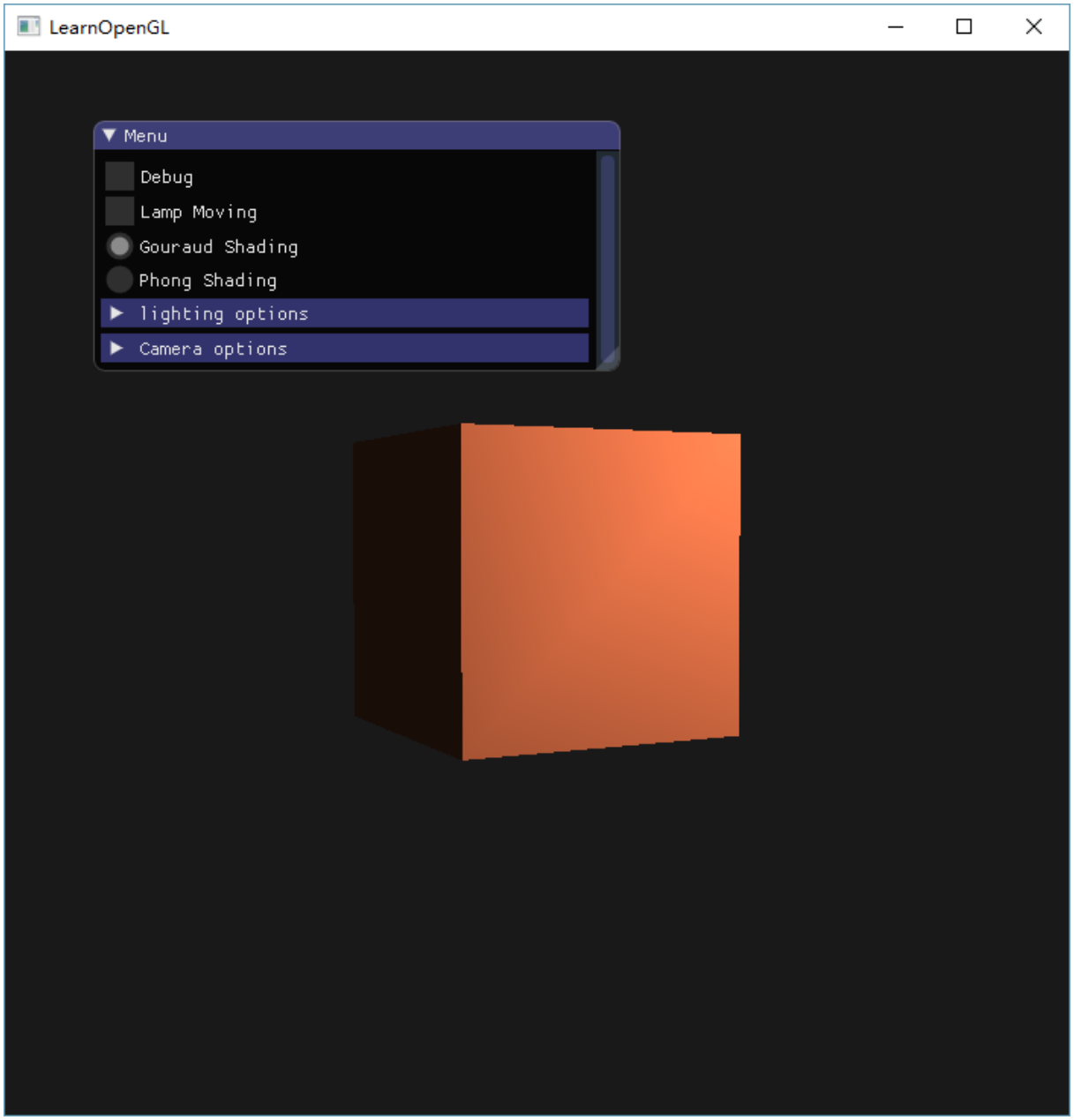
Homework 6 - Lights and Shading

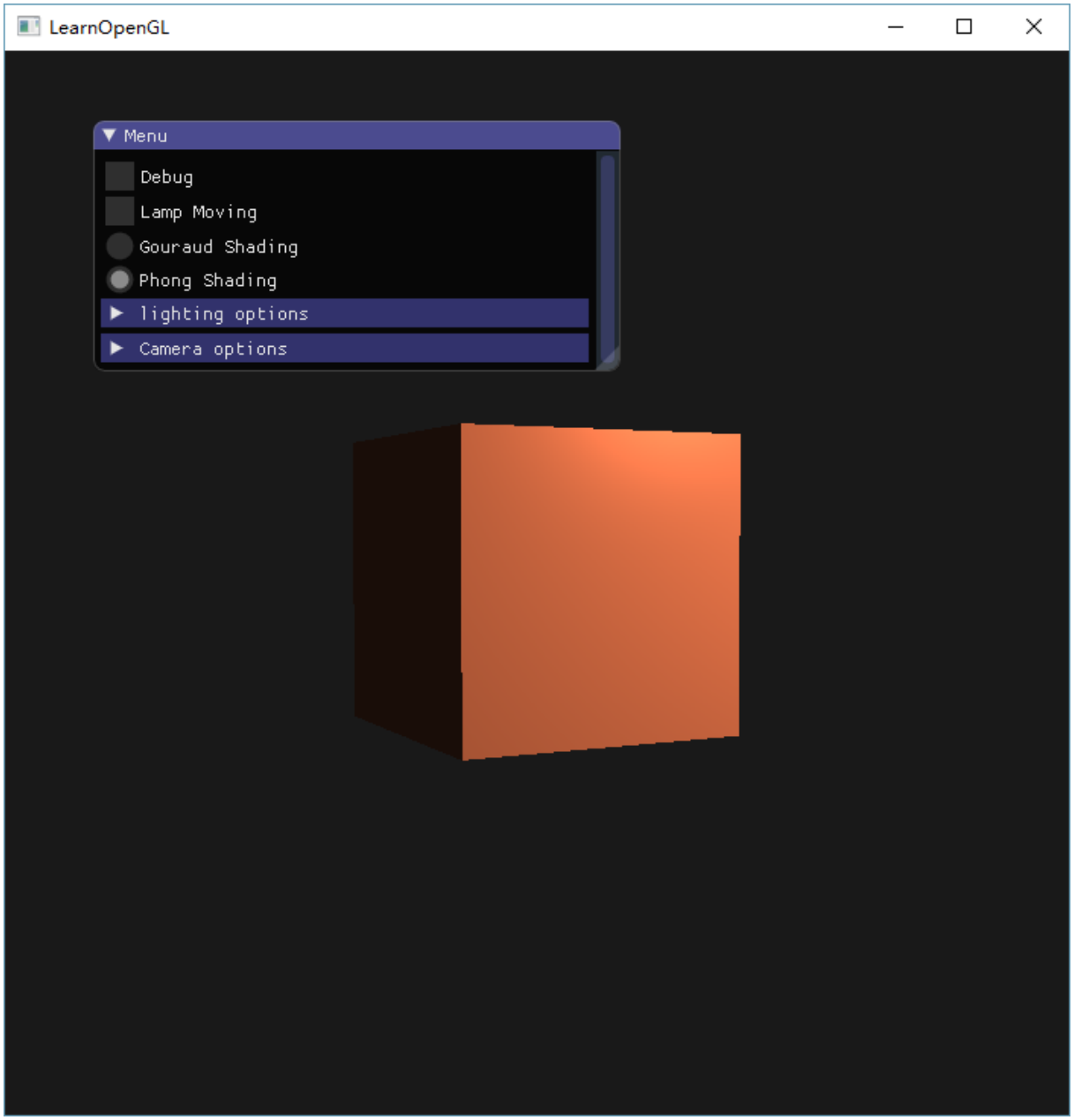
Basic

1. 实现一个Phong光照模型，并分别实现Phong 和 Gouraud Shading

实验截图



Gouraud Shading



Phong Shading

> 通过手动调整 Camera 和 Lamp 的参数可以切换不同视角和改变光线属性与位置

总体来看 Gouraud Shading 的效果可能看的不是太清晰，可以通过使用 GUI 不断切换 Phong 和 Gouraud 模式，可以对比看出 Gouraud 模式还是能隐约看到那条对角线的。

