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 计算机科学与技术

指导教师 周虹

报 告 人 林浩晟　学号 2022280310

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

1. 先获取面的下标，得到对应的坐标，再各自相减得到两个平面上的向量，由向量计算得到法向量并对其归一化；

```
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 edge1 = v0 - v1;//平面上的两个向量
        glm::vec3 edge2 = v1 - v2;

        glm::vec3 norm;
        norm = glm::normalize(glm::cross(edge1, edge2));//计算法向量
        face_normals[i] = norm;

    }
}
```

2. 按照提示分别累加面的法向量即可，使用 normalize 函数归一化；

```
121     for (size_t i = 0; i < faces.size(); i++) {
122         auto& face = faces[i];
123         // @TODO: 先累加面的法向量
124         // vertex_normals[face.x] += face_normals[i];
125         // ...
126         vertex_normals[face.x] += face_normals[i];
127         vertex_normals[face.y] += face_normals[i];
128         vertex_normals[face.z] += face_normals[i];
129     }
130     // @TODO 对累加的法向量归一化
131     for (size_t i = 0; i < vertex_normals.size(); i++) {
132         vertex_normals[i] = glm::normalize(vertex_normals[i]);
133     }
134 }
```

3. 将 main.cpp 里面的注释解开；

```
// @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]);
```

4. 再按照其他的着色器初始化部分编写该部分代码；

```
// @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() + mesh->getColors().size()) * sizeof(glm::vec3)));
```

5. 分别计算各个向量以及漫反射系数和高光系数；

```
// @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);
vec3 H = normalize(V + L);
```

```
// @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);;
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(N, H), 0.0), material.shininess);
vec4 I_s = specular_dot_pow * light.specular * material.specular;
```

6. 设置光源位置，设置了两个方案，一个凸显光亮效果，一个凸显阴影效果；

```
// @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
int choose = 1;
if(choose==1)
{//明亮效果
    light->setTranslation(glm::vec3(2.0, 2.0, 5.0));
    light->setAmbient(glm::vec4(1.0, 1.0, 1.0, 1.0));
    light->setDiffuse(glm::vec4(0.9, 0.9, 0.9, 1.0));
    light->setSpecular(glm::vec4(0.8, 0.8, 0.8, 1.0));
}
else
{
    //对比的光源  阴影效果
    light->setAmbient(glm::vec4(0.2, 0.2, 0.2, 1.0));  // 环境光较低
    light->setDiffuse(glm::vec4(0.7, 0.7, 0.7, 1.0));  // 较弱的漫反射
    light->setSpecular(glm::vec4(1.0, 1.0, 1.0, 1.0)); // 强镜面反射
    light->setTranslation(glm::vec3(5.0, 5.0, 3.0));   // 高侧位光源位置
}
```

7. 设置三个材质，分别对于圆球、皮卡丘和杰尼龟三个模型；

```
int cnt = 3;
if(cnt==1)
{
    //紫色材质
    mesh->setAmbient(glm::vec4(0.3, 0.0, 0.3, 1.0));    // 环境光
    mesh->setDiffuse(glm::vec4(0.6, 0.0, 0.6, 1.0));    // 漫反射
    mesh->setSpecular(glm::vec4(0.5, 0.5, 0.5, 1.0));   // 镜面反射
    mesh->setShininess(32.0);                           // 高光系数
}else if(cnt==2)
{
    //皮卡丘
    mesh->setAmbient(glm::vec4(0.2, 0.2, 0.0, 1.0));    // 环境光
    mesh->setDiffuse(glm::vec4(1.0, 0.85, 0.0, 1.0));   // 漫反射（黄色）
    mesh->setSpecular(glm::vec4(0.5, 0.5, 0.0, 1.0));   // 镜面反射
    mesh->setShininess(16.0);                           // 高光系数
}else
{//杰尼龟
    mesh->setAmbient(glm::vec4(0.2, 0.3, 0.4, 1.0));    // 环境光
    mesh->setDiffuse(glm::vec4(0.4, 0.7, 0.8, 1.0));    // 漫反射（淡蓝色）
    mesh->setSpecular(glm::vec4(0.4, 0.4, 0.5, 1.0));   // 镜面反射
    mesh->setShininess(20.0);                           // 高光系数
}
```

