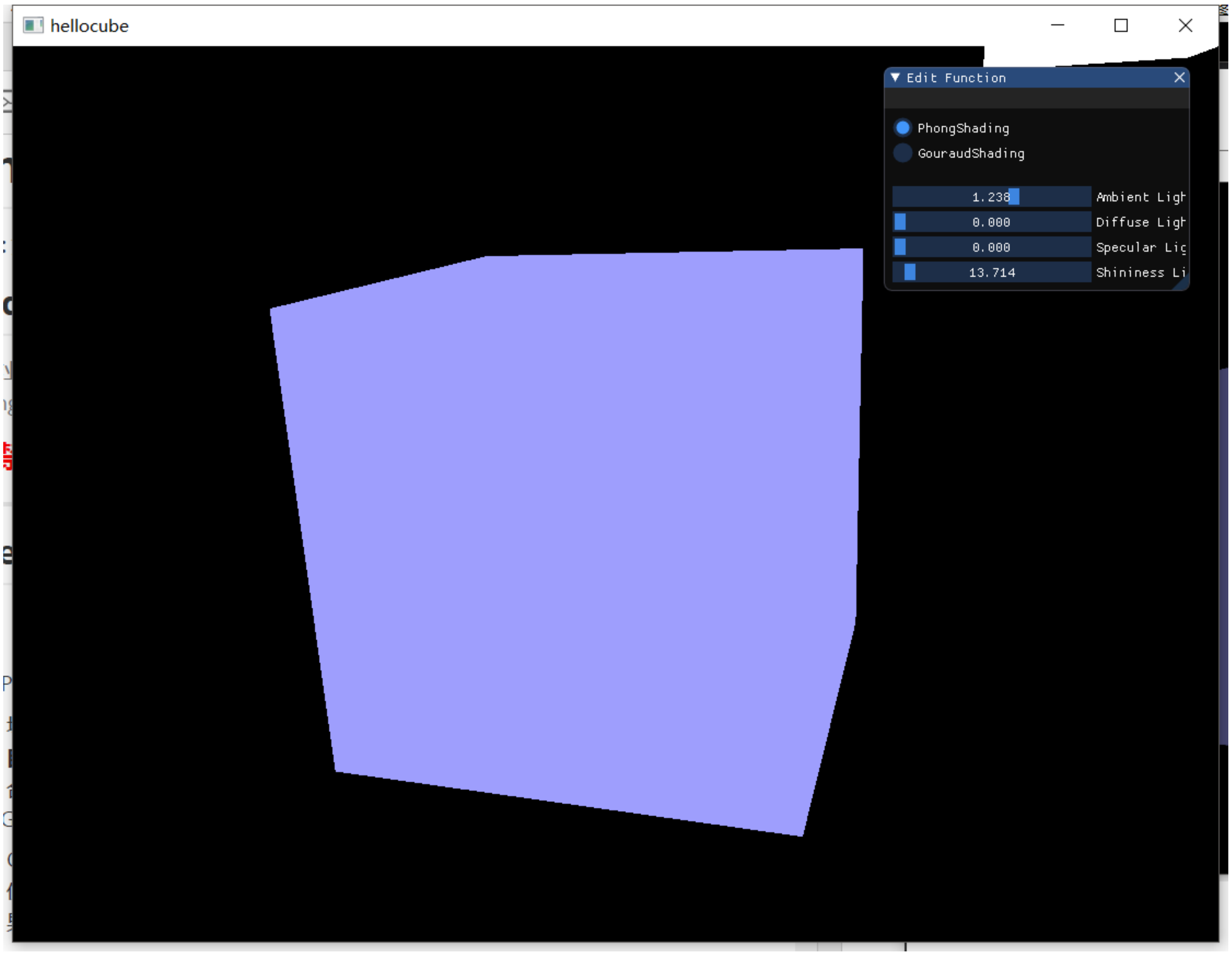
0.500 Ambient Ligh
1.000 Diffuse Ligh
0.500 Specular Lig
32.000 Shininess Li

