**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： 胡瑞珍**

**报告人： 学号： 班级：**

**实验时间： 2024 年 11 月 12 日**

**实验报告提交时间： 2024 年 11 月 12 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：   1. **读入三维场景文件**  * **读入off文件**   以下代码实现了从指定的 .off 文件读取三维模型数据并将其加载到内存中的功能。在 TriMesh::readOff 函数中，首先检查文件名是否为空，并尝试打开文件。如果文件成功打开，则读取文件头信息，包括顶点数、面片数和边数。然后根据顶点数读取每个顶点的坐标，并将其存入 vertex\_positions 向量（注意，由于后续需要将阴影投影到y=0平面，所以这里需要把模型整体向y轴正方向移动适当距离，以避免阴影和模型重叠，在代码中移动距离设置为0.7）。同时，将顶点坐标的副本也存入 vertex\_colors 向量。接着，依据面片数读取每个面片的数据（即面片所包含的顶点索引），并将其存入 faces 向量。最后，调用 storeFacesPoints 函数来处理面片的数据。     * **切换不同off模型**   在主程序中，通过键盘输入响应事件，用户按下不同的键时，会加载不同的 .off 文件（如 sphere.off、Pikachu.off、Squirtle.off、sphere\_coarse.off），并初始化模型显示。每次加载新模型时，都会创建一个新的 TriMesh 对象，读取文件并渲染对应的三维模型。     * **设置背景**   将主函数中init()的glClearColor设置为（0.5，0.5，0.5，0.0），即设置背景为灰色。     * **效果展示**        1. **设置相机**  * **更新相机位置和观察方向**   以下这段代码实现了更新相机位置和观察方向的功能。相机的位置是基于球坐标系计算的，其中 radius 决定相机到原点的距离，upAngle 和 rotateAngle 控制相机的俯仰角度和水平旋转角度。首先，up 向量的 y 分量根据 upAngle 的值进行调整，如果角度超过一定范围，则改变相机的上方向。然后，通过 upAngle 和 rotateAngle 计算出相机在三维空间中的位置 (eye 向量)，并将目标点 at 固定在 (0.0, 0.7, 0.0)，这表示相机始终注视一个位于 y = 0.7 的点（这里目标点的y轴量需要对应前面模型向y轴正方向移动的距离）。最终，这些信息将用于更新视图矩阵，以确保渲染时相机的视角是准确的。     * **计算相机的视图矩阵**   以下这段代码实现了通过 getViewMatrix 和 lookAt 函数计算相机的视图矩阵。视图矩阵用于将世界坐标系中的物体转换到相机的视角下，从而确定相机的观察方向和位置。lookAt 函数通过计算视点（eye）和目标点（at）之间的向量来确定相机的朝向（前向向量 n），并通过叉积计算出相机的右向量（u）和上向量（v），确保这三者是正交的。  接着，通过将这些向量（u、v 和 n）与相机位置的点积，构造视图矩阵的前三列，表示相机的旋转部分，而第四列则是齐次坐标，代表相机的位置偏移。最后，矩阵被转置，以符合 OpenGL 中列主序的存储方式，再返回矩阵。  getViewMatrix 只是调用 lookAt 函数来返回最终的视图矩阵，该矩阵可以用来更新渲染中的视图变换，确保物体以相机的位置和朝向正确地显示。       * **计算相机的投影矩阵**   以下这段代码实现了相机的投影矩阵计算功能，支持正交投影和透视投影。getProjectionMatrix 函数根据传入的 isOrtho 参数决定使用正交投影还是透视投影。  当 isOrtho 为 true 时，ortho 函数会计算并返回一个正交投影矩阵。正交投影矩阵是通过平行投影的方式，将物体按比例缩放到视图空间中。正交投影的特点是不会有透视缩小效果，所有物体在视图中的大小保持一致。矩阵的构造基于传入的左、右、下、上、近远裁剪面的距离，计算出对应的投影矩阵。  当 isOrtho 为 false 时，perspective 函数会计算并返回一个透视投影矩阵。透视投影矩阵模拟人眼或相机的视角，物体离相机越远，显得越小，产生透视缩小效果。矩阵的构造基于视场角（fov）、宽高比（aspect）、以及近远裁剪面的距离，计算出对应的透视投影矩阵。  两种投影矩阵都通过转置操作转换为列主序格式，以符合 OpenGL 使用的矩阵存储方式。最终，getProjectionMatrix 函数根据isOrtho返回相应的投影矩阵。         * **控制相机**   以下这段代码实现了键盘输入事件的处理，通过键盘按键控制相机的各项参数，例如相机的旋转角度、视野范围、相机位置和投影方式等。       * **相机视角和投影设置**   在 display 函数中，首先清除屏幕和深度缓冲区，然后调用相机的 updateCamera 方法更新相机参数，再通过 getViewMatrix 和 getProjectionMatrix 获取相机的视图矩阵和投影矩阵。通过这些矩阵来进行相机视角和投影设置，以保证渲染的场景正确显示。       * **效果展示**   左边为正交投影，右图为透视投影。    正交投影下调整相机的水平旋转角度、相机的俯仰角度、相机的缩放比例。    