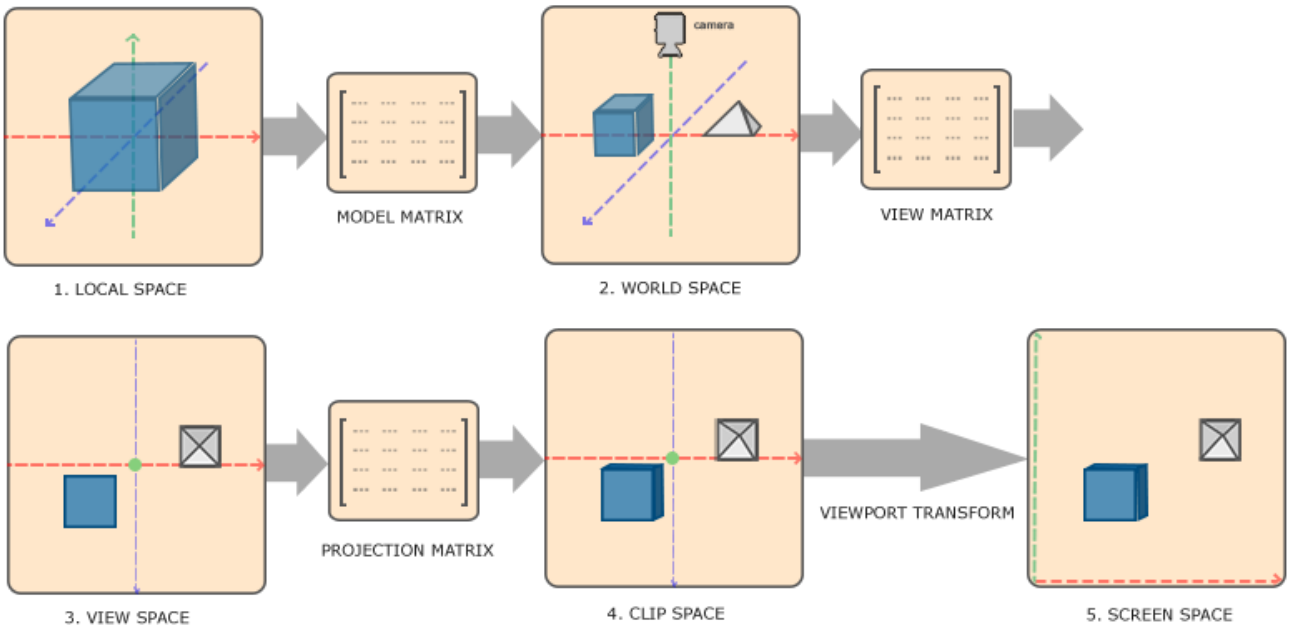
Homework 4

此次作业需要使用opengl绘制3D物体，并对它进行一些计算机图形学变换的基本操作，如平移、旋转、缩放等。

画立方体

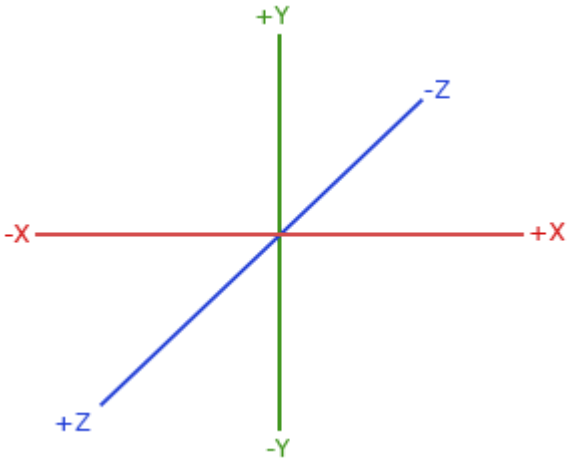
绘制立方体与绘制平面图形相同，也是由一个个三角形组成而来的。为了绘制一个立方体（边长为4），我们需要36个顶点。这里与平面图形不同，要用到坐标的第三个值—深度。

