练 习 题 报 告

课程名称 计算机图形学

项目名称 Phong 光照模型（2）

学　　院 计算机与软件学院

专　　业 软件工程（腾班）

指导教师 熊卫丹

报 告 人 洪子敬 学号 2022155033

一、 练习目的

1. 了解 OpenGL 中基本的光照模型
2. 掌握 OpenGL 中实现基于片元的光照计算（Phong 着色）

二．练习完成过程及主要代码说明

练习要求：类似实验 3.3，完善 main.cpp、Trimesh.cpp 和 fshader.gsl，实现 Phong 光照模型；**区别是实现片元着色器，即 fshader.gsl，而不是顶点着色器，即 vshader.gsl；因此本报告主要说明与实验 3.3 不同的地方，其他相同地方只展示代码，详细请见提交的实验 3.3 报告。**

Task-1：在 Trimesh.cpp 中完善 computeTriangleNormals()和 computeVertexNormals()，接着在 main.cpp 中完善 bindObjectAndData().

computeTriangleNormals 函数补充：

```
void TriMesh::computeTriangleNormals()
{
    // 这里的resize函数会给face_normals分配一个和faces一样大的空间
    face_normals.resize(faces.size());
    for (size_t i = 0; i < faces.size(); i++) {
        auto& face = faces[i];
        // @TODO: Task1 计算每个面片的法向量并归一化
        glm::vec3 v0 = vertex_positions[face.x];
        glm::vec3 v1 = vertex_positions[face.y];
        glm::vec3 v2 = vertex_positions[face.z];
        glm::vec3 norm;
        norm = glm::normalize(glm::cross(v1 - v0, v2 - v0));
        face_normals[i] = norm;
    }
}
```

HZJ

computeVertexNormals 函数补充：

```
void TriMesh::computeVertexNormals()
{
    // 计算面片的法向量
    if (face_normals.size() == 0 && faces.size() > 0) {
        computeTriangleNormals();
    }
    // 这里的resize函数会给vertex_normals分配一个和vertex_positions一样大的空间
    // 并初始化法向量为0
    vertex_normals.resize(vertex_positions.size(), glm::vec3(0, 0, 0));
    // @TODO: Task1 求法向量均值
    for (size_t i = 0; i < faces.size(); i++) {
        auto& face = faces[i];
        // @TODO: 先累加面的法向量
        vertex_normals[face.x] += face_normals[i];
        vertex_normals[face.y] += face_normals[i];
        vertex_normals[face.z] += face_normals[i];
    }
    // @TODO 对累加的法向量归一化
    for (size_t i = 0; i < vertex_normals.size(); i++) {
        vertex_normals[i] = glm::normalize(vertex_normals[i]);
    }
}
```

HZJ

bindObjectAndData 函数的补充：

添加以下代码传递数据给着色器（在顶点着色器中进行坐标系转换再进入片元着色器）

```
// @TODO: Task1 从顶点着色器中初始化顶点的法向量
object.nLocation = glGetAttribLocation(object.program, "vNormal");
glEnableVertexAttribArray(object.nLocation);
glVertexAttribPointer(object.nLocation, 3, GL_FLOAT, GL_FALSE, 0, BUFFER_OFFSET(mesh->getPoints().size() * sizeof(glm::vec3)));
```

编写完 TriMesh.cpp 代码后打开注释内容：

```
// @TODO: Task1 修改完TriMesh.cpp的代码成后再打开下面注释，否则程序会报错
glBufferSubData(GL_ARRAY_BUFFER, (mesh->getPoints().size() + mesh->getColors().size()) * sizeof(glm::vec3), mesh->getNormals().size() * sizeof(glm::vec3), &mesh->getNormals()[0]);
```

Task-2：在 fshader.glsl 中完善 main()

