计算机图形学

朱俊凯
16340315

1. 把运行结果截图贴到报告里，并回答作业里提出的问题。

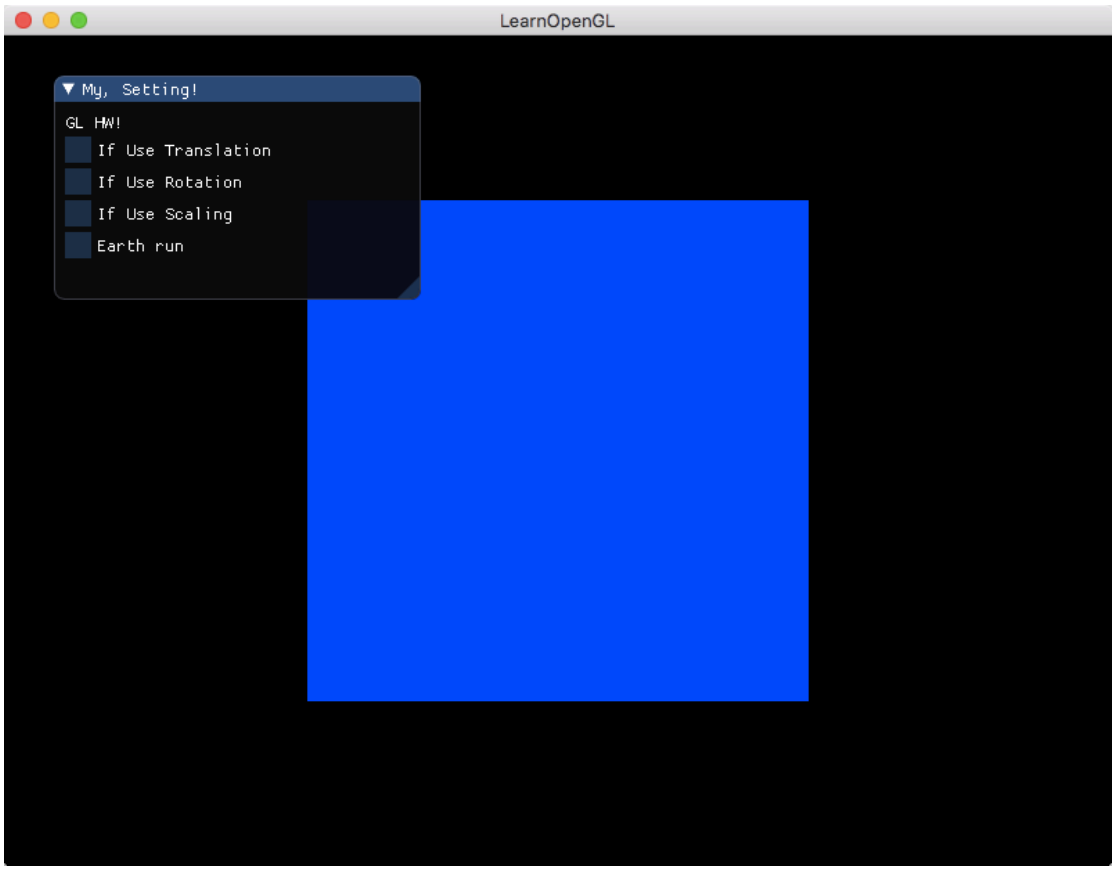
------运行 gif 在文件里面，自行查看----

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。

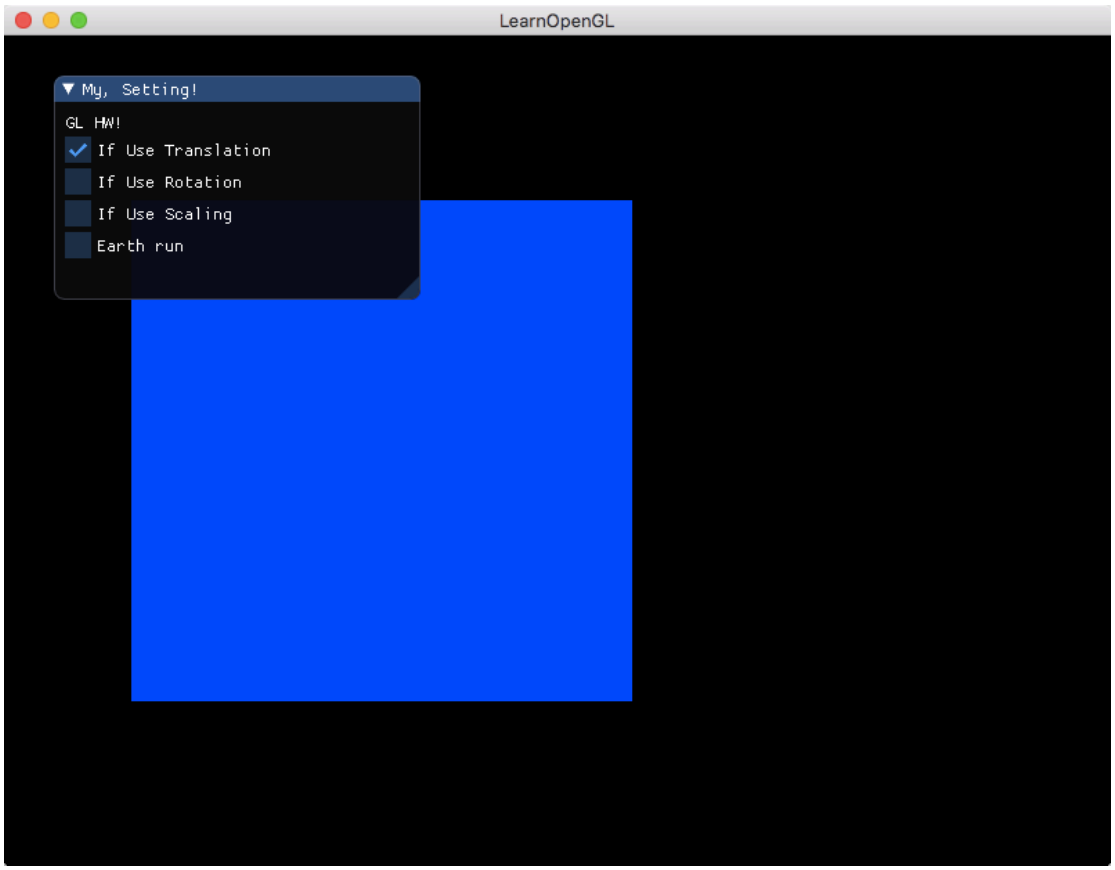
边长为 4 时，可能个人不太能理解具体的细节操作,为了能显示，就在 shader 的 gl_Position 的等式 transform 前面再乘上一个 projection 和 view 的矩阵，实现一个透视摄像机的效果。gl_Position =projection* view * transform * vec4(aPos, 1.0);

启动和关闭深度测试的区别：在关闭时，图像其实是没有深度的，某种程度上，深度信息是没有用的，也就是说，在我们简单的程序里面（z 轴值为深度信息），拥有相同 x，y 值的像素，无论 z 的值大小，总是之后渲染的像素会将之前渲染的像素覆盖掉，总是显示之后渲染的像素。

