计算机图形学 实验报告 Lab4

作者：陈靖辉 时间：2024.4.25

实验要求

代码、程序界面、报告都很专业 (bonus)

上机任务

1. 修改程序 7.2 以使其包括两个位于不同位置的位置光，其中一 个是蓝光，另外一个是红光。
2. 片段着色器需要混合每个光的漫反射和镜面反射分量。可以尝 试简单地将它们加起来或者加权求和，并限制结果不超出光照 值的上限。
3. 撰写实验报告，报告中应包含完成任务的核心代码（注意不要 大段复制粘贴代码），运行结果的屏幕截图以及必要的讨论分 析。打包上传实验报告和原始代码，注意代码只需 要.h、.cpp、.glsh以及3D模型和纹理图片文件，不要包含Visual Studio工程文件以及 生成的临时文件。
4. 将压缩包上传到http://xzc.cn/SWhktJ3RU5 作业提交截止时间4 月30日23:59

修改流程

设置第二个光源的位置和颜色

对于当前任务，我们需要参照原先的光源，构造第二个光源

```
// variable allocation for display
GLuint mvLoc, projLoc, nLoc;

GLuint globalAmbLoc,
ambLoc, diffLoc, specLoc, posLoc,
mambLoc, mdiffLoc, mspecLoc, mshiLoc,  //材质

ambLo1, diffLo1, specLo1, posLo1; //第二光源

int width, height;
float aspect;
glm::mat4 pMat, vMat, mMat, mvMat, invTrMat, rMat, rMa1;
glm::vec3 currentLightPos, transformed;
glm::vec3 currentLightPo1, transforme1; //第二光源


float lightPos[3];
float lightPo1[3]; //第二光源位置

glm::vec3 initialLightLoc = glm::vec3(7.0f, 2.0f, 2.0f);
glm::vec3 initialLightLo1 = glm::vec3(-7.0f, -2.0f, -2.0f); //第二光源初始位置

// red light
float globalAmbient[4] = { 0.7f, 0.7f, 0.7f, 1.0f };
float lightAmbient[4] = { 0.0f, 0.0f, 0.0f, 1.0f };
float lightDiffuse[4] = { 1.0f, 0.0f, 0.0f, 1.0f };
float lightSpecular[4] = { 1.0f, 0.0f, 0.0f, 1.0f };
// blue light
float globalAmbien1[4] = { 0.7f, 0.7f, 0.7f, 1.0f };
float lightAmbien1[4] = { 0.0f, 0.0f, 0.0f, 1.0f };
float lightDiffus1[4] = { 0.0f, 0.0f, 1.0f, 1.0f };
float lightSpecula1[4] = { 0.0f, 0.0f, 1.0f, 1.0f };
```

```
void installLights(glm::mat4 vMatrix) {
    //使用视图矩阵将光源变化到另一个矩阵空间
    transformed = glm::vec3(vMatrix * glm::vec4(currentLightPos, 1.0));
    lightPos[0] = transformed.x;
    lightPos[1] = transformed.y;
    lightPos[2] = transformed.z;

    transforme1 = glm::vec3(vMatrix * glm::vec4(currentLightPo1, 1.0));
    lightPo1[0] = transforme1.x;
    lightPo1[1] = transforme1.y;
    lightPo1[2] = transforme1.z;

    // get the locations of the light and material fields in the shader
    globalAmbLoc = glGetUniformLocation(renderingProgram, "globalAmbient");
    ambLoc = glGetUniformLocation(renderingProgram, "light.ambient");
    diffLoc = glGetUniformLocation(renderingProgram, "light.diffuse");
    specLoc = glGetUniformLocation(renderingProgram, "light.specular");
    posLoc = glGetUniformLocation(renderingProgram, "light.position");

    ambLo1 = glGetUniformLocation(renderingProgram, "ligh1.ambient");
    diffLo1 = glGetUniformLocation(renderingProgram, "ligh1.diffuse");
    specLo1 = glGetUniformLocation(renderingProgram, "ligh1.specular");
    posLo1 = glGetUniformLocation(renderingProgram, "ligh1.position");

    mambLoc = glGetUniformLocation(renderingProgram, "material.ambient");
    mdiffLoc = glGetUniformLocation(renderingProgram, "material.diffuse");
    mspecLoc = glGetUniformLocation(renderingProgram, "material.specular");
    mshiLoc = glGetUniformLocation(renderingProgram, "material.shininess");

    //  set the uniform light and material values in the shader
    glProgramUniform4fv(renderingProgram, globalAmbLoc, 1, globalAmbient);
    glProgramUniform4fv(renderingProgram, ambLoc, 1, lightAmbient);  //红光光照
    glProgramUniform4fv(renderingProgram, diffLoc, 1, lightDiffuse);
    glProgramUniform4fv(renderingProgram, specLoc, 1, lightSpecular);
    glProgramUniform3fv(renderingProgram, posLoc, 1, lightPos);

    glProgramUniform4fv(renderingProgram, ambLo1, 1, lightAmbien1); //蓝光光照
    glProgramUniform4fv(renderingProgram, diffLo1, 1, lightDiffus1);
    glProgramUniform4fv(renderingProgram, specLo1, 1, lightSpecula1);
    glProgramUniform3fv(renderingProgram, posLo1, 1, lightPo1);

    glProgramUniform4fv(renderingProgram, mambLoc, 1, matAmb);
    glProgramUniform4fv(renderingProgram, mdiffLoc, 1, matDif);
    glProgramUniform4fv(renderingProgram, mspecLoc, 1, matSpe);
    glProgramUniform1f(renderingProgram, mshiLoc, matShi);
}
```

