计算机图形学作业报告

标签： 廖蕾 16340135

Basic

- 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

答：

- cube的绘制方法和之前用过的一样，就是定义顶点和定义顶点的颜色值
  然后使用VAO、VBO去渲染图形。这里就不赘述了。
- Phong Shading的实现：
  Phong模型中主要的计算公式如下：

$$I = K_a I_a + \sum_{i=1}^{m} I_i (K_d(\vec{n}, \vec{l}) + k_s(\vec{v}, \vec{r})^n)$$

  可以看到这里是有环境光、漫反射光和镜面反射光组成。
  环境光照，使用很小的一个常量光照颜色，添加到物体片段的最终颜色中，这样场景中即使没有直接光源，也能看起来存在有一些发散的光。
  漫反射光照的计算需要定义一个法向量和一个定向光照。在计算的时候，还需要知道光源的位置向量和片段位置向量。计算的步骤是需要先计算光源和片段位置之间的方向向量，然后计算光源对当前片段实际的漫发射影响。结果值再乘以光的颜色，得到漫反射分量。
  环境光照：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。
  这样实现的fShader如下：
  这里的主要的实现想法是，通过传入所需要的一些变量，计算需要的环境光照、漫反射和镜面光照。
  在下面的代码中，计算环境光照主要是用了一个常量，计算漫反射和镜面反射光照效果的时候，需要先用光线向量和观察者向量，然后进行正则化，在计算得到漫反射和镜面反射的向量之后，再乘以环境光照颜色，最后将三种光照效果得到的值相加。

```
#version 330 core
in vec3 Normal;
in vec3 FragPos;
out vec4 FragColor;

uniform vec3 objectColor;
uniform vec3 lightColor;
uniform vec3 lightPos;
uniform vec3 viewPos;
uniform float ambientStrength;
uniform float specularStrength;
uniform float concentrate;
uniform float diffuseMutiple;

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

  // vector calculatation
  vec3 norm = normalize(Normal);
  vec3 lightDirection = normalize(lightPos - FragPos);
  vec3 viewDirection = normalize(viewPos - FragPos);
  vec3 reflectDirection = reflect(-lightDirection, norm);

  // diffuse effect
  float diff = max(dot(norm, lightDirection), 0.0);
  vec3 diffuse = diff * lightColor;

  // specular effect
  float spec = pow(max(dot(viewDirection, reflectDirection), 0.0), concentrate);
  vec3 specular = specularStrength * spec * lightColor;

  // combining all factors & output
  vec3 result  = (ambient + diffuseMutiple * diffuse + specular) * objectColor;
  FragColor = vec4(result, 1.0);
}
```