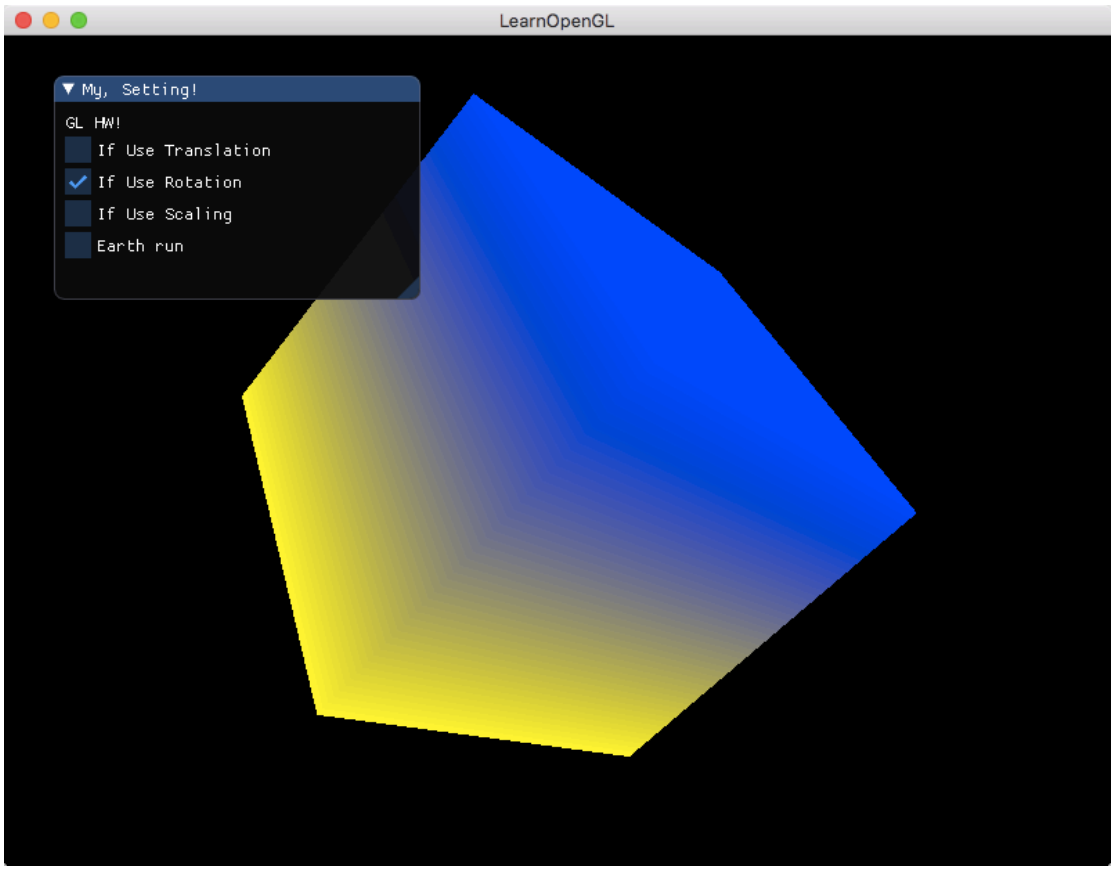
因此，启动深度测试后，深度信息（z）也就变成有用的信息了，程序会比较两个像素的深度信息，并在屏幕上渲染出相同位置深度信息小的像素来，这样就能感受到图形的相对深度了。



