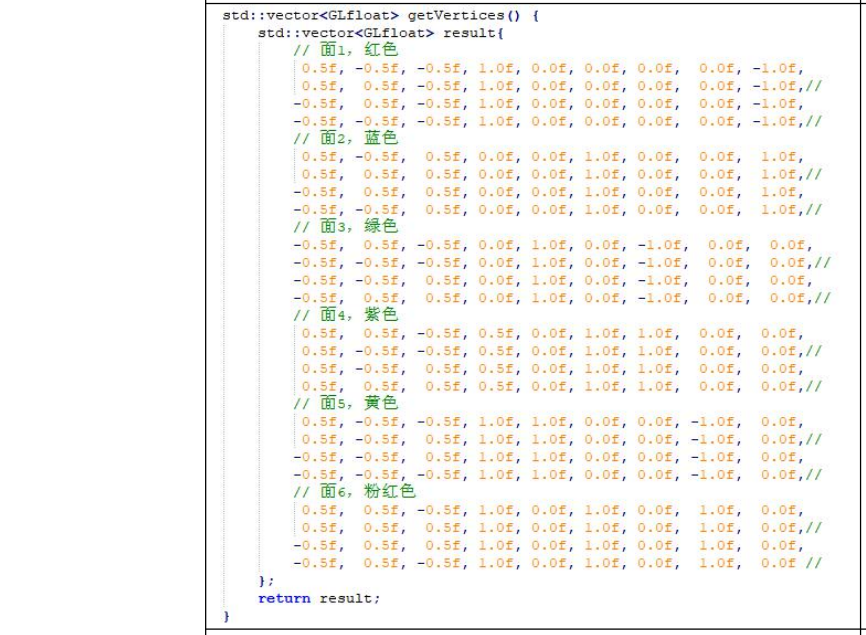
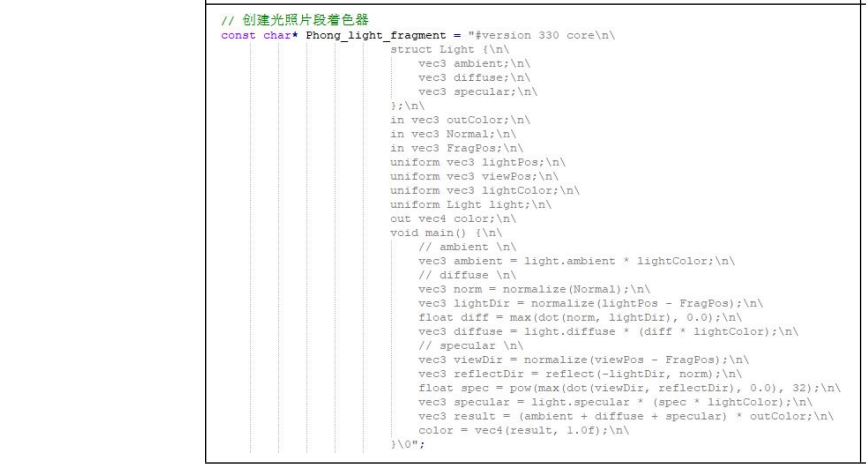
计算机图形学实验报告

第一部分：实现思路和代码分析

代码截图	代码说明
	在前面作业的基础上，为立方体的每一个面添加法向量。此时，顶点数组中的每个点使用九维向量表示：前三维表示位置（x,y,z），中间三维表示点的颜色（R,G,B），最后三维表示该点所在面的法向量（x’,y’,z’）。
	Phong 光照模型的顶点着色器 GLSL 代码。增加平面法向量的输入，把顶点位置属性乘以模型矩阵变换到世界空间坐标，同时计算平面的法线矩阵，与原法向量相乘得到新的平面法向量。
	Phong 光照模型的片段着色器 GLSL 代码。相较于以前的版本增加了 ambient 因子、diffuse 因子、specular 因子，光源位置，光的颜色以及观察者的世界空间坐标等参数，可以进行调节从而演示 Phong 光照模型的不同效果。

glm::vec3 a_vec = ambient_Factor * glm::vec3(1.0f),
d_vec = diffuse_Factor * glm::vec3(1.0f),
s_vec = specular_Factor * glm::vec3(1.0f);
 	
if (modeChange == 1 || modeChange == 3) {
// 使用着色器程序
glUseProgram(shaderProgram_Phong);
// 将变换矩阵应用到坐标中
// 位置
unsigned int transformLoc = glGetUniformLocation(shaderProgram_Phong, "transform");
unsigned int viewLoc = glGetUniformLocation(shaderProgram_Phong, "view");
unsigned int projectionLoc = glGetUniformLocation(shaderProgram_Phong, "projection");
 	
