深 圳 大 学 实 验 报 告

课程名称：计算机图形学

实验项目名称：实验二 三维模型读取与控制

学院：计算机与软件学院

专业：软件工程（腾班）

指导教师：熊卫丹

报告人：洪子敬 学号：2022155033 班级：腾班

实验时间：2024 年 10 月 17 日 - 2024 年 10 月 23 日

实验报告提交时间：2024 年 10 月 20 日

教务部制

实验目的与要求：

1. 熟悉 OpenGL 三维模型的读取与处理；理解三维模型的基本变换操作；掌握鼠标键盘交互控制逻辑；掌握着色器中 uniform 关键字的使用以及数据传输方法。
2. OFF 格式三维模型文件的读取：完成对 OFF 格式三维模型文件的读取与显示，可改变物体的显示颜色。
3. 三维模型的旋转动画：结合模型进行旋转变换的过程，为模型添加自动的旋转动画。
4. 键盘鼠标的交互：通过键盘设定选择绕 x、y、z 轴进行旋转，鼠标左右键控制动画的开始与暂停。

实验过程及内容：

1. OFF 格式三维模型文件的读取

此实验提供了两个 OFF 格式的三维模型在“Models”文件中，分别对应立方体模型和牛模型，由于之前实验已经做过了关于立方体的实验，这里主要详细说明牛模型的细节。根据之前的实验，我们延续实验 2.3 的框架（**注意不能用实验 2.2 的，因为 shaders 文件中不含有旋转的代码**），不过不用其自定义类，而是与之前一样读取 OFF 文件到顶点和面片集合中，再去将它们存储到点和颜色集合中。代码如下所示：

```
// 读取OFF字符串
string str;
fin >> str;
// 读取文件中顶点数、面片数、边数
fin >> nVertices >> nFaces >> nEdges;

// 根据顶点数，循环读取每个顶点坐标，将其保存到vertices
float x1, x2, x3;
for (int i = 0; i < nVertices; i++) {
    fin >> x1 >> x2 >> x3;
    vertices.push_back(glm::vec3(x1, x2, x3));
}

// 根据面片数，循环读取每个面片信息，并用构建的vec3i结构体保存到faces
int n, y1, y2, y3;
for (int i = 0; i < nFaces; i++) {
    fin >> n >> y1 >> y2 >> y3;
    vec3i tmp(y1, y2, y3);
    faces.push_back(tmp);
}
```

HZJ

```
void storeFacesPoints()
{
    points.clear();
    colors.clear();
    glm::vec3 colorA = vertex_colors[0];
    glm::vec3 colorB = vertex_colors[1];
    glm::vec3 colorC = vertex_colors[2];
    //存储point和colors
    for (int i = 0; i < nFaces; i++) {
        vec3i index = faces[i];
        points.push_back(vertices[index.a]);
        points.push_back(vertices[index.b]);
        points.push_back(vertices[index.c]);

        colors.push_back(vertices[index.a] + glm::vec3(0.4, 0.4, 0.4));
        colors.push_back(vertices[index.b] + glm::vec3(0.4, 0.4, 0.4));
        colors.push_back(vertices[index.c] + glm::vec3(0.4, 0.4, 0.4));
        /*colors.push_back(vertex_colors[index.a]);
        colors.push_back(vertex_colors[index.b]);
        colors.push_back(vertex_colors[index.c]);*/
    }
}
```

HZJ

注意：对应牛模型的颜色，这里的是将每个顶点直接传入作为 RGB 值，使得相近位置点具有相近的颜色，同时加上些许 RGB 值，使其不会太暗。（立方体模型与之前一致）

2. 三维模型的旋转动画

此实验同样需要对三维模型进行旋转，所以我们只需要沿用 2.3 的旋转变换即可。主要的变换过程如下：（详细函数可见提交代码）

```
// 调用函数传入三种变化的变化量，累加得到变化矩阵
// 注意三种变化累加的顺序
m = glm::scale(m, scaleTheta); // 缩放
m = glm::rotate(m, glm::radians(rotateTheta.x), glm::vec3(1, 0, 0)); // 绕X轴旋转
m = glm::rotate(m, glm::radians(rotateTheta.y), glm::vec3(0, 1, 0)); // 绕Y轴旋转
m = glm::rotate(m, glm::radians(rotateTheta.z), glm::vec3(0, 0, 1)); // 绕Z轴旋转
m = glm::translate(m, translateTheta); // 平移
```

