郑国林 16340304

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

首先是为正方体的设定顶点的坐标（前三个是位置坐标，后两个是纹理坐标），然后就是常规操作创建 VAO,VBO 并进行绑定，并在 glVertexAttribPointer 函数中要告诉 openGL 顶点数据的格式，和之前有点不一样。

要做成边长为 4 的话，在顶点矩阵那里设了坐标后，还要在后面的 view 矩阵中进行调整一下，否则的话就会占了整个屏幕

为了看起来好看一点，跟着教程弄了一下纹理的设置，并且为了显示 3d 的效果，那个着色器那里也要发生改变，不再是之前那样直接进行光栅化，而是传入了纹理坐标，在顶点着色器中进行一下那个坐标的变换，就是矩阵的相乘，然后传给片段着色器，片段着色器再通过纹理的图片进行插值（教程中用的是两张图片然后混在一起的效果）。并且为了代码更加的有可读性，就学着教程那里把着色器单独抽象成了一个类，并封装了不同的方法，虽然他那个读取着色器源码那里弄不太懂，还是自己按之前的方法把着色器的代码弄成字符串进行编译的。

另外关于纹理的话，教程中还说到了纹理的环绕方式和过滤。环绕方式是值纹理坐标超出了范围的情况，纹理坐标的范围通常是从(0, 0)到(1, 1)，如果超出范围的话默认是重复纹理图像，但是通过设定不同的参数超出范围会有不同的效果

环绕方式	描述
GL_REPEAT	对纹理的默认行为。重复纹理图像。
GL_MIRRORED_REPEAT	和GL_REPEAT一样，但每次重复图片是镜像放置的。
GL_CLAMP_TO_EDGE	纹理坐标会被约束在0到1之间，超出的部分会重复纹理坐标的边缘，产生一种边缘被拉伸的效果。
GL_CLAMP_TO_BORDER	超出的坐标为用户指定的边缘颜色。

至于纹理过滤是由于纹理坐标不依赖于分辨率，它可以是任意浮点值，所以 OpenGL 需要知道怎样将纹理像素映射到纹理坐标。当你有一个很大的物体但是纹理的分辨率很低的时候这就变得很重要了，因为每个坐标不一定能对应到一个像素点。GL_NEAREST 的时候，OpenGL 会选择中心点最接近纹理坐标的那个像素；GL_LINEAR（也叫线性过滤，(Bi)linear Filtering）它会基于纹理坐标附近的纹理像素，计算出一个插值，近似出这些纹理像素之间的颜色。GL_NEAREST 产生了颗粒状的图案，我们能够清晰看到组成纹理的像素，而 GL_LINEAR 能够产生更平滑的图案，很难看出单个的纹理像素。

还有一个多级渐远纹理的概念，简单的理解就是距观察者的距离超过一定的阈值，OpenGL 会使用不同的多级渐远纹理，即最适合物体的距离的那个。由于距离远，解析度不高也不会被用户注意到。

过滤方式	描述
GL_NEAREST_MIPMAP_NEAREST	使用最邻近的多级渐远纹理来匹配像素大小，并使用邻近插值进行纹理采样
GL_LINEAR_MIPMAP_NEAREST	使用最邻近的多级渐远纹理级别，并使用线性插值进行采样
GL_NEAREST_MIPMAP_LINEAR	在两个最匹配像素大小的多级渐远纹理之间进行线性插值，使用邻近插值进行采样
GL_LINEAR_MIPMAP_LINEAR	在两个邻近的多级渐远纹理之间使用线性插值，并使用线性插值进行采样