// 光照
unsigned int lightLoc = glGetUniformLocation(shaderProgram_Phong, "lightColor");
unsigned int lightViewLoc = glGetUniformLocation(shaderProgram_Phong, "viewPos");
unsigned int lightPosLoc = glGetUniformLocation(shaderProgram_Phong, "lightPos");
 	
// 光照参数
unsigned int ambient = glGetUniformLocation(shaderProgram_Phong, "light.ambient");
unsigned int diffuse = glGetUniformLocation(shaderProgram_Phong, "light.diffuse");
unsigned int specular = glGetUniformLocation(shaderProgram_Phong, "light.specular");
 	
glUniformMatrix4fv(transformLoc, 1, GL_FALSE, glm::value_ptr(transform));
glUniformMatrix4fv(viewLoc, 1, GL_FALSE, &view[0][0]);
glUniformMatrix4fv(projectionLoc, 1, GL_FALSE, &projection[0][0]);
glUniform3fv(lightLoc, 1, glm::value_ptr(lightColor));
glUniform3fv(lightViewLoc, 1, &lightView[0]);
glUniform3fv(lightPosLoc, 1, &lightPos[0]);
glUniform3fv(ambient, 1, &a_vec[0]);
glUniform3fv(diffuse, 1, &d_vec[0]);
glUniform3fv(specular, 1, &s_vec[0]);
}
	将相应的参数绑定到 Phong 方法的 Shader 上，然后进行模型渲染。

// 创建顶点着色器
const char* Gouraud_light_vertex = "#version 330 core\n\
layout(location = 0) in vec3 pos;\n\
layout(location = 1) in vec3 color;\n\
layout(location = 2) in vec3 aNormal;\n\
struct Light {\n\
vec3 ambient;\n\
vec3 diffuse;\n\
vec3 specular;\n\
};\n\
uniform mat4 projection;\n\
uniform mat4 view;\n\
uniform mat4 transform;\n\
uniform vec3 lightPos;\n\
uniform vec3 viewPos;\n\
uniform vec3 lightColor;\n\
uniform Light light;\n\
out vec3 outColor;\n\
out vec3 outLightColor;\n\
void main() {\n\
gl_Position = projection * view * transform * vec4(pos, 1.0);\n\
outColor = color;\n\
vec3 Position = vec3(transform * vec4(pos, 1.0));\n\
vec3 Normal = mat3(transpose(inverse(transform))) * aNormal;\n\
// ambient \n\
vec3 ambient = light.ambient * lightColor; \n\
// diffuse \n\
vec3 norm = normalize(Normal);\n\
vec3 lightDir = normalize(lightPos - Position); \n\
float diff = max(dot(norm, lightDir), 0.0); \n\
vec3 diffuse = light.diffuse * (diff * lightColor); \n\
// specular \n\
vec3 viewDir = normalize(viewPos - Position);\n\
vec3 reflectDir = reflect(-lightDir, norm);\n\
float spec = pow(max(dot(viewDir, reflectDir), 0.0), 32);\n\
vec3 specular = light.specular * (spec * lightColor);\n\
outLightColor = ambient + diffuse + specular; \n\
}\0";
	Gouraud 光照模型的顶点着色器 GLSL 代码。增加平面法向量的输入，把顶点位置属性乘以模型矩阵变换到世界空间坐标，同时计算平面的法线矩阵，与原法向量相乘得到新的平面法向量。Gouraud 光照模型与 Phong 模型最大的不同在于，顶点着色器增加了 ambient 因子、diffuse 因子、specular 因子，光源位置，光的颜色以及观察者的世界空间坐标等参数，也就是说，模型中顶点的颜色 RGB 比例是在此阶段就已经被计算出来了。

// 创建片段着色器 const char* Gouraud_light_fragment = "#version 330 core\n\ in vec3 outColor;\n\ in vec3 outLightColor;\n\ out vec4 color;\n\ void main() {\n\ color = vec4(outLightColor * outColor, 1.0f);\n\ }\0";	Gouraud 光照模型的片段着色器 GLSL 代码。这里把顶点着色器算出的色彩比例与真实颜色值相乘得到各顶点的光照值。

