B类大作业实验报告

冯志远 2024311588

B.1 实验

实验内容

1. 绘制出星星旋转并扩散的效果（OpenGL 15 分）
   - **1)** 多个星星绕着一个中心旋转；
   - **2)** 每个星星具有不同的颜色；
   - **3)** 星星的效果呈现为螺旋扩散，逐渐向外圈扩展；
   - **4)** 至少实现以下螺线效果之一：
     - 阿基米德螺线 （**本实验选择的**）
     - 对数螺线
     - 费马螺线
   - **5)** 使用的星星图片见附件 `Star.bmp` ：

实验报告

实验环境

- **编程语言**：C++
- **开发环境**：Visual Studio Code
- **依赖库**：GLFW、GLAD、GLM
- **运行平台**：Windows

实验步骤

1. 环境配置

- 配置 OpenGL 环境，安装并链接必要的库：GLFW、GLAD 和 GLM。
- 在项目目录中包含 `Star.bmp` 文件，用于星星的纹理加载。
- 创建一个宽 800 像素、高 600 像素的窗口，用于显示旋转星星的动画效果。

2. 着色器设计

- **顶点着色器**：
  - 使用 MVP 矩阵（模型矩阵、视图矩阵、投影矩阵）对每个星星的位置和旋转进行变换。
  - 顶点着色器通过纹理坐标将纹理传递到片段着色器中。
- **片段着色器**：
  - 实现纹理映射，显示星星的纹理。
  - 使用 `uniform` 变量传递颜色，实现每个星星的颜色独立变化。
  - 增加亮度调整因子，使星星看起来更亮。

3. 纹理加载

- 使用 `stb_image` 库加载 `Star.bmp` 纹理文件。
- 设置纹理的采样参数（如重复方式、过滤方式），并生成多级纹理（Mipmap）。

4. 星星位置与颜色初始化

- 使用一个动态数组管理星星的生命周期，每颗星星都有独立的初始位置和颜色：
  - **位置**：基于阿基米德螺线公式 `r = b * θ` 计算，星星在螺旋路径上运动。
  - **颜色**：为每颗星星随机生成 RGB 值，使其颜色多样化。
- 初始生命周期为 `0`，星星从中心点开始扩散。

5. 循环渲染

1. **更新星星的位置**：
   - 通过时间步长（`deltaTime`）逐帧更新星星的位置，模拟阿基米德螺线的运动轨迹。
   - 控制星星的最大可视半径，超出范围的星星不再绘制。
2. **绘制每颗星星**：
   - 使用模型矩阵对每颗星星进行平移和缩放变换，将其移动到指定位置并设置大小。
   - 设置颜色 `uniform`，使每颗星星具有独立的颜色。
3. **动态添加新星星**：
   - 每秒添加一个新星星，初始位置为中心，颜色随机生成。
4. **交换缓冲区**：
   - 在每一帧结束时调用 `glfwSwapBuffers` 刷新窗口，显示更新后的动画效果。

6. 资源清理

- 程序退出前，释放顶点数组对象 (VAO)、顶点缓冲对象 (VBO) 和着色器相关资源。

- 调用 `glfwTerminate` 退出 OpenGL 环境，销毁所有窗口和上下文资源。

实验结果

1. 程序运行后，窗口中显示一个动态旋转的星星螺旋动画。
2. 星星从中心沿着阿基米德螺线逐渐扩散，形成动态的螺旋运动效果。
3. 每颗星星的颜色随机，整个动画具有丰富的视觉效果。
4. 程序运行稳定，关闭窗口后所有资源被正确释放。

实验总结

通过本实验，我实现了一个以阿基米德螺线为基础的动态星星动画效果。实验过程中，掌握了以下技能：

1. **OpenGL 基础操作**：学习了顶点缓冲区对象 (VBO)、顶点数组对象 (VAO) 的使用，熟悉了纹理加载与采样的基本操作。
2. **着色器编程**：掌握了顶点着色器与片段着色器的基本编写方法，理解了矩阵变换与颜色控制的实现过程。
3. **动态效果实现**：通过时间步长模拟星星的旋转与扩散运动，学习了如何控制对象的动态行为。

本实验完成了一个具有实时交互与视觉效果的图形学案例，为进一步学习复杂图形渲染打下了基础。在调试过程中，解决了纹理加载、坐标变换与动态资源管理等问题，提升了对 OpenGL 和图形学的实践能力。

B.3 实验

实验内容

三维模型的显示（OpenGL 15 分）

1. **a.** 使用半边结构（half-edge data structure）读入一个三维网格模型，并在屏幕上显示；
2. **b.** 支持以下四种显示模式：
   - **仅显示网格效果**
   - **仅显示顶点效果**（顶点颜色由程序自行设定）
   - **仅显示三角面效果**（面颜色由程序自行设定）
   - **同时显示面和边的效果**
3. **c.** 要读入的模型文件见附件 `eight.uniform.obj`；
4. **d.** 四种显示方式的效果见附件 `模型显示.png` 。

