Homework 6 - Lights and Shading

Basic:

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

实现原理：

Phong Shading：

一种经验模型，综合了环境光、漫反射及镜面反射。根据顶点的法向量插值计算出表面内各点的法向量，再根据光照模型逐像素计算光照值。

$$I = K_a I_a + K_d I_e \cos\alpha + k_s I_e \cos^n \gamma$$

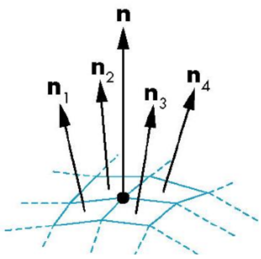
- Involves the following steps
    1. Normals are computed at the vertex as the average of the normals of all the faces meeting at that vertex
    2. For each polygon
       the value of the normal for the surface occupied by each interior pixel is calculated by linear interpolation of the normals at the vertices

Gouraud Shading：

根据顶点法向量计算出光照，再插值计算出整个面的光照；效果比 Flat shading 好，尤其是被模拟的值本就是线性的时候；但由于内插值总小于顶点的最大值，故得到的高光部分只能在顶点出现，在高光面中甚至可以分辨出各个小的面元。

- Involves the following steps
  - Normals are computed at the vertex as the average of the normals of all the faces meeting at that vertex
  - Intensity at each vertex is calculated using the normal and an illumination model
  - For each polygon the intensity values for the interior pixels are calculated by linear interpolation of the intensities at the vertices

