科目	计算机图形学
学号	16340294
姓名	张星
邮箱	dukestar@foxmail.com

Basic

题目一

实验要求

> 画一个立方体(cube)：边长为4， 中心位置为(0, 0, 0)。分别启动和关闭深度测试glEnable(GL_DEPTH_TEST) 、glDisable(GL_DEPTH_TEST) ，查看区别，并分析原因。

实验思路

- 绘制正方体：正方体有六个面，在之前画三角形的时候，就有做过用两个三角形拼成一个长方形，所以这次只需要画12个三角形即可。为了使深度测试的结果更加明显，给六个面分别设置了不同的颜色，矩阵如下：

```
float vertices[] = {
       -2.0f, -2.0f, -2.0f, 0.5f, 0.0f, 0.0f,
        2.0f, -2.0f, -2.0f, 0.5f, 0.0f, 0.0f,
        2.0f,  2.0f, -2.0f, 0.5f, 0.0f, 0.0f,
        2.0f,  2.0f, -2.0f, 0.5f, 0.0f, 0.0f,
       -2.0f,  2.0f, -2.0f, 0.5f, 0.0f, 0.0f,
       -2.0f, -2.0f, -2.0f, 0.5f, 0.0f, 0.0f,

       -2.0f, -2.0f,  2.0f, 0.0f, 0.5f, 0.0f,
        2.0f, -2.0f,  2.0f, 0.0f, 0.5f, 0.0f,
        2.0f,  2.0f,  2.0f, 0.0f, 0.5f, 0.0f,
        2.0f,  2.0f,  2.0f, 0.0f, 0.5f, 0.0f,
       -2.0f,  2.0f,  2.0f, 0.0f, 0.5f, 0.0f,
       -2.0f, -2.0f,  2.0f, 0.0f, 0.5f, 0.0f,

       -2.0f,  2.0f,  2.0f, 0.0f, 0.0f, 0.5f,
       -2.0f,  2.0f, -2.0f, 0.0f, 0.0f, 0.5f,
       -2.0f, -2.0f, -2.0f, 0.0f, 0.0f, 0.5f,
       -2.0f, -2.0f, -2.0f, 0.0f, 0.0f, 0.5f,
       -2.0f, -2.0f,  2.0f, 0.0f, 0.0f, 0.5f,
       -2.0f,  2.0f,  2.0f, 0.0f, 0.0f, 0.5f,

        2.0f,  2.0f,  2.0f, 0.5f, 0.5f, 0.0f,
        2.0f,  2.0f, -2.0f, 0.5f, 0.5f, 0.0f,
        2.0f, -2.0f, -2.0f, 0.5f, 0.5f, 0.0f,
        2.0f, -2.0f, -2.0f, 0.5f, 0.5f, 0.0f,
        2.0f, -2.0f,  2.0f, 0.5f, 0.5f, 0.0f,
```

```
         2.0f,  2.0f,  2.0f, 0.5f, 0.5f, 0.0f,

        -2.0f, -2.0f, -2.0f, 0.5f, 0.0f, 0.5f,
         2.0f, -2.0f, -2.0f, 0.5f, 0.0f, 0.5f,
         2.0f, -2.0f,  2.0f, 0.5f, 0.0f, 0.5f,
         2.0f, -2.0f,  2.0f, 0.5f, 0.0f, 0.5f,
        -2.0f, -2.0f,  2.0f, 0.5f, 0.0f, 0.5f,
        -2.0f, -2.0f, -2.0f, 0.5f, 0.0f, 0.5f,

        -2.0f,  2.0f, -2.0f, 0.0f, 0.5f, 0.5f,
         2.0f,  2.0f, -2.0f, 0.0f, 0.5f, 0.5f,
         2.0f,  2.0f,  2.0f, 0.0f, 0.5f, 0.5f,
         2.0f,  2.0f,  2.0f, 0.0f, 0.5f, 0.5f,
        -2.0f,  2.0f,  2.0f, 0.0f, 0.5f, 0.5f,
        -2.0f,  2.0f, -2.0f, 0.0f, 0.5f, 0.5f
    };
```

- 深度测试：未启动深度测试时，正方体只能看到后面的区块颜色，覆盖了不应该覆盖的颜色，这样的原因是OpenGL是一个接一个地绘制三角形，即使之前那里已经绘制也会被新的覆盖。启动了深度测试后，就符合我们的日常认知，是因为启动了深度测试后，每当片段想要输出其颜色时，OpenGL就会将它的深度值和一个用于存储深度信息的深度缓冲比较，如果当前片段在其他片段之后，则该片段被丢弃，否则就会覆盖之前的片段，这个过程称之为**深度测试。**