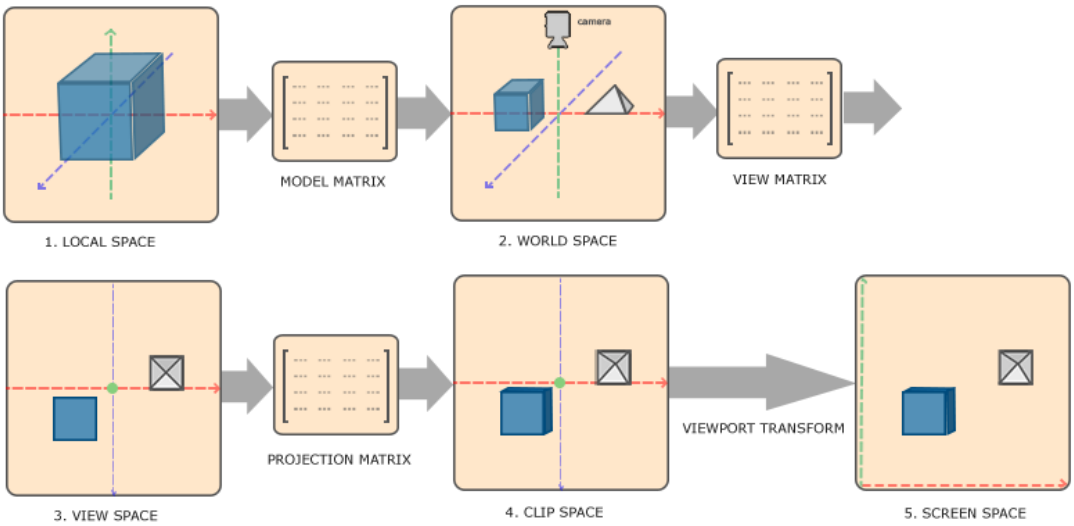
为坐标超出范围设定不同处理方式，为放大过滤和放小过滤设定不同的处理方式

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
// 过滤方式
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
// 加载图片
```

然后为了创建纹理，还用到了 stb_image.h 这个库，就下载下来并且要新建一个文件进行宏定义。具体的步骤就是先生成纹理 ID,进行绑定（像 VBO 那些一样），再通过库的函数加载图片生成到纹理中，然后释放加载的图片。然后就在 gldraw 之前激活，绑定一下纹理，就会使用上面改过的着色器进行渲染了。(教程中将两个纹理混合就是激活了两次纹理单元，然后绑定，通过片段着色器中 mix 函数进行线性插值完成)