ambient光照



PhongShading [✓]
GouraudShading

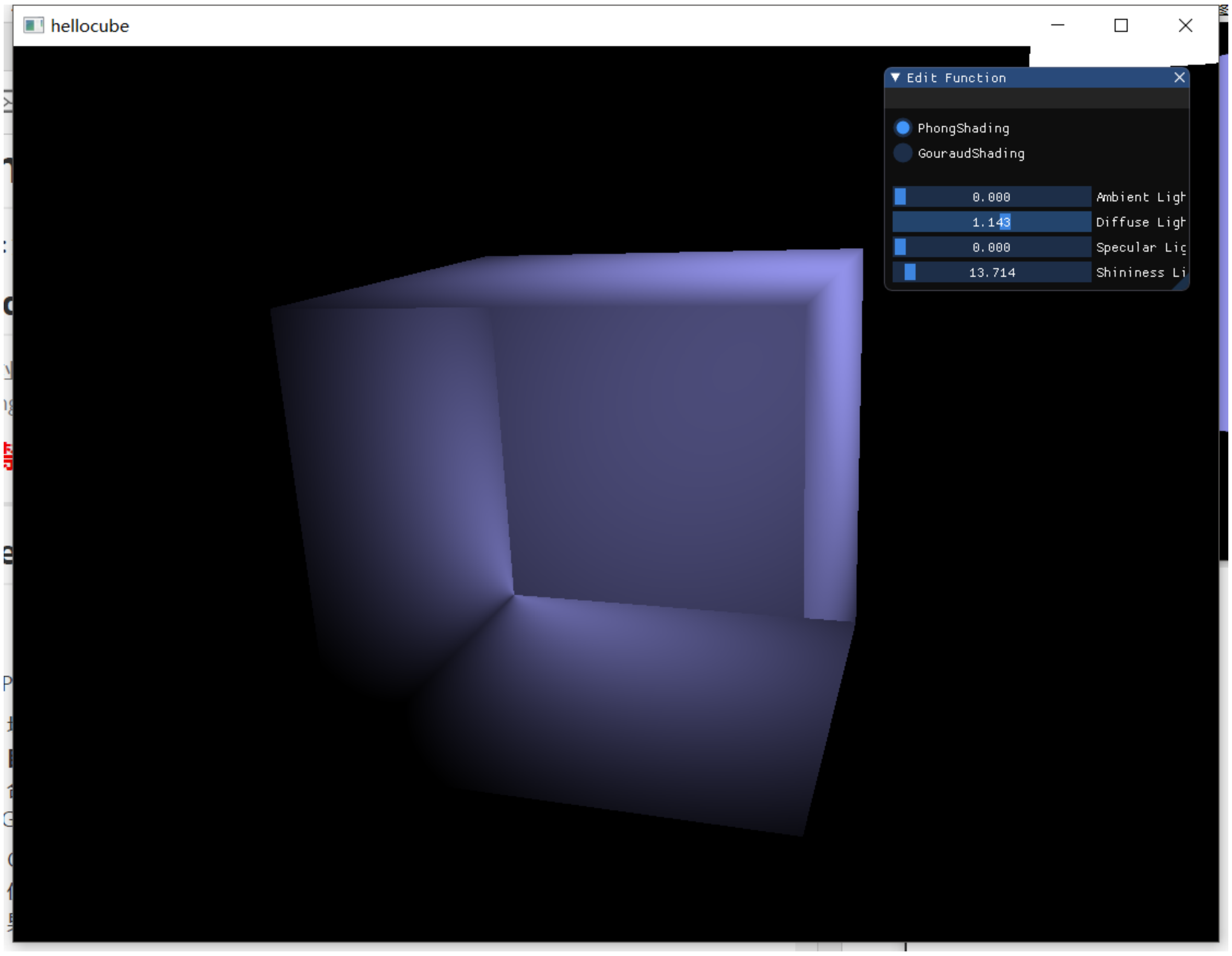
```
0.583   Ambient Ligh
0.000   Diffuse Ligh
0.000   Specular Lig
13.714  Shininess Li
```