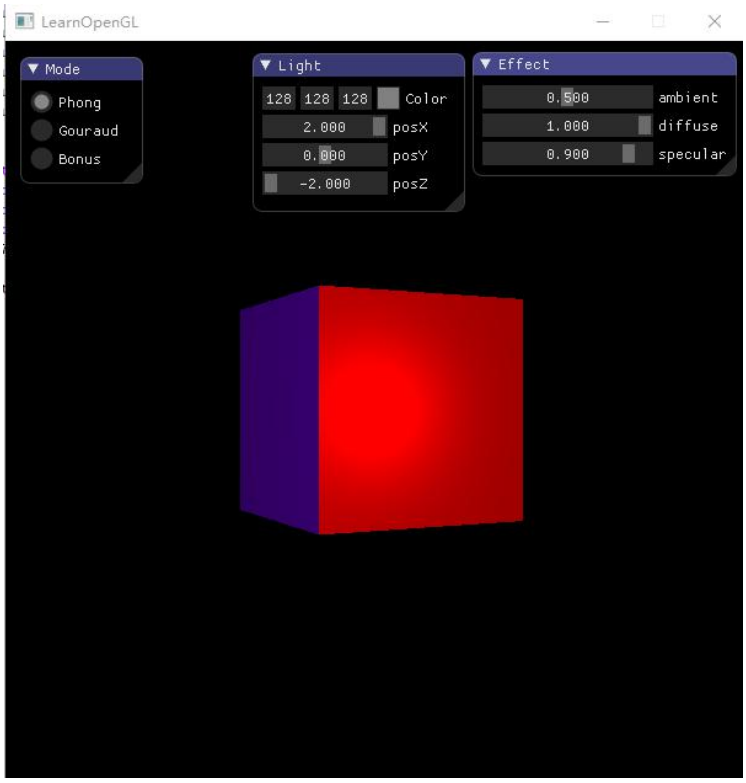
else {
// 使用着色器程序
glUseProgram(shaderProgram_Gouraud);
// 将变换矩阵应用到坐标中
// 位置
unsigned int transformLoc = glGetUniformLocation(shaderProgram_Gouraud, "transform");
unsigned int viewLoc = glGetUniformLocation(shaderProgram_Gouraud, "view");
unsigned int projectionLoc = glGetUniformLocation(shaderProgram_Gouraud, "projection");
 	
// 光照
unsigned int lightLoc = glGetUniformLocation(shaderProgram_Gouraud, "lightColor");
unsigned int lightViewLoc = glGetUniformLocation(shaderProgram_Gouraud, "viewPos");
unsigned int lightPosLoc = glGetUniformLocation(shaderProgram_Gouraud, "lightPos");
 	
// 光照参数
unsigned int ambient = glGetUniformLocation(shaderProgram_Gouraud, "light.ambient");
unsigned int diffuse = glGetUniformLocation(shaderProgram_Gouraud, "light.diffuse");
unsigned int specular = glGetUniformLocation(shaderProgram_Gouraud, "light.specular");
 	
glUniformMatrix4fv(transformLoc, 1, GL_FALSE, glm::value_ptr(transform));
glUniformMatrix4fv(viewLoc, 1, GL_FALSE, &view[0][0]);
glUniformMatrix4fv(projectionLoc, 1, GL_FALSE, &projection[0][0]);
glUniform3fv(lightLoc, 1, glm::value_ptr(lightColor));
glUniform3fv(lightViewLoc, 1, &lightView[0]);
glUniform3fv(lightPosLoc, 1, &lightPos[0]);
glUniform3fv(ambient, 1, &a_vec[0]);
glUniform3fv(diffuse, 1, &d_vec[0]);
glUniform3fv(specular, 1, &s_vec[0]);
}
	将相应的参数绑定到 Gouraud 方法的 Shader 上，然后进行模型渲染。

if (modeChange == 3) {
lightPos.x = sin(glfwGetTime()) * 3.0f;
lightPos.z = cos(glfwGetTime()) * 3.0f;
view = glm::lookAt(glm::vec3(3.0f, 3.0f, -3.0f),
glm::vec3(0.0f, 0.0f, 0.0f),
glm::vec3(0.0f, 1.0f, 0.0f));
}
else {
transform = glm::rotate(transform, (float)objectAngle, glm::vec3(0.0f, 1.0f, 0.0f));
view = glm::lookAt(glm::vec3(3.0f, 0.0f, -3.0f),
glm::vec3(0.0f, 0.0f, 0.0f),
glm::vec3(0.0f, 1.0f, 0.0f));
}
	设置一般场景和 Bonus 场景中光源的位置。在一般场景下，光源位置手动调节；在 Bonus 场景下，光源位置作圆周运动。

