计算机图形学报告

58122310 唐梓烨

在一个学期的实验中，我们小组完成了4次实验和最终的综合实验，从零一步步搭建出了一个还说的过去的画面。

实验内容：

在第1次实验中，我们学习了OpenGL编程的基本规范，完成了一个简单的二维卡通图像。

在第2次实验中，我们进入三维的世界，因此我们重新搭建，通过导入模型和自己定义三维图形的顶点坐标，展示了一个鸡块和棒棒糖的组合画面，并让它们旋转平移动了起来。

在第3次实验中，我们学习了阴影和明暗，因此在第2次实验的基础上，我们添加了光源，通过光源和光照函数的实现，我们的模型上有了明暗变化。

在第4次实验中，我们给模型加上了纹理，通过在片元着色器中混合纹理颜色和光照颜色，实现了纹理和光照的结合，模型和背景也更加生动。

在最终的组合实验中，我们给画面加上了天空盒，并实现了场景漫游。用天空盒替换原来的黑色背景后明显更有真实感了，简单的第一人称场景漫游也给画面加上了一些交互要素。

交互按键标注：

1. 相机移动控制：

- `W` – 向前移动
- `S` – 向后移动
- `A` – 向左平移
- `D` – 向右平移
- `空格键(SPACE)` – 向上移动
- `SHIFT` – 向下移动

2. 视角控制：

- `↑(UP)` – 向上看
- `↓(DOWN)` – 向下看

- `←(LEFT)` – 向左看
- `→(RIGHT)` – 向右看

3. 场景控制：

- `ENTER` – 按住时光源旋转，松开停止
- `+` – 增加场景中棒棒糖的数量（最多4个）
- `–` – 减少场景中棒棒糖的数量（最少1个）

设计思路和原理：

下面是我认为在实验中的关键点的设计思路：

1. 对物体运动的矩阵的设计

   基于计算机图形学中的坐标变换理论，需要将物体从局部坐标系转换到世界坐标系(M)，再转换到观察坐标系(V)，最后投影到屏幕(P)

   模型矩阵(M)控制物体本身的变换：位置、旋转、缩放

   视图矩阵(V)模拟相机移动，本质是将世界坐标系中的物体转换到相机坐标系

   投影矩阵(P)将3D场景投影到2D平面，模拟人眼或相机的视觉效果

   矩阵乘法顺序很重要：P*(V*M)，因为变换是从右向左应用的

2. 对光照模型的设计

   基于Phong光照模型，将光照分解为三个组成部分：

   - 环境光(Ambient)：模拟间接光照，使物体不会完全黑暗
   - 漫反射光(Diffuse)：模拟光线照射到粗糙表面的散射效果
   - 镜面反射光(Specular)：模拟光线在光滑表面的反射，产生高光

   通过设置以下四个参数就能为每个物体设置不同的光照材质：环境光反射系数；漫反射系数；镜面反射系数；光泽度。

   在片元着色器中

   a. 环境光计算：

   `全局环境光× 材质环境反射系数 + 光源环境光 × 材质环境反射系数`

   用于模拟来自所有方向的间接光照，不考虑位置和方向，确保物体在阴影处仍可见。

b. 漫反射计算：

光源方向(L)和表面法向量(N)的点积决定漫反射强度，即用光源方向表面法向量的夹角来模拟现实世界的反射强度。

c. 镜面反射计算：

反射方向(R)和视线方向(V)的点积决定高光强度，反射强度和上面相同，当反射方向射入摄像机便形成了高光，光泽度越大，高光区域越小越集中。

3. 对法向量和曲面细分的设计

从OpenGL光照实现原理可以看出，模型的法向量和曲面细分是十分重要的。

a. 法向量的物理意义：

表示物体表面的朝向，是光照计算的基础，用于计算光线入射角，决定漫反射强度；用于计算反射方向，决定镜面反射效果。

b. 曲面细分的必要性：

增加多边形数量，使曲面更光滑；提供更精确的法向量，改善光照效果，使其更加真实，特别是对球体等曲面物体很重要。

4. 对纹理映射系统的设计

纹理坐标系统原理：

UV坐标范围在[0,1]之间，独立于物体实际大小，U坐标对应水平方向，V坐标对应垂直方向。通过为每个顶点指定纹理坐标，GPU通过插值计算片段的纹理坐标，根据纹理坐标从纹理图像采样颜色，我们便实现了纹理映射。

5. 对纹理和明暗结合的设计

a. 基本原理：

纹理提供基础颜色信息，光照计算提供明暗变化，通过两者相乘得到最终效果。通过纹理采样获取物体表面的基础颜色信息，纹理颜色表示表面对不同波长光的反射率。再结合光照计算，光照颜色表示入射光的强度和颜色，将两者相乘得到最终反射光的颜色和强度。

b. 同时我们还实现了渲染模式控制：

通过useLight开关控制渲染模式，对于天空盒等不需要光照计算的物体，我们只使用纹理颜色。

6. 对天空盒的设计

为什么使用天空盒?