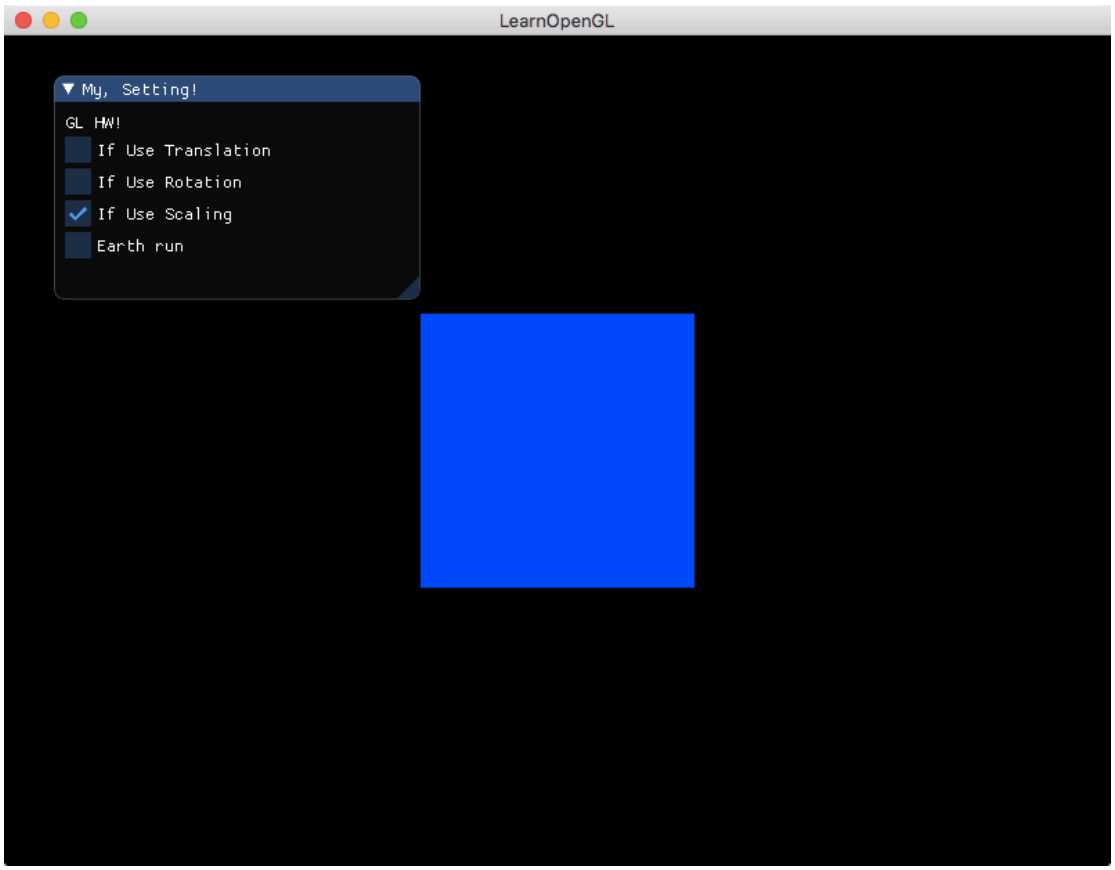
2. 平移(Translation):使画好的 cube 沿着水平或垂直方向来回移动。