实验环境

1. **编程语言**：C++
2. **开发环境**：Visual Studio Code
3. **依赖库**：GLFW、GLAD、GLM、Assimp
4. **运行平台**：Windows

实验步骤

1. 环境配置

- 配置 OpenGL 环境，安装并链接 GLFW、GLAD 和 GLM 库，用于图形学渲染和矩阵操作。
- 使用 Assimp 库解析 `eight.uniform.obj` 文件，提取模型的顶点、法线和面信息。
- 创建一个 800×600 像素的窗口，并启用深度测试以确保正确的 3D 渲染。

2. Half-Edge 数据结构设计

- 实现半边数据结构，用于存储和操作模型的网格结构：
  - **顶点（Vertex）：** 存储顶点的坐标和与该顶点相关的一条半边。
  - **半边（HalfEdge）：** 存储与网格拓扑结构相关的指针，包括起点顶点、下一条边、对偶边、所属面及关联的边。
  - **边（Edge）：** 存储一条半边的指针。
  - **面（Face）：** 存储与该面相关的颜色和一条半边的指针。

3. OBJ 模型加载与 Half-Edge 构建

- 使用 Assimp 加载 `eight.uniform.obj` 文件，提取模型的顶点和三角面索引数据。
- 通过以下步骤构建 Half-Edge 数据结构：
  - 遍历每个三角面，创建 3 条半边，并将它们链接为一个环。
  - 为每条半边创建对应的 `Edge` 对象。
  - 使用 `std::unordered_map` 查找并设置对偶边（twin）。
- 为模型的每个面随机分配一个颜色值，用于渲染。

4. 着色器设计

- **顶点着色器**：
  - 实现 MVP（模型-视图-投影）矩阵变换，将顶点从模型空间转换到屏幕空间。
  - 接收顶点位置，传递到片段着色器。
- **片段着色器**：
  - **面着色（Faces）：** 接收面颜色并将其作为片段颜色。

- **边着色（Edges）：** 使用固定黑色绘制网格边框。
- **顶点着色（Vertices）：** 为顶点分配固定的红色。

5. 数据绑定与 VAO/VBO 配置

- **面数据（Faces）：**
  - 遍历每个面，提取其所有顶点位置，存储到一个数组中，并绑定到 VAO/VBO。
- **边数据（Edges）：**
  - 遍历每条边，提取起点和终点位置，存储到一个数组中，并绑定到 VAO/VBO。
- **顶点数据（Vertices）：**
  - 提取每个顶点的位置，存储到一个数组中，并绑定到 VAO/VBO。

6. 循环渲染

1. **用户选择显示模式：** 提供以下四种显示模式：
   - **网格显示（Wireframe）：** 仅绘制网格边框；
   - **顶点显示（Vertices）：** 仅绘制顶点；
   - **面显示（Faces）：** 仅绘制模型的三角面；
   - **面与边显示（Faces and Edges）：** 同时绘制面和网格边框。
2. **动态渲染：**
   - 在每一帧中，根据用户选择的模式调用不同的着色器程序：
     - **网格显示：** 使用线框模式（ `GL_LINES` ）绘制边框。
     - **顶点显示：** 绘制顶点（ `GL_POINTS` ）。
     - **面显示：** 绘制面（ `GL_TRIANGLES` ）。
     - **面与边显示：** 同时调用面绘制和网格绘制函数。
3. **相机视角与投影：**
   - 通过 `glm::lookAt` 设置相机位置，使视角位于模型上方并观察模型中心。
   - 使用 `glm::perspective` 设置透视投影矩阵，确保正确的 3D 投影效果。

7. 资源清理

- 程序退出前，释放 VAO、VBO 和着色器资源，销毁窗口。
- 清理 Half-Edge 数据结构中动态分配的顶点、半边、边和面对象。

实验结果

1. **OBJ 文件加载：** 成功加载 `eight.uniform.obj` 模型，并构建对应的 Half-Edge 数据结构。

2. **显示效果：**
    - **网格显示（Wireframe）：** 清晰显示模型的边框结构，边颜色为黑色。
    - **顶点显示（Vertices）：** 显示模型的所有顶点，颜色为红色。
    - **面显示（Faces）：** 显示模型的三角面，颜色为浅灰色。
    - **面与边显示（Faces and Edges）：** 同时显示面和边，面为浅灰色，边为黑色。

实验总结

通过本实验，熟悉了以下技术：

1. **Half-Edge 数据结构的实现：** 学习了网格数据的存储方式及其高效操作方法，掌握了半边数据结构的基本原理和构建过程。
2. **OBJ 文件的加载与解析：** 使用 Assimp 库解析 OBJ 文件，提取顶点和面信息，为网格渲染提供数据支持。
3. **OpenGL 渲染技术：** 学会了通过 VAO 和 VBO 绑定顶点和边数据，并使用不同着色器实现多种显示模式（面、边和顶点）。
4. **动态渲染与模式切换：** 实现了基于键盘输入的模式切换功能，进一步理解了 OpenGL 的动态渲染流程。

