计算机图形学 作业4

徐达烽

16340260

效果见transform.mp4视频。

Basic

1. 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。
2. 平移(Translation)：使画好的cube沿着水平或垂直方向来回移动。
3. 旋转(Rotation)：使画好的cube沿着XoZ平面的x=z轴持续旋转。
4. 放缩(Scaling)：使画好的cube持续放大缩小。
5. 在GUI里添加菜单栏，可以选择各种变换。
6. 结合Shader谈谈对渲染管线的理解

实现思路

1. 画正方体

为了画一个立方体，需要画6个面，一共12个三角形。将这12个三角形的36个顶点的位置的值写入一个数组里。

```
float vertices[] = {
    -2.0f, -2.0f, -2.0f,  0.0f, 0.0f,
     2.0f, -2.0f, -2.0f,  1.0f, 0.0f,
     2.0f,  2.0f, -2.0f,  1.0f, 1.0f,
     2.0f,  2.0f, -2.0f,  1.0f, 1.0f,
    -2.0f,  2.0f, -2.0f,  0.0f, 1.0f,
    -2.0f, -2.0f, -2.0f,  0.0f, 0.0f,

    -2.0f, -2.0f,  2.0f,  0.0f, 0.0f,
     2.0f, -2.0f,  2.0f,  1.0f, 0.0f,
```

```
        2.0f,  2.0f,  2.0f,  1.0f, 1.0f,
        2.0f,  2.0f,  2.0f,  1.0f, 1.0f,
       -2.0f,  2.0f,  2.0f,  0.0f, 1.0f,
       -2.0f, -2.0f,  2.0f,  0.0f, 0.0f,

       -2.0f,  2.0f,  2.0f,  1.0f, 0.0f,
       -2.0f,  2.0f, -2.0f,  1.0f, 1.0f,
       -2.0f, -2.0f, -2.0f,  0.0f, 1.0f,
       -2.0f, -2.0f, -2.0f,  0.0f, 1.0f,
       -2.0f, -2.0f,  2.0f,  0.0f, 0.0f,
       -2.0f,  2.0f,  2.0f,  1.0f, 0.0f,

        2.0f,  2.0f,  2.0f,  1.0f, 0.0f,
        2.0f,  2.0f, -2.0f,  1.0f, 1.0f,
        2.0f, -2.0f, -2.0f,  0.0f, 1.0f,
        2.0f, -2.0f, -2.0f,  0.0f, 1.0f,
        2.0f, -2.0f,  2.0f,  0.0f, 0.0f,
        2.0f,  2.0f,  2.0f,  1.0f, 0.0f,

       -2.0f, -2.0f, -2.0f,  0.0f, 1.0f,
        2.0f, -2.0f, -2.0f,  1.0f, 1.0f,
        2.0f, -2.0f,  2.0f,  1.0f, 0.0f,
        2.0f, -2.0f,  2.0f,  1.0f, 0.0f,
       -2.0f, -2.0f,  2.0f,  0.0f, 0.0f,
       -2.0f, -2.0f, -2.0f,  0.0f, 1.0f,

       -2.0f,  2.0f, -2.0f,  0.0f, 1.0f,
        2.0f,  2.0f, -2.0f,  1.0f, 1.0f,
        2.0f,  2.0f,  2.0f,  1.0f, 0.0f,
        2.0f,  2.0f,  2.0f,  1.0f, 0.0f,
       -2.0f,  2.0f,  2.0f,  0.0f, 0.0f,
       -2.0f,  2.0f, -2.0f,  0.0f, 1.0f
   };
```

每个顶点用float的五元组来描述，前三维是空间坐标，后两维是纹理坐标。
定义好变换矩阵后，使用DrawElements函数画出12个三角形，即可显示出一个正方体。
不开启深度检测时，一些不应该被看到的面也会显示出来。因为如果没有开启深度检测，openGL就是简单将后面画的图元覆盖到之前画的图元上，于是乎就导致了后面的面跑到前面来的奇怪现象，开启深度检测后openGL会根据元素的前后关系进行显示，这里通过一种Z缓冲的机制，于是实现了正常的正方体的效果。

2. 平移

使用glm::translate构造平移变换的矩阵，很简单。

实现来回移动的方法：这里我用过glfwGetTime()获取当前的时间t，然后将t映射到`[-10, 10]`的值域上，并且先从-10到10线性递增，然后从10到-10线性递减。这样就能实现位置的来回移动。

具体实现如下：

```
int translate_range = 10;
float cur_time = (float)glfwGetTime();
int segment = static_cast<int>(cur_time / translate_range);
float pos = cur_time - segment * translate_range - translate_range / 2.0f;
if (segment % 2 == 1) {
    pos = -pos;
}
pos *= 2;
...
if (move_vertical) {
    model = glm::translate(model, glm::vec3(0.0f, pos, 0.0f));
}
if (move_horizontal) {
    model = glm::translate(model, glm::vec3(pos, 0.0f, 0.0f));
}
```

3. 旋转(Rotation)

使画好的cube沿着XoZ平面的x=z轴持续旋转。

```
model = glm::rotate(model, cur_time * glm::radians(50.0f), glm::vec3(1.0f, 0.0f, 1.0f));
```

4. 放缩(Scaling)

使画好的cube持续放大缩小。

```
model = glm::scale(model, glm::vec3(scale_size, scale_size, scale_size));
```

