**深 圳 大 学 实 验 报 告**

**课程名称：­ 计算机图形学**

**实验项目名称： 实验三 光照与阴影**

**学院： 计算机与软件学院**

**专业： 计算机科学与技术**

**指导教师： 徐鹏飞**

**报告人： 郑雨婷 学号： 2021150122 班级： 高性能**

**实验时间：2023年 11月27日 -- 2023年 11月30日**

**实验报告提交时间： 2023年 11月30日**

**教务部制**

|  |
| --- |
| 实验目的与要求：   1. 掌握OpenGL三维场景的读取与绘制方法，理解光照和物体材质对渲染结果的影响，强化场景坐标系转换过程中常见矩阵的计算方法，熟悉阴影的绘制方法。 2. 创建OpenGL绘制窗口，读入三维场景文件并绘制。 3. 设置相机并添加交互，实现从不同位置/角度、以正交或透视投影方式观察场景。 4. 实现Phong光照效果和物体材质效果。 5. 自定义投影平面（为计算方便，推荐使用y=0平面），计算阴影投影矩阵，为三维物体生成阴影。 6. 使用鼠标点击（或其他方式）控制光源位置并更新光照效果，并同时更新三维物体的阴影。 |
| 实验过程及内容：  **一、实验内容**  实现场景的光照和阴影绘制。示例模型是一个球，需要实现的目标是为场景添加光照，并以光源为投影中心生成阴影，程序要可以控制相机从不同角度观察，如下图1所示。可以根据自己喜好绘制不同物体甚至多个物体。    图1 实验示例  **二、具体内容**  1. 绘制场景、模型  创建OpenGL绘制窗口，然后参考实验2.2内容读入三维场景文件（可以使用之前实验课提供的几何体的\*.off文件）并绘制。为了和后期的阴影颜色区分，可以将窗口背景色设置为灰色。  2. 设置相机  参考实验3.1，设置相机并添加交互，实现从不同位置/角度、以正交或透视投影方式观察场景。  3. 添加光照和材质效果  参考实验3.3或实验3.4，实现Phong光照效果和物体材质效果。  4. 添加阴影效果  参考实验3.2，以步骤3中的光源位置作为投影中心，自定义投影平面（为计算方便，推荐使用y=0平面），计算阴影投影矩阵，为三维物体生成阴影。  5. 交互控制光源位置并更新阴影  参考实验2.1，使用鼠标点击（或其他方式）控制光源位置并更新光照效果，并同时更新三维物体的阴影。  **三、实验步骤**  1.绘制场景、模型  创建OpenGL绘制窗口，然后参考实验2.2内容读入三维场景文件（可以使用之前实验课提供的几何体的\*.off文件）并绘制。为了和后期的阴影颜色区分，可以将窗口背景色设置为灰色。  （1）创建窗口，并为窗口设置标题为”2021150122-郑雨婷-实验三”。  在main函数中调用glfwCreateWindoew()函数，设置标题。但因为标题中含有中文，在程序前添加#pragma execution\_character\_set("utf-8")，将编码方式改为utf-8,这样就可以避免出现乱码。      （2）读入三维场景文件。  我们仿照实验2.2读取OFF文件。OFF文件通过描述物体表面的多边形来表示一个模型的几何结构，这里的多边形可以有任意数量的顶点。之前的实验3.4提供了多个OFF文件，放在assets文件夹下。  在TriMesh类中，创建**read\_off函数**，按照OFF文件的格式，先读取顶点数、面片数、边数。根据顶点数nVertices，循环读取每个顶点坐标，把读入的坐标写入向量vertex\_positions中，并把颜色（0，0，0）写入向量vertex\_colors中。虽然自行写入了颜色，但是物体呈现的颜色其实是光照效果的叠加，这里只是为了让vertex\_colors的数量与顶点数保持一致。    然后根据面片数nFaces，循环读取每个面片信息，并用构建的vec3i结构体保存每个面片信息，将读取到的信息写入faces向量中。    在TriMesh类中，创建**storeFacesPoints函数**。除了将每个面片顶点的坐标和颜色储存之外，还需计算每个顶点的法向量并且将数据存储到用于 GPU 的缓冲区中。**调用computeVertexNormals函数计算每个顶点的法向量**，其具体计算方法是：先计算出每个面片的法向量，累加面的法向量，再对累加的法向量并归一化。    (3)实现绘制多个物体。  添加键盘交互，分别按下qwas键可以切换不同的图形。再mainWindow\_key\_callback函数中，添加如下代码，实现按键Q读取特定的OFF文件。A、W、S这三个键与Q相似，不再给出。    （4）窗口背景色设置为灰色。  为了和后期的阴影颜色区分，可以将窗口背景色设置为灰色。在init函数中调用glClearColor()函数，设置窗口背景颜色为灰色将参数改为(0.2, 0.2, 0.2, 1.0)。    此时，运行程序的结果如下图2所示，因为未添加光照，此时模型是黑色的，但是OFF文件的读取、模型的绘制成功。    图2 读取OFF文件，但未添加光照  2. 设置相机  参考实验3.1，设置相机并添加交互，实现从不同位置/角度、以正交或透视投影方式观察场景。我们在include文件夹内创建了一个Camera.h文件，并定义了一个相机Camera类，里面声明了一些函数与相机参数，然后在Camera.cpp中实现这些函数。  （1）设置相机位置和方向  updateCamera函数用于设置相机位置和方向。eye 是相机的位置向量，表示相机在世界坐标系中的位置。at 是相机看向的点的向量，这里设定为原点 (0.0, 0.0, 0.0)。up 是相机的上方向向量，这里设定为 (0.0, 1.0, 0.0)，表示相机坐标系的上方向。在updateCamera()中可以根据角度与距离半径更新计算相机eye的位置，设置相机的方式按球形轨迹移动。    （2）计算相机观察矩阵  通过采用u,n,v即可定义出相机的观察变换矩阵（其实就是相机的局部坐标系）如下viewMatrix。由于在最开始，我们还需要将相机从坐标原点移动到视点，所以还需要一个平移矩阵如下T    所以最终的相机观察矩阵viewMatrix为：  在LookAt函数中按照公式取得计算相机观察矩阵。  （3）添加键盘交互，实现从不同位置/角度观察场景。  为了实现从不同角度观察模型，我们再camera类中实现一个keyboard函数，按下u的话可以增加旋转角度rotateAngle，如果是同时按下shift+u的话，可以减少旋转角度。同理按下I、shift+I、O、shift+O可以分别增加和减少向上的倾角upAngle和相机的半径radius。    给出按下I的代码，I控制的是向上的倾角upAngle，设置最大不超过180度，最小不小于180度。按键O和按键U是代码类似，这里不再给出。此时运行程序，可以得到如下的运行结果，成功实现了通过按键调整相机位置：    图3 改变相机的位置和方向  （4）正交投影  在camera.cpp中实现计算正交投影的ortho函数。按照计算公式为正交投影矩阵，函数接受left、right、bottom、top、zNearzFar六个参数，然后返回对应的正交投影矩阵**。**    （5）透视投影  在camera.cpp中实现计算透视投影的perspective函数。按照公式计算透视投影矩阵函数接受四个参数：视场角 fov、宽高比 aspect、近裁剪面距离 zNear 和远裁剪面距离 zFar。然后返回对应的透视投影矩阵。    （6）添加交互，实现正交和透视投影的切换  在camera.cpp中实现计算投影矩阵getProjectionMatrix时，根据Camera中的成员isOrtho来选择的投影方式。当isOrtho为true时，选择正交投影，当isOrtho为false时，选择透视投影。    按下p的话可以将投影方式设置为透视投影，如果是同时按下shift+p的话，可以将投影方式设置为正交投影。    此时运行程序，按下p和shift+p，可以看到显示出的图像略有不同，是因为正交投影和透视投影大体相似，只是正交投影的焦点在无穷远处，透视投影的焦点较近。正交投影与透视投影的切换实现成功。    图4 改变投影方式，有略微不同  3. 添加光照和材质效果  参考实验3.4，实现Phong光照效果和物体材质效果。实验3.4中的Phong反射模型在片元着色器上实现的。Phong反射模型考虑了光线和材料之间的三种相互作用，环境光反射、漫反射和镜面反射叠加组成Phong反射模型。因此，在片元着色器中依次计算环境光分量I\_a、漫反射分量I-d和镜面反射分量I\_s。  （1）在顶点着色器fshader中执行为每个顶点执行光照计算.  为了简单考虑，我们这里假设衰减系数。  按照公式计算环境光分量I\_a、漫反射分量I\_d、漫反射分量I\_s。    最后为累加三个部分的颜色分量，得到每个顶点的颜色如下，颜色相加后最后一维的透明度需要进行修正。    （2）设置光源和物体材质。  光源对象由光源位置环境光、漫反射光、镜面反射光组成。物体材质由高光系数、环境光、漫反射光、镜面反射光参数组成。在Init函数中，添加如下初始化设置：    此时运行程序可以得到，模型不再是黑色，而是环境光分量I\_a、漫反射分量I\_d、漫反射分量I\_s累加的效果，设置光源和物体材质成功。    图5 添加光源和物体材质  （3）添加交互，实现为材质material设置交互。  1~9增减反射系数，0增减高光指数。在mainWindow\_key\_callback中，实现了按下1、2、3\shift+1、2、3可以增加\减少环境光反射系数，按下4、5、6\shift+4、5、6可以增加\减少可以改变漫反射系数、按下7、8、9\shift+7、8、9可以增加\减少可以改变镜面反射系数。    给出按下1的代码，1控制的是环境光反射系数，设置最大不超过1，最小不小于0。其他按键2~9代码类似，这里不再给出。现在运行程序，通过1~9s设置反射系数，0设置高光系数，可以得到不同颜色效果的模型，为材质material设置交互成功。    图6 按键改变反射系数和高光指数  4. 添加阴影效果  参考实验3.2，以步骤3中的光源位置作为投影中心，自定义投影平面（为计算方便，推荐使用y=0平面），计算阴影投影矩阵，为三维物体生成阴影。  （1）平移物体至Y=0平面上方  原OFF文件中顶点的坐标是以(0, 0, 0)为中心的，所以。为了方便后面计算阴影矩阵，我们可以把点整体都平移到y=0上。所以再TriMesh类中再添加一个函数translationPositions用于整体平移，在函数中先是找到所有顶点的y的最小值imin，之后再把所有顶点的y值减去imin做整体平移到y=0平面上。    在storeFacePoints中，先调用translationPositions进行整体平移，然后存储顶点的信息，这样就可以使整个给物体都在Y=0平面上方。  （2）计算阴影投影矩阵  绘制阴影，其实就是对模型制两次。第一次是按照常规方式绘制，第二次是使用了阴影变换矩阵之后对新的阴影模型进行绘制。也就是说，第一次绘制时，将阴影投影矩阵设置为单位矩阵，而第二次绘制是计算出矩阵值之后变得到投影模型。  getShadowProjectionMatrix实现了生成阴影投影矩阵的功能。通过将光源位置经过模型矩阵变换后，取其x、y、z坐标分别记为lx,ly,lz。然后构建一个特定形式的4x4矩阵，该矩阵用于将场景投影到Y=0d的平面上。    （3）绘制阴影  原本的模型矩阵与阴影投影矩阵相乘后，就得到了阴影的矩阵，将其传入着色器，即可绘制出阴影。在display函数中添加以下代码，以绘制阴影。    此时运行程序，可以看到不同模型的阴影都绘制成功。    图7 绘制阴影  5. 交互控制光源位置并更新阴影  参考实验2.1，使用鼠标点击（或其他方式）控制光源位置并更新光照效果，并同时更新三维物体的阴影。  （1）添加键盘交互，实现光源位置的改变。  在键盘按键监听函数中，分别按下xyz可以分别增加光源的xyz位置，每次变化的步长为0.05，按下shift+xyz可以减少光源的xyz的位置。但是为了避免光源到y=0的平面上时观察不到阴影，我给光源设定y的最小值为1，如果变化后的值比1还小的话，是不能移动的。    x方向和z方向上的位置变换与y方向上的相似，这里就不再给出，此时运行程序，通过按下XYZ键可以改变光源的位置，也可以看的相应的阴影。    图8 改变光源位置 |