$$n = \frac{n_1 + n_2 + n_3 + n_4}{|n_1 + n_2 + n_3 + n_4|}$$

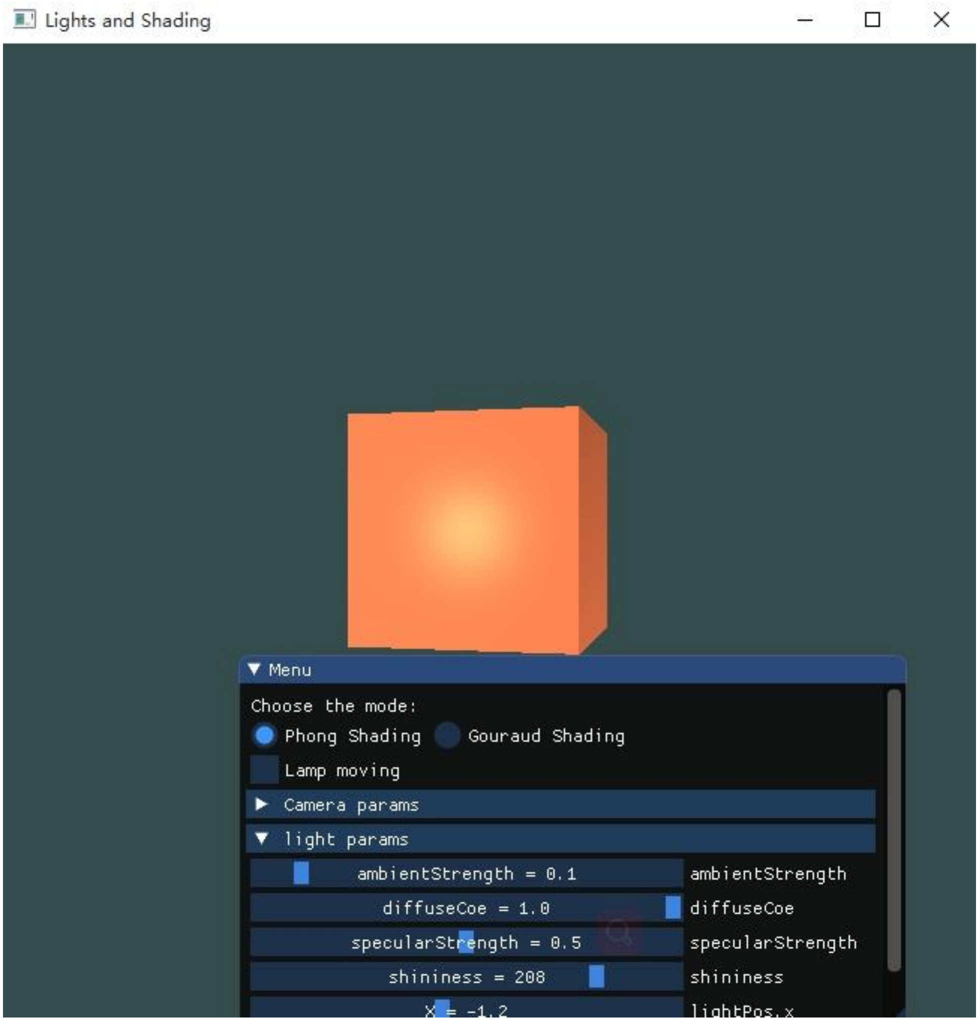


Notice：

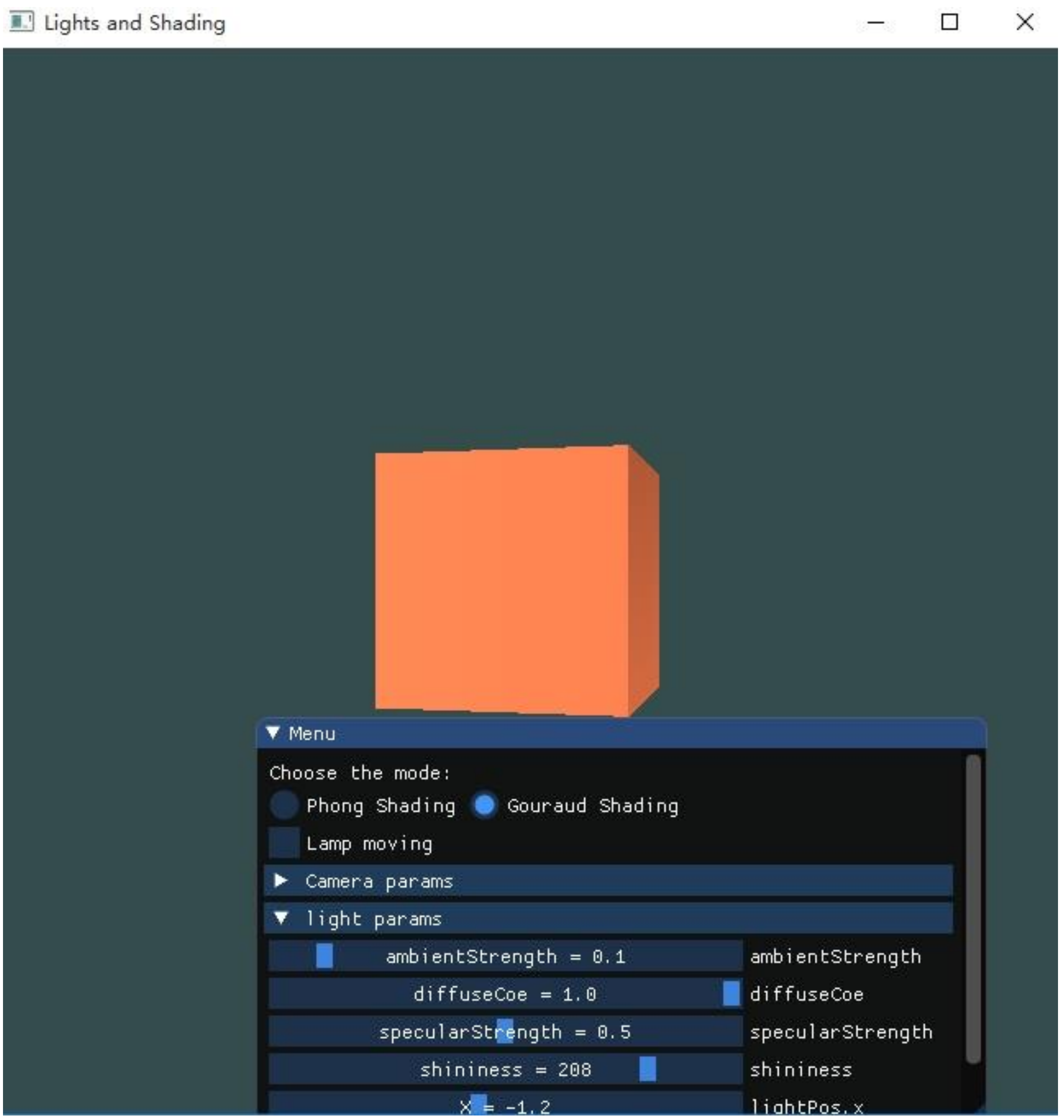
通过高光情况下使用 Phong shading，镜面反射弱的情况下使用 Gouraud shading，可以既保证速度，又保证质量。