算法实现

Cube的画法和以前的作业基本一样。不过这次对于每个顶点都引入了顶点的法向量坐标作为顶点数据传入。

对于 Gouraud Shading 和 Phong Shading 以及光源 Lamp，分别为它们设置3个不同的着色器。

```
1. Shader phongShader = Shader(".\\Shader\\Phong.vs", ".\\Shader\\Phon
   g.fs");
2. Shader gouraudShader = Shader(".\\Shader\\Gouraud.vs", ".\\Shader\\G
   ouraud.fs");
```

```
3. Shader lampShader = Shader(".\\Shader\\lamp.vs", ".\\Shader\\lamp.f
   s");
```

通过GUI来选择不同的模式，并且修改相关的参数，相关参数通过 uniform 的形式传入着色器里面进行修改。

Phong Shading 着色器实现

1. 在顶点着色器中计算好 Frag 段的位置和法向量 Normal

```
1. // Vertex Shader
2. void main()
3. {
4.     gl_Position = projection * view * model * vec4(aPos, 1.0);
5.     FragPos = vec3(model * vec4(aPos, 1.0));
6.     Normal = mat3(transpose(inverse(model))) * aNormal;
7. }
```

2. 在片段着色器中使用 Phong 模型来计算光线分量。分别**在世界坐标下**计算环境光、漫反射、镜面反射的光亮再把它们给综合起来。

```
1. // Fragment Shader
2. void main()
3. {
4.     vec3 ambient = ambientStrength * lightColor;
5.
6.     vec3 norm = normalize(Normal);
7.     vec3 lightDir = normalize(lightPos - FragPos);
8.     float diff = max(dot(norm, lightDir), 0.0);
9.     vec3 diffuse = diff * lightColor;
10.
11.    vec3 viewDir = normalize(viewPos - FragPos);
12.    vec3 reflectDir = reflect(-lightDir, norm);
13.    float spec = pow(max(dot(viewDir, reflectDir), 0.0), shininess);
14.    vec3 specular = specularStrength * spec * lightColor;
15.
16.    vec3 result = (ambient + diffuse + specular) * objectColor;
17.    FragColor = vec4(result, 1.0);
18. }
```

其中，与三种光模型有关的参数如 `ambientStrength`，`specularStrength`，`shininess`， `lightPos` 等都是 uniform 类型，与主程序进行交互。

Gouraud Shading 着色器实现

Gouraud Shading 区别于 Phong Shading 的地方在于，**它是在顶点着色器中计算光模型的，然后在片段着色器中进行插值计算来得到立方体的颜色。**