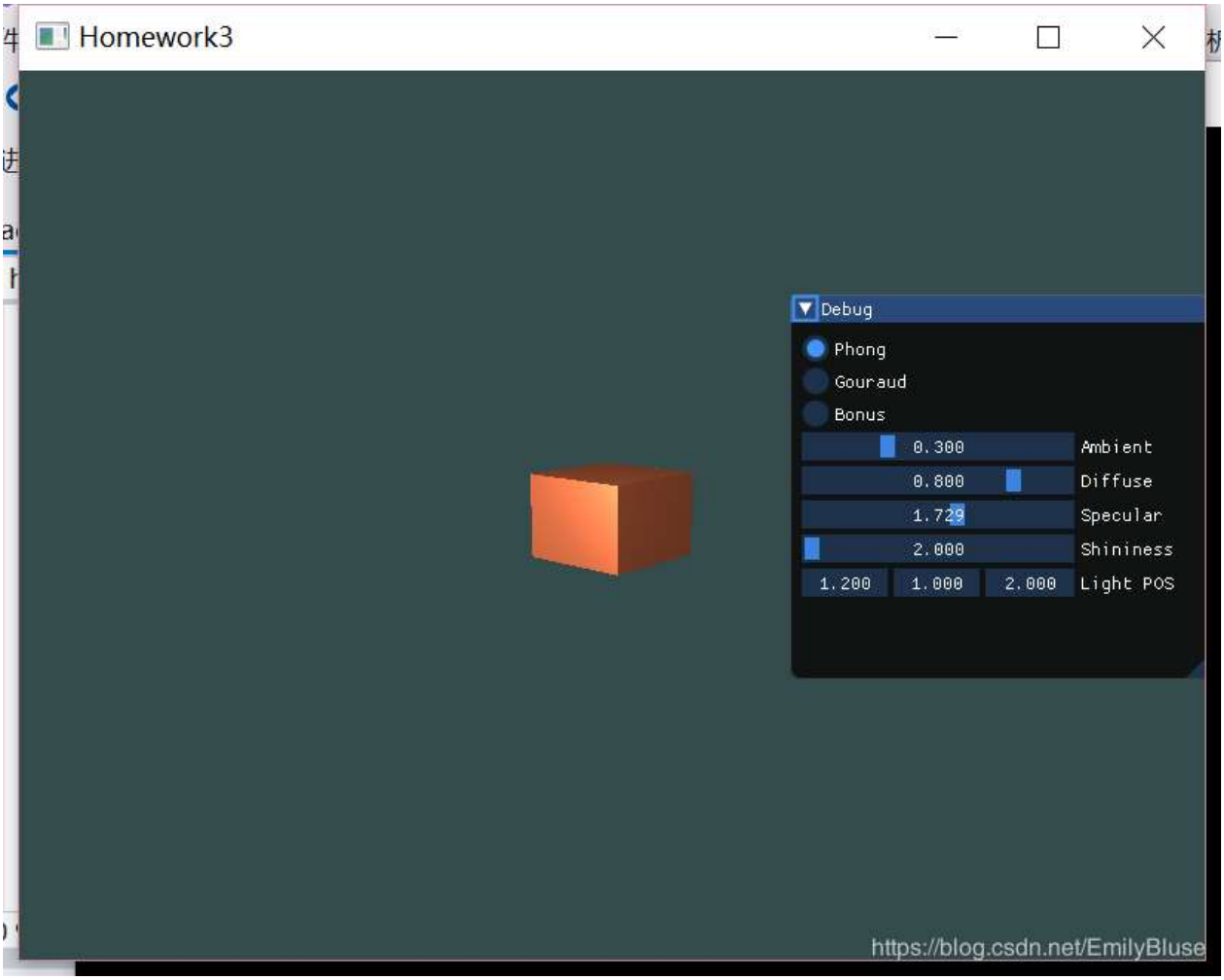
其他的和现在是一样的。
在渲染的循环中，会用到四个变量，这四个变量对应了控制光照的一些参数：

```
float ambientStrength = 0.3;
float specularStrength = 0.9;
float concentrate = 32;
float diffuseMutiple = 0.8;
```