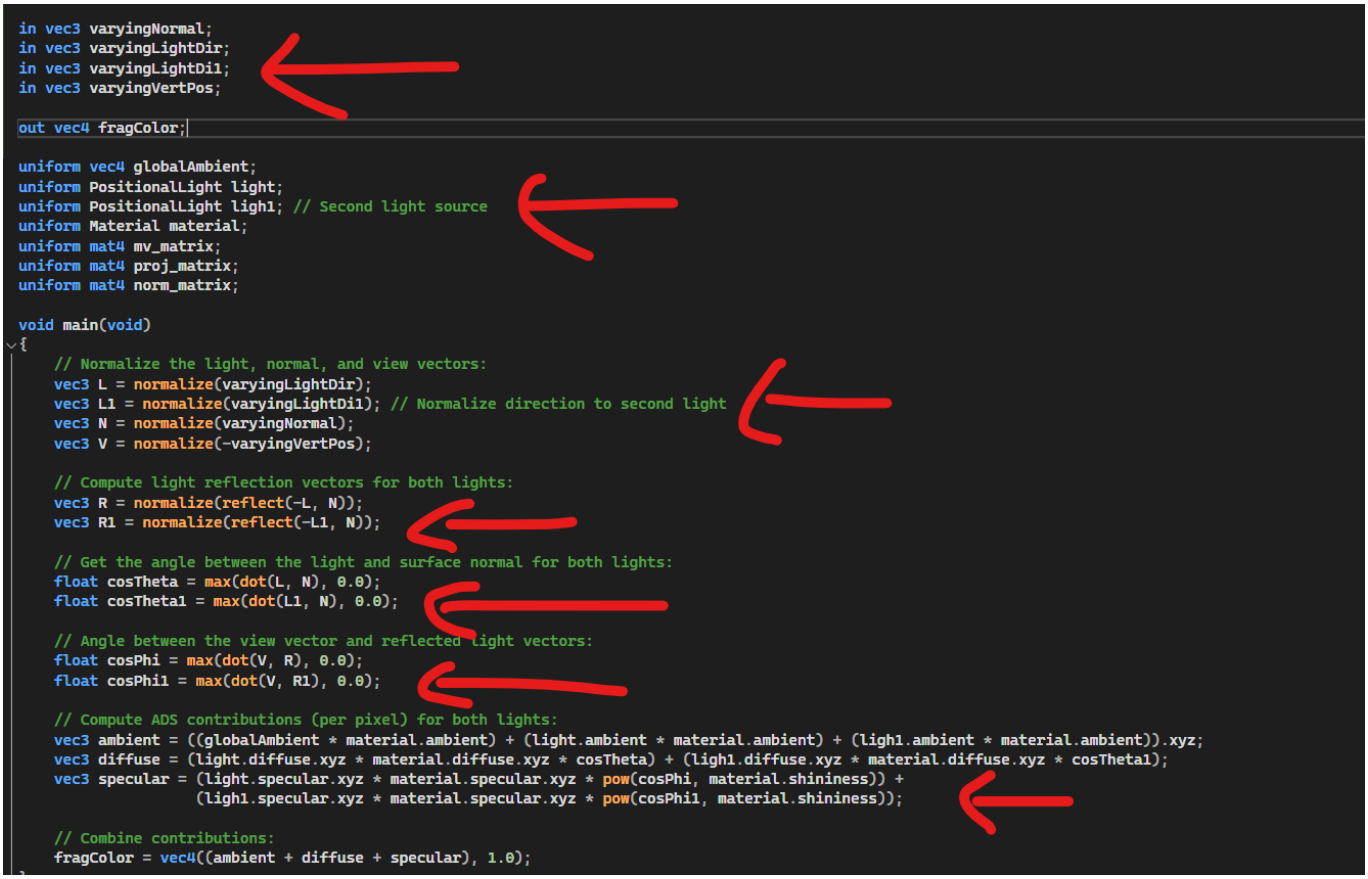
```
void display(GLFWwindow* window, double currentTime) {
    glClear(GL_DEPTH_BUFFER_BIT);
    glClear(GL_COLOR_BUFFER_BIT);

    glUseProgram(renderingProgram);

    mvLoc = glGetUniformLocation(renderingProgram, "mv_matrix");
    projLoc = glGetUniformLocation(renderingProgram, "proj_matrix");
    nLoc = glGetUniformLocation(renderingProgram, "norm_matrix");

    vMat = glm::translate(glm::mat4(1.0f), glm::vec3(-cameraX, -cameraY, -cameraZ));

    mMat = glm::translate(glm::mat4(1.0f), glm::vec3(torLocX, torLocY, torLocZ));
    mMat *= glm::rotate(mMat, toRadians(35.0f), glm::vec3(1.0f, 0.0f, 0.0f));

    currentLightPos = glm::vec3(initialLightLoc.x, initialLightLoc.y, initialLightLoc.z);
    currentLightPo1 = glm::vec3(initialLightLo1.x, initialLightLo1.y, initialLightLo1.z); //位置设置
    amt = currentTime * 25.0f;
    rMat = glm::rotate(glm::mat4(1.0f), toRadians(amt), glm::vec3(0.0f, 0.0f, 1.0f));
    rMa1 = glm::rotate(glm::mat4(1.0f), toRadians(-amt), glm::vec3(0.0f, 0.0f, 1.0f)); //反方向旋转
    currentLightPos = glm::vec3(rMat * glm::vec4(initialLightLoc, 1.0f));
    currentLightPo1 = glm::vec3(rMa1 * glm::vec4(initialLightLo1, 1.0f));

    installLights(vMat);

    mvMat = vMat * mMat;
    invTrMat = glm::transpose(glm::inverse(mvMat));
```

