计算机图形学基础 C 类作业报告

姓名/学号：钟静仪 2025403465

联系方式：zjy25@mails.tsinghua.edu.cn

日期：11/25/2025

基础实验环境

- 系统与工具：Windows, VSCode, C++
- 核心库：OpenGL，GLFW (窗口管理), GLAD (函数加载), GLM (数学运算)
- 着色器语言：GLSL 330 Core

作业 1 显示旋转的三角形

实验效果

显示一个绕 Y 轴垂直中心线进行顺时针旋转的三角形，初始颜色蓝色，用户可用键盘上的 1、2、3 分别对应红绿蓝，切换三角形的颜色。由于启用了透视投影，三角形在旋转时会有近大远小的 3D 视觉效果。

实验原理:

1. **图形渲染管线**
    - **顶点着色器 (Vertex Shader)**：负责处理顶点坐标。通过模型矩阵 (Model)、观察矩阵 (View) 和投影矩阵 (Projection) 将局部坐标转换为裁剪空间坐标 (gl_Position = projection * view * model * vec4(position, 1.0f);)
    - **片段着色器 (Fragment Shader)**：负责计算像素颜色。片段着色器接收一个来自 CPU 的 uniform 变量 triangleColor，直接将该颜色赋值给输出，实现纯色填充。
2. **坐标变换 (Coordinate Transformations)**:
    - **模型变换 (Model Transform)**：使用 glm::rotate 函数构建旋转矩阵。为了实现动画效果，旋转角度随时间 (glfwGetTime()) 变化。代码中使用负号 -(GLfloat)glfwGetTime() 配合 Y 轴 (0.0f, 1.0f, 0.0f) 实现**顺时针**旋转。

    - **观察变换 (View Transform)**：使用 glm::translate 将相机向 Z 轴负方向移动 (-3.0f)，像是往后退了几步，以便在视野中看到三角形。
    - **投影变换 (Projection Transform)**：使用 glm::perspective 创建透视投影矩阵，赋予场景深度感。
3. **交互机制 (Interaction)**：
    - 利用 glfwGetKey 接受键盘状态。
    - 当检测到特定按键（1、2、3）被按下时，更新 C++端的 triangleColor 变量。
    - 每一帧渲染循环中，通过 glUniform3fv 将最新的颜色值传递给着色器。

实验步骤

1. **初始化**：配置 GLFW 上下文及窗口，初始化 GLAD。
2. **资源加载**：编写顶点着色器 (main.vert) 和片段着色器 (main.frag)。使用 Shader 类读取、编译并链接着色器程序。定义三角形的三个顶点坐标（NDC 范围内）。创建并绑定 VAO (Vertex Array Object) 和 VBO (Vertex Buffer Object)。将顶点数据复制到缓冲内存中，并配置顶点属性指针 (Vertex Attribute Pointer)。
3. **渲染循环**：
    - **输入**：检测按键 1 (红), 2 (绿), 3 (蓝) 修改颜色变量。
    - **更新**：计算旋转矩阵（顺时针：绕 Y 轴负方向旋转）。
    - **绘制**：glUniform 传递颜色与矩阵，调用 glDrawArrays。
4. **循环结束后**，删除 VAO、VBO 和着色器程序，销毁窗口并终止 GLFW。

作业 2 绘制三角形和四边形并着色，同时旋转和平移

实验效果

左侧：不等边三角形一边逆时针自转，一边在水平方向左右往复平移。表面颜色均一（Flat 效果）。右侧：四边形（正方形）一边顺时针自转，一边在垂直方向上下往复平移。表面呈现四角颜色的平滑渐变（Smooth 效果）。

实验原理

若要让物体绕自身中心旋转，而不是绕世界坐标原点旋转，需要先将物体移回原点，旋转，再移回原位（或者在建模时就将物体中心置于原点）。本实验通过 translate(center) ->

rotate -> translate(-center) 的方式实现了绕任意中心的自转。利用 sin(time) 函数构建周期性位移向量。

着色模式 (Shading Mode):

- **Flat Shading**：使用 flat 关键字禁止插值，面片呈现单一颜色（取自第一个顶点）。
- **Smooth Shading**：默认插值模式，顶点颜色在面片表面平滑过渡。

实验步骤

1. **数据准备**：在同一 VBO 中存储左侧三角形（3 顶点）和右侧四边形（6 顶点）的数据。
2. **着色器逻辑**:
    - Vertex Shader 输出 smoothColor 和 flat flatColor。
    - Fragment Shader 根据 uniform int shadingMode 选择输出哪种颜色。
3. **渲染逻辑**:
    - **左侧三角形**：应用 sin 函数控制 X 轴左右平移，设置 shadingMode=1 (Flat)。
    - **右侧四边形**：应用 sin 函数控制 Y 轴上下平移，设置 shadingMode=0 (Smooth)。

作业 3 对正方体加载纹理

实验环境

新增库：stb_image.h (用于图像加载)。

实验效果

显示一个正方体，每一面有不同的纹理。用鼠标拖动旋转正方体。默认的纹理加载过滤方式是最近点采样，按键 1 切换到最近点采样，按键 2 切换到线性采样，按键 3 切换到 mipmap。但可能我选的 6 个图片本身是像素风格的图片，所以线性采样和 mipmap 看起来一样模糊。

实验原理

1. **纹理映射**：利用 UV 坐标将 2D 图像贴合至 3D 立方体表面。
2. **纹理滤波 (Filtering)**：
    - **Nearest**：最近邻采样，有像素感。
    - **Linear**：线性插值，边缘平滑。
    - **Mipmap**：多级渐远纹理，解决远处纹理噪点问题。
3. **鼠标交互**：通过捕捉鼠标位移量 (Offset) 计算欧拉角 (Yaw, Pitch) 来控制模型旋转。

实验步骤

1. **纹理加载**：使用 stb_image 加载 6 张 PNG 图片，生成纹理对象。
2. **立方体绘制**：
    - 构建包含位置与 UV 坐标的立方体顶点数据。
    - 在绘制每个面前绑定对应的纹理单元 (glBindTexture)。
3. **交互实现**：
    - mouseMoveCallback：计算鼠标拖拽偏移量，更新旋转角度。
    - keyCallback：按键 1/2/3 调用 glTexParameteri 切换滤波模式。