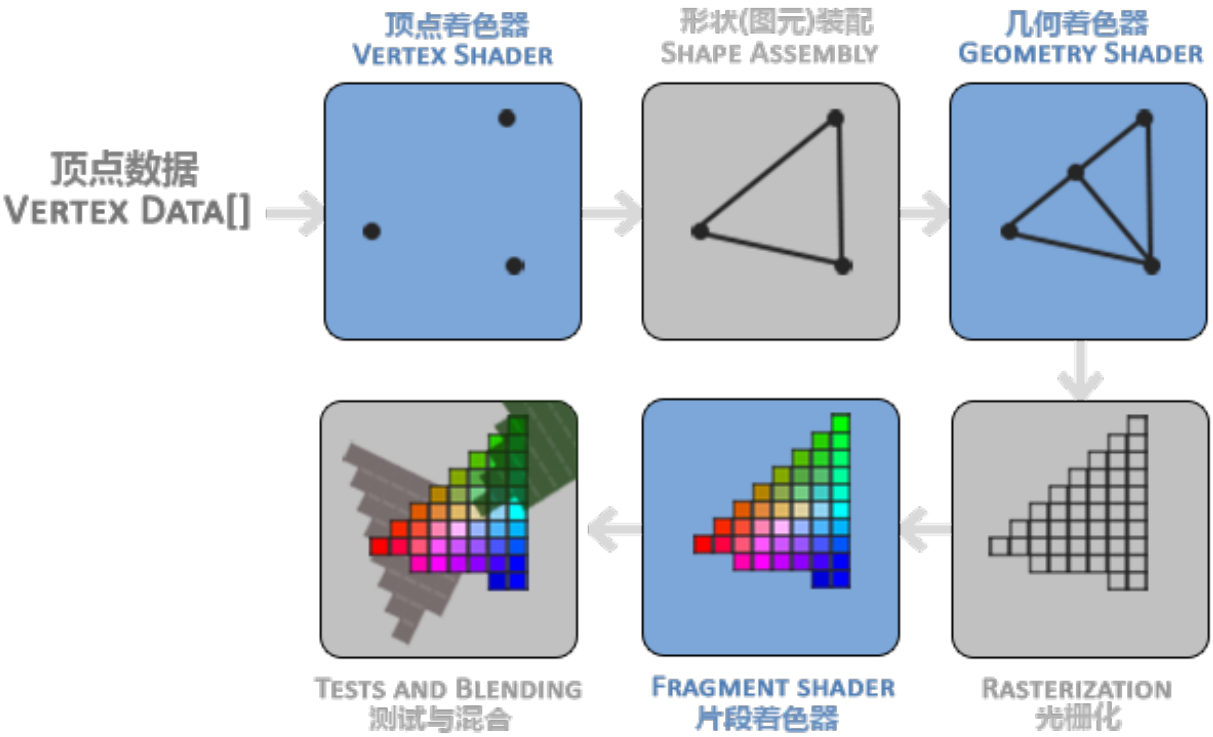
来回放大缩小的实现方法与translate类似。

5. GUI添加菜单栏。

绑定布尔值变量即可。

```
{
    ImGui::Begin("Hello, world!");                          // Create a window called "Hello, world!" and append into it.
    ImGui::Checkbox("Rotate", &can_rotate);
    ImGui::Checkbox("Move Vertical", &move_vertical);
    ImGui::Checkbox("Move Horizontal", &move_horizontal);
    ImGui::Checkbox("Scale", &can_scale);
    ImGui::Text("Application average %.3f ms/frame (%.1f FPS)", 1000.0f / ImGui::GetIO().Framerate, ImGui::GetIO().Framerate);
    ImGui::End();
}
```

6. 结合Shader谈谈对渲染管线的理解



图形渲染管线的第一个部分是顶点着色器(Vertex Shader)，它把一个单独的顶点作为输入。顶点着色器主要的目的是把3D坐标转为另一种3D坐标（后面会解释），同时顶点着色器允许

我们对顶点属性进行一些基本处理。

图元装配(Primitive Assembly)阶段将顶点着色器输出的所有顶点作为输入，并所有的点装配成指定图元的形状；

图元装配阶段的输出会传递给几何着色器(Geometry Shader)。几何着色器把图元形式的一系列顶点的集合作为输入，它可以通过产生新顶点构造出新的（或是其它的）图元来生成其他形状。

几何着色器的输出会被传入光栅化阶段(Rasterization Stage)，这里它会把图元映射为最终屏幕上相应的像素，生成供片段着色器(Fragment Shader)使用的片段(Fragment)。在片段着色器运行之前会执行裁切(Clipping)。裁切会丢弃超出视图以外的所有像素，用来提升执行效率。

OpenGL中的一个片段是OpenGL渲染一个像素所需的所有数据。

片段着色器的主要目的是计算一个像素的最终颜色，这也是所有OpenGL高级效果产生的地方。通常，片段着色器包含3D场景的数据（比如光照、阴影、光的颜色等等），这些数据可以被用来计算最终像素的颜色。在片段着色器中会进行片段插值(Fragment Interpolation)。光栅化(Rasterization)阶段通常会造成比原指定顶点更多的片段。光栅会根据每个片段在几何形状上所处相对位置决定这些片段的位置。基于这些位置，它会插值(Interpolate)所有片段着色器的输入变量。

在所有对应颜色值确定以后，最终的对象将会被传到最后一个阶段，Alpha测试和混合(Blending)阶段。