练 习 题 报 告

课程名称 计算机图形学

项目名称 三维模型的平移、缩放和旋转

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

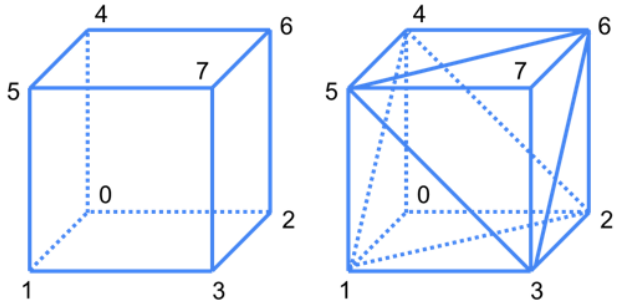
一、 练习目的

1. 掌握三维模型顶点与三角面片之间关系。
2. 了解和掌握三维模型的基本变换操作。
3. 掌握在着色器中使用变换矩阵。

二．练习完成过程及主要代码说明

实验要求：本实验将绘制一个立方体，并对其进行旋转平移等变换。

首先是补写 TriMesh 类，此类里面包含保存顶点坐标、顶点颜色、面片顶点下标的变量，代码中预先定义好了立方体顶点坐标的数组 cube_vertices 和颜色数组 colors，要求根据下图的立方体示意图在 TriMesh.cpp 中完善 storeFacesPoints 和 generateCube 函数。



对于 storeFacesPoints 函数：要求我们在 points 和 colors 容器中存储每个三角形面片的各个顶点和颜色信息。由于 faces 中存储着所有的面片信息，而每个面片对应着三个顶点信息，而也与前一个练习一致，系统回默认三个点一个面片，不需要我们特别指定，所以我们遍历每个面片，将每个索引对应的顶点传给 points 中即可。至于颜色的选择，类似地，我们为了方便直接每个顶点索引对应一种颜色进行传入即可。代码如下所示：

```
void TriMesh::storeFacesPoints() {
    // @TODO: Task-2修改此函数在points和colors容器中存储每个三角面片的各个点和颜色信息
    // 根据每个三角面片的顶点下标存储要传入GPU的数据
    points = getPoints();
    colors = getColors();
    for (const vec3i face : faces) {
        //获取面片索引
        int a = face.x;
        int b = face.y;
        int c = face.z;
        //存储顶点坐标
        points.push_back(vertex_positions[a]);
        points.push_back(vertex_positions[b]);
        points.push_back(vertex_positions[c]);
        //存储顶点颜色
        colors.push_back(vertex_colors[a]);
        colors.push_back(vertex_colors[b]);
        colors.push_back(vertex_colors[c]);
    }
```

HZJ

对于 generateCube 函数：要求我们在 vertex_positions 和 vertex_colors 中保存每个顶点的信息并记录每个面片的信息到 faces 中。顶点题目已经给出，我们只需要循环将它们传入 vertex_positions 中即可，顶点颜色信息也与之前一样，每个顶点对应所给的代码的颜色，

从而存储在 vertex_colors。由上面的立方体可知，每个面对应两个三角形面片，一共有 12 个三角形面片，每个面片对应三个顶点坐标（一一对应），根据上图手动创建 12 个面片，存储在 faces 中即可。代码如下所示：

```
// @TODO: Task1-修改此函数，存储立方体的各个面信息
// vertex_positions和vertex_colors先保存每个顶点的数据
vertex_positions = getVertexPositions();
faces = getFaces();
//定义立方体的各个顶点颜色
for (int i = 0; i < 8; i++) {
    vertex_positions.push_back(cube_vertices[i]);
    vertex_colors.push_back(basic_colors[i]);
}
// faces再记录每个面片上顶点的下标
//记录立方体的面片
faces.push_back(vec3i(0, 1, 2)); // 第一个三角形
faces.push_back(vec3i(1, 3, 2)); // 第二个三角形
faces.push_back(vec3i(4, 5, 6)); // 第三个三角形
faces.push_back(vec3i(5, 7, 6)); // 第四个三角形
faces.push_back(vec3i(0, 2, 4)); // 第五个三角形
faces.push_back(vec3i(2, 6, 4)); // 第六个三角形
faces.push_back(vec3i(1, 3, 5)); // 第七个三角形
faces.push_back(vec3i(3, 7, 5)); // 第八个三角形
faces.push_back(vec3i(0, 4, 1)); // 第九个三角形
faces.push_back(vec3i(4, 5, 1)); // 第十个三角形
faces.push_back(vec3i(2, 3, 6)); // 第十一个三角形
faces.push_back(vec3i(3, 7, 6)); // 第十二个三角形

storeFacesPoints();
```

HZJ

在完成上述代码的编写之后，打开 main.cpp 中 bindObjectAndData()函数的注释，否则程序会报错，如下所示：

```
// @TODO: Task3-修改完TriMesh.cpp的代码成后再打开下面注释，否则程序会报错
 glBufferSubData(GL_ARRAY_BUFFER, 0, mesh->getPoints().size() * sizeof(glm::vec3), &mesh->getPoints()[0]);
 glBufferSubData(GL_ARRAY_BUFFER, mesh->getPoints().size() * sizeof(glm::vec3), mesh->getColors().size() * sizeof(glm::vec3), &mesh->getColors()[0]);
```