透视投影下调整相机的半径、相机的视野角度、视野宽高比。       1. **添加光照和材质效果**  * **设置光源位置、光源和模型材质参数**   以下这段代码设置了光源（注意，由于后续要将阴影投影到y=0平面，所以需要将光源向y轴正方向移动适当距离，这里移动量为2.0）和和模型材质的属性。     * **将物体的材质属性和光源信息传递给着色器**   以下这段代码的作用是将物体的材质属性和光源信息传递给着色器程序，以便渲染时进行光照计算。首先，代码使用 glUniform3fv 将相机的位置传递给着色器的 eye\_position uniform 变量，然后获取物体的环境光、漫反射光、镜面反射光以及高光系数，并通过 glUniform4fv 和 glUniform1f 函数将这些材质属性传递给着色器。接着，从光源对象中获取光源的环境光、漫反射光、镜面反射光和光源位置，使用 glUniform4fv 和 glUniform3fv 将光源信息传递给着色器。这样，着色器就可以根据传入的材质和光源数据进行正确的光照计算，渲染出物体的光照效果。  在主函数的display函数中，将材质和光源数据传递给着色器。       * **基于Phong光照模型的片元着色器**   以下这段代码实现了基于Phong光照模型的片元着色器，在计算片元颜色时考虑了环境光、漫反射光、镜面反射光和阴影效果。首先，如果启用了阴影（isShadow == 1），则输出纯黑色，否则，计算光照贡献。法向量 N 是顶点法线的归一化结果，视线向量 V 是从片元到眼睛的方向，光源向量 L 是从片元指向光源的方向，反射向量 R 是从表面反射到观察方向的向量。接着，计算环境光、漫反射光和镜面反射光的贡献，分别通过环境光分量 I\_a、漫反射光分量 I\_d 和镜面反射光分量 I\_s 来实现。漫反射部分根据光源与法线的点积计算，镜面反射部分则考虑了反射向量与视线向量的关系。最后，所有光照分量相加并设置透明度为 1，得到最终的片元颜色。如果光源在背面，则去除镜面反射的贡献。     * **效果展示**        1. **添加阴影效果**  * **创建正方形平面**   以下这段代码用于创建和配置一个平面对象，并为其设置多个属性。首先，使用 generateSquare 函数生成一个正方形平面，并设置其颜色为白色。接着，平面被旋转了-90度绕X轴（注意正方形平面翻转角度，要使得正方形平面的法向量朝y轴正方向，否则渲染时由于正方形平面的法向量朝y轴负方向，光源在正方形平面背面，平面渲染缺少镜面反射的贡献），然后平面的位置被稍微下移（由于后续会将物体阴影投影到y=0平面，为了避免正方形平面与阴影重叠，故将正方形平面略微向y轴负方向移动0.001）。平面大小被放大为原来的五倍。此外，平面的光照属性也被配置，包括环境光、漫反射光、镜面反射光以及高光系数。最后，调用 bindObjectAndData 函数将平面的数据和着色器绑定到OpenGL对象上，为后续的渲染做准备。     * **绘制正方形平面**   以下这段代码用于绘制一个平面对象，首先绑定平面的顶点数组对象（VAO）并激活相应的着色器程序。然后，获取并传递平面的模型变换矩阵、视图矩阵和投影矩阵到着色器中，用于在正确的位置渲染平面。接着，传递阴影开关值，0 表示不启用阴影效果。接下来，调用 bindLightAndMaterial 函数将光源和材质的信息传递到着色器中，确保光照效果正确计算。最后，使用 glDrawArrays 函数绘制平面，采用三角形图元进行渲染。     * **绘制阴影**   以下这段代码首先定义了一个函数 getShadowProjectionMatrix() 用于计算光源的阴影投影矩阵。该矩阵基于光源在世界坐标系中的位置以及其模型矩阵生成，计算的结果用于将物体的投影映射到一个平面（Y=0 平面）。接着，在绘制阴影的部分，首先从光源对象中获取计算出的阴影矩阵，并将其转置以适应OpenGL的列主序存储。然后将这个阴影矩阵与物体的模型矩阵相乘，得到最终的阴影变换矩阵。接着，这些矩阵被传递到OpenGL着色器中，使用 glUniformMatrix4fv 函数传递模型、视图和投影矩阵。通过设置 isShadow 变量为1，标记为阴影渲染，最后使用 glDrawArrays 绘制该阴影。       * **效果展示**        1. **交互控制光源位置并更新阴影**  * **鼠标点击控制光源位置**   以下这段代码定义了一个鼠标按钮回调函数，当鼠标左键被按下时触发。首先，通过 glfwGetCursorPos 获取当前鼠标在窗口中的坐标。然后，通过窗口的宽度和高度将鼠标位置转换为标准化坐标系（即X和Y轴的值范围在 -1 到 1 之间）。接着，获取当前光源的位置并更新其X和Z坐标为鼠标在窗口中的标准化位置。最后，使用 light->setTranslation 设置新的光源位置，从而通过鼠标点击来改变光源的位置。     * **键盘调整材质属性和光源位置**   以下这段代码实现了通过键盘输入调整材质属性（环境光、漫反射、镜面反射、高光系数）和光源位置的功能。adjustMaterialProperty 函数根据传入的参数调整不同的属性。每个属性（如环境光、漫反射、镜面反射、高光系数）都有不同的处理逻辑，允许通过键盘按键对每个属性进行调整。当按下特定的键（如1-9、A、S、D、0）时，adjustMaterialProperty 会根据按键来调整对应的属性，例如增加或减少光源位置的坐标，或者增减材质的不同属性。同时，按下 - 键会将所有材质属性重置为默认值。         * **效果展示** |