第二部分：实验截图分析

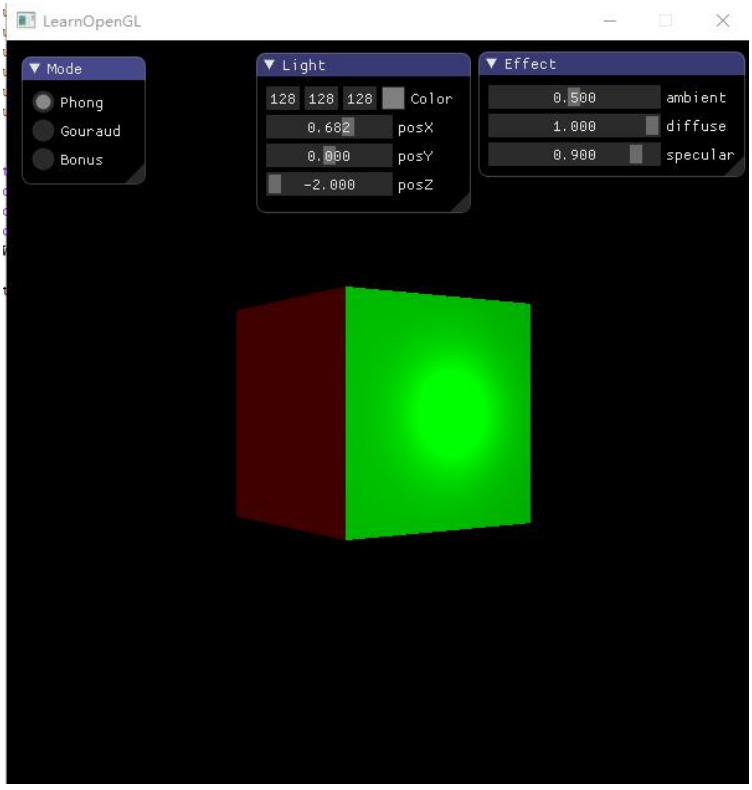
1. 实现 Phong 光照模型：

（1）场景中绘制一个 cube

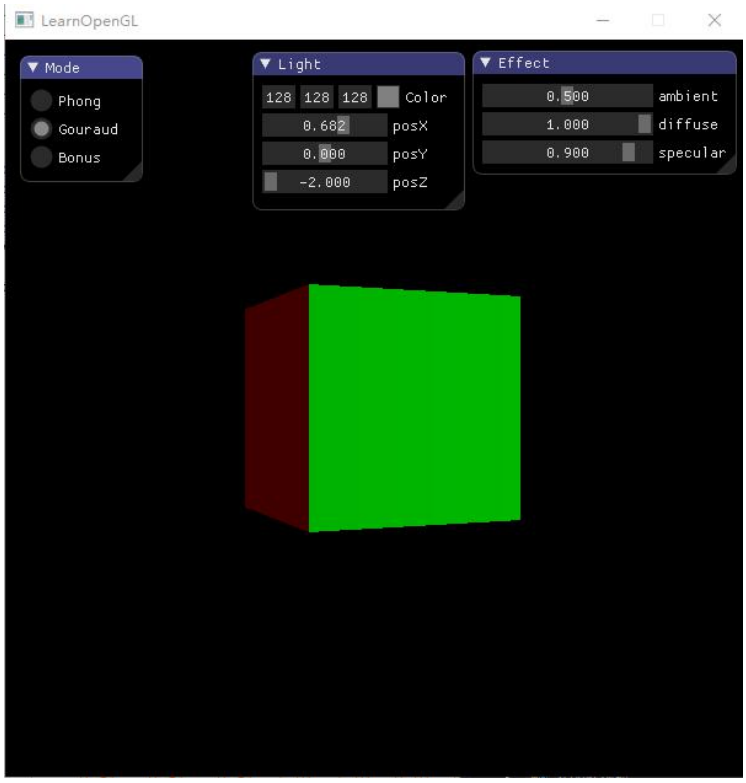


（2）自己写 shader 实现两种 shading: Phong Shading 和 Gouraud Shading，并解释两种 shading 的实现原理

Phong Shading：



Gouraud Shading：



实现原理：

Phong Shading：冯氏光照模型在片段着色器中实现。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对应的光照值。

Gouraud Shading：冯氏光照模型在顶点着色器中实现。根据顶点的法向量和光照模型计算出各顶点处的光照值，再通过插值计算出整个平面上其他像素所对应的光照值。

2. 使用 GUI，使参数可调节，效果实时更改：

（1）GUI 里可以切换两种 shading

（2）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

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