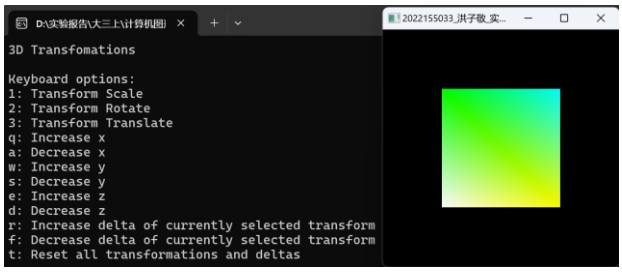
HZJ

接着是立方体的变换，main.cpp 中已经给出了旋转平移的参数，要求完善 display 函数，在 display 中使用这些参数计算变换矩阵，并且实时将最新的变换矩阵传递给渲染管道，让立方体可以进行旋转平移等变化。变化的顺序是缩放、旋转、平移。用函数 scale 和变化量 scaleTheta 可以对立方体进行缩放，用函数 rotate 并制定按哪个轴和对应的变化量可以按该轴进行旋转，利用 translate 函数和偏移量 translateTheta 可以对立方体进行平移。代码如下所示：

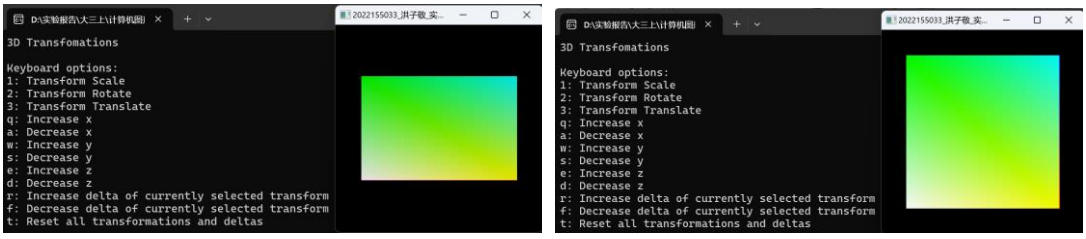
```
// @TODO: Task4-在此处修改函数，计算最终的变换矩阵
// 调用函数传入三种变化的变化量，累加得到变化矩阵
// 注意三种变化累加的顺序
m = glm::scale(m, scaleTheta); // 缩放
m = glm::rotate(m, glm::radians(rotateTheta.x), glm::vec3(1, 0, 0)); // 绕X轴旋转
m = glm::rotate(m, glm::radians(rotateTheta.y), glm::vec3(0, 1, 0)); // 绕Y轴旋转
m = glm::rotate(m, glm::radians(rotateTheta.z), glm::vec3(0, 0, 1)); // 绕Z轴旋转
m = glm::translate(m, translateTheta); // 平移
```

HZJ

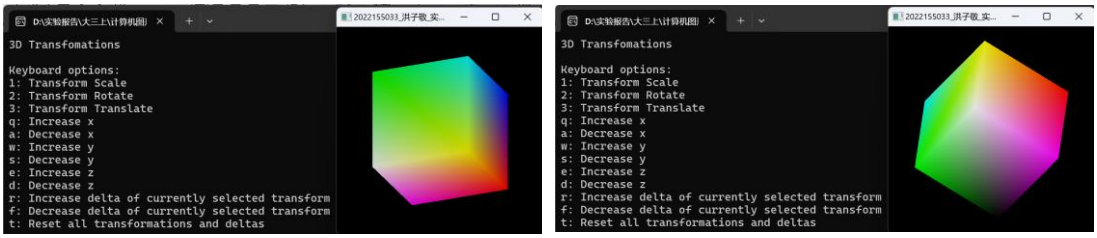
最后是结果展示，下面是程序运行后的初始效果：



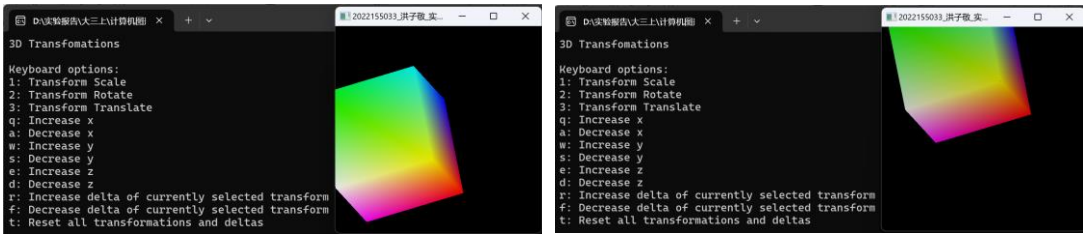
选择模式“1”对立方体进行缩放，部分效果如下所示：



选择模式“2”对立方体进行旋转，部分效果如下所示：



选择模式“3”对图像进行平移，部分效果如下所示：



拓展练习 1：Uniform 应用。在片元着色器 fshader 中加入一个 uniform 控制亮暗的浮点变量 brightness，将着色器颜色改为乘以此亮暗变量。代码如下；

```
uniform float brightness;
void main()
{
    fColor = vec4(color*brightness, 1.0);
}
```