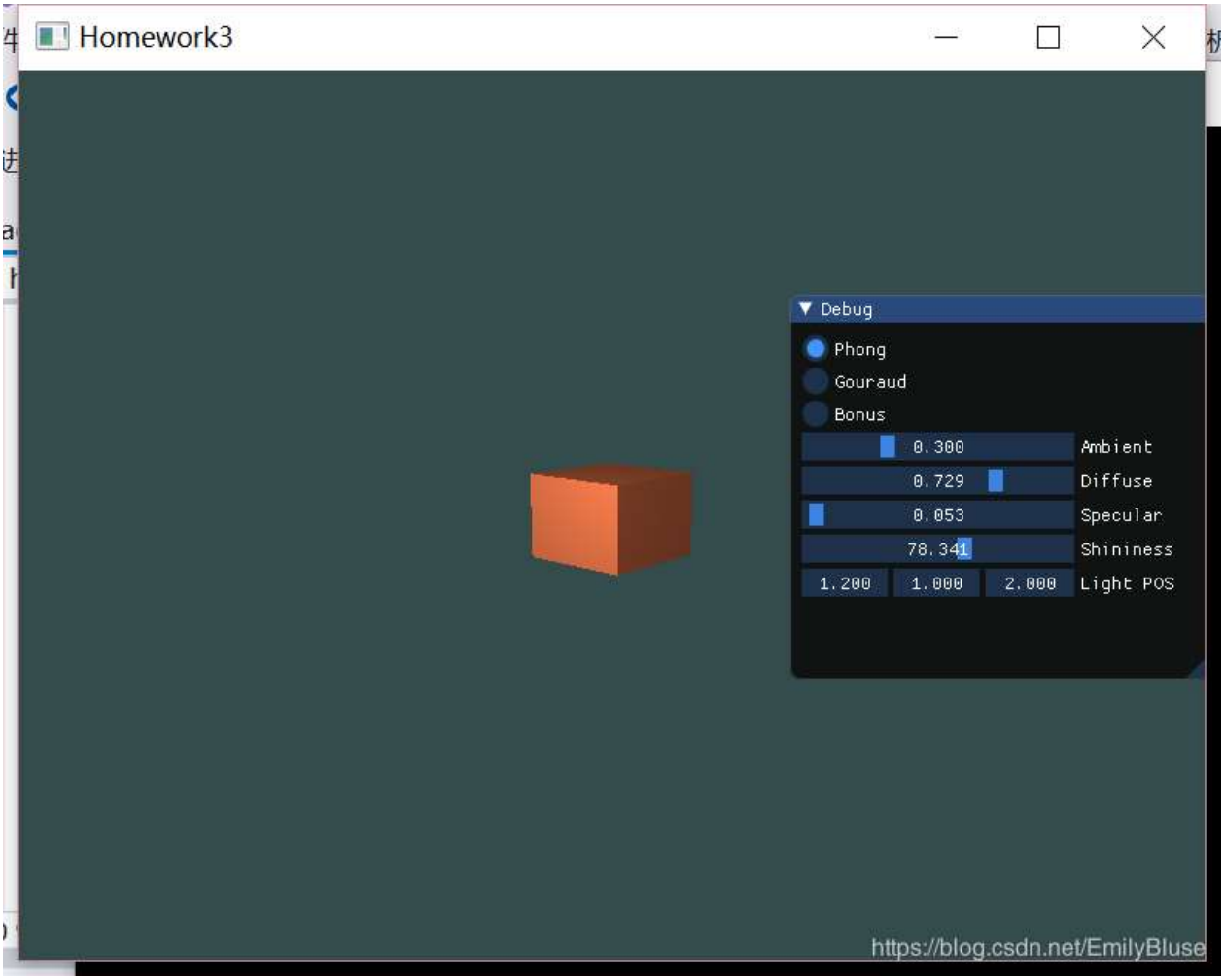
然后通过shader去设置他们的值：

```
shader.setFloat("ambientStrength", ambientStrength);
shader.setFloat("diffuseMutiple", diffuseMutiple);
shader.setFloat("specularStrength", specularStrength);
shader.setFloat("concentrate", concentrate);
```

这样一个Phong模型就写好了。
得到效果如下：
带高光效果：


Homework3
Debug
Phong
Gouraud
Bonus
0.300
Ambient
0.800
Diffuse
1.729
Specular
2.000
Shininess
1.200
1.000
2.000
Light POS
https://blog.csdn.net/EmilyBluse