这里的坐标是局部空间内的坐标，设置了坐标后我们还要设置几个变换矩阵以对应世界空间、观察空间、裁剪空间与屏幕空间（最后输出）。其中最重要的几个分别是模型矩阵、观察矩阵与投影矩阵。它们对应的过程：



从局部空间到世界空间的转换是由模型矩阵实现的，它是一种变换矩阵，能通过对物体进行位移、旋转、缩放来将它置于相应的位置或是朝向。本次作业中平移、旋转和放缩部分基本都是通过模型矩阵实现的。

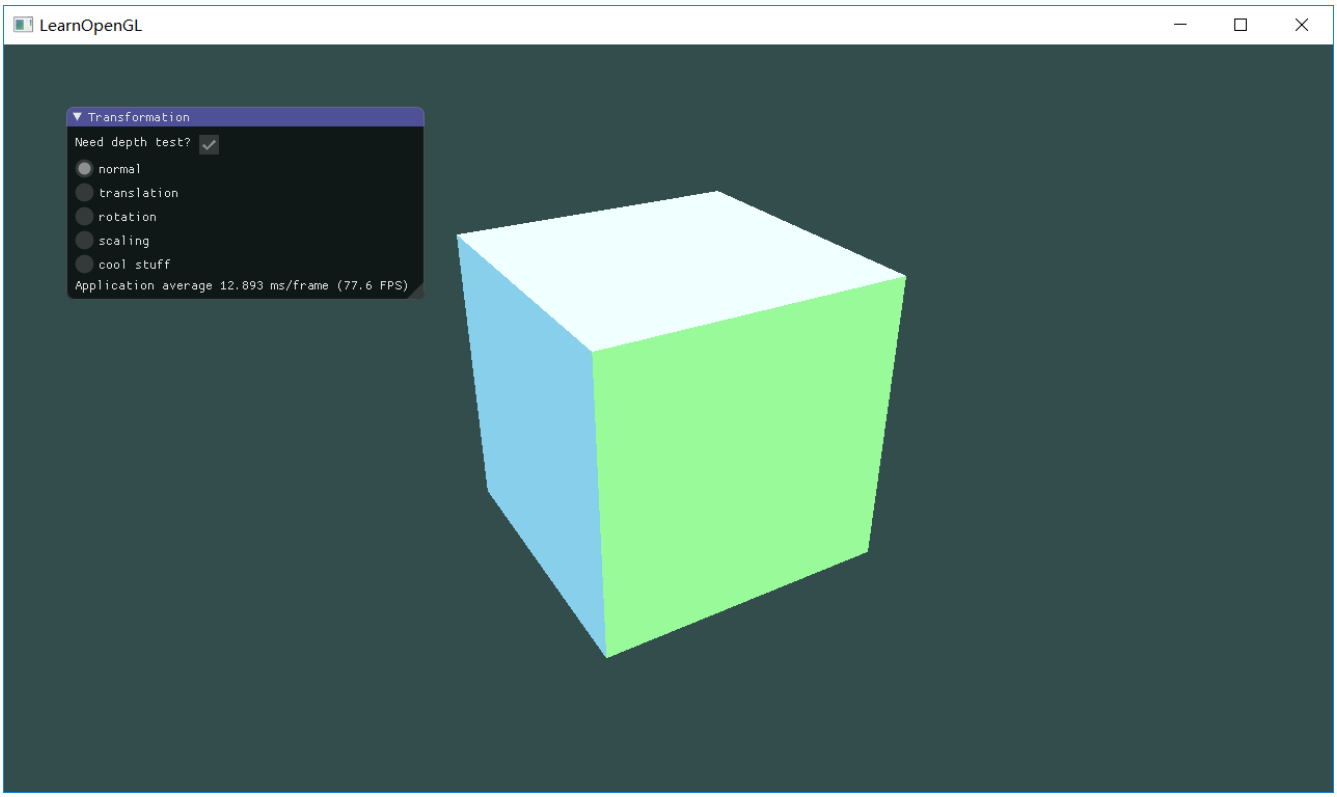
为了更好的呈现立方体的3D效果，这里首先使立方体分别沿x轴、y轴旋转一个角度，注意opengl是一个右手坐标系：