实现效果：

Phong Shading：



Gouraud Shading：



可以看到，相同参数下，Phong Shading 的平面内有镜面反射的光亮区域，而 Gouraud Shading 则没有。这是由于 Gouraud Shading 平面内像素点的亮度是通过顶点插值而来的，若光亮部分不是落在顶点，则会损失这部分光亮区域，而 Phong Shading 则不会有这个问题。

实现细节：

Cube 的渲染和以前的作业基本一样。不过，对于每个顶点，将颜色属性改成了顶点的法向量坐标作为顶点数据传入。

对于 Gouraud Shading、Phong Shading 以及光源 Lamp，为它们设置不同的着色器。

```
//shader
Shader phong_shader("Phong.vs", "Phong.fs");
Shader gouraud_shader("Gouraud.vs", "Gouraud.fs");
Shader lamp_shader("Lamp.vs", "Lamp.fs");
```

Phong Shading 着色器：

在顶点着色器中计算好片段的位置和法向量 Normal

```
#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 FragPos;
out vec3 Normal;

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

在片段着色器中使用 Phong 光照模型来计算光线分量。在世界坐标下计算环境光、漫反射、镜面反射的光亮，然后进行综合。

```
in vec3 Normal;
in vec3 FragPos;
out vec4 FragColor;

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

uniform float ambientStrength;
uniform float specularStrength;
uniform int shininess;
uniform float diffuseCoe;

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 = diff * lightColor * diffuseCoe;
    //specular
    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);
}
```

Gouraud Shading 着色器：

Gouraud Shading 与 Phong Shading 的不同点的是 Gouraud Shading 在顶点着色器中计算光模型，在片段着色器中进行插值计算来得到立方体的颜色。则在实现中，只需要把 Phong Shading 片段着色器中的计算光照的逻辑迁移到 Gouraud Shading 顶点着色器中即可。

顶点着色器：

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

out vec3 result;

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

uniform float ambientStrength;
uniform float specularStrength;
uniform int shininess;

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 = diff * lightColor;

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

    result = (ambient + diffuse + specular) * objectColor;
}
```