漫反射效果：



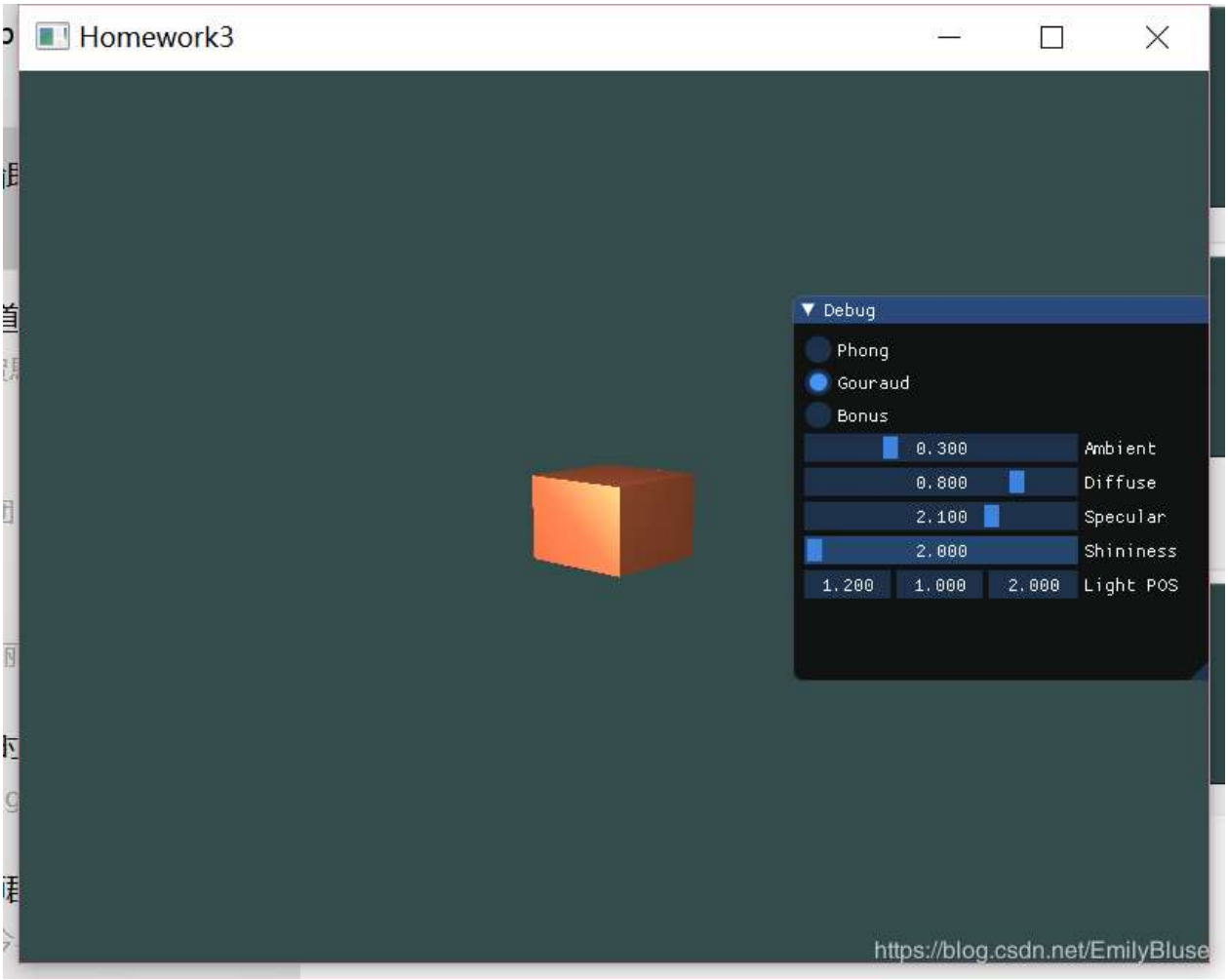
- Gouraud Shading：
  和Phong模型类似，我们可以用需要的变量，然后定义对应的vShader和fShader。Gouraud的基本思想是：已知三角形的顶点在屏幕上的坐标和对应的顶点的颜色，那么在计算的时候，根据每个像素在三角形内部的位置，对颜色进行插值。
  实现的vShader如下：

```
#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 lightColor;
uniform vec3 lightPos;
uniform vec3 viewPos;

uniform float ambientStrength;
uniform float specularStrength;
uniform float concentrate;

out vec3 result;

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

  // vector calculation
  vec3 norm = normalize(Normal);
  vec3 lightDirection = normalize(lightPos - FragPos);
  vec3 viewDirection = normalize(viewPos - FragPos);
  vec3 reflectDirection = reflect(-lightDirection, norm);

  // ambent effect
  vec3 ambient = ambientStrength * lightColor;


  // diffuse effect
  float diff = max(dot(norm, lightDirection), 0.0);
  vec3 diffuse = diff * lightColor;

  // specular effect
  float spec = pow(max(dot(viewDirection, reflectDirection), 0.0), concentrate);
  vec3 specular = specularStrength * spec * lightColor;

  // combine and output to fragment shader
  result = ambient + diffuse + specular;
}
```