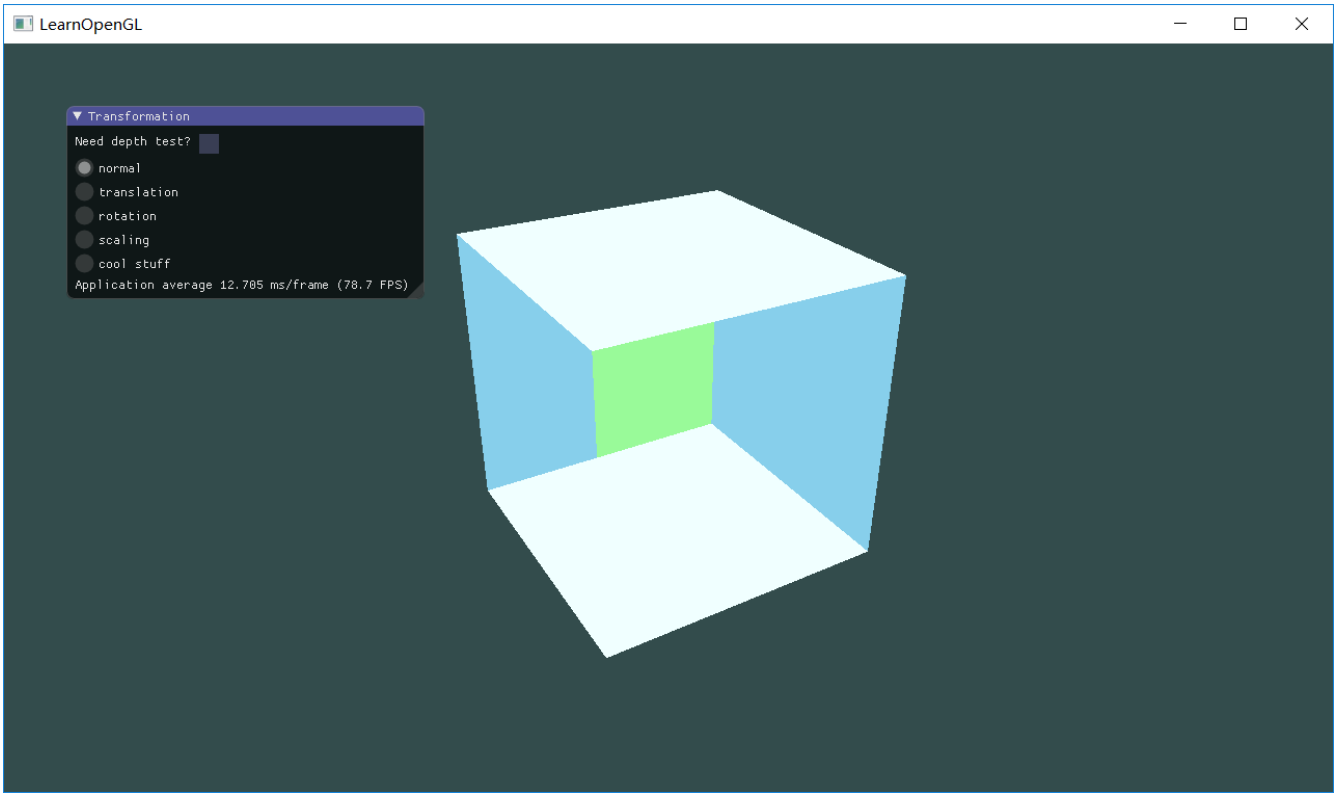
然后进行旋转操作：

```
model = glm::rotate(model, glm::radians(30.0f), glm::vec3(1.0f, 0.0f, 0.0f));   //x轴
model = glm::rotate(model, glm::radians(30.0f), glm::vec3(0.0f, 1.0f, 0.0f));   //y轴
```