目的是创造无限远的环境错觉，提供场景的背景上下文，增加场景的真实感。

使用立方体贴图，每个面对应一张图片，绘制时关闭深度测试让它永远绘制在其他物体之后，并移除天空盒视图矩阵的位移部分，使天空盒跟随相机移动。

7. 对场景漫游系统设计
    a. 相机位置和视角基于：位置(Position)——相机在世界空间的位置，方向(Front)——相机朝向的方向，上向量(Up)——定义相机的垂直方向，这三个lookat参数决定。
    b. 视角控制系统：使用水平角(Horizon)控制左右视角，垂直角(Vertical)控制上下视角，两个角度共同决定观察方向，使用三角函数计算实际的方向向量。
    c. 移动控制系统：通过键盘改变相机坐标，WASD控制前后左右移动，空格和Shift控制上下移动。

OpenGL代码基本框架

在实验过过程中，老师提供的示例程序对我提供了很大的帮助，下面我想总结一下OpenGL代码基本框架，便于我后续使用能再次快速上手。

OpenGL中的重要专有名词：

1. 缓冲区相关

- VAO (Vertex Array Object)：顶点数组对象，存储顶点属性配置的容器，可以保存多个顶点属性的格式和对应VBO的引用。
- VBO (Vertex Buffer Object)：顶点缓冲对象，在GPU内存中存储大量顶点数据的缓冲区，可以包含位置、颜色、纹理坐标等数据。
- EBO/IBO (Element Buffer Object/Index Buffer Object)：索引缓冲对象，存储顶点索引的缓冲区，用于复用顶点数据，减少内存使用。

2. 着色器相关

- Shader（着色器）：在GPU上运行的小程序，用于处理图形渲染管线中的特定阶段。
- Vertex Shader（顶点着色器）：处理单个顶点的着色器，负责顶点坐标变换和顶点属性计算。
- Fragment Shader（片段着色器）：处理像素颜色的着色器，也称为像素着色器，决定每个像素的最终颜色。

3. 矩阵变换相关

- Model Matrix（模型矩阵）：将物体从局部坐标变换到世界坐标，包含平移、旋转、缩放操作。
- View Matrix（视图矩阵）：将世界坐标变换到相机视角，定义观察者的位置和方向。
- Projection Matrix（投影矩阵）：将3D场景投影到2D平面，包括透视投影和正交投影两种。
- MVP Matrix：Model-View-Projection矩阵，三种矩阵的组合变换。

4. 纹理相关

- Texture（纹理）：用于给3D模型表面贴图的2D图像，可以存储颜色、法线等信息。
- Texture Coordinates（纹理坐标）：也称UV坐标，定义纹理如何映射到3D模型表面。

5. 渲染相关

- Frame Buffer（帧缓冲区）：存储最终渲染图像的内存区域，包含颜色缓冲、深度缓冲等。
- Depth Buffer（深度缓冲区）：存储每个像素的深度信息，用于处理3D物体的遮挡关系。

6. 其他重要概念

- Uniform变量：着色器程序中的全局变量，在渲染过程中保持不变。
- Attribute（属性）：顶点的各种属性数据，如位置、颜色、法线等。
- Primitive（图元）：基本绘制单位，包括点、线、三角形等。

OpenGL开发的API框架

1. init函数的作用

- 初始化着色器程序并链接，加载纹理
- 设置相机初始位置和方向，设置物体初始位置，设置投影方式，初始化光照参数
- 设置顶点数据：包括创建VAO(顶点数组对象)和VBO(顶点缓冲对象)，指定顶点位置、纹理坐标、法线等数据，配置顶点属性指针，设置索引缓冲对象(EBO)

2. display函数的作用

- 缓冲区管理：包括清除颜色缓冲区和深度缓冲区，设置清除颜色，启用/禁用深度测试。
- 着色器程序使用：激活着色器程序，，获取uniform变量位置，传递uniform值到着色器。
- 矩阵变换：包括计算和更新模型矩阵(位移、旋转、缩放)，更新视图矩阵(相机位置和方向)，计算MVP矩阵，传递变换矩阵到着色器。
- 纹理绑定与使用：包括激活纹理单元，绑定纹理对象，设置纹理uniform采样器。

3. main函数的作用

- GLFW窗口管理：包括初始化GLFW，创建窗口和OpenGL上下文，设置窗口属性和回调函数，处理窗口事件。
- 维护主渲染循环：包括处理输入事件，更新场景状态，调用渲染函数，交换缓冲区。
- 资源清理：包括释放OpenGL资源，清理GLFW资源，正确退出程序。

碰到的问题总结

1. 在实验中碰到的耗费了最多时间的问题是，在绘制自定义的三维物体时，总是出现绘制残缺的情况。一开始检查了很多地方都没检查出来问题，最后发现是在绘制时本应该使用 `glDrawArrays(GL_TRIANGLES, 0, myModel.getNumIndices())` ，结果错误的使用了 `myModel.getNumVertices()` 导致了错误，由于少绘制了顶点，从而导致模型残缺。
2. 遇到的第二个问题是，在绘制阴影时，发现阴影效果不明显，检查了法向量的设置也没有发现什么问题。最终发现是因为没有绑定法向量的VBO导致的，修改后因为各面的法向量不同，使得阴影更有层次感了。
3. 遇到的第三个问题就是，绘制天空盒时天空盒不显示，最终发现是由于深度测试的问题，由于天空盒应该显示在所有其他物体的最后面，我们可以在渲染时关闭深度测试，或者使用使用GL_LEQUAL深度测试函数来渲染最远端（最深）的物体。
4. 最新问题：在最小化窗口时出现runtime error，发现原因是由于最小化，在计算视图比例aspect=height/width时出现了除以0的情况，增加了一个异常判断排除后，问题得以解决。

个人总结

在本次实验中，我和小组成员一同合作，利用所学的知识，完成了一个小demo，至少是能显示出简单的三维图像了。在实验中有困难也有收获，对于我以后尝试进行游戏开发有很大的帮助，等我有时间了，应该会进一步学习，从计算机图形学这门课程入门，做一点自己的东西。