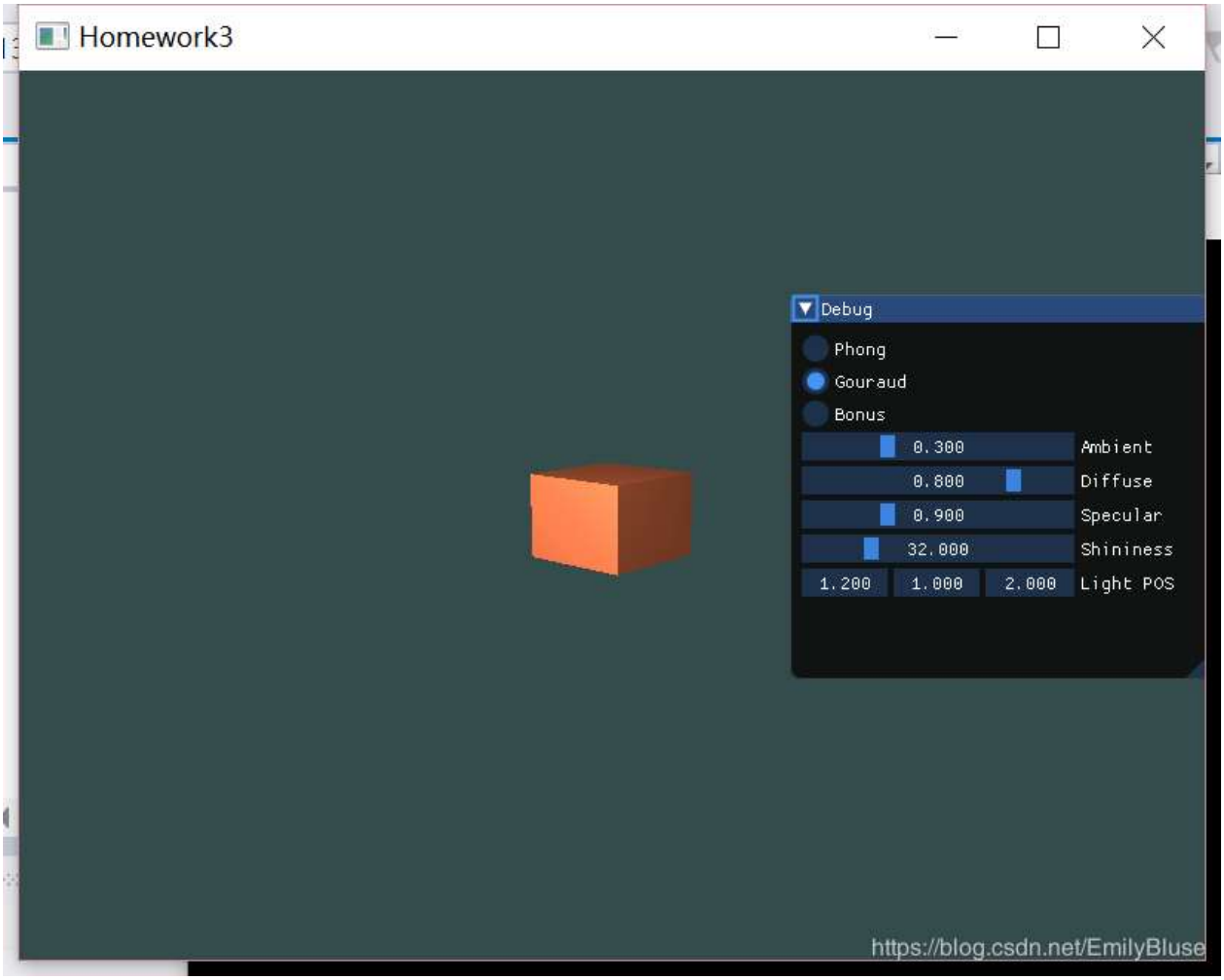
实现的fShader如下：

```
#version 330 core
in vec3 result;
out vec4 FragColor;

uniform vec3 objectColor;

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

可以看出在Gouraud Shading中主要用于计算的是vShader，在fShader中则只是将结果乘物体颜色，然后得到齐次坐标。
实现的效果如下：
带高光效果：



漫反射效果：



- 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
    使用GUI的方法和之前一样，在渲染过程中使用GUI的代码如下：