与实验 3.3 顶点着色器光照模型的绘制类似，main 函数里面主要是 Phong 反射模型数据的计算包括四个归一化向量的计算和归一化、环境光的计算（已给）、漫反射的计算、镜面反射计算和高光系数的计算。主要区别是这里的顶点坐标、光源坐标和法向量不需要在此处转换到相机坐标系，因为在顶点坐标系已经进行了转换。

同时实验 3.3 原理相同，四个向量计算和归一化过程如下所示：

```
// @TODO: 计算四个归一化的向量 N,V,L,R(或半角向量H)
vec3 N=normalize(norm);
vec3 V=normalize(eye_position-position);
vec3 L=normalize(light.position-position);
vec3 R=reflect(-L,N);
```

HZJ

漫反射系数与分量、镜面反射分量和高光系数的计算过程如下所示：

```
// @TODO: Task2 计算系数和漫反射分量I_d
float diffuse_dot = max(dot(N,L),0.0);
vec4 I_d = diffuse_dot *  light.diffuse * material.diffuse;

// @TODO: Task2 计算系数和镜面反射分量I_s
 float specular_dot_pow = pow(max(dot(R,V),0.0),material.shininess);
 vec4 I_s = specular_dot_pow * light.specular * material.specular;
```

HZJ

Task-3：在 main.cpp 中 init()函数里改变材质参数

由于此实验是在片元着色器里面绘制的，与实验 3.3 有些许不同，在光源位置、材质的选取上略有差异，本实验同样提高光源的位置并使漫反射的光偏黄色一点，整体运行结果像是停电后点蜡烛的温馨效果。详细代码如下：

```
void init()
{
    std::string vshader, fshader;
    // 读取着色器并使用
    vshader = "shaders/vshader.glsl";
    fshader = "shaders/fshader.glsl";

    // @TODO: Task3 请自己调整光源参数和物体材质参数来达到不同视觉效果
    // 设置光源位置
    light->setTranslation(glm::vec3(0.0, 1.0, 2.0)); // 调整光源位置，使其更接近物体
    light->setAmbient(glm::vec4(0.5, 0.5, 0.5, 1.0)); // 环境光
    light->setDiffuse(glm::vec4(1.0, 1.0, 0.8, 1.0)); // 漫反射
    light->setSpecular(glm::vec4(1.0, 1.0, 0.8, 1.0)); // 镜面反射

    // 设置物体的旋转位移
    mesh->setTranslation(glm::vec3(0.0, 0.0, 0.0));
    mesh->setRotation(glm::vec3(0, 0.0, 0.0));
    mesh->setScale(glm::vec3(1.0, 1.0, 1.0));

    // 设置材质
    mesh->setAmbient(glm::vec4(0.3, 0.2, 0.1, 1.0)); // 暖色调环境光
    mesh->setDiffuse(glm::vec4(0.8, 0.5, 0.3, 1.0)); // 温暖的漫反射
    mesh->setSpecular(glm::vec4(1.0, 1.0, 1.0, 1.0)); // 明亮的镜面反射
    mesh->setShininess(32.0); // 增加高光系数，以获得更明显的高光效果
    // 将物体的顶点数据传递
    bindObjectAndData(mesh, mesh_object, vshader, fshader);

    glClearColor(0.0, 0.0, 0.0, 0.0);
```

HZJ

Task-4：在 main.cpp 中 mainWindow_key_callback ()函数完善键盘交互

4-6 键实现漫反射光的增强和减弱的代码如下：

```
else if (key == GLFW_KEY_4 && action == GLFW_PRESS && mode == 0x0000)
{
    diffuse = mesh->getDiffuse();
    tmp = diffuse.x;
    diffuse.x = std::min(tmp + 0.1, 1.0);
    mesh->setDiffuse(diffuse);
}
else if (key == GLFW_KEY_4 && action == GLFW_PRESS && mode == GLFW_MOD_SHIFT)
{
    diffuse = mesh->getDiffuse();
    tmp = diffuse.x;
    diffuse.x = std::max(tmp - 0.1, 0.0);
    mesh->setDiffuse(diffuse);
}
else if (key == GLFW_KEY_5 && action == GLFW_PRESS && mode == 0x0000)
{
    diffuse = mesh->getDiffuse();
    tmp = diffuse.y;
    diffuse.y = std::min(tmp + 0.1, 1.0);
    mesh->setDiffuse(diffuse);
}
```