本实验提供了对 3D 模型加载与显示的全面实践，为深入学习复杂网格编辑和渲染技术打下了坚实基础。

B.4 实验

实验内容

三维模型的交互与光照效果（OpenGL 15 分）

- **a.** 对第 3 题读入的模型使用鼠标进行旋转、缩放操作；
- **b.** 利用光照模型实现简单光照效果，使用物体材质和环境光、漫反射、镜面反射结合产生光照效果；
- **c.** 光照参考效果见附件“模型光照效果.jpg”。

实验环境

1. **编程语言**：C++
2. **开发环境**：Visual Studio Code
3. **依赖库**：GLFW、GLAD、GLM
4. **运行平台**：Windows

实验目的

实现一个三维网格模型的交互与光照效果，通过鼠标操作实现模型的旋转和缩放，结合光照模型（环境光、散射光和镜面光）为模型添加材质效果。

实验步骤

1. 环境配置

- 配置 OpenGL 环境，安装并链接 GLFW 和 GLAD，用于窗口管理和 OpenGL 函数加载。
- 使用 GLM 库完成矩阵变换和法向量计算。
- 初始化窗口大小为 800×600，启用深度测试以确保三维场景的正确显示。

2. 光照模型设计

- **顶点着色器**：
  - 接收模型的顶点位置和法向量，通过 `model`、`view` 和 `projection` 矩阵完成顶点的坐标变换。
  - 使用 `transpose(inverse(model))` 计算法向量的正确变换，传递到片段着色器。
- **片段着色器**：
  - **环境光（Ambient Light）**：
    - 模拟全局光源影响，计算公式：`ambient = ambientStrength * lightColor`。
  - **散射光（Diffuse Light）**：
    - 使用 Lambert 光照模型，根据光源方向与法向量夹角调整亮度，计算公式：
      `diffuse = diffuseStrength * max(dot(norm, lightDir), 0.0) * lightColor`。
  - **镜面光（Specular Light）**：
    - 使用 Blinn-Phong 模型模拟高光效果，根据视角方向和反射方向计算反射强度，计算公式：
      `specular = specularStrength * pow(max(dot(viewDir, reflectDir), 0.0), shininess) * lightColor`。
  - 将三种光的叠加结果与材质颜色相乘，作为片段的最终颜色输出。

3. 半边数据结构构建

- 使用半边数据结构解析 `.obj` 文件，存储顶点、边和面的关系。
- **顶点数据**：包含位置和法向量。
- **法向量计算**：为每个面计算法向量，并将其累加到面对应的顶点，最终为每个顶点生成归一化的法向量。

4. 用户交互

- **鼠标控制**：
  - 捕获鼠标拖动事件，更新 `rotationX` 和 `rotationY` 变量，实现模型绕 x 和 y 轴旋转。
  - 捕获滚轮事件，更新 `fov` （视野角度），实现模型的缩放。
- **控制台交互**：
  - 运行程序时，允许用户输入光照参数（环境光、散射光、镜面光强度和高光值）。
  - 提供选项选择模型颜色，支持红、绿、蓝、黄、紫、青和白色等预设。

5. 渲染流程

1. 初始化渲染管线，生成 VAO 和 VBO 存储顶点位置和法向量数据，绑定到着色器程序的输入变量。
2. 通过 `model` 矩阵更新模型的旋转和缩放变换，结合固定的 `view` 和 `projection` 矩阵实现三维渲染。
3. 在片段着色器中，根据用户设置的材质和光源参数，逐片段计算颜色并显示。

实验结果

1. **显示效果**：
   - 加载并显示 `eight.uniform.obj` 模型，模型可绕 x 和 y 轴旋转，支持滚轮缩放操作。
   - 光照效果显著，不同的环境光、散射光和镜面光参数影响模型的明暗程度。
   - 材质颜色由用户控制，模型表面表现出动态光照效果。
2. **光照演示**：
   - **环境光强度**：增大环境光强度会使模型整体变亮，适合暗光场景。
   - **散射光强度**：增加散射光强度使模型表面随光源角度变化更加明显。
   - **镜面光强度和高光值**：增强镜面光强度和减小高光值，会使模型表现出更亮的高光点。

实验分析

1. **成功点**：
   - 成功加载并显示三维模型，实现了鼠标旋转和缩放操作。
   - 光照模型结合材质参数，展示了真实的光照效果，包括环境光、散射光和镜面光的交互影响。
   - 通过控制台交互和鼠标操作增强了程序的可玩性和灵活性。
2. **问题与解决**：
   - **问题**：法向量计算错误导致散射光效果异常。
     - **解决**：使用 `transpose(inverse(model))` 矩阵变换法向量，确保其正确性。
   - **问题**：模型过暗，细节不明显。
     - **解决**：增大环境光和散射光强度，改善视觉效果。

实验总结

通过本实验，掌握了以下关键技术：

1. 半边数据结构的实现及其在模型解析中的应用。
2. 光照模型的实现，包括环境光、散射光和镜面光的组合使用。
3. 鼠标交互操作的绑定与实现，增强了用户体验。

实验结果验证了光照模型与交互功能在三维渲染中的重要性，为进一步实现动态光源和高级效果（如阴影和点光源）奠定了基础。