那么我们只需要把 Phong Shading 片段着色器中的计算光照的逻辑迁移到顶点着色器中即可以完成 Gouraud Shading 的实现。

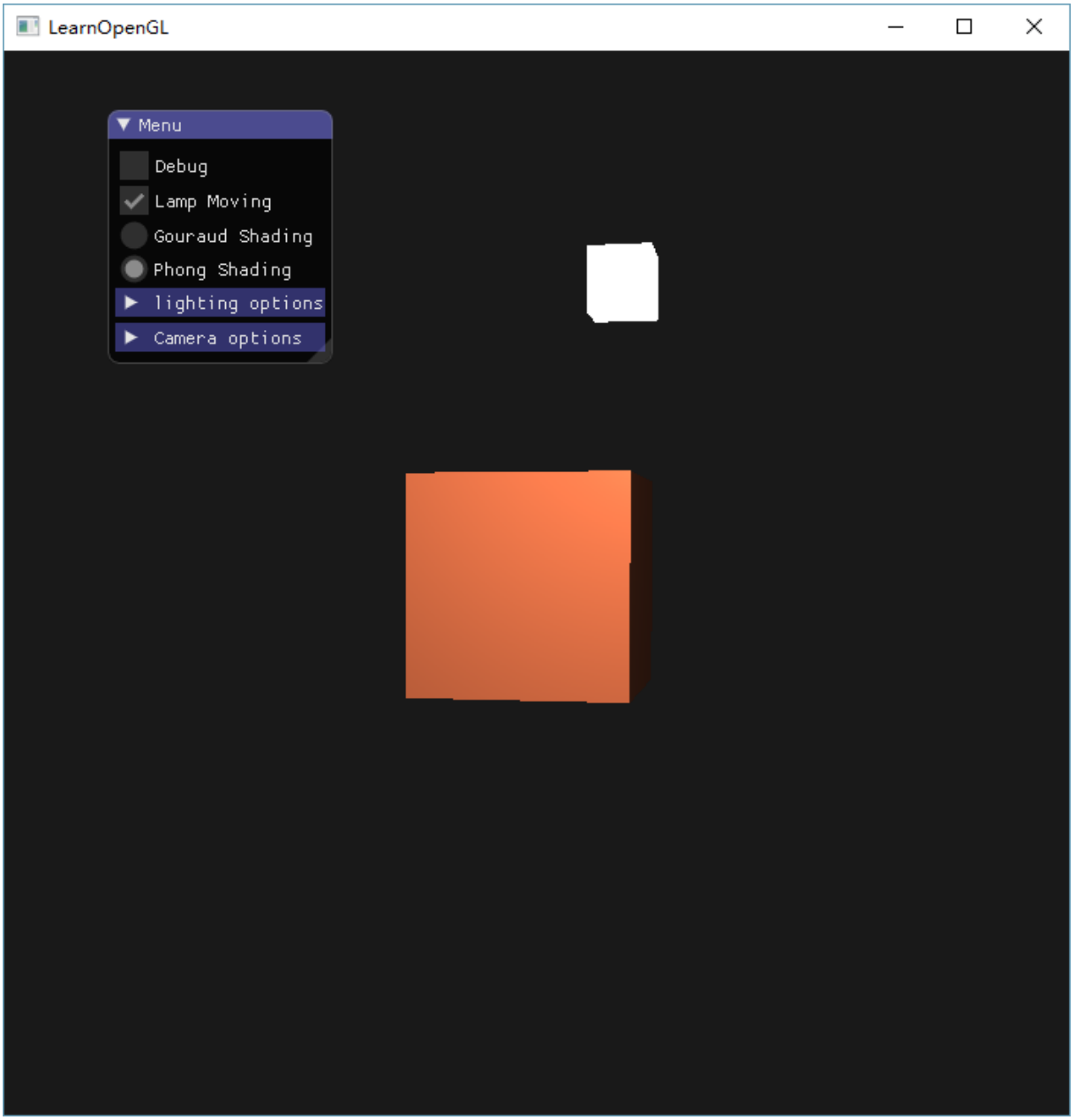
```
// Vertex Shading
void main()
{
    gl_Position = projection * view * model * vec4(aPos, 1.0);
    vec3 Position = vec3(model * vec4(aPos, 1.0));
    vec3 Normal = mat3(transpose(inverse(model))) * aNormal;

    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;
}
```

```
// Fragment Shading
#version 330 core

in vec3 result;
out vec4 FragColor;

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

Bonus

使光源移动

实验截图



> 更加清晰的实验结果可以看 gif 图

算法实现

只需要在每一帧中使用 `glfwGetTime()` 函数，结合 sin 和 cos， 就可以实现光源的移动。

```
if (is_lamp_moving) {
    lightPos.x = 0.5f + abs(sin(glfwGetTime())) * 1.0f;
    lightPos.y = abs(sin(glfwGetTime() / 2.0f)) * 1.0f;
}
```