得到立方体：



注意到左边的imgui窗口，当取消depth test后，立方体变成了：



可以看到，这个立方体的某些本应该被遮挡住的面被绘制在了这个立方体其他面上。这是因为关闭深度测试后，坐标对应的深度信息没有被使用，而opengl又是一个三角形一个三角形地来绘制立方体，那些本应被遮挡住的面就可能会覆盖已绘制的像素。

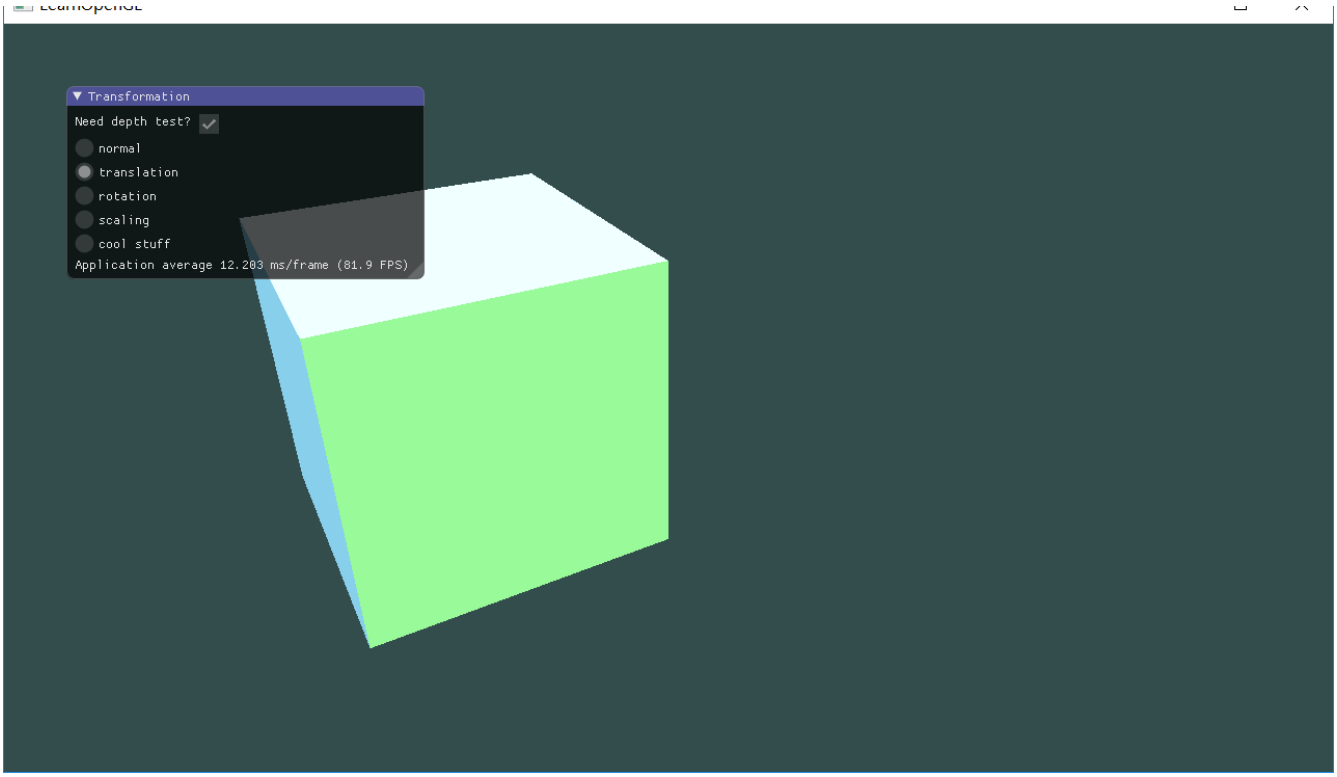
启用GL_DEPTH_TEST后，opengl将存储它的所有深度信息于一个z缓冲中，当片段想要输出它的颜色时，opengl就将它的深度值与z缓冲做比较，若当前的片段在其他片段之后，则丢弃，否则将会覆盖。