添加对应变量入两个着色器文件

顶点着色器

```glsl
#version 430

layout (location = 0) in vec3 vertPos;
layout (location = 1) in vec3 vertNormal;

// Outputs to the fragment shader
out vec3 varyingNormal;
out vec3 varyingLightDir;
out vec3 varyingLightDi1; // Direction to the second light
out vec3 varyingVertPos;

struct PositionalLight
{
    vec4 ambient;
    vec4 diffuse;
    vec4 specular;
    vec3 position;
};

struct Material
{
    vec4 ambient;
    vec4 diffuse;
    vec4 specular;
    float shininess;
};

uniform vec4 globalAmbient;
uniform PositionalLight light;
uniform PositionalLight ligh1; // Second light source
uniform Material material;
uniform mat4 mv_matrix;
uniform mat4 proj_matrix;
uniform mat4 norm_matrix;

void main(void)
{
    vec4 worldPos = mv_matrix * vec4(vertPos, 1.0);
    varyingVertPos = worldPos.xyz;
    varyingLightDir = light.position - varyingVertPos;
    varyingLightDi1 = ligh1.position - varyingVertPos; // Calculate the direction to ligh1
    varyingNormal = (norm_matrix * vec4(vertNormal, 1.0)).xyz;

    gl_Position = proj_matrix * worldPos;
}
```

片段着色器



```
in vec3 varyingNormal;
in vec3 varyingLightDir;
in vec3 varyingLightDi1;
in vec3 varyingVertPos;

out vec4 fragColor;

uniform vec4 globalAmbient;
uniform PositionalLight light;
uniform PositionalLight ligh1; // Second light source
uniform Material material;
uniform mat4 mv_matrix;
uniform mat4 proj_matrix;
uniform mat4 norm_matrix;

void main(void)
{
    // Normalize the light, normal, and view vectors:
    vec3 L = normalize(varyingLightDir);
    vec3 L1 = normalize(varyingLightDi1); // Normalize direction to second light
    vec3 N = normalize(varyingNormal);
    vec3 V = normalize(-varyingVertPos);

    // Compute light reflection vectors for both lights:
    vec3 R = normalize(reflect(-L, N));
    vec3 R1 = normalize(reflect(-L1, N));

    // Get the angle between the light and surface normal for both lights:
    float cosTheta = max(dot(L, N), 0.0);
    float cosTheta1 = max(dot(L1, N), 0.0);

    // Angle between the view vector and reflected light vectors:
    float cosPhi = max(dot(V, R), 0.0);
    float cosPhi1 = max(dot(V, R1), 0.0);

    // Compute ADS contributions (per pixel) for both lights:
    vec3 ambient = ((globalAmbient * material.ambient) + (light.ambient * material.ambient) + (ligh1.ambient * material.ambient)).xyz;
    vec3 diffuse = (light.diffuse.xyz * material.diffuse.xyz * cosTheta) + (ligh1.diffuse.xyz * material.diffuse.xyz * cosTheta1);
    vec3 specular = (light.specular.xyz * material.specular.xyz * pow(cosPhi, material.shininess)) +
                    (ligh1.specular.xyz * material.specular.xyz * pow(cosPhi1, material.shininess));

    // Combine contributions:
    fragColor = vec4((ambient + diffuse + specular), 1.0);
}
```

