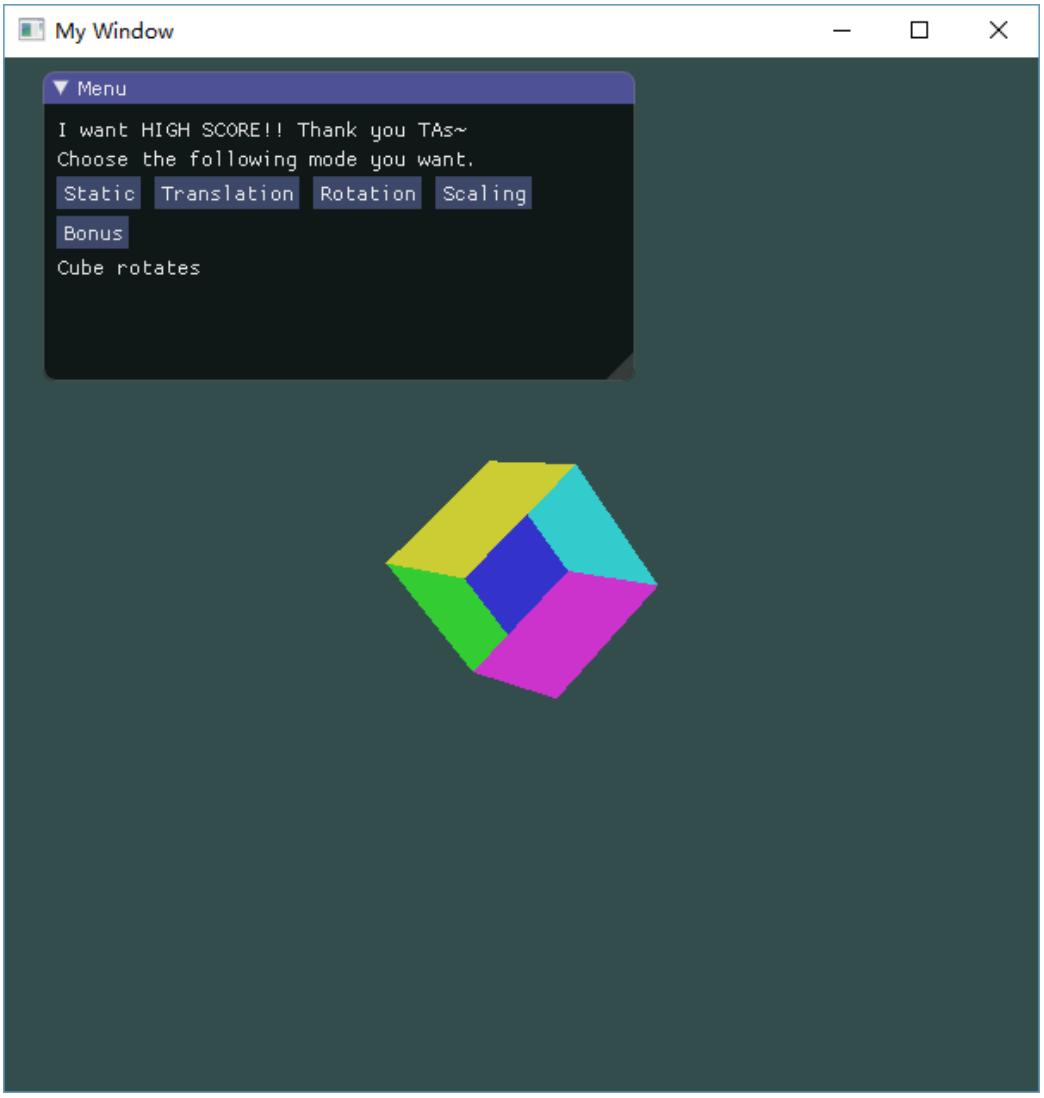
Homework 4

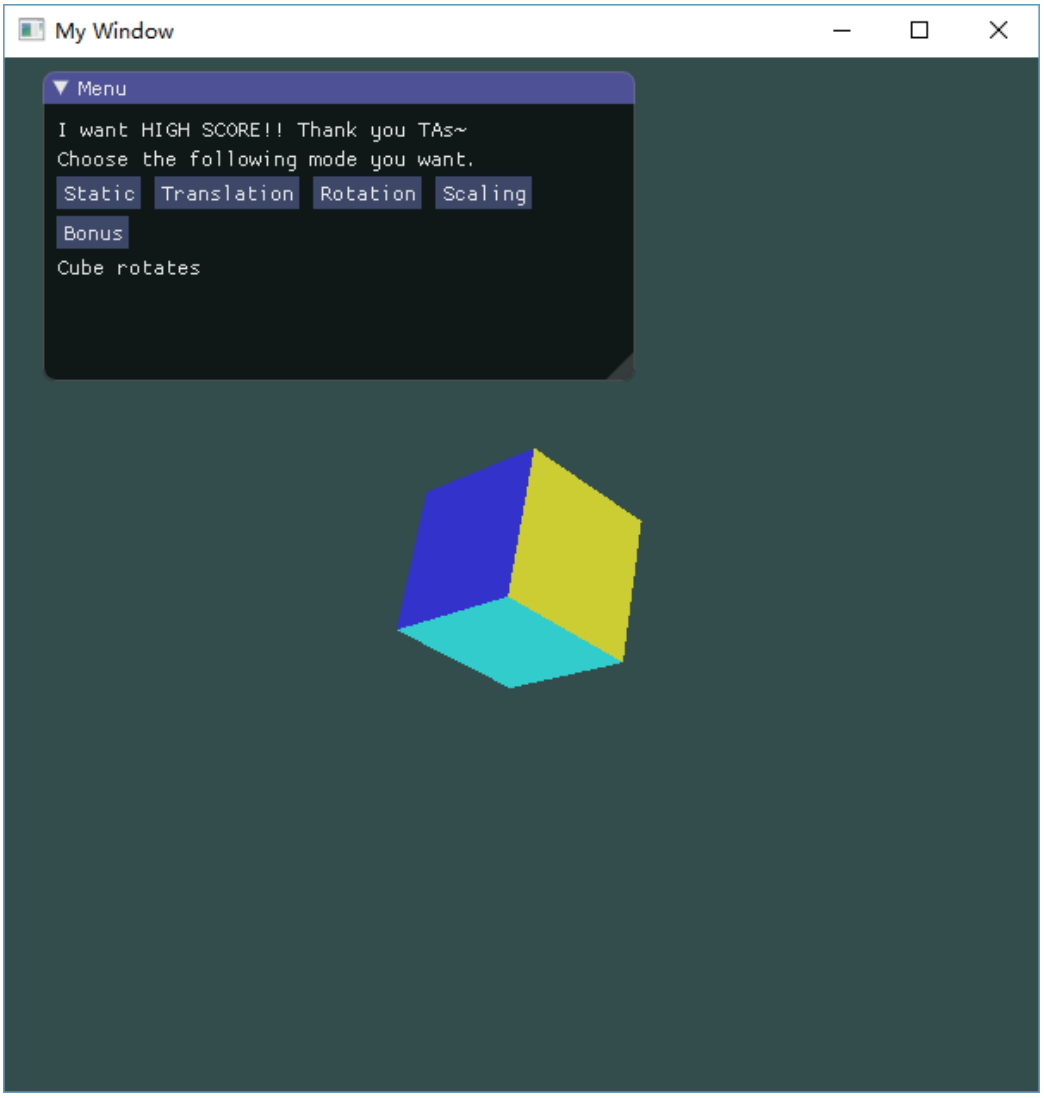
Basic

1. 画一个 Cube， 并观察有无开深度测试的区别

实验截图


My Window
Menu
I want HIGH SCORE!! Thank you TAs~
Choose the following mode you want.
Static
Translation
Rotation
Scaling
Bonus
Cube rotates

没有开启深度测试



开启深度测试

算法实现

设置好顶点信息，并且按照要求修改一下着色器的程序。

```
    float vertices[] = {
        // 背面
        // 顶点位置              颜色
        -0.2f, -0.2f, -0.2f,  0.8f, 0.2f, 0.2f,
        0.2f, -0.2f, -0.2f,   0.8f, 0.2f, 0.2f,
        0.2f,  0.2f, -0.2f,   0.8f, 0.2f, 0.2f,
        0.2f,  0.2f, -0.2f,   0.8f, 0.2f, 0.2f,
        -0.2f,  0.2f, -0.2f,  0.8f, 0.2f, 0.2f,
        -0.2f, -0.2f, -0.2f,  0.8f, 0.2f, 0.2f,
        ...
    }
```

