练 习 题 报 告

课程名称 计算机图形学

项目名称 Phong 光照模型（1）

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 中实现基于顶点的光照计算
3. 掌握法向量的计算

二．练习完成过程及主要代码说明

练习要求：完善 main.cpp、Trimesh.cpp 和 vshader.gsl，实现 Phong 光照模型。

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：

此函数作用主要是计算面片的法向量，给定三角形的面片，要求计算每个面片的法向量并进行归一化。因此，我们只需要计算给定的每个面片三个顶点坐标，并根据公式：$(p_1 - p_0) \times (p_2 - p_0)$进行求解（三个变量为一个面片的三个顶点坐标），最后进行归一化即可，不过注意要将归一化的向量存储在自定义类变量 face_norm 中进行传递。详细函数代码如下：

```
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：

此函数的作用主要是计算顶点法向量，给定面片的法向量（前面函数计算得到），顶点的平均法向量为法向量的和。具体通过对每个面片的法向量累加到面片上每个顶点上，从而的得到每个顶点的法向量，最后进行归一化即可。详细代码如下所示：

```
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：

此函数主要的作用是将计算好的顶点法向量数据传递给着色器，参照 vPosition 的写法，我们只需要修改传递对象为“vNormal”，并且指定传递位置为“nLocation”即可。

```
// @TODO: Task1 从顶点着色器中初始化顶点的法向量                                                    HZJ
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)));
```

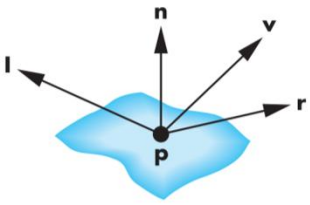
此外，我们还需要打开此函数中的一句注释进行法向量数据的传递。

```
// @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() * size
```

Task-2：在 vshader.glsl 中完善 main()

mian 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，而通过实验指导书我们可以知道四个归一化向量分布如右下图所示：

p 为三维物体表面上的一点，l 是从点 p 指向光源位置的向量，n 表示 p 点的法向量，v 是从 p 点指向相机（观察者）的向量，r 是沿着 l 方向入射光线按照反射定律的出射方向。



因此按照此原理，四个向量计算和归一化过程如下所示：

```
// @TODO: Task2 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-pos);
vec3 L=normalize(l_pos-pos);
vec3 R=reflect(-L,N);                                                    HZJ
```

由于实验设定衰减系数$\frac{1}{a+bd+cd^2}=1$，因此漫反射分量计算可以表示为：$\max\big((l*n),0\big)L_d$，而镜面反射分量可以表示为：$k_sL_smax\ (r\cdot v,\ 0)^{\alpha}$。按照公式编写代码如下所示：

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

// @TODO: Task2 计算高光系数beta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;
```

Task-3：在 main.cpp 中 init()函数里改变材质参数

曲面上某一点的颜色是光照和材质共同作用的结果，相同颜色的光照射在不同材质的物体上会得到不同的颜色，在代码中创建材质对象并应用到光照的计算中设置材质对象，材质的属性有环境反射率、漫反射率、镜面反射率、（自发光系数）、高光系数等组成。

本实验为了使物体变化明显，设置环境光暗一些，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。详细设置如下；

```
// @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));
light->setAmbient(glm::vec4(0.2, 0.2, 0.2, 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, 1.0, 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(0.9, 0.9, 0.9, 1.0)); // 明亮的镜面反射
mesh->setShininess(32.0); // 高光系数
```

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

此函数已经实现了通过鼠标设置光源的位置功能、旋转和平移相机功能以及设置了一半材质的交互功能，剩下一半需要我们自己去实现。通过观察可知，1-3 键实现的是环境光的增强和减弱，包括三个方向（x、y 和 z）的增强（普通按键）和减弱（按键加 shift）。因此，4-6 键我们打算实现漫反射光的增强和减弱，同样包括三个方向，也是通过增加 shift 键来实现增强减弱之分，具体实现如下所示：

```
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

同理，7-9 键实现镜面反射光的增强和减弱，思想基本相同，代码如下所示：

```
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

此外，还有按键 0 来实现高光指数的大小，我们指定其范围在[1.0, 128.0]之间波动，每次普

通按键，系数增加 1；加上 shift 键，系数减少 1。系数越大，表面会越加光滑。

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

结果展示：运行程序，部分实验结果展示如下，实验圆满结束：