PhongShading [✓] / GouraudShading

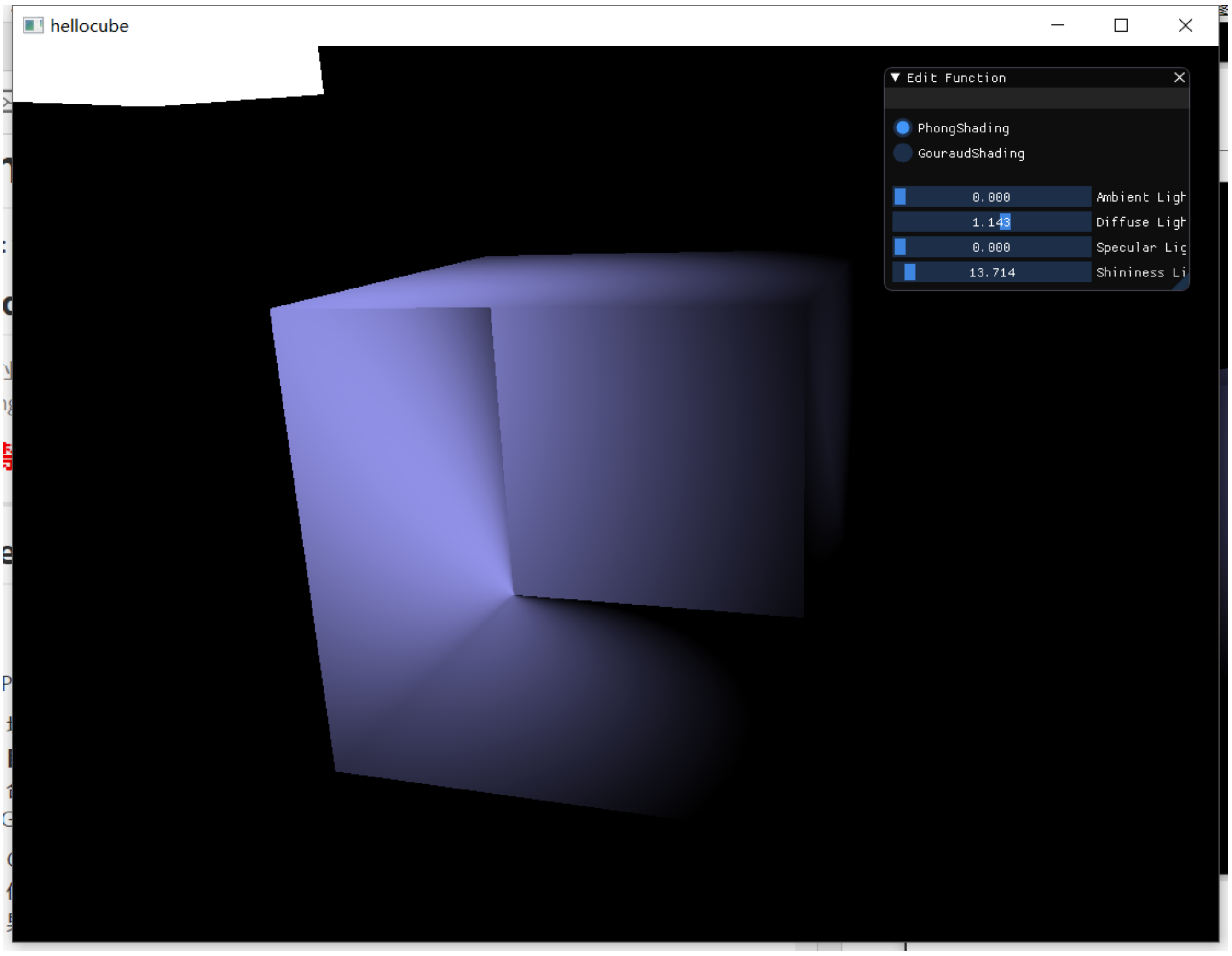
```
1.238   Ambient Ligh
0.000   Diffuse Ligh
0.000   Specular Lig
13.714  Shininess Li
```