顶点着色器加上 3 个 uniform 的 mat4 分别表示 model，view 和 projection 矩阵。

```
#version 330 core

```

```
layout (location = 0) in vec3 aPos;
layout (location = 1) in vec3 aColor;

out vec4 vColor;
uniform mat4 model;
uniform mat4 view;
uniform mat4 projection;

void main() {
    gl_Position = projection * view * model * vec4(aPos, 1.0);
    vColor = vec4(aColor, 1.0);
}
```

然后正常绑定 VAO 和 VBO 并且画出来。和以前作业的不同之处在于现在顶点加入了深度信息，z 方向上的变量不在恒为0，同时需要开启深度测试 `glEnable(GL_DEPTH_TEST)` ，并在 render loop 中使用 `glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)` 。

view 和 projection 的矩阵在本次作业中可以保持不变，将 view 稍微抬高一点以看到立方体的不同面。

```
view = glm::lookAt(glm::vec3(0.0f, 1.0f, 3.0f),
            glm::vec3(0.0f, 0.0f, 0.0f),
            glm::vec3(0.0f, 1.0f, 0.0f));
projection = glm::perspective(45.0f, (float)SCR_WIDTH / (float)SCR_H
EIGHT, 0.1f, 100.0f);
```

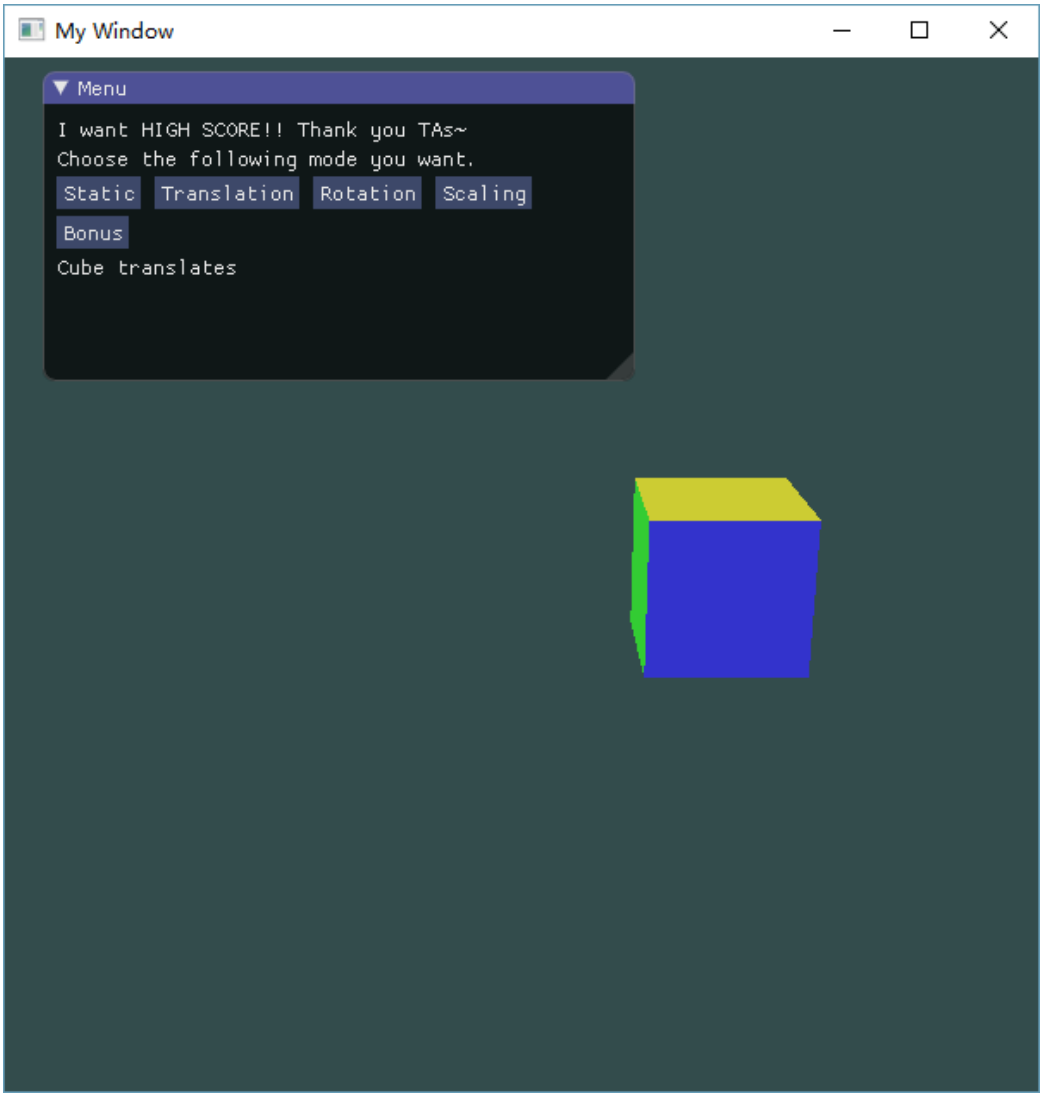
现象解释

没有开启深度测试的时候输出的确有点像是一个立方体，但又有种说不出的奇怪。立方体的某些本应被遮挡住的面被绘制在了这个立方体其他面之上。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。而当你开启了深度测试后，就不会出现这种奇怪的现象。

