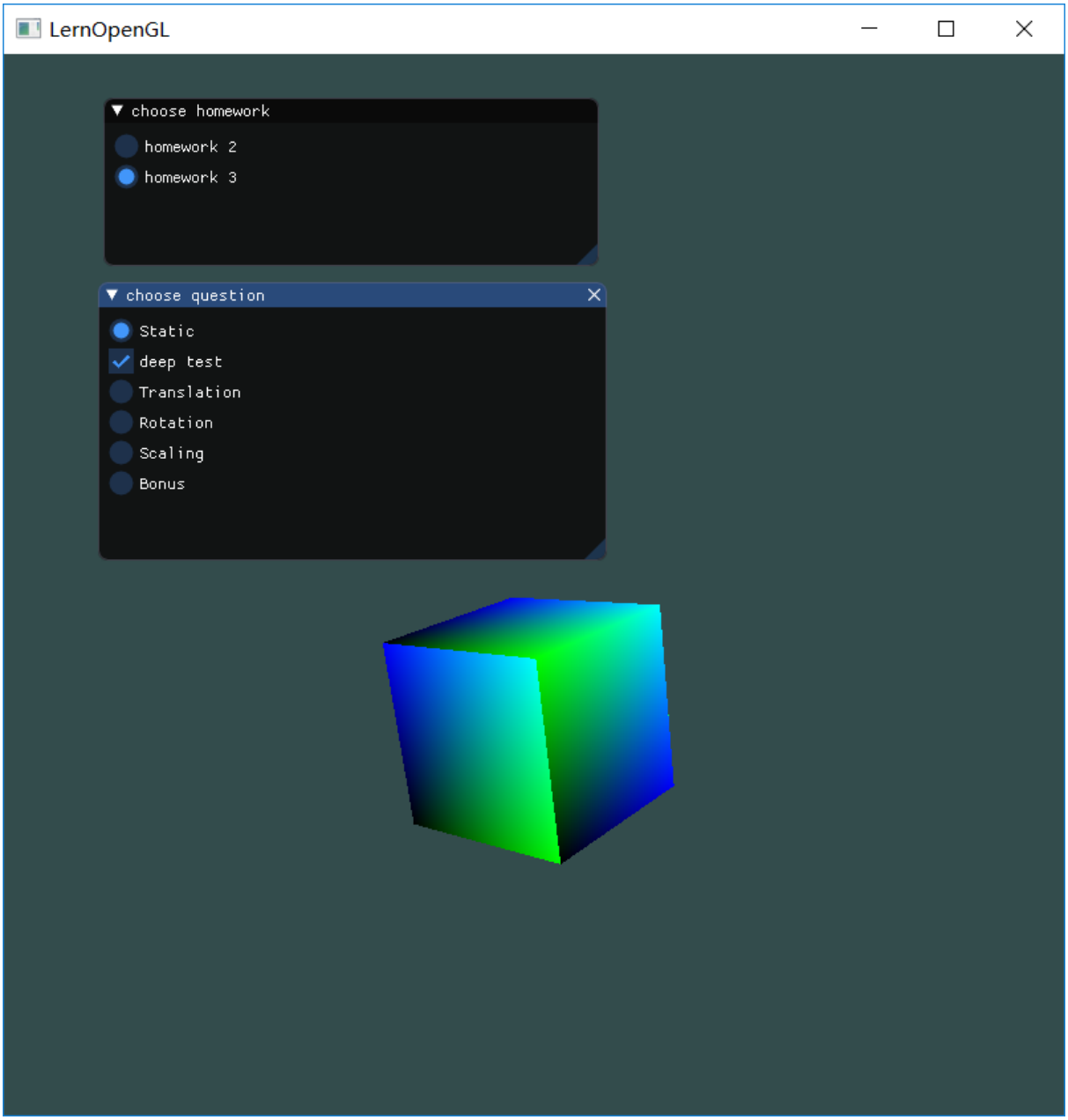
实验报告

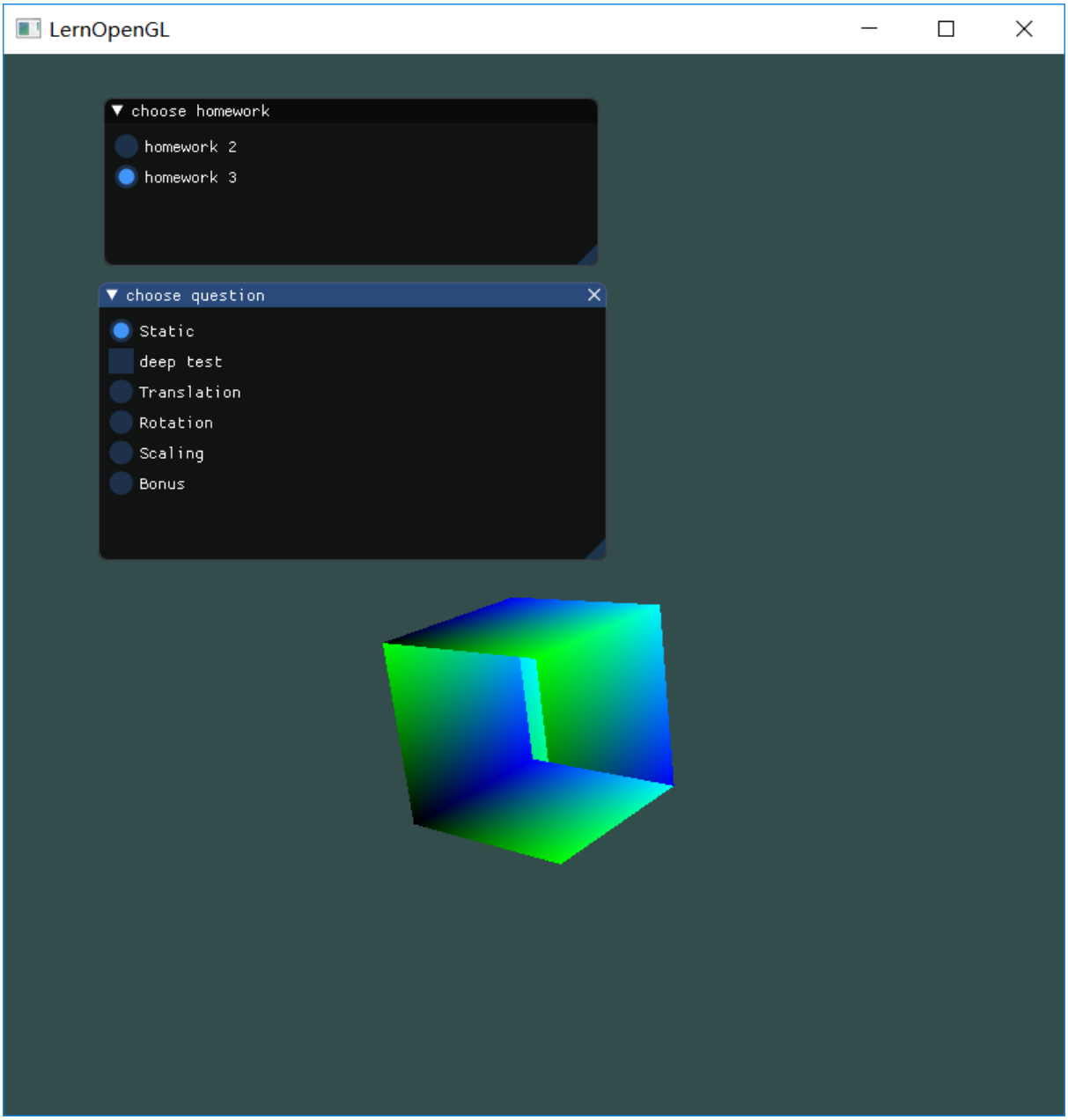
1. 画正方形

画一个正方形，边长为4，中心位置(0, 0, 0)。分别启动和关闭深度测试

1. 实验结果
    1. 开启深度测试


LernOpenGL
choose homework
homework 2
homework 3
choose question
Static
deep test
Translation
Rotation
Scaling
Bonus

2. 关闭深度测试



2. 实现思路

    1. 创建窗口、初始化ImGui等（和上一次作业一样）
    2. 创建立方体的顶点数组，每一面用两个三角形模拟，后面跟着是颜色参数
    3. 创建shader类，其中，顶点着色器的position为：