```
int radioMark = 0;

glm::vec3 lightPosition = glm::vec3(1.2f, 1.0f, 2.0f);
glm::vec3 viewPosition = glm::vec3(6.0f, 2.0f, 5.0f);
float ambientStrength = 0.3;
float specularStrength = 0.9;
float concentrate = 32;
float diffuseMutiple = 0.8;

while (!glfwWindowShouldClose(window))
{
    //...

        // render
        // ------
        ImGui_ImplGlfwGL3_NewFrame();
        glClearColor(0.2f, 0.3f, 0.3f, 1.0f);
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT); // also clear the depth buffer now

        {
                ImGui::RadioButton("Phong", &radioMark, 0);
                ImGui::RadioButton("Gouraud", &radioMark, 1);
                ImGui::RadioButton("Bonus", &radioMark, 2);
                ImGui::SliderFloat("Ambient", &ambientStrength, 0.0f, 1.0f);
                ImGui::SliderFloat("Diffuse", &diffuseMutiple, 0.0f, 1.0f);
                ImGui::SliderFloat("Specular", &specularStrength, 0.0f, 3.0f);
                ImGui::SliderFloat("Shininess", &concentrate, 2.0f, 128.0f);
                ImGui::DragFloat3("Light POS", &lightPosition.x, 2, 0);
                //...
                //...

                if (radioMark == 0 || radioMark == 2) {
                        if (radioMark == 2) {

                                lightPosition = glm::vec3(camX, lightPosition.y, camZ);
                        }
                        //...

                }
                else if (radioMark == 1) {
                        //...
                }
                //...
        }