参考资料

2. 实现平移

实验截图



平移

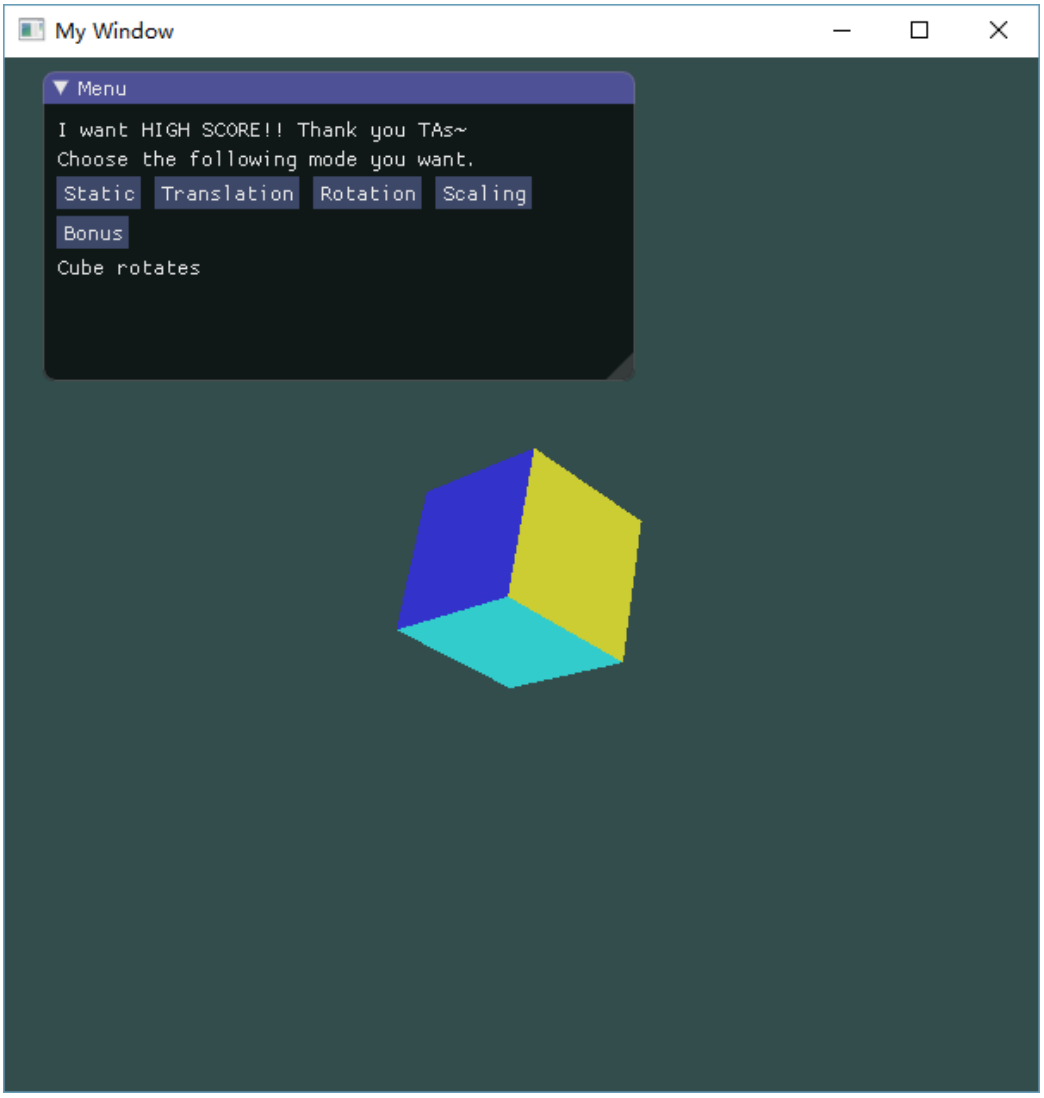
算法实现

view 和 projection 保持不变，使用 `glm::translate` 函数来改变 model 矩阵，实现水平平移，其中，使用 `sin` 和 `glfwGetTime()` 函数来实现水平**来回**平移。

```
1. model = glm::translate(model, (float)sin(glfwGetTime()) * glm::vec3(
   0.5f, 0.0f, 0.0f));
```