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

    shader类包含函数：设置shader、glUseProgram、setMat4

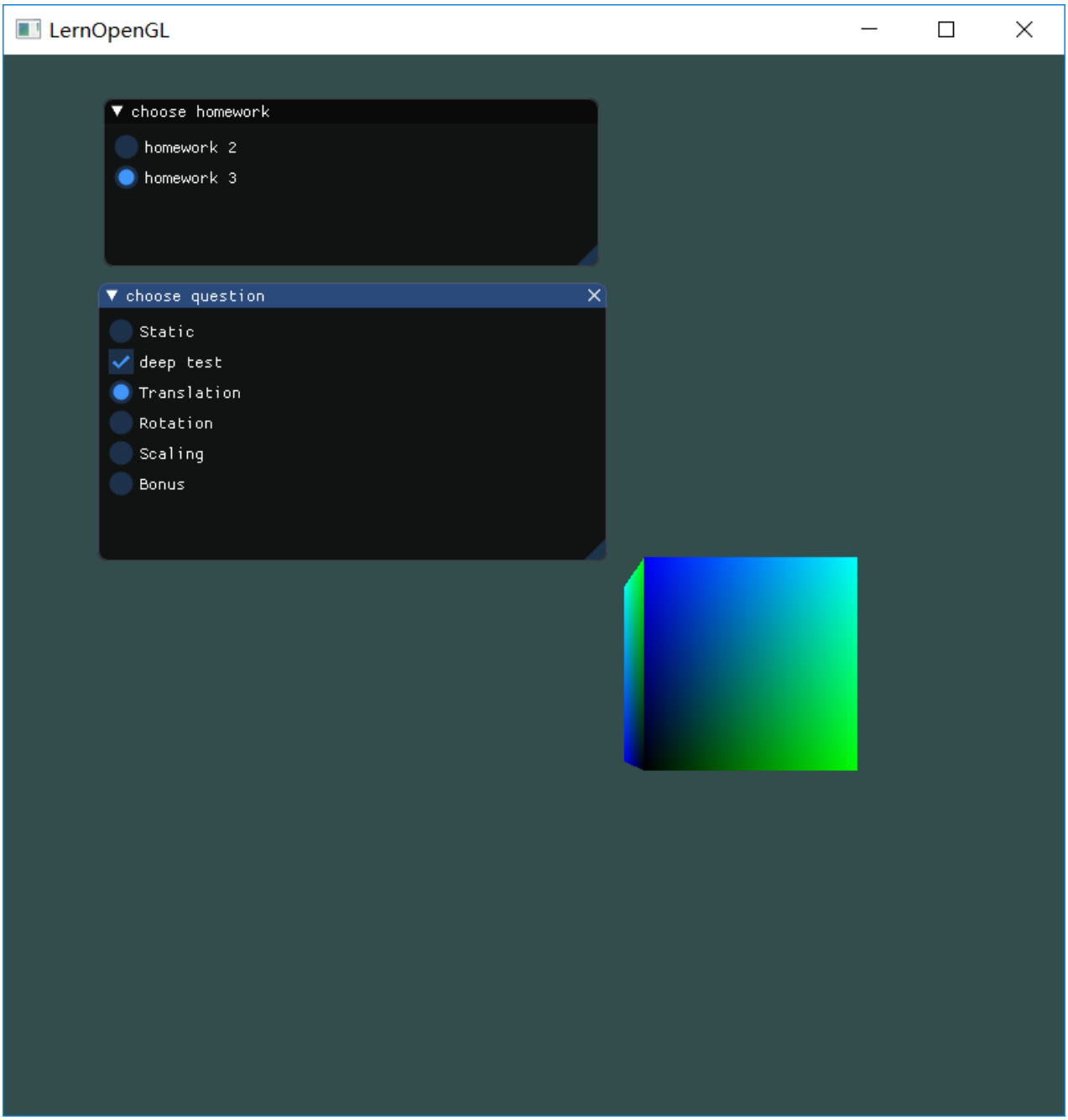
    4. 根据选择使用glEnable和函数glDisable设置或者取消深度测试
    5. 和之前方法一样，将数组的内容添加到缓存里
    6. 设置model（模型矩阵）、view（观察矩阵）、projection（投影矩阵）
        1. 其中，模型矩阵绕y轴旋转45°，绕x轴旋转22.5°使得能看到三个面
        2. 观察矩阵在z轴平移-20，即在z轴远离物体20的地方观察它
        3. 投影使用透视投影
    7. 通过函数 glGetUniformLocation 和 glUniformMatrix4fv 函数将矩阵传入着色器

8. 通过函数glBindVertexArray 和 glDrawArrays 函数将点画出

2. 平移

使画好的cube沿着水平或垂直方向来回移动。

1. 实验结果



2. 实现思路

    1. 其余部分和上一题一样，只是修改模型矩阵：模型矩阵随时间在x轴上左右运动