深圳大学学生实验报告用纸

|  |
| --- |
| 实验结论：    通过本次实验，我成功地掌握了OpenGL在三维场景中的光照与阴影处理技术。实验过程中，我首先学习了如何读取并绘制三维场景文件，并通过键盘输入实现了模型的加载与显示。接着，我设置了相机并添加了交互功能，使得可以从不同位置和角度观察场景，包括正交和透视投影方式。  在光照效果方面，我实现了Phong光照模型，并能够根据物体材质属性进行渲染。通过自定义投影平面，我计算了阴影投影矩阵，并成功为三维物体生成了阴影。此外，我还实现了使用鼠标点击控制光源位置的功能，这使得光照效果和物体阴影能够实时更新，增强了场景的真实感。  在技术实现上，我深入理解了场景坐标系转换过程中的矩阵计算方法，包括视图矩阵和投影矩阵的计算。通过键盘和鼠标的交互控制，我能够动态调整相机参数和光源位置，从而实时观察到不同光照和阴影效果对场景渲染的影响。  总体来说，本次实验不仅加深了我对OpenGL和三维图形编程的理解，而且提高了我解决实际问题的能力。通过实践，我学会了如何将理论知识应用到具体的图形编程任务中，为未来在计算机图形学领域的进一步学习和研究打下了坚实的基础。 |
| 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。