3. 实现旋转

实验截图



旋转

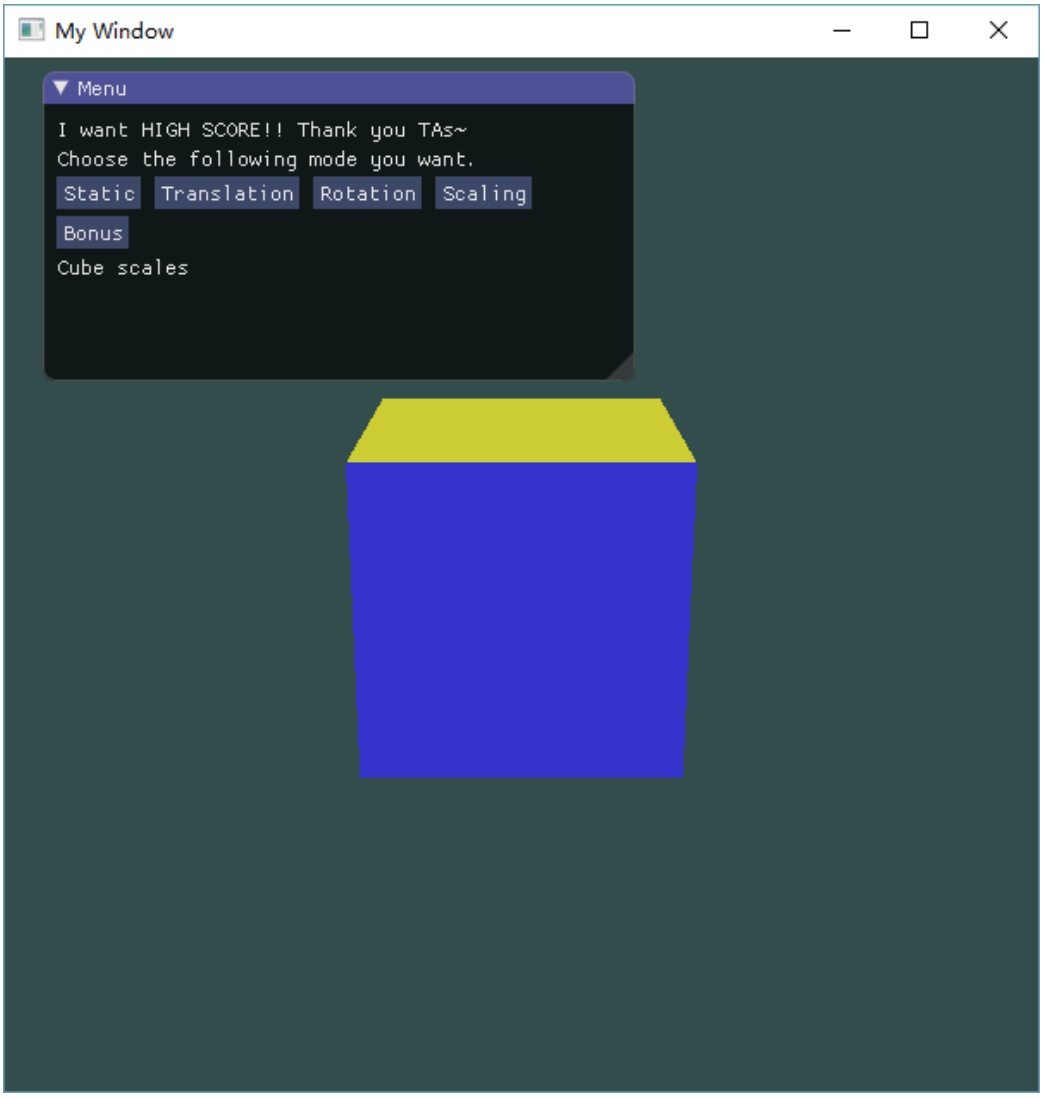
算法实现

view 和 projection 保持不变，使用 `glm::rotate` 函数来改变 model 矩阵，实现旋转，其中，在 0.9.5.4 版本的 glm 中， `glm::rotate` 函数里面的角度参数是用角度制而不是用弧度制，和作业给出的教程有出入，需要注意。使用 `glfwGetTime()` 来实现持续旋转。

```
1. model = glm::rotate(model, (float)glfwGetTime() * 80.0f, glm::vec3(
   0.0f, 1.0f, 1.0f));
```

4. 实现放缩

实验截图



放缩

算法实现

view 和 projection 保持不变，使用 `glm::scale` 函数来改变 model 矩阵，实现放缩，其中，使用 `abs(sin(glfwGetTime()))` 来实现持续的放大和缩小。

```
1. model = glm::scale(model, (float)abs(sin(glfwGetTime())) * glm::vec3
   (2.0f, 2.0f, 2.0f));
```

Bonus

将以上三种变换相结合，实现有创意的动画。 比如： 地球绕太阳转等。

实验截图

简化实现太阳系星体转动动的模拟，用立方体来代替模拟星球的球形。
中心最大的立方体是太阳，太阳自转。

第二大的立方体是地球，围绕太阳转。
最小的是月球，围绕地球转。

算法实现

对于画多个立方体，只需要对每一个不同的立方体定制属于它的 model 矩阵即可。

> PS：一开始写得比较着急，另外又写了一组顶点信息，但是到后来细想其实没有必要，**先放缩就可以得到不同大小的立方体**。这里的代码有点冗余，**去掉重新调参需要花费挺多时间的，所以这里就不删了。**

1. 首先设置好一个 model 矩阵 vector， 准备用来存储三个星体的变换。

```
1. std::vector<glm::mat4> carModel;
2. carModel.assign(carNum, glm::mat4(1));
```