3. 键盘鼠标的交互

要求：在前面的基础上，添加键盘和鼠标的交互，使得通过键盘的设定，可以选择 x、y、z 轴哪个进行旋转，同时可以通过鼠标左右键控制模型进行自动的旋转。

思路：在键盘中添加响应，使得按不同键可以切换不同的轴；定义鼠标响应函数，点击左键标志开始自动旋转，点击右键标志停止自动旋转，注意在主函数中绑定鼠标响应函数；其次在主函数中通过判断是否自动旋转的标志来决定自动旋转；最后修改提示函数，修改为设置的键位。（注意：代码中定义了全局标志变量 autoRotation（bool 类型）以及当前旋转轴 currentRotationAxis（int 类型））

（1）键盘响应函数

在实验 2.3 的键盘响应函数上，我们去除了其他可切换的变换，留下旋转这一变换，保留键位如“Esc”（退出）、“Q”（X 轴正向旋转）、“A”（X 轴负向旋转）等方便用于手动的旋转变换，并添加了“X”用于“改变当前旋转轴为 X_AIS”、“C”用于“改变当前旋转轴为 Y_AIS”和“V”用于“改变当前旋转轴为 Z_AIS”。键位表汇总如下：

键位	作用	键位	作用
“Esc”	退出	“R”	旋转速度加快
“Q”	X轴正向旋转	“F”	旋转速度减慢
“A”	X轴负向旋转	“T”	重置旋转参数
“W”	Y轴正向旋转	“X”	改变当前旋转轴为X_AIS
“S”	Y轴负向旋转	“C”	改变当前旋转轴为Y_AIS
“E”	Z轴正向旋转	“V”	改变当前旋转轴为Z_AIS
“D”	Z轴负向旋转		

（详细请见提交代码）

（2）鼠标响应函数与自动旋转

首先通过鼠标捕获当前的动作是左键还是右键得出不同的决策，左键则设置自动旋转标志变量为 true，右键则设置为 false；函数代码如下：

```
void mouse_button_callback(GLFWwindow* window, int button, int action, int mode) {
    if (action == GLFW_PRESS)
        switch (button) {
        case GLFW_MOUSE_BUTTON_LEFT:
            currentTransform = TRANSFORM_ROTATE;
            autoRotation = true;
            break;
        case GLFW_MOUSE_BUTTON_RIGHT:
            autoRotation = false;
        }
    return;
}
```

HZJ

接着在主函数中通过 glfwSetMouseButtonCallback 函数绑定鼠标事件；

最后在主函数循环中添加自动旋转的代码逻辑，实际上就是在每次 display 前判断标志变量是否为 true，若为 true 则在当前旋转轴上更新旋转角度；代码如下所示：

```
init();
printHelp();
glEnable(GL_DEPTH_TEST);
while (!glfwWindowShouldClose(window))
{
    // 交换颜色缓冲 以及 检查有没有触发什么事件（比如键盘输入、鼠标移动等）
    if (autoRotation) {
        updateTheta(currentRotationAxis, 1);
    }
    display();

    glfwSwapBuffers(window);
    glfwPollEvents();
}
return 0;
```