修改步骤（省略源代码已有部分）

代码讲解部分

main函数当中与着色器变量的互动

在光照函数当中，需要有两个与缓冲区互动的步骤

```
// get the locations of the light and material fields in the shader
globalAmbLoc = glGetUniformLocation(renderingProgram, "globalAmbient");
ambLoc = glGetUniformLocation(renderingProgram, "light.ambient");
diffLoc = glGetUniformLocation(renderingProgram, "light.diffuse");
specLoc = glGetUniformLocation(renderingProgram, "light.specular");
posLoc = glGetUniformLocation(renderingProgram, "light.position");

ambLo1 = glGetUniformLocation(renderingProgram, "ligh1.ambient");
diffLo1 = glGetUniformLocation(renderingProgram, "ligh1.diffuse");
specLo1 = glGetUniformLocation(renderingProgram, "ligh1.specular");
posLo1 = glGetUniformLocation(renderingProgram, "ligh1.position");
```

`light`和`ligh1`分别对应获取顶点着色器的两个uniform变量 `glGetUniformLocation`可用于传递颜色、变换矩阵、光照参数等数据。

```
//  set the uniform light and material values in the shader
glProgramUniform4fv(renderingProgram, globalAmbLoc, 1, globalAmbient);
glProgramUniform4fv(renderingProgram, ambLoc, 1, lightAmbient);  //红光光照
glProgramUniform4fv(renderingProgram, diffLoc, 1, lightDiffuse);
glProgramUniform4fv(renderingProgram, specLoc, 1, lightSpecular);
glProgramUniform3fv(renderingProgram, posLoc, 1, lightPos);

glProgramUniform4fv(renderingProgram, ambLo1, 1, lightAmbien1); //蓝光光照
glProgramUniform4fv(renderingProgram, diffLo1, 1, lightDiffus1);
glProgramUniform4fv(renderingProgram, specLo1, 1, lightSpecula1);
glProgramUniform3fv(renderingProgram, posLo1, 1, lightPo1);
```