```
#define STB_IMAGE_IMPLEMENTATION
#include "stb_image.h"
```

接下来就是进行坐标变换的部分了，根据教程中所说的，主要用到的变换矩阵是模型(Model)、观察(View)、投影(Projection)三个矩阵，顶点坐标起始于局部空间(Local Space)，在这里它称为局部坐标(Local Coordinate)，它在之后会变为世界坐标(World Coordinate)，观察坐标(View Coordinate)，裁剪坐标(Clip Coordinate)，并最后以屏幕坐标(Screen Coordinate)的形式结束。



局部坐标是相对于局部原点的坐标，一般是建模时候的相对坐标，然后把模型放到我们的世界中，就得到了世界坐标，原点发生变化，接下来我们将世界坐标变换为观察空间坐标，使得每个坐标都是从摄像机或者说观察者的角度进行观察的，坐标到达观察空间之后，我们需要将其投影到裁剪坐标。裁剪坐标会被处理至-1.0 到 1.0 的范围内，并判断哪些顶点将会

出现在屏幕上。最后，我们将裁剪坐标变换为屏幕坐标，我们将使用一个叫做视口变换(Viewport Transform)的过程。视口变换将位于-1.0 到 1.0 范围的坐标变换到由 glViewport 函数所定义的坐标范围内。最后变换出来的坐标将会送到光栅器，将其转化为片段。

在顶点着色器的源码中，就是利用了上面说的三个矩阵，对坐标进行变化

$$V_{clip} = M_{projection} \cdot M_{view} \cdot M_{model} \cdot V_{local}$$

然后在构建这三个矩阵的时候，还用到了 glm 这个库，同样是把那个源码文件下载下来加到项目里用就行了。模型矩阵。这个模型矩阵包含了位移、缩放与旋转操作，它们会被应用到所有物体的顶点上，以变换它们到全局的世界空间。至于观察矩阵，将摄像机向后移动，和将整个场景向前移动是一样的，所以说这个变换的话是有点相反方向的感觉的。投影矩阵的话教程中用的是透视投影。

得到的截图如下

但是这样看起来有点奇怪，后面的面本来应该是要被阻挡住的，但是却显示了出来绘制了在前面的面上

分析：

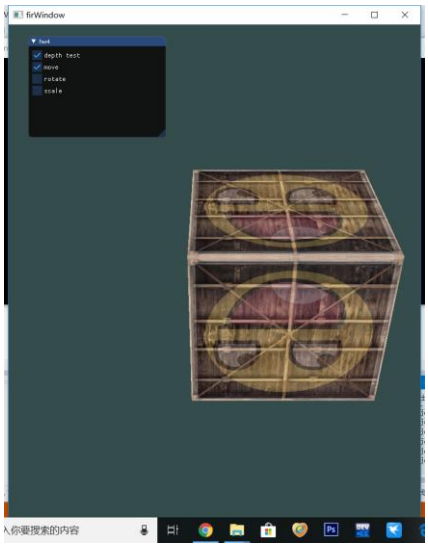