diffuse光照



PhongShading [✓]
GouraudShading

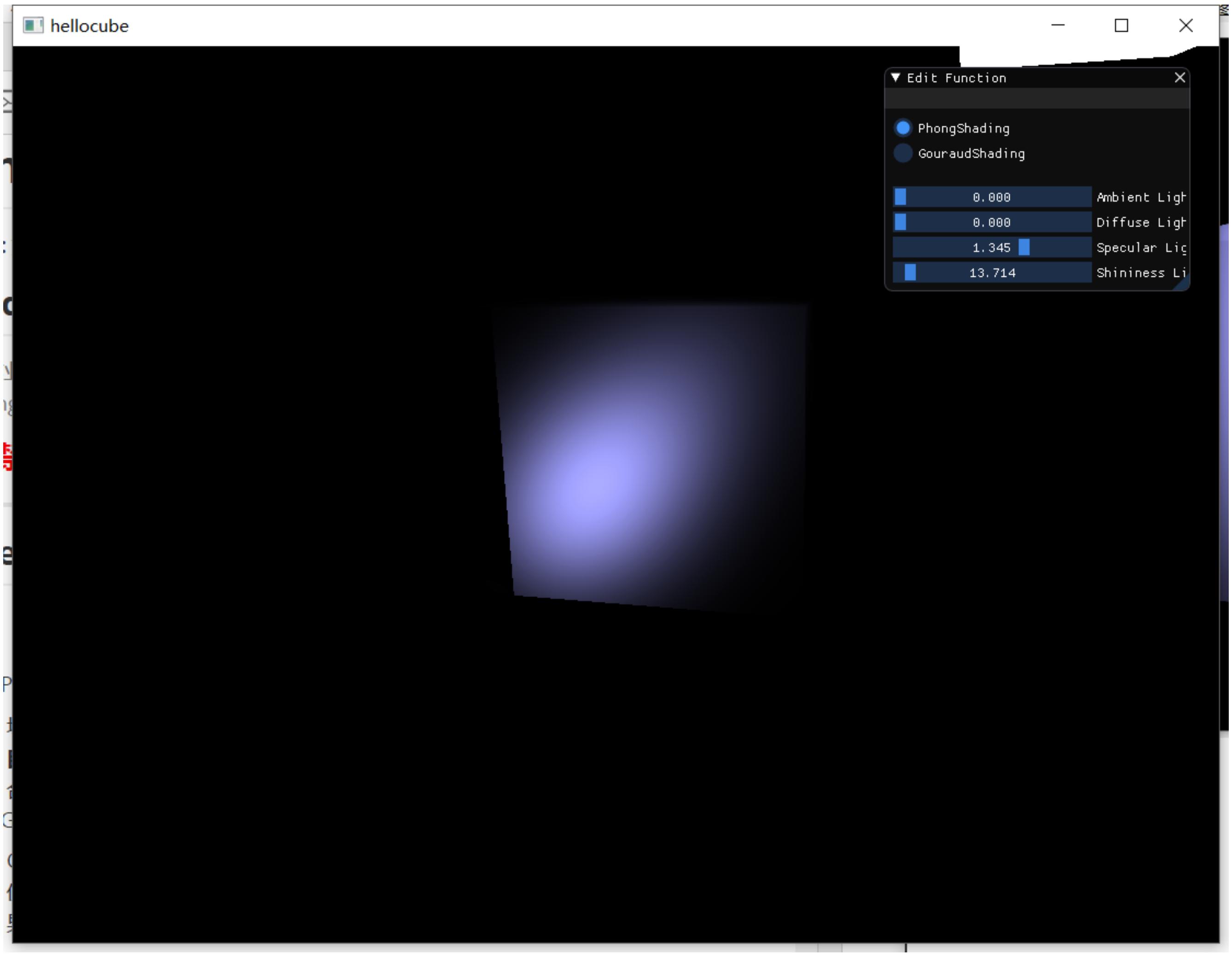
```
0.000   Ambient Ligh
1.143   Diffuse Ligh
0.000   Specular Lig
13.714  Shininess Li
```



PhongShading [✓]
GouraudShading

```
0.000    Ambient Ligh
1.143    Diffuse Ligh
0.000    Specular Lig
13.714   Shininess Li
```