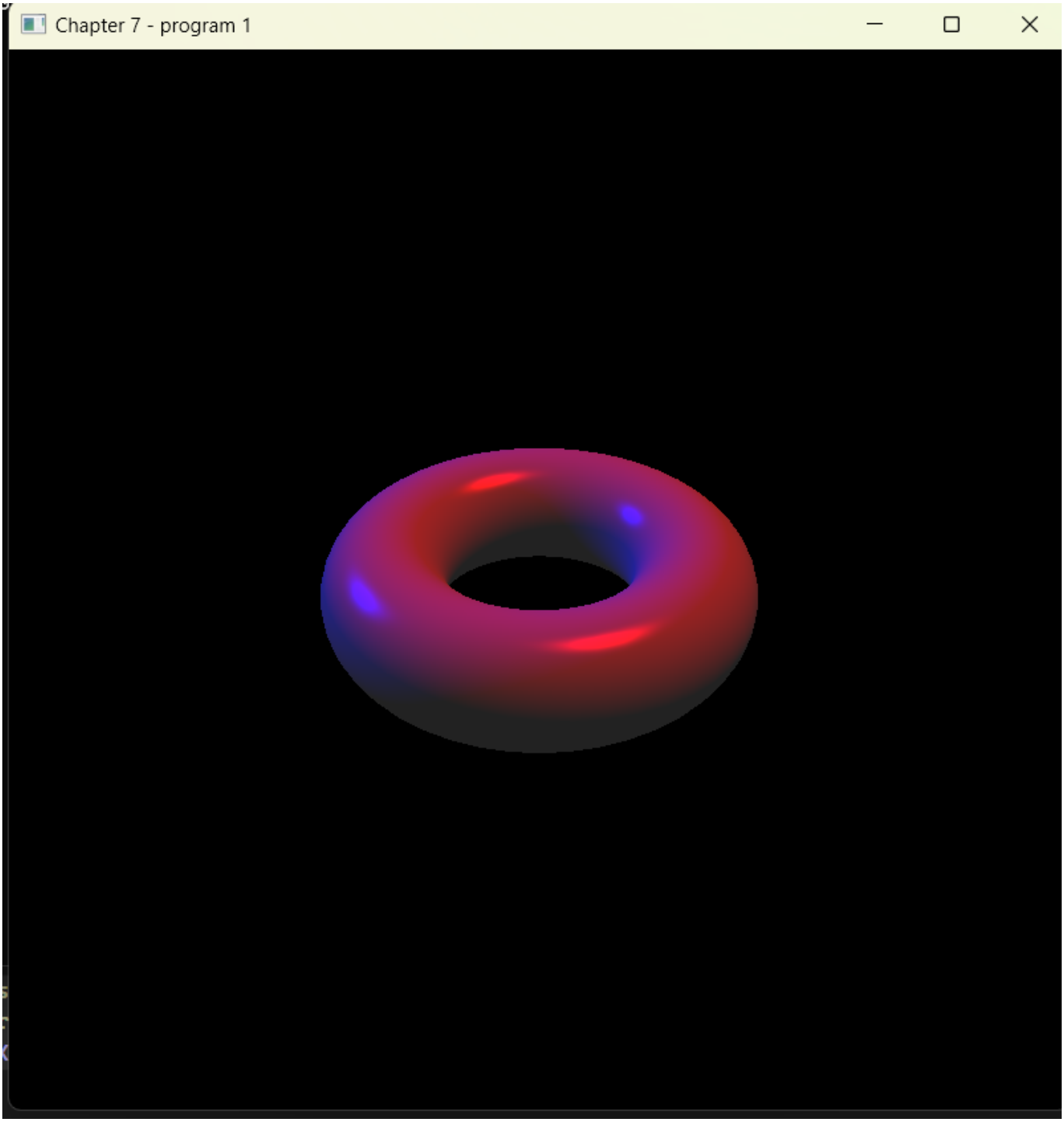
作用是让两个光照与顶点互动 `glProgramUniform4fv`用于直接设置着色器程序中uniform变量的值

光源旋转

```
currentLightPos = glm::vec3(initialLightLoc.x, initialLightLoc.y,
initialLightLoc.z);
currentLightPo1 = glm::vec3(initialLightLo1.x, initialLightLo1.y,
initialLightLo1.z); //位置设置
amt = currentTime * 25.0f;
rMat = glm::rotate(glm::mat4(1.0f), toRadians(amt), glm::vec3(0.0f, 0.0f, 1.0f));
rMa1 = glm::rotate(glm::mat4(1.0f), toRadians(-amt), glm::vec3(0.0f, 0.0f, 1.0f));
//反方向旋转
currentLightPos = glm::vec3(rMat * glm::vec4(initialLightLoc, 1.0f));
currentLightPo1 = glm::vec3(rMa1 * glm::vec4(initialLightLo1, 1.0f));
```

注意，要使光源旋转，需要改变的是`toRadians()`函数，后面那个参数是用来设置旋转围绕的轴。

结果图



实验收获：

学习和应用多光源 通过本次实验，我首次尝试在同一个场景中设置两个独立的光源，并对它们的属性如颜色、强度和位置进行了控制。实验中一个光源发出红光，另一个发出蓝光，这使得整个场景呈现出非常独特和有趣的视觉效果。通过对每个光源的漫反射和镜面反射分量进行混合和加权求和，我得以观察到不同光源设置对物体表面影响的变化，这增强了我的光照处理能力。

着色器的灵活应用 在本次实验中，修改和应用顶点着色器和片段着色器是一个挑战。我学习了如何传递多个光源信息给着色器，并在着色器中编写代码来处理这些信息。通过这种方式，我能够控制场景中不同光源的影响，使我对着色器编程的理解更加深刻。

代码结构和组织 在进行实验的过程中，我意识到良好的代码结构和注释的重要性。为了使代码清晰和易于理解，我花了额外的时间来优化代码结构并添加了详尽的注释。这不仅帮助我在编写时保持思路清晰，也使得其他人（包括教师和同学们）能够更容易地理解我的工作。 **实践与理论的结合** 通过这次实验，我将课堂上学到的

理论知识应用于实践中，特别是在光照模型和着色器的具体应用上。这种实践经验是非常宝贵的，它不仅加深了我对图形学概念的理解，还提高了我解决实际问题的能力。