HZJ

```
else if (key == GLFW_KEY_5 && action == GLFW_PRESS && mode == GLFW_MOD_SHIFT)
{
    diffuse = mesh->getDiffuse();
    tmp = diffuse.y;
    diffuse.y = std::max(tmp - 0.1, 0.0);
    mesh->setDiffuse(diffuse);
}
else if (key == GLFW_KEY_6 && action == GLFW_PRESS && mode == 0x0000)
{
    diffuse = mesh->getDiffuse();
    tmp = diffuse.z;
    diffuse.z = std::min(tmp + 0.1, 1.0);
    mesh->setDiffuse(diffuse);
}
else if (key == GLFW_KEY_6 && action == GLFW_PRESS && mode == GLFW_MOD_SHIFT)
{
    diffuse = mesh->getDiffuse();
    tmp = diffuse.z;
    diffuse.z = std::max(tmp - 0.1, 0.0);
    mesh->setDiffuse(diffuse);
}
```

HZJ

7-9 键实现镜面反射光的增强和减弱的代码如下：

```
else if (key == GLFW_KEY_7 && action == GLFW_PRESS && mode == 0x0000)
{
    specular = mesh->getSpecular();
    tmp = specular.x;
    specular.x = std::min(tmp + 0.1, 1.0);
    mesh->setSpecular(specular);
}
else if (key == GLFW_KEY_7 && action == GLFW_PRESS && mode == GLFW_MOD_SHIFT)
{
    specular = mesh->getSpecular();
    tmp = specular.x;
    specular.x = std::max(tmp - 0.1, 0.0);
    mesh->setSpecular(specular);
}
else if (key == GLFW_KEY_8 && action == GLFW_PRESS && mode == 0x0000)
{
    specular = mesh->getSpecular();
    tmp = specular.y;
    specular.y = std::min(tmp + 0.1, 1.0);
    mesh->setSpecular(specular);
}
```

HZJ

```
else if (key == GLFW_KEY_8 && action == GLFW_PRESS && mode == GLFW_MOD_SHIFT)
{
    specular = mesh->getSpecular();
    tmp = specular.y;
    specular.y = std::max(tmp - 0.1, 0.0);
    mesh->setSpecular(specular);
}
else if (key == GLFW_KEY_9 && action == GLFW_PRESS && mode == 0x0000)
{
    specular = mesh->getSpecular();
    tmp = specular.z;
    specular.z = std::min(tmp + 0.1, 1.0);
    mesh->setSpecular(specular);
}
else if (key == GLFW_KEY_9 && action == GLFW_PRESS && mode == GLFW_MOD_SHIFT)
{
    specular = mesh->getSpecular();
    tmp = specular.z;
    specular.z = std::max(tmp - 0.1, 0.0);
    mesh->setSpecular(specular);
}
```

HZJ

0 键来实现高光指数的大小，指定其范围在[1.0, 128.0]之间波动的代码如下：

```
else if (key == GLFW_KEY_0 && action == GLFW_PRESS && mode == 0x0000)
{
    shininess = mesh->getShininess();
    shininess = std::min(shininess + 1.0, 128.0);
    mesh->setShininess(shininess);
}
else if (key == GLFW_KEY_0 && action == GLFW_PRESS && mode == GLFW_MOD_SHIFT)
{
    shininess = mesh->getShininess();
    specular.z = std::max(shininess - 1.0, 1.0);
    mesh->setShininess(shininess);
}
```

HZJ

结果展示：运行程序，部分实验结果展示如下，实验圆满结束：