8. 先定义以下参数，再根据上面的代码编写 4~6 的键鼠交互，该部分功能为 Change diffuse parameters；

```
glm::vec4 ambient;
glm::vec4 diffuse;
glm::vec4 specular;
float shininess;
```

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

9. 7~9 的键鼠交互，该部分功能为 Change specular parameters；

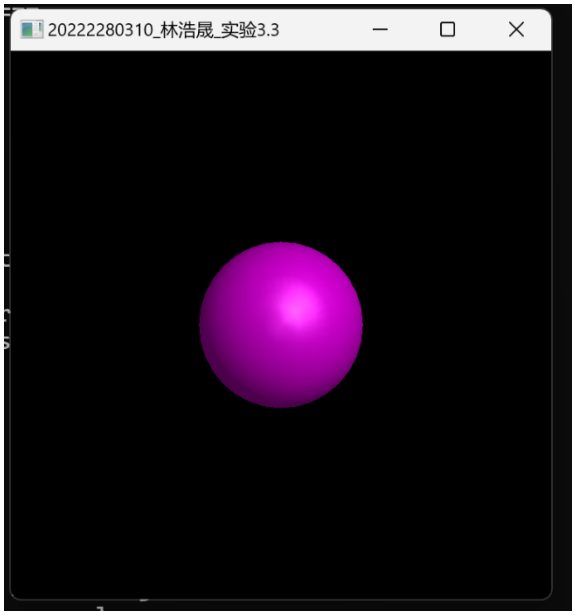
```
else if (key == GLFW_KEY_7 && action == GLFW_PRESS && mode == 0x0000)
{
    specular = mesh->getSpecular();
    tmp = specular.x;
    specular.x = std::min(tmp + 0.1, 1.0);
    mesh->setSpecular(specular);
}
else if (key == GLFW_KEY_7 && action == GLFW_PRESS && mode == GLFW_MOD_SHIFT)
{
    specular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

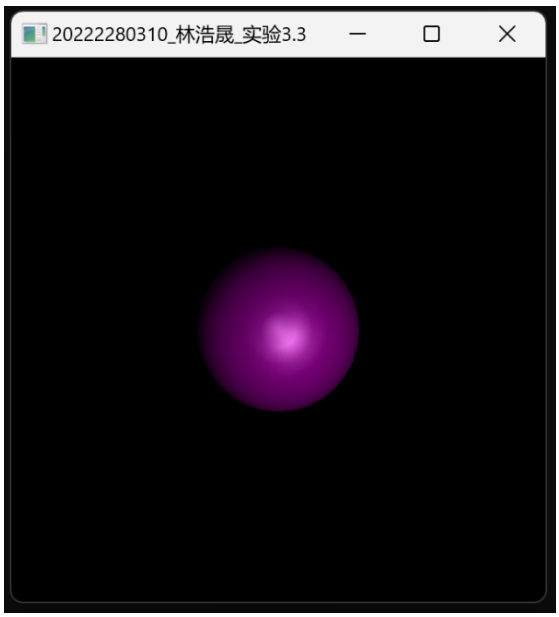
10. 0 键的键鼠交互，功能为 Change shininess parameters；