        // glfw: swap buffers and poll IO events (keys pressed/released, mouse moved etc.)
        // -------------------------------------------------------------------------------
        ImGui::Render();
        ImGui_ImplGlfwGL3_RenderDrawData(ImGui::GetDrawData());
        glfwSwapBuffers(window);
        glfwPollEvents();
}
```

实现的结果如上面所示，这里就不放图了。

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。

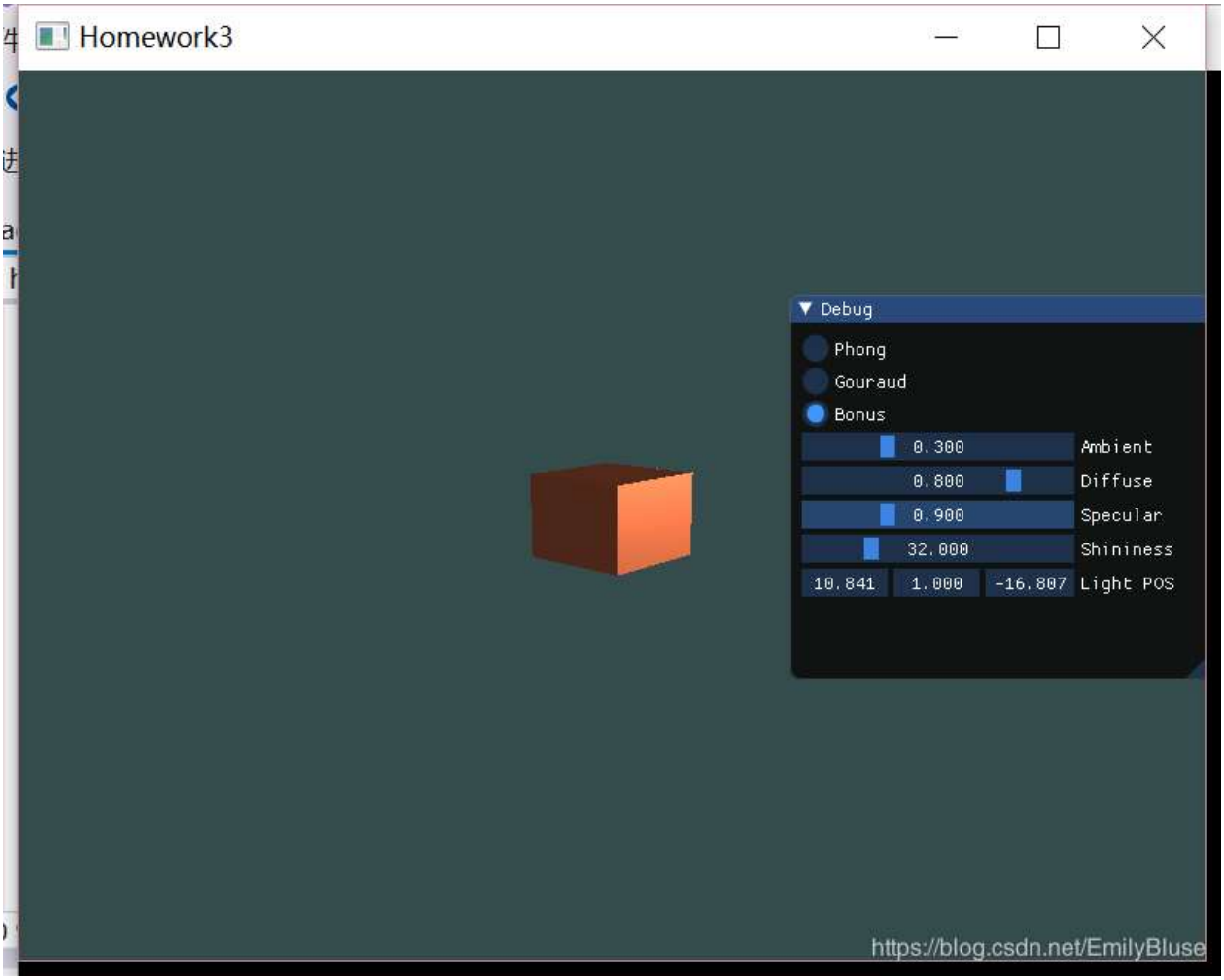
答：

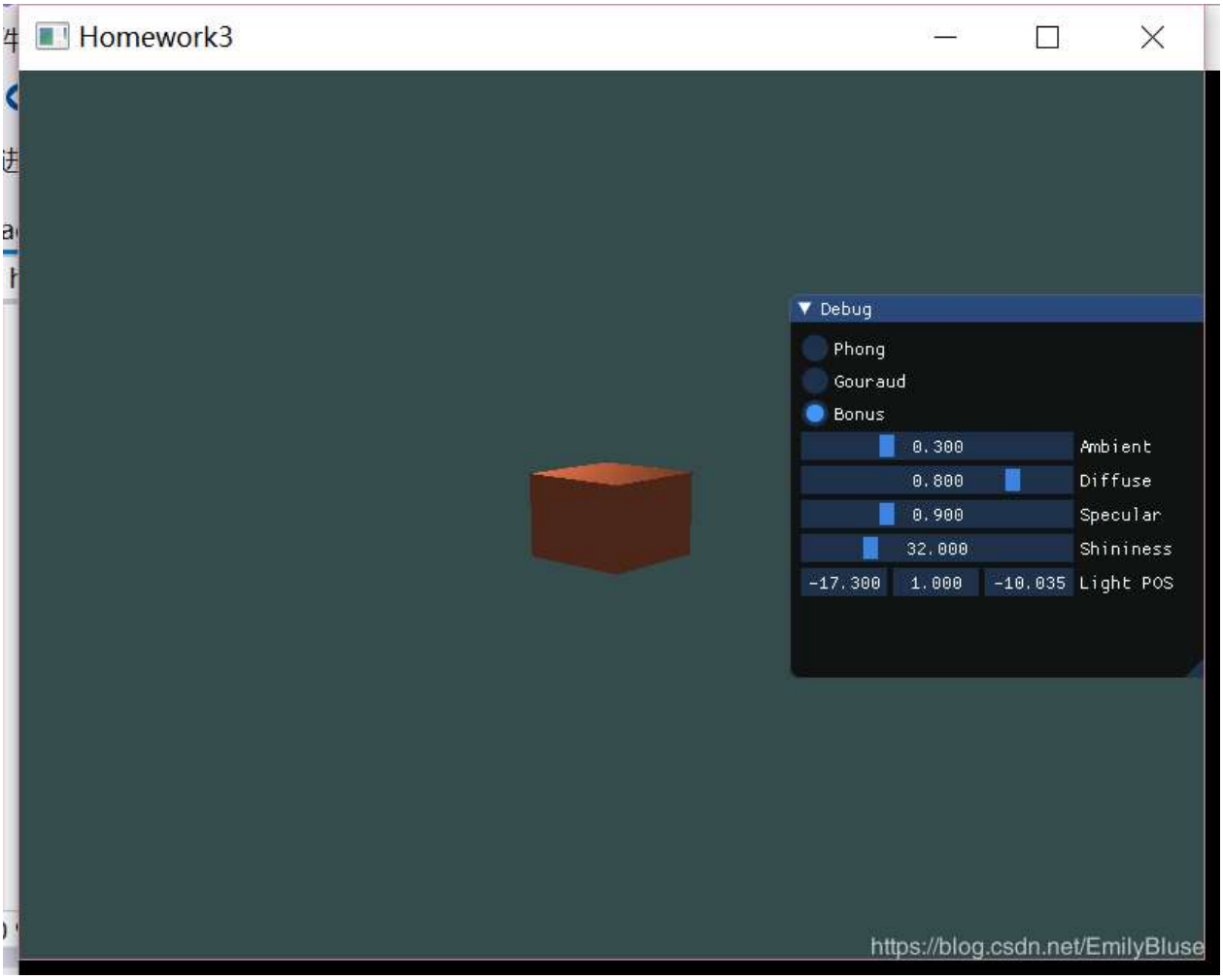
这里用到的就是利用时间在每次渲染的时候，用时间去改变光照的位置，然后其他的渲染就和之前一样。这里我选择在Phong模型中去改变光照位置。

主要实现的想法就是：计算改变光照的三个坐标位置，通过时间去计算。代码如下所示：