之所以这样是因为 OpenGL 是一个三角形一个三角形地来绘制你的立方体的，所以即便之前那里有东西它也会覆盖之前的像素。因为这个原因，有些三角形会被绘制在其它三角形上面，虽然它们本不应该是被覆盖的。

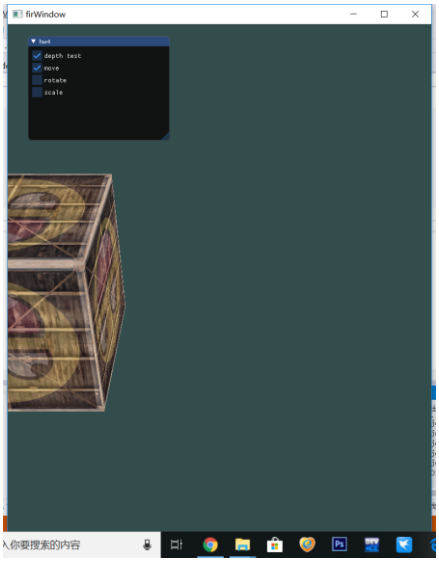
OpenGL 存储深度信息在一个叫做 Z 缓冲(Z-buffer)的缓冲中，所以可以开启深度测试，深度值存储在每个片段里面（作为片段的 z 值），当片段想要输出它的颜色时，OpenGL 会将它的深度值和 z 缓冲进行比较，如果当前的片段在其它片段之后，它将会被丢弃，否则将会覆盖，由 OpenGL 自动完成的。

加了深度测试之后的截图就正常很多了。

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

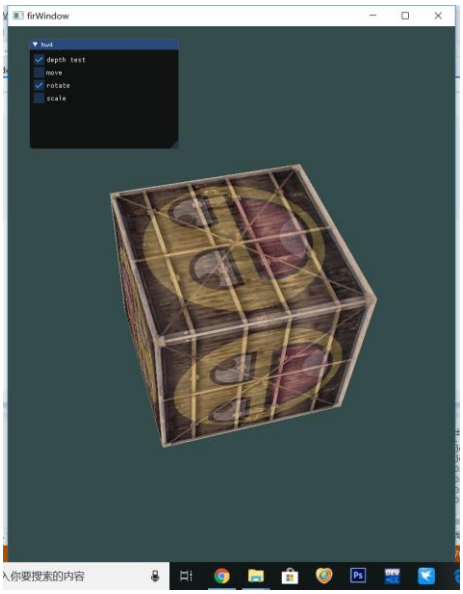
就利用 glm 中的 translate，然后一个参数是 model 矩阵，一个参数是移动的位置坐标，其实应该就相当于在 model 矩阵中和这个平移矩阵相加了。所以就弄了个额外的变量 x，每帧渲染循环中就 x+0.01，到达某个阈值就把 x 重置到最左边就行了。





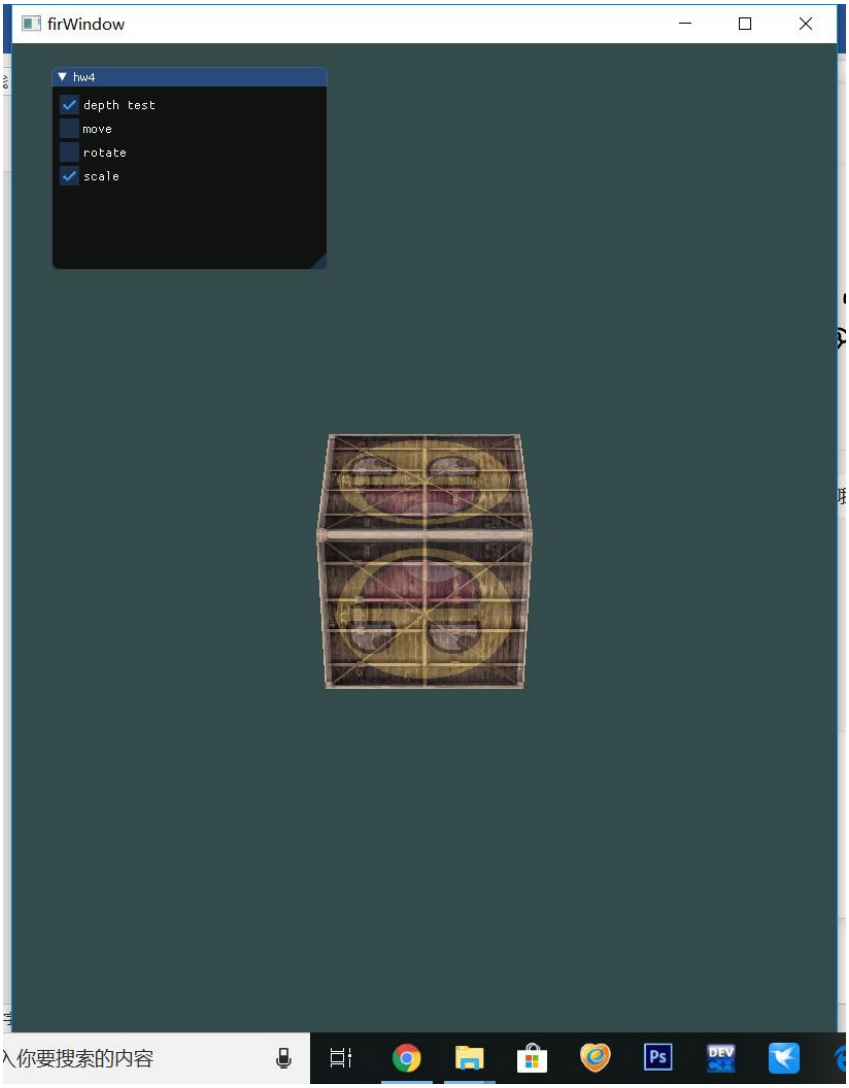
3. 旋转(Rotation)：使画好的 cube 沿着 XoZ 平面的 x=z 轴持续旋转。

利用 glm 库中的 rotate 函数，一个参数是 model 矩阵，一个参数是旋转的角度，一个参数是旋转轴，然后就会得到一个新的 model 矩阵。这里用到了 glfwGetTime()，然后转成 float 来作为旋转的角度，因为是周期的。



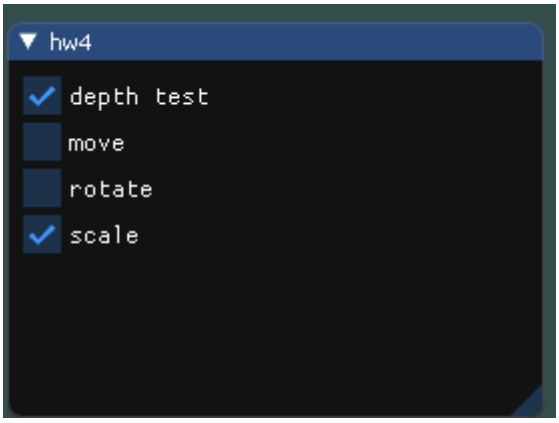
4. 放缩(Scaling)：使画好的 cube 持续放大缩小

用到了 glm 中的 scale 函数，一个参数是 model 矩阵，一个参数是三个方向变换的系数向量，用了另外一个变量 s 来纪录这个系数，当到达 0 或 1 的时候就转换变化的方向，加变成减 0.01



5. 在 GUI 里添加菜单栏，可以选择各种变换。

就加了几个 checkbox，然后就可以把上面的集中变化都混在一起了。



6. 结合 Shader 谈谈对渲染管线的理解