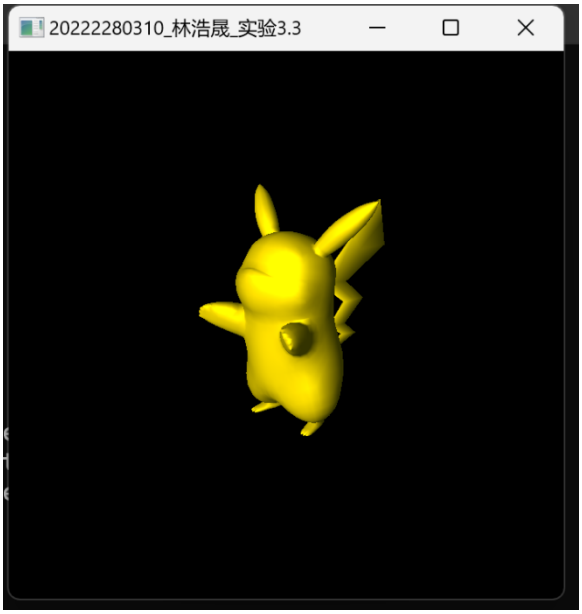
```
else if (key == GLFW_KEY_0 && action == GLFW_PRESS && mode == 0x0000)
{
    shininess = mesh->getShininess();
    tmp = shininess;
    shininess = std::min(tmp + 0.1, 1.0);
    mesh->setShininess(shininess);
}
else if (key == GLFW_KEY_0 && action == GLFW_PRESS && mode == GLFW_MOD_SHIFT)
{
    shininess = mesh->getShininess();
    tmp = shininess;
    shininess = std::min(tmp - 0.1, 1.0);
    mesh->setShininess(shininess);
}
```

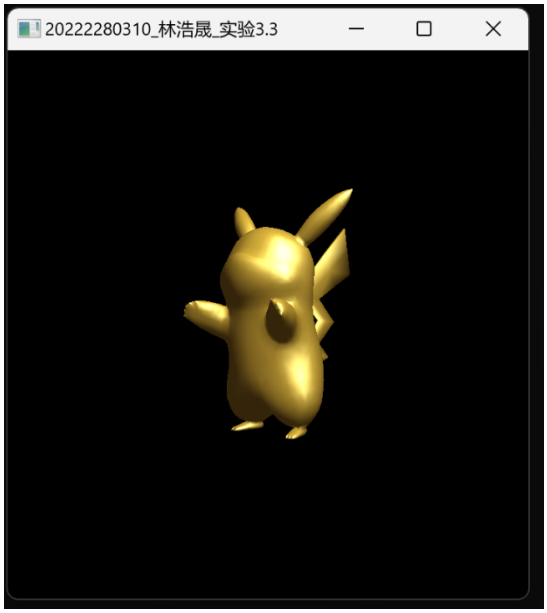
11. 运行结果如下：

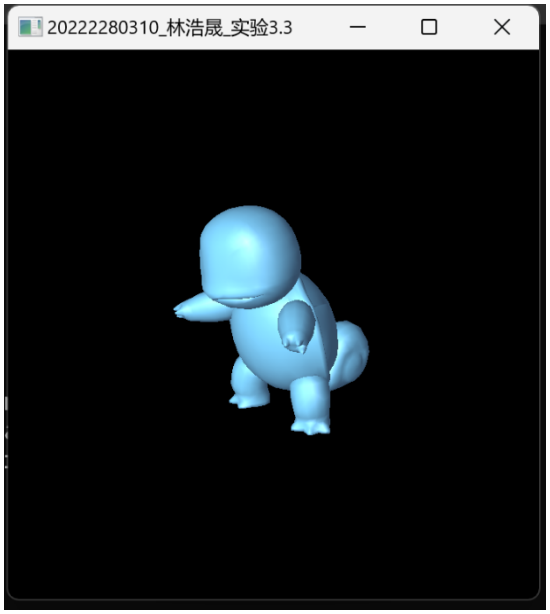
可见紫球成功显示，一张为光亮效果，一张为阴影效果(皮卡丘和杰尼龟同样)

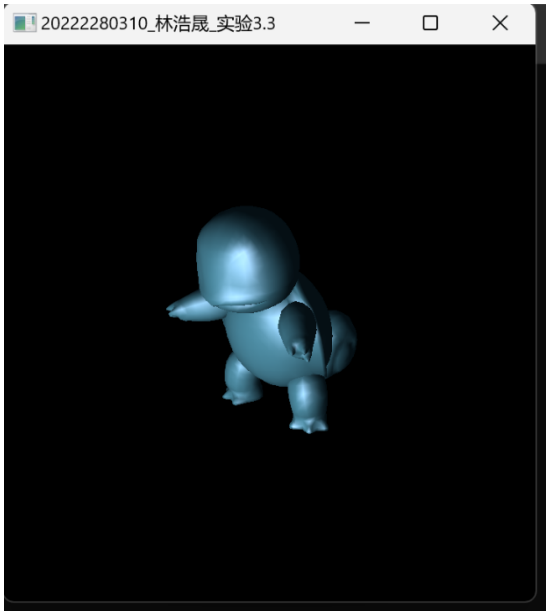





20222280310_林浩晟_实验3.3


20222280310_林浩晟_实验3.3





可以看见粗糙球体模型也成功显示