难点

一开始拿到题目，看到正方体的边长为4时我有点懵，因为之前的实验中屏幕的坐标为-1.0f~1.0f之间，超出这个范围就会无法显示。但通过看教程，学会了局部坐标到屏幕坐标之间的转换，只需要调整view的坐标，类似Unity中的摄像机坐标，就可以使正方体正常显示了。

题目二

实验要求

> 平移(Translation)：使画好的cube沿着水平或垂直方向来回移动。

实验思路

本题目使用了GLM，关键代码如下：

```
glm::mat4 model = glm::mat4(1.0f);
glm::mat4 view = glm::mat4(1.0f);
glm::mat4 projection = glm::mat4(1.0f);
model = glm::translate(model, glm::vec3(0.0f, 0.0f, 0.0f));

//move the cube on the horizontal direction
model = glm::translate(model, (float)speed*glm::vec3(0.01f, 0.0f, 0.0f));
if (speed >= 300) {
    direction = 1;
}
if (speed <= -300) {
    direction = 0;
}
//move to the right or left
```

```
if (direction == 0) {
    speed++;
}
else {
    speed--;
}
```

由于题目中要求来回移动，所以我用两个变量，一个表示长度，一个表示方向，然后在一次次循环中不断更新数据，使用translate函数，达到移动的目的。

题目三

实验要求

> 旋转(Rotation)：使画好的cube沿着XoZ平面的x=z轴持续旋转。

实验思路

本题依旧使用GLM，关键代码如下：

```
//rotate the cube
model = glm::rotate(model, (float)glfwGetTime()*glm::radians(angle), glm::vec3(1.0f, 0.0f, 1.0f));
```

angle是一个可以调整的参数，用于控制旋转的速度快慢，glfwGetTime()是为了使其持续，连续地旋转，最后一个参数是旋转的中心轴，这里是x=z的向量。

题目四

实验要求

> 放缩(Scaling)：使画好的cube持续放大缩小。

实验思路

本题可以使用GLM的scale函数实现，也可以通过调整摄像机等参数实现，这里我选择的方式是调整透视投影的参数来实现。关键代码如下：

```
projection = glm::perspective(glm::radians(scale), (float)height / (float)width, 0.1f, 100.0f);
```

scale为参数，表示观察空间的大小，我们可以通过调整其来实现缩放或方法正方体的目的。

题目五

实验要求

> 在GUI里添加菜单栏，可以选择各种变换。

实验思路

使用ImGui实现，用好几次了，不再赘述，不过这周发现了一个新的部件，可以实现滑动条的效果：

```
ImGui::SliderFloat("Scale", &scale, 35.0f, 100.0f);
```

其最后两个参数表示滑动条的范围。

题目六

实验要求

> 结合Shader谈谈对渲染管线的理解。

答案

Shader即控制GPU的一系列指令集，使用GLSL编写。OpenGL的Shader渲染管线框图包括：

1. Vertex Processor：对顶点进行Shader操作，入坐标空间变换等。
2. Geometry Processor：可以改变输出体元语义类型。
3. Clipper：固定管线模块，将用裁剪空间的6个面对体元进行裁剪操作。
4. Rasterizer：光栅化，产生fragment。
5. Fragment Processor：装入纹理，颜色等。

我们经常声明Shader程序对象，然后声明Shader对象，以此使用可编程管线的功能。最常用的Vertex Shader和Fragment Shader，但若不进行定义，则这两个Shader则使用固定管线的功能。

编程的过程为：

1. 声明Shader程序对象和Shader对象。
2. 指定Shader源代码。
3. 编译Shader对象。
4. 检验是否正常。
5. 绑定Shader对象和Shader程序对象。
6. 验证Shader程序对象的有效性。（可略去）
7. 将链接好的Shader程序对象送入Shader管线。

Bonus

题目一

实验要求

> 将以上三种变换相结合，打开你们的脑洞，实现有创意的动画。比如：地球绕太阳转等。

实验思路

我的目的是实现地球绕太阳的效果，但是没有绘制球体，直接以正方体代替，有三个正方体，分别为太阳，地球以及另一颗行星，绕着太阳转。实现的难点是如何使其实现公转，我试了很多次，最后以先旋转，再平移的操作，实现了公转，具体代码如下：

```
modelBonus1 = glm::rotate(modelBonus1, (float)glfwGetTime()*glm::radians(angle),
glm::vec3(0.0f, 0.0f, 1.0f));
modelBonus1 = glm::translate(modelBonus1, glm::vec3(0.0f, 10.0f, 0.0f));
```

先使在原点的正方体绕Z轴旋转，然后将其平移(0.0f, 10.0f, 0.0f)，即可实现公转的操作。但并非所有的旋转+平移都能够实现公转，经我实验，平移的向量必须与旋转轴垂直，才可以达到效果。