Shader 着色器是一种较短小的程序片段，用于告诉图形硬件怎么计算和输出图像，现在可以用高级语言来变现。

渲染管线也叫渲染流水线，是显示芯片内部能够并行的处理单元的处理过程，就是一序

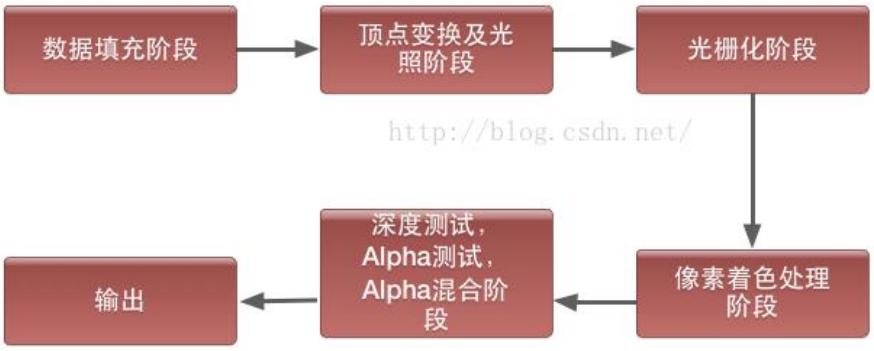
列可以并行的渲染阶段。

而 shader 就是这个流水线中某些阶段的控制程序。

这是我从网上找的图，渲染管线大概可以分成下面的阶段，数据填充阶段大概就是建模阶段，像那些顶点信息和转换矩阵那些。而顶点处理阶段应该就是根据我们的模型矩阵，观察矩阵和投影矩阵将局部坐标空间转换成世界空间，观察坐标空间到投影空间，屏幕空间等的坐标变换过程，而且这个过程是可编程的，就是我们常用的顶点着色器。光栅操作阶段属于不可编程阶段，即完全由硬件实现，包含视口变换 即把投影变换后的顶点 X,Y 坐标，根据用户设定的视口参数，变换到屏幕上对应的坐标，还有隐藏面消除 即使经过了裁剪操作，对于每个物体，摄像机只能看到正对着摄像机的一面等功能。而像素着色变换也是可编程的一部分，就是我们常用的片段着色器，进行纹理映射，纹理混合，模糊，扩散等效果。像素着色计算。这一阶段接受的数据是经过插值计算后的顶点属性。输出的是颜色值，以提供给下一阶段处理。接下来的阶段就是我们上面提到的深度测试阶段了，也是由硬件完成的，当像素经过像素着色阶段处理后，并不能都有机会输出到屏幕上，因为它们还要经过深度测试和模板测试，Alpha 测试，经过这些测试后，还要经过 Alpha 混合，这是于目标缓冲区的混合，就能够实现变透明效果。

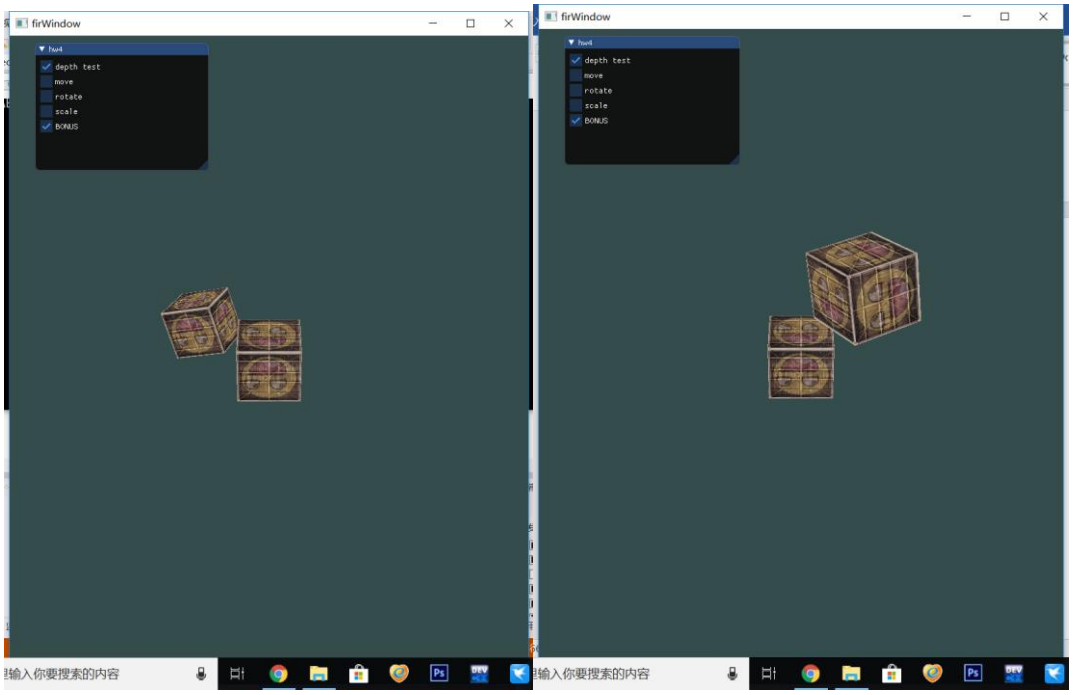
而上面所说的几个阶段是并行的流水线操作，加快处理的速度。

渲染管线流程



Bonus：

画了两个正方体，一个在中央，一个绕着中间正方题进行旋转，要画两个立方体的话，由于是一样的，就是位置那些不一样，所以就直接用了不同的 model 矩阵来转到不同的世界坐标中，然后 view 和 project 矩阵是一样的，中间那个就不动，然后另外一个就组合上面的几种动画来实现环绕的效果


firWindow
hw4
depth test
move
rotate
scale
BONUS
输入你要搜索的内容
firWindow
hw4
depth test
move
rotate
scale
BONUS
输入你要搜索的内容