HZJ

接着在 openGLObject 中新建一个成员变量 brightnessLocation，用于记录此变量的位置，同时便于后续的实时更新亮暗程度。而亮暗程度的变化，通过 glfwGetTime 函数获取当前时间，并使用 sin 函数使其在[-1,1]之间变化，并通过缩放和平移使其在[0,1]之间变化，这样可以使得亮度动态地发生变化，并使用 glUniform1f(GLint location, GLfloat v0)函

数将变化结果传递给着色器，代码如下：

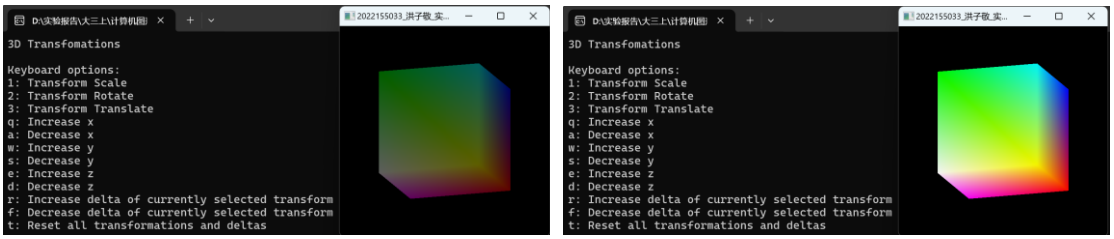
```
float timeValue = glfwGetTime();
// 计算亮度在0到1之间变化
float brightness = (sin(timeValue) / 2.0f) + 0.5f;
glUniform1f(cube_object.brightnessLocation, brightness);
```

HZJ

最后还需要在 bindObjectAndData 函数中，还需要获取其位置以便于动态的更新亮暗程度，代码如下：

```
//获取 brightness 的位置
object.brightnessLocation = glGetUniformLocation(object.program, "brightness");
```

整体部分运行效果如下所示：



拓展练习 2：变换矩阵与鼠标交互，要求实现对立方体的拖拽。首先要清楚需要编写两个函数，一个是鼠标点击拖动函数，一个是感知鼠标移动，更新立方体位置的函数。对于鼠标拖动函数 mouse_button_call_back，通过判断鼠标按动开始拖拽，释放停止拖拽来更新全局标志变量 dragging；而另一个鼠标移动函数 mouse_move_call_back，通过全局变量 dragging 是否为 true 来判断是否更新立方体的位置，为了能改变立方体的位置，还得定义其初始位置变量 rectanglePosition。

```
bool dragging = false;
glm::vec3 rectanglePosition(0.0f, 0.0f, 0.0f); // 矩形的初始位置
```

此时要注意窗口的坐标系与 openGL 标准空间坐标系之间的转换（转换到[-1,1]范围内，特别是 y 轴，两个坐标系方向相反，所以计算时要取反）。代码如下：

```
//鼠标拖动函数
void mouse_button_callback(GLFWwindow* window, int button, int action, int mods)
{
    if (button == GLFW_MOUSE_BUTTON_LEFT)
    {
        if (action == GLFW_PRESS)
        {
            dragging = true; // 开始拖拽
        }
        else if (action == GLFW_RELEASE)
        {
            dragging = false; // 结束拖拽
        }
    }
}
```

HZJ

```
//鼠标移动函数
void mouse_move_callback(GLFWwindow* window, double xpos, double ypos)
{
    if (dragging)
    {
        // 获取窗口大小
        int width, height;
        glfwGetWindowSize(window, &width, &height);

        // 将鼠标位置转换为OpenGL坐标
        float x = (2.0f * xpos / width) - 1.0f; // 转换到[-1, 1]
        float y = 1.0f - (2.0f * ypos / height); // 转换到[-1, 1]

        // 更新矩形位置
        rectanglePosition = glm::vec3(x, y, 0.0f);
    }
}
```

HZJ

接着还得注意绑定鼠标事件，在 main 函数中使用 glfwSetMouseButtonCallBack 函数绑定鼠标拖动函数，用 glfwCursorPosCallBack 函数更新鼠标的移动（实际上可以将两者合

一，但为了逻辑更加清晰，这里分为两部分），代码如下所示：

```
glfwMakeContextCurrent(window);
glfwSetKeyCallback(window, key_callback);
glfwSetFramebufferSizeCallback(window, framebuffer_size_callback);
//绑定鼠标
glfwSetMouseButtonCallback(window, mouse_button_callback);
glfwSetCursorPosCallback(window, mouse_move_callback);
```

HZJ

最后别忘了我们是通过改变该立方体位置来改变位置的，所以还得在 display 函数中对立方体进行平移，平移量为 rectanglePosition。

整体部分效果展示如下：（点击鼠标左键进行拖动）