深圳大学学生实验报告用纸

|  |
| --- |
| 实验结论：  本次实验成功实现了OpenGL绘制窗口，并通过读入三维场景文件，绘制了具有灰色背景的场景和模型。通过设置相机参数，实现了从不同位置和角度以正交或透视投影方式观察场景。光照和材质效果得以添加，成功实现了Phong光照模型，增强了物体的真实感和观感。  在阴影效果方面，根据光源位置，自定义了投影平面（y=0平面），成功计算并生成了阴影投影矩阵。通过该阴影投影矩阵，为三维物体生成了阴影，增强了场景的逼真度。  通过交互控制光源位置的实现，使用鼠标点击等方式成功地更新了光源位置，同时动态更新了光照效果和三维物体的阴影。这使得用户能够实时调整光源位置，从而更直观地感受到光照对场景的影响，并观察阴影在物体上的动态变化。  综合而言，实验成功实现了一个包括绘制场景、设置相机、添加光照和材质效果、添加阴影效果以及通过交互控制光源位置的完整OpenGL渲染系统。这为进一步研究和开发基于图形学的交互式三维场景提供了基础。 |
| 指导教师批阅意见：  成绩评定：  指导教师签字：  年 月 日 |
| 备注： |

注：1、报告内的项目或内容设置，可根据实际情况加以调整和补充。

2、教师批改学生实验报告时间应在学生提交实验报告时间后10日内。