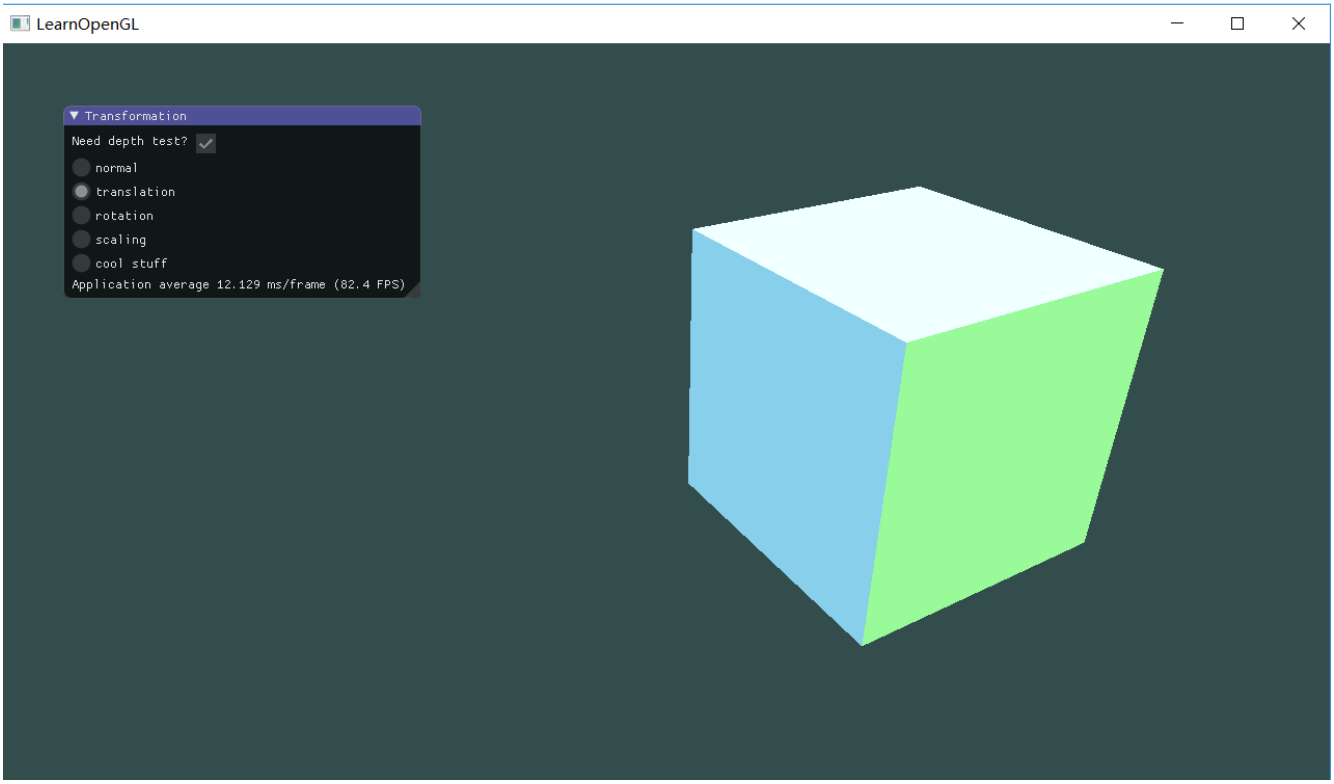
平移、旋转与缩放

要使得立方体来回移动、持续旋转或是持续放大缩小，需要用到时间函数（这里用的是glfwGetTime()）使得移动的值或是旋转的角度随时间变化而变化。但时间函数的变化是单向的，那么，怎么让立方体来回移动呢？考虑到来回变化的性质，就可以想到三角函数了，这里使用了sin与cos加上时间函数来完成这三种变换：