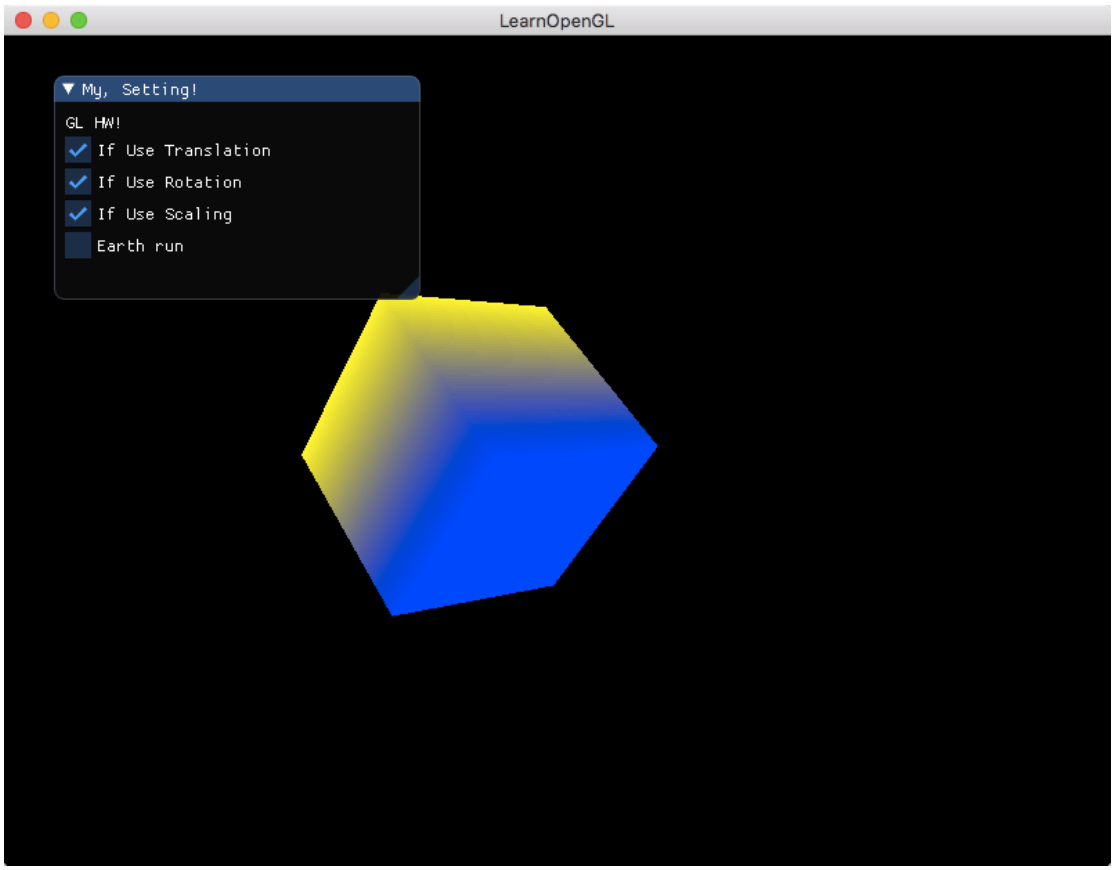
3. 旋转(Rotation):使画好的 cube 沿着 XoZ 平面的 x=z 轴持续旋转。