片段着色器：

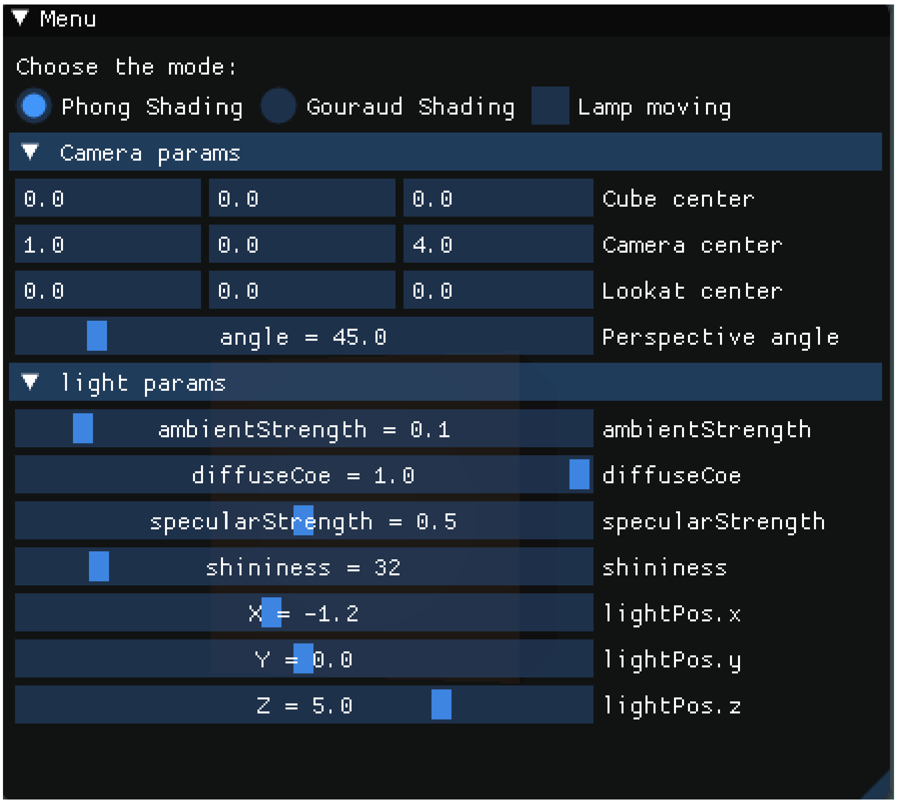
```
in vec3 result;
out vec4 FragColor;

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

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

实现结果：



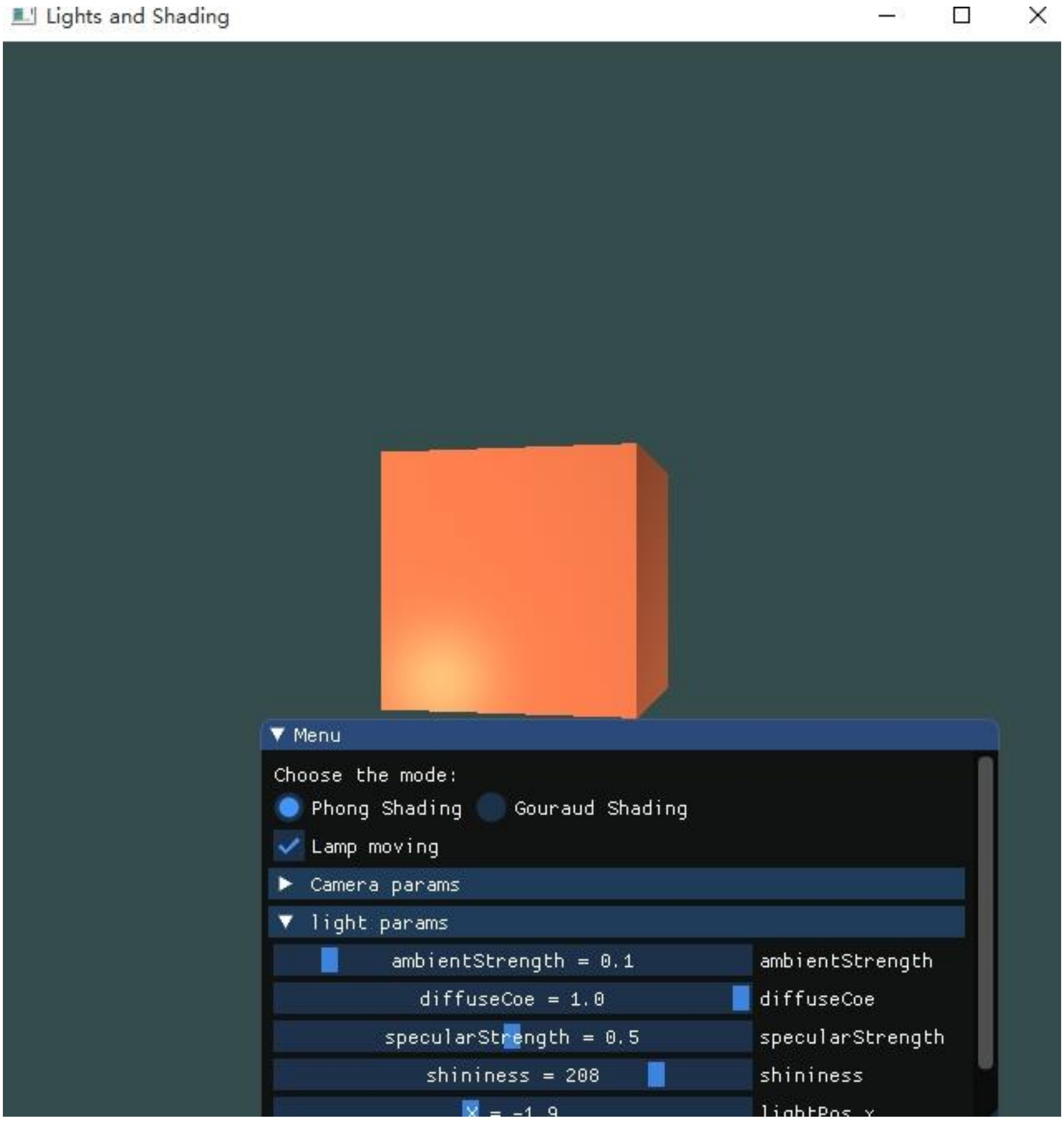
具体实现：