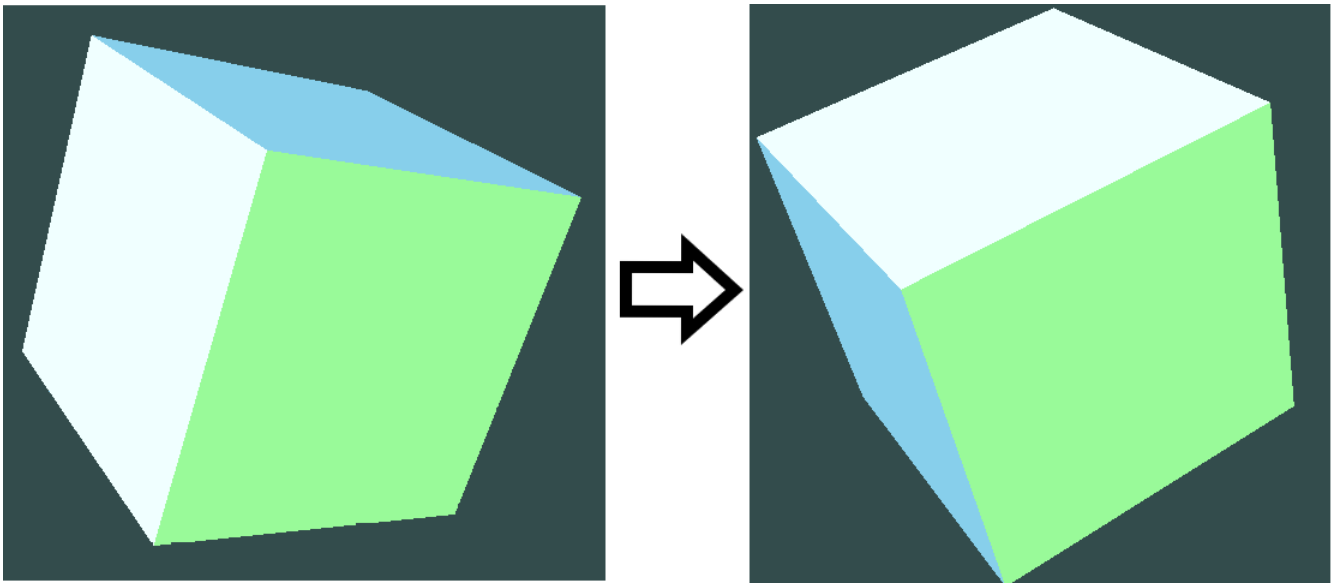
```
//perform translation
model = glm::translate(model, 3 * sin((float)glfwGetTime()) * glm::vec3(1.0f, 0.0f, 0.5f));

//perform rotation
model = glm::rotate(model, (float)glfwGetTime(), glm::vec3(0.0f, 0.0f, 1.0f));

//perform scaling
model = glm::scale(model, 2 * abs(sin((float)glfwGetTime())) * glm::vec3(0.5f, 0.5f,
0.5f));
```

