作业 4

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。

绘制立方体

1. 在这里采用的绘制方法是利用索引缓冲对象(EBO)，只储存不同的顶点，并设定绘制这些顶点的顺序，即可减少开销。

```
//顶点输入
float vertices[] = {
    //坐标              //颜色
    2.0f, 2.0f, 2.0f, 1.0f, 0.0f, 0.0f,
    2.0f, -2.0f, 2.0f, 0.0f, 1.0f, 0.0f,
    -2.0f, 2.0f, 2.0f, 0.0f, 0.0f, 1.0f,
    -2.0f, -2.0f, 2.0f, 1.0f, 1.0f, 1.0f,
    2.0f, 2.0f, -2.0f, 1.0f, 1.0f, 1.0f,
    2.0f, -2.0f, -2.0f, 0.0f, 0.0f, 1.0f,
    -2.0f, 2.0f, -2.0f, 0.0f, 1.0f, 0.0f,
    -2.0f, -2.0f, -2.0f, 1.0f, 0.0f, 0.0f,
};
```

```
unsigned int indices[] = { // 注意索引从0开始!
    0, 1, 2,
    1, 2, 3,

    4, 5, 6,
    5, 6, 7,

    0, 1, 4,
    1, 4, 5,

    2, 3, 6,
    3, 6, 7,

    0, 2, 4,
    2, 4, 6,

    1, 3, 5,
    3, 5, 7
};
```

2. 在定义了立方体的八个顶点之后，我们需要创建索引缓冲对象。

```
unsigned int VBO, VAO, EBO;
glGenVertexArrays(1, &VAO);
glGenBuffers(1, &VBO);
glGenBuffers(1, &EBO);

glBindVertexArray(VAO);

//把新创建的缓冲绑定到GL_ARRAY_BUFFER
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EBO);
glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(indices), indices, GL_STATIC_DRAW);

glBindBuffer(GL_ARRAY_BUFFER, VBO);
glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW);
```

3. 用 glDrawElements 时，我们会使用当前绑定的索引缓冲对象中的索引进行绘制

```
glBindVertexArray(VAO);
glDrawElements(GL_TRIANGLES, 24, GL_UNSIGNED_INT, 0);
```

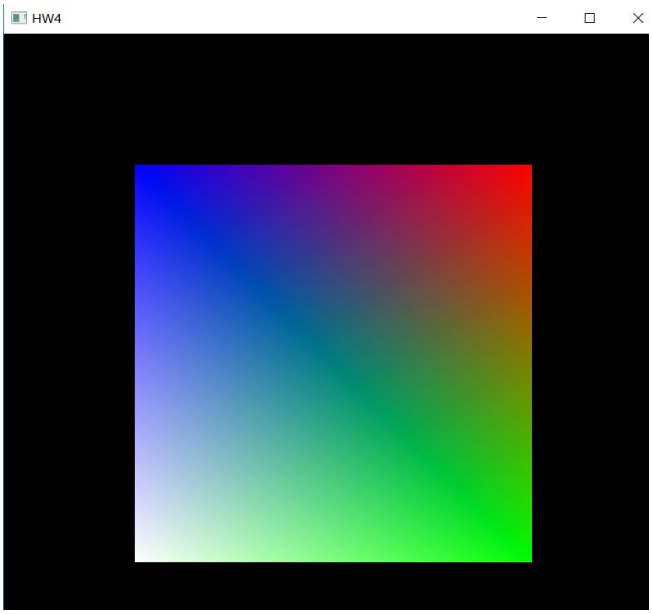
视图管理

由于边长为 4，中心位置为(0, 0, 0)，所以需要更改 view 的位置·以便看到完整的立方体。

```
glm::mat4 view = glm::mat4(1.0f);

view = glm::translate(view, glm::vec3(0.0f, 0.0f, -15.0f));

unsigned int viewLoc = glGetUniformLocation(shaderProgram, "view");

glUniformMatrix4fv(viewLoc, 1, GL_FALSE, glm::value_ptr(view));
```