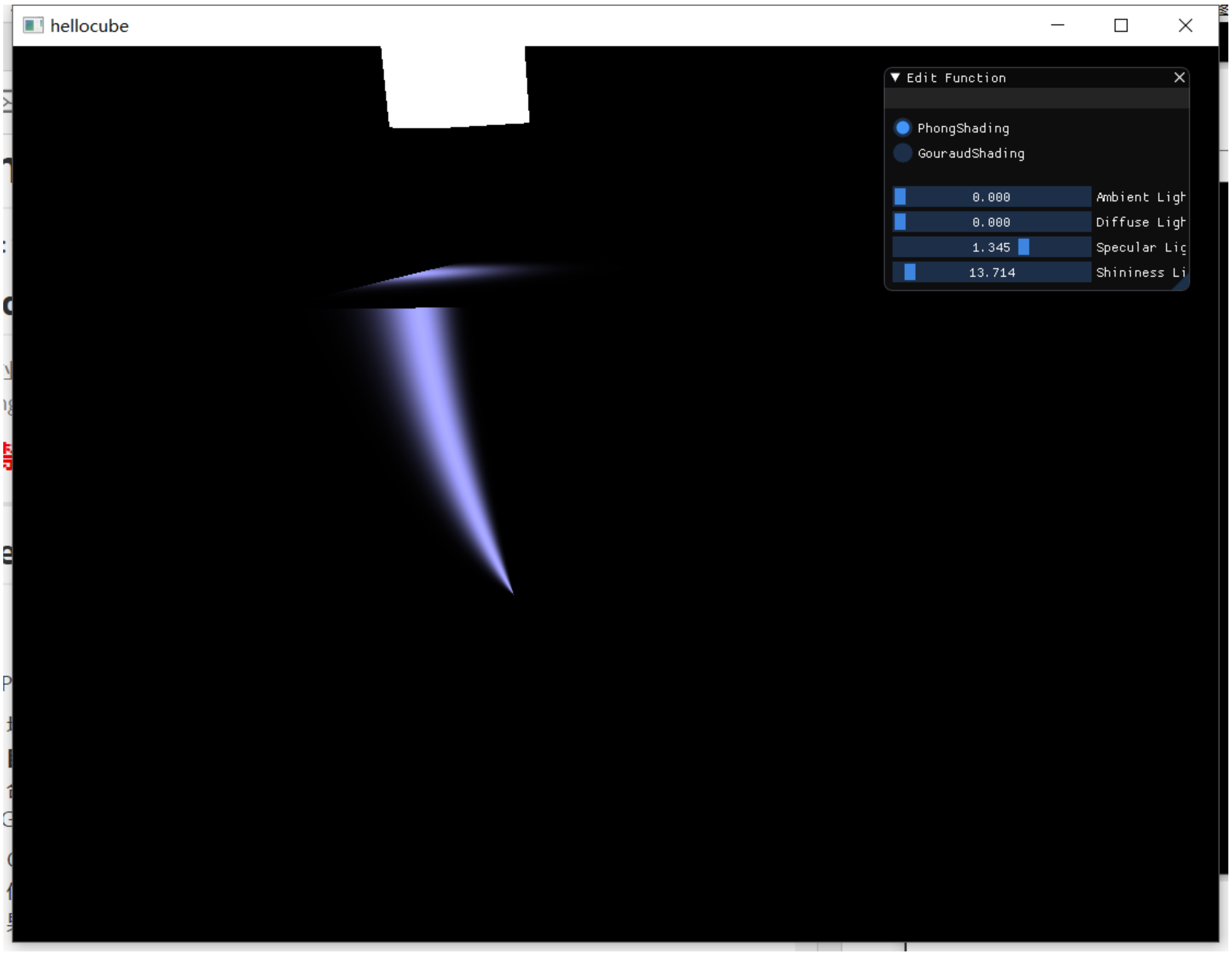
specular光照



Edit Function

[✓ PhongShading]
GouraudShading

```
0.000    Ambient Ligh
0.000    Diffuse Ligh
1.345    Specular Lig
13.714   Shininess Li
```