平移：

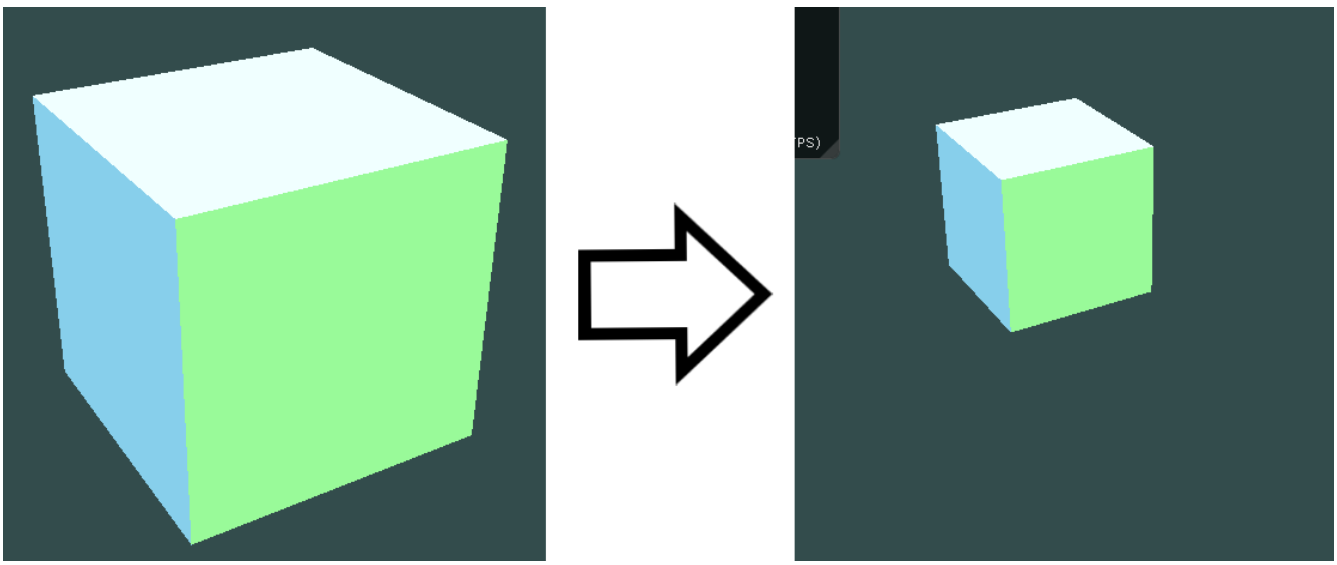

Transformation
Need depth test?
normal
translation
rotation
scaling
cool stuff
Application average 12.203 ms/frame (81.9 FPS)


LearnOpenGL
Transformation
Need depth test?
normal
translation
rotation
scaling
cool stuff
Application average 12.129 ms/frame (82.4 FPS)

旋转：



缩放：



shader与渲染管线

在本次作业（以及前几次作业中），都用到了顶点shader对象、片元shader对象。这两个对象就对应了opengl渲染管线中的顶点处理、片元操作阶段。实际上中间还存在几何处理、PA、裁剪阶段、光栅化阶段等，我们一般只控制顶点处理与片元操作就够了。

在顶点处理阶段中，对每个顶点执行shader操作，顶点数量在draw函数中指定。shader操作包括坐标空间变化、纹理坐标变化等等，如这次作业的顶点shader程序中：

```
gl_Position = projection * view * model * vec4(aPos, 1.0);
```

这里就通过硬编码来改变顶点位置（通过左乘变换矩阵做空间变换）。gl_Position是一个内置的特殊变量，用来存储顶点在齐次裁剪空间的坐标。

接着是几何处理阶段，此时顶点的邻接关系与体元语义都被传入shader，在几何shader中不仅处理顶点本身，还要考虑很多附加信息。而后又经过PA阶段，顶点shader或几何shader处理过的顶点被重新装配成为三角形。

最后是光栅化与片元操作阶段，光栅化后产生许多的片元，接着执行片元shader。在片元shader中可能会装入纹理、产生颜色，比如这次作业的片元shader程序中：

```
FragColor = vec4(oColor, 1.0f);
```

设定了顶点shader中传来的颜色。