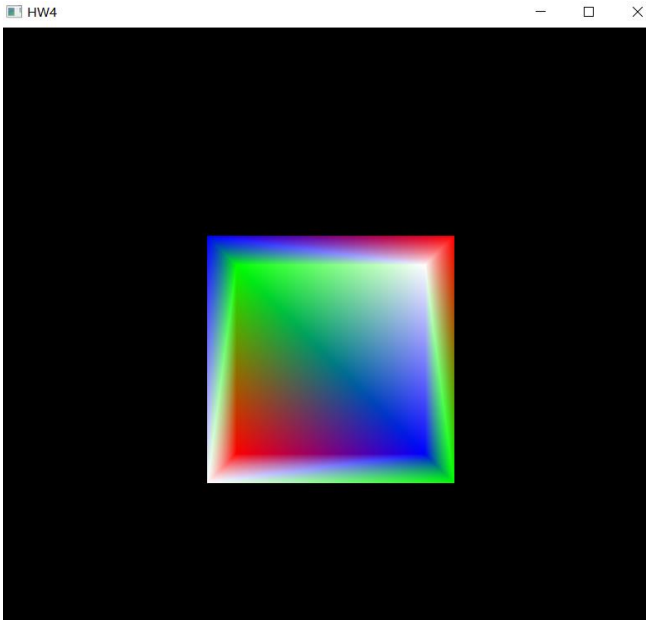
启动深度测试

由下图可以看出，在开启深度测试的情况下，面向镜头的这一面将不会被穿透，当片段想要输出它的颜色时，OpenGL 会将它的深度值和 z 缓冲进行比较，如果当前的片段在其它片段之后，它将会被丢弃，否则将会覆盖。



关闭深度测试

由下图可以看出，在没有开启深度测试的情况下，面向镜头的这一面将被穿透，可以看到立方体内部的五个面的情况，这是由于被阻挡的面将渲染到其它面的前面。



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

由于要实现来回移动，所以首先想到采用正弦函数进行计算。

```
model = glm::translate(model, glm::vec3(1.0f, 0.0f, 0.0f) * (float)sin(time * PI / 200) * 5.0f);
```

效果如文档中 Basic-2 录屏所示。

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

沿 x=z 轴持续旋转

```
model = glm::rotate(model, (float)glfwGetTime(), glm::vec3(1.0f, 0.0f, 1.0f));
```

效果如文档中 Basic-3 录屏所示。

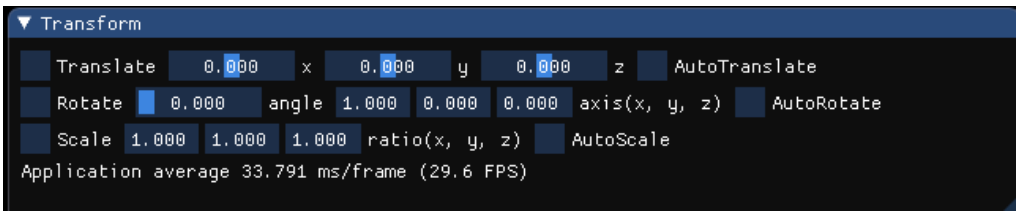
4. 放缩(Scaling)：使画好的 cube 持续放大缩小。

实现持续放大缩小

```
model = glm::scale(model, glm::vec3(1.0f, 1.0f, 1.0f) * ((float)sin(time * PI / 200) * 1.0f + 0.5f));
```

效果如文档中 Basic-4 录屏所示。

5. 在 GUI 里添加菜单栏，可以选择各种变换。



由上图可知，在菜单栏里面实现了以下三种功能

1. 平移：可以利用拖动条对立方体进行平移。当选中 AutoTranslate 时，立方体将会沿设定方向来回移动。
2. 旋转：拖动条用来调整旋转的角度，后面的输入框设置旋转围绕的轴线。当选中 AutoRotate 时，立方体将会沿设定方向持续旋转。
3. 放缩：输入框设置各边放缩的比例。当选中 AutoScale 时，立方体将会持续变大变小。