HZJ

（3）修改提示函数 printHelp

将提示函数的键位改为预设定的键位，代码如下：

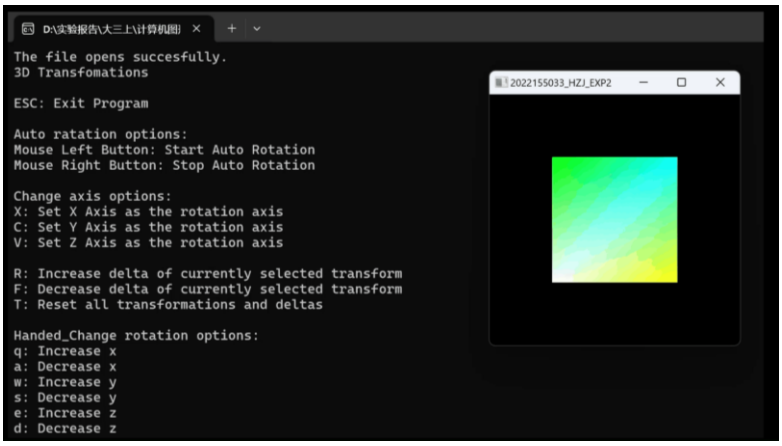
```
void printHelp() {
    printf("%s\n\n", "3D Transfomations");
    printf("ESC: Exit Program\n");
    printf("\n");
    printf("Auto ratation options:\n");
    printf("Mouse Left Button: Start Auto Rotation\n");
    printf("Mouse Right Button: Stop Auto Rotation\n");
    printf("\n");
    printf("Change axis options:\n");
    printf("X: Set X Axis as the rotation axis\n");
    printf("C: Set Y Axis as the rotation axis\n");
    printf("V: Set Z Axis as the rotation axis\n");
    printf("\n");
```

```
printf("R: Increase delta of currently selected transform\n");
printf("F: Decrease delta of currently selected transform\n");
printf("T: Reset all transformations and deltas\n");
printf("\n");
printf("Handed_Change rotation options:\n");
printf("q: Increase x\n");
printf("a: Decrease x\n");
printf("w: Increase y\n");
printf("s: Decrease y\n");
printf("e: Increase z\n");
printf("d: Decrease z\n");
```

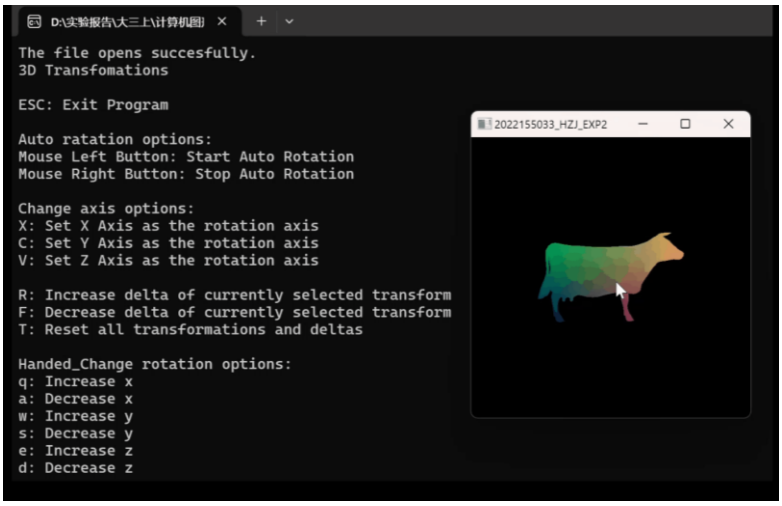
通过上述步骤，我们就完成了此实验基本的代码编写和修改，实现了键盘和鼠标的交互功能。（三维模型的自动旋转）

结果展示：

对于 cube.off（立方体模型），注意修改预定义顶点数为 8，结果 gif 如下所示：



对于 cow.off（牛模型），注意修改预定义顶点数为 2904，结果 gif 如下所示：



上述 gif 可能会有像素上的损失，详细结果可见提交的 main.exe 文件；

且刚开始旋转较快，可以“shift”加“F”降低一下速度。

实验结论：

本次实验对三维模型进行了读取和控制，主要是回顾了先前对 off 格式模型的读取以及三维模型的旋转操作，并在此基础上，对其添加了鼠标和键盘事件来实现三维模型的自动旋转操作。实验结果表明，结果较为成功，不仅能用鼠标用控制自动旋转和停止，还能用键盘改变旋转轴，还能用键盘手动控制其旋转。本次实验圆满结束。

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。