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



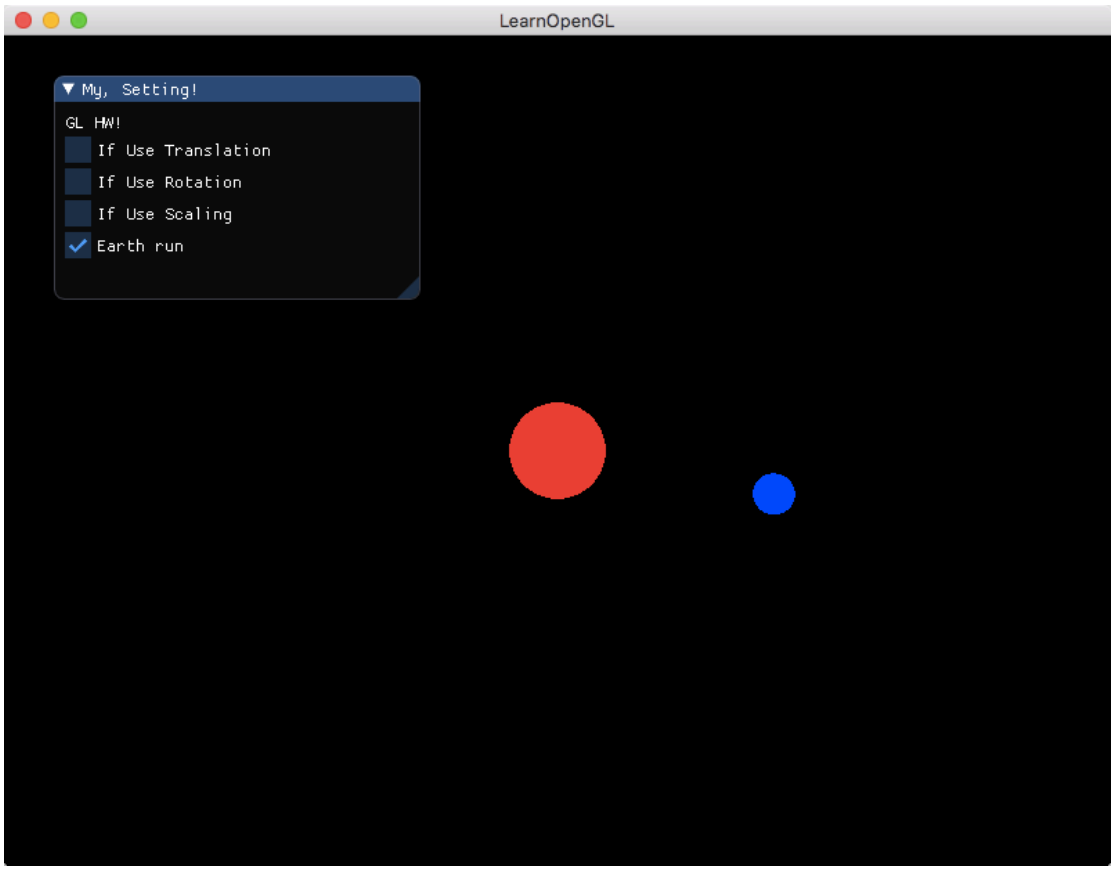
6. 结合 Shader 谈谈对渲染管线的理解

Shader 就是着色器，是一种短小的程序片段，用于告诉图形硬件如何计算和输出图像，在 opengl 里面，主要有 Vertex Shader 和 Fragment Shader 这两种，因此，总的来说，Shader 就是可编程图形管线的算法片段。

所以个人理解上，渲染管线其实就是图像处理过程中的几个相互独立的并行处理单元，可以帮忙快速的对一些输入的点，图形啊之类的进行加工处理，产生对应的输出，给其他模块，因此可以通过对渲染管线的编程，来实现对图像处理。

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。



实现思路以及主要 function/algorithm 的解释：

透视摄像机实现：

1.修改 shader 为 gl_Position =projection* view * transform *　vec4(aPos, 1.0);

2.view 为摄像头位置的变换矩阵

3.projection 为透视模式的矩阵

开启深度测试：

1. glClear(GL_DEPTH_BUFFER_BIT); glEnable(GL_DEPTH_TEST)；即可

立方体的绘制：

1.设定立方体的 vertices 顶点位置组，

2.其实立方体每个面都是由 2 个三角形构成的，因此生成对应的 12 组三角形的 indices，绑定到 EBO 上去。

3.渲染这些三角形就形成立方体。

管线编程：

1.其实比较简单，首先对 shader 代码进行修改，增加一个 transform 的变量，并且修改 gl_Position 的计算公式，让原来的位置信息加上一个 1.0 形成四元数，并且乘上 transform，就形成了新的位置信息。

形变实现：

1.接着就是产生对应的 transform 来达到对应的形变效果

2. glm::mat4 transform = glm::mat4(1.0f);生成基础的矩阵

3. 根据 glfwGetTime()通过一些除余和绝对值之类的操作，达到一个根据时间循环变化的数

4. 平移，其中 x_trans 收时间影响，因此达到随时间变化偏移量 transform = glm::translate(transform,glm::vec3(x_trans, 0.0f, 0.0f));

5. 旋转，旋转的周期基本上就是 6 秒，后面的向量参数是对应的旋转轴，transform = glm::rotate(transform, (float)glfwGetTime(), glm::vec3(1.0f, 0.0f, 1.0f));

6. 缩放，同平移，根据随时间变量，来修改缩放值的大小，实现对应的缩放动画效果，transform=glm::scale(transform, glm::vec3(scalesize, scalesize, scalesize));

7 获取 shader 里面的 transform 变量的位置，并且将我们刚获得的 transform 提交上去，即渲染时使用我们生成的 transform 来处理图像. glUniformMatrix4fv(transformLoc, 1, GL_FALSE, glm::value_ptr(transform));

球的绘制：

1.其实就是通过算法生成更多的点，渲染更多的三角形面，从而实现近似球的巨多面体。