具体效果如文档中 Basic-5 录屏所示

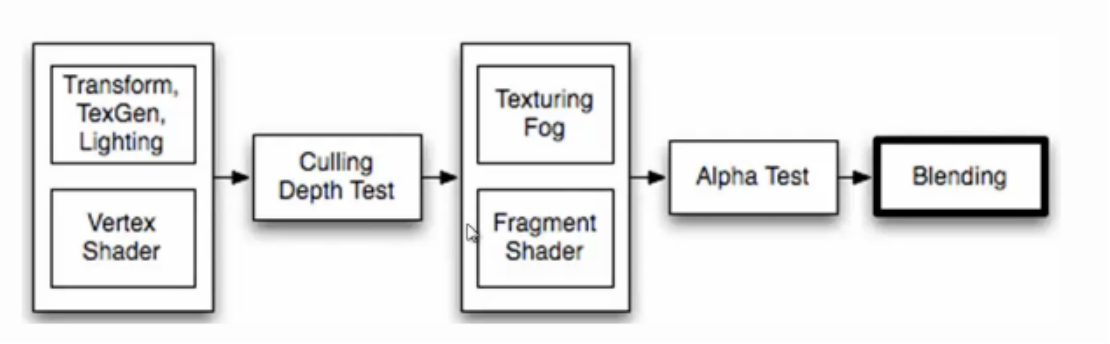
6. 结合 Shader 谈谈对渲染管线的理解

Shader，中文翻译即着色器，是运行在 GPU 上的小程序。这些小程序为图形渲染管线的某个特定部分而运行。用于告诉图形硬件如何计算和输出图像，过去由汇编语言来编写，现在也可以使用高级语言来编写。从基本意义上来说，着色器只是一种把输入转化为输出的程序。着色器也是一种非常独立的程序，因为它们之间不能相互通信；它们之间唯一的沟通只有通过输入和输出。一句话概括：Shader 是可编程图形管线的算法片段。它主要分为两类：Vertex Shader 和 Fragment Shader。

渲染管线也称为渲染流水线，是显示芯片内部处理图形信号相互独立的并行处理单元。一个流水线是一序列可以并行和按照固定顺序进行的阶段。就像一个在同一时间内，

不同阶段不同的汽车一起制造的装配线，传统的图形硬件流水线以流水的方式处理大量的顶点、几何图元和片段。

下方为 Unity 官方手册中的一张渲染管线图示。



渲染管道线中最左边的这个部分中 Transform 指的是模型的空间变换，主要针对的是顶点的空间几何变换；TexGen 即 Texture Generator，表示的是纹理坐标的生成，主要用于在顶点当中去取得纹理坐标，再转换为 UV 取值的范围；Lighting 指的是光照。因此这个部分就是过去就是 T&L 几何变换光照流水线，当图形硬件具有了可编程能力后，这个固定的模块就被【Vertex Shader】顶点着色器代替了。

在顶点着色器处理过后，Unity 就进入【Culling & Depth Test】裁剪和深度测试过程。裁剪和深度测试描述的是如果一个物体在摄像机前展示，它向着摄像机的面会被观察到，它背对着摄像机的面不会被观察到，在这样的情况下，为了减少 GPU 处理数据量就进行了一个裁剪（Culling），把看不见的面直接剔除，不需要去处理的这些面所涉及的顶点数据，从而加速图形处理。第二个方面深度测试（Depth Test），指的是摄像机有一个特性，在计算机当中没有无限这个概念，计算机处理的数据都是离散化的，它有一个范围，当超过最近和最远这个范围的这部分会被剔除。

接下来就进入到纹理采样（Texturing）和雾化处理（Fog）阶段。在这阶段实际上就是在进行光栅化处理，描述的就是如何在屏幕上显示每一个像素的颜色。这里需要去纹理采样，一张贴图有很多数据，我们去采集纹理上某一点的颜色值，这个就叫做纹理采样。雾化就是根据最后计算的数据后需不需要进行一个雾化处理，近处的很清晰，远处的有种朦胧感，这个部分就是片段着色器可编程的能力范围。

之后还需要【Alpha Test】，指的是去绘制那些半透明的或全透明的物体。经过【Alpha Test】之后还需要进行【Blending】处理，这阶段会混合最终的图像。

(本题部分内容参照网上博客作答)

Bonus

1. 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

在这里实现的两个物体围绕着中间一个物体旋转。具体的实现方法是，每在图中构建一个物体时，都重新加载一次 model，view 和 projection 并调用一次着色器，以便于每个物体都可以按照想要的方式运行。具体效果如文档中 Bonus 录屏所示