```
//设置imgui
ImGui_ImplOpenGL3_NewFrame();
ImGui_ImplGlfw_NewFrame();
ImGui::NewFrame();
//设置菜单样式
{
    ImGui::Begin("Menu");
    ImGui::Text("Choose the mode:");
    ImGui::RadioButton("Phong Shading", &mode, 0);
    ImGui::SameLine();
    ImGui::RadioButton("Gouraud Shading", &mode, 1);
    ImGui::SameLine();
    ImGui::Checkbox("Lamp moving", &is_moving_lamp);

    if (ImGui::CollapsingHeader("Camera params")) {
        ImGui::InputFloat3("Cube center", cubePosition, 1);
        ImGui::InputFloat3("Camera center", cameraPos, 1);
        ImGui::InputFloat3("Lookat center", lookAtCenter, 1);
        ImGui::SliderFloat("Perspective angle", &angle, 0.0f, 360.0f, "angle = %.1f");
    }

    if (ImGui::CollapsingHeader("light params")) {
        ImGui::SliderFloat("ambientStrength", &ambientStrength, 0.0f, 1.0f, "ambientStrength = %.1f"); //ambient因子
        ImGui::SliderFloat("diffuseCoe", &diffuseCoe, 0.0f, 1.0f, "diffuseCoe = %.1f");  //diffuse因子
        ImGui::SliderFloat("specularStrength", &specularStrength, 0.0f, 1.0f, "specularStrength = %.1f"); //specular因子
        ImGui::SliderInt("shininess", &shininess, 0, 255, "shininess = %d");            //反光度
        ImGui::SliderFloat("lightPos.x", &lightPos.x, -10.0f, 10.0f, "X = %.1f");
        ImGui::SliderFloat("lightPos.y", &lightPos.y, -10.0f, 10.0f, "Y = %.1f");
        ImGui::SliderFloat("lightPos.z", &lightPos.z, -10.0f, 10.0f, "Z = %.1f");
    }

    ImGui::End();
```

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。

实现效果：具体效果见<u>演示视频.mp4</u>



具体实现：

在每帧中使用 glfwGetTime()函数，结合 sin 函数，改变光源的位置实现光源的移动。

```
if (is_moving_lamp) {
    lightPos.x = -1.0f + sin(glfwGetTime()) * 1.0f;
    lightPos.y = sin(glfwGetTime()) * 1.0f;
}
```