2. 用放缩设置各个星体的大小，用平移设置各个星体的初始在世界坐标系中的位置，用旋转来模拟星球转动的动画。其中，

太阳：

```
carModel[0] = glm::scale(carModel[0], glm::vec3(2.0, 2.0, 2.0));
carModel[0] = glm::rotate(carModel[0], (float)glfwGetTime() * 30.0f,
    glm::vec3(0.0f, 0.0f, 1.0f));
```

地球：

```
carModel[1] = glm::rotate(carModel[1], (float)glfwGetTime() * 40.0f,
    glm::vec3(0.0, 0.0, 1.0));
carModel[1] = glm::translate(carModel[1], glm::vec3(0.55, 0.55, 0.0
));
```

月球：

```
carModel[2] = glm::scale(carModel[2], glm::vec3(0.7, 0.7, 0.7));
// 因为月球需要围绕着地球转动，所以缩放成合适大小后，需要先乘一个地球的 model 矩
阵，将坐标系转到以地球为中心。
carModel[2] = carModel[1] * carModel[2];
carModel[2] = glm::rotate(carModel[2], (float)glfwGetTime() * 50.0f,
    glm::vec3(0.0, 0.0, 1.0));
carModel[2] = glm::translate(carModel[2], glm::vec3(0.3, 0.3, 0.0));
```

上面代码的一些重要的解释已经在注释中说明，不再赘述。

行星模拟这里使用了一个新的 view 使得展示效果更好一点。

```
view = glm::lookAt(glm::vec3(0.0f, -2.0f, 3.0f),
                   glm::vec3(0.0f, 0.0f, 0.0f),
                   glm::vec3(0.0f, 1.0f, 0.0f));
```