```
float radius = 20.0f;
float camX = sin(glfwGetTime()) * radius;
float camZ = cos(glfwGetTime()) * radius;

if (radioMark == 0 || radioMark == 2) {
        if (radioMark == 2) {
            //改变光照位置
                lightPosition = glm::vec3(camX, lightPosition.y, camZ);
        }
        shader.use();
        shader.setVec3("objectColor", 1.0f, 0.5f, 0.31f);
        shader.setVec3("lightColor", 1.0f, 1.0f, 1.0f);
        model = glm::rotate(model, 30.0f, glm::vec3(0, 1, 0));
        shader.setVec3("lightPos", lightPosition.x, lightPosition.y, lightPosition.z);
        shader.setVec3("viewPos", viewPosition.x, viewPosition.y, viewPosition.z);
        shader.setFloat("ambientStrength", ambientStrength);
        shader.setFloat("diffuseMutiple", diffuseMutiple);
        shader.setFloat("specularStrength", specularStrength);
        shader.setFloat("concentrate", concentrate);
        shader.setMat4("model", model);
        shader.setMat4("view", view);
        shader.setMat4("projection", projection);
}
```

实现的效果请看展示视频，这里放几张改变光照的效果图：


Homework3
Debug
Phong
Gouraud
Bonus
0.300
Ambient
0.800
Diffuse
0.900
Specular
32.000
Shininess
10.841
1.000
-16.807
Light POS
https://blog.csdn.net/EmilyBluse


Homework3
Debug
Phong
Gouraud
Bonus
0.300
Ambient
0.800
Diffuse
0.900
Specular
32.000
Shininess
-17.300
1.000
-10.035
Light POS
https://blog.csdn.net/EmilyBluse