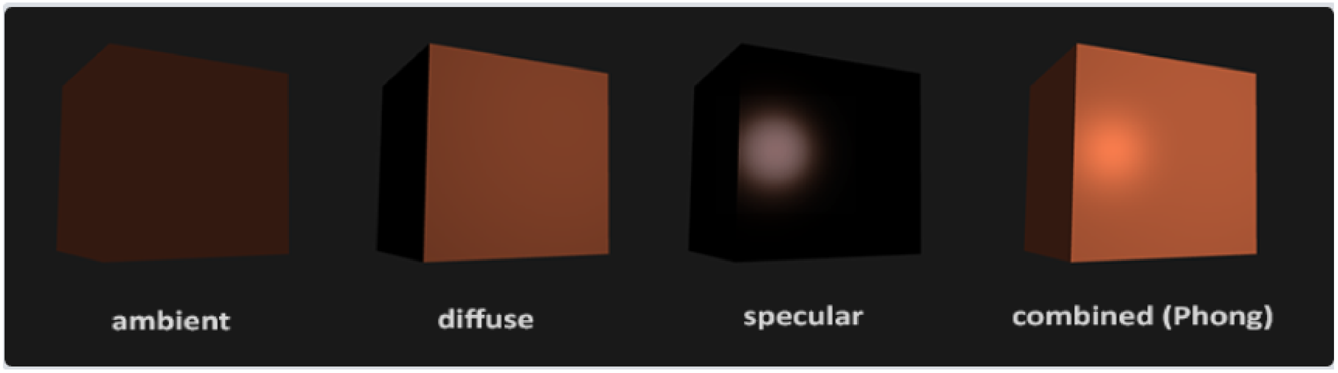


[● PhongShading] GouraudShading

0.000	Ambient Ligh
0.000	Diffuse Ligh
1.345	Specular Lig
13.714	Shininess Li

实现思路

光照模型由三个部分组成，分别是环境光、漫反射和镜面反射



环境光

环境光表现的是周边环境中难以完全屏蔽的光亮如月光、远处微弱的光等，故即使在黑夜人们也可以看见物体，物体因为环境光的存在而总是可以被看见

通过调节环境光因子来控制，环境光因子越大则环境光的强度越大

漫反射

光源对物体方向性的影响，物体正对着光源的部分通常会较亮

通过调节漫反射因子来控制，漫反射因子越大则漫反射的强度越大，不同位置的亮度差越小，使得物体整体越亮

镜面反射

模拟光泽物体上反射光源的亮块

通过调节镜面反射因子来控制，越大则镜面反射的强度越大

Phong Shading

将多边形的每个顶点的法向量线性插值，在每个像素上计算该像素的法向量并进行Phong局部关照明模型的计算来得到其颜色

Gouraud Shading

在每个多边形的顶点使用Phong局部光照明模型计算出顶点颜色，用线性插值得到整个多边形中像素的颜色

关键代码

main函数中使用着色器

```
        lightPos.x = cos(glfwGetTime()) * 1.0f;
        lightPos.z = sin(glfwGetTime()) * 1.0f;
        if (func == FUNC::PHONGSHADING) {
            lightingShader = phongShader;
        }
        else if (func == FUNC::GOURAUDSHADING) {
            lightingShader = gouraudShader;
        }

        lightingShader.use();
        lightingShader.setVec3("objectColor", 0.5f, 0.5f, 0.8f);
        lightingShader.setVec3("lightColor", 1.0f, 1.0f, 1.0f);
        lightingShader.setVec3("lightPos", lightPos);
        lightingShader.setVec3("viewPos", camera.getPosition());
        lightingShader.setFloat("ambientStrength", ambientStrength);
        lightingShader.setFloat("diffuseStrength", diffuseStrength);
        lightingShader.setFloat("specularStrength", specularStrength);
        lightingShader.setFloat("shininess", shininess);

        glm::mat4 projection = glm::perspective(glm::radians(45.0f), (float)SCR_WIDTH /
(float)SCR_HEIGHT, 0.1f, 100.0f);
        glm::mat4 view = glm::mat4(1.0f);
        view = glm::translate(view, glm::vec3(0.0f, 0.0f, -2.5f));
        view = glm::rotate(view, glm::radians(25.0f), glm::vec3(1.0f, -1.0f, 0.0f));    // 调整观
察视角
        glm::mat4 model = glm::mat4(1.0f);
        lightingShader.setMat4("projection", projection);
        lightingShader.setMat4("view", view);
        lightingShader.setMat4("model", model);

        glBindVertexArray(cubeVAO);
        glDrawArrays(GL_TRIANGLES, 0, 36);
        lampShader.use();
        model = glm::mat4(1.0f);
        model = glm::translate(model, lightPos);
        model = glm::scale(model, glm::vec3(0.3f));

        lampShader.setMat4("projection", projection);
        lampShader.setMat4("view", view);
        lampShader.setMat4("model", model);

        glBindVertexArray(lightVAO);
        glDrawArrays(GL_TRIANGLES, 0, 36);
```

Gouraud Shading

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

out vec3 LightingColor;
out vec3 FragPos;
out vec3 Normal;

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

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

uniform float ambientStrength;
uniform float diffuseStrength;
uniform float specularStrength;
uniform float shininess;

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

    gl_Position = projection * view * vec4(FragPos, 1.0);
    //环境光照
    vec3 ambient = ambientStrength * lightColor;
    //漫反射光照
    vec3 norm = normalize(Normal);
    vec3 lightDir = normalize(lightPos - FragPos);
    float diff = max(dot(norm, lightDir), 0.0);
    vec3 diffuse = diffuseStrength * diff * lightColor;
    //镜面光照
    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;
    LightingColor = ambient + diffuse + specular;
}
```

Phong Shading

```
#version 330 core
out vec4 FragColor;

in vec3 Normal;
in vec3 FragPos;

uniform float ambientStrength;
uniform float diffuseStrength;
uniform float specularStrength;
uniform float shininess;

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

void main() {
    //环境光照
    vec3 ambient = ambientStrength * lightColor;
    //漫反射光照
    vec3 norm = normalize(Normal);
    vec3 lightDir = normalize(lightPos - FragPos);
    float diff = max(dot(norm, lightDir), 0.0);
    vec3 diffuse = diffuseStrength * diff * lightColor;
    //镜面光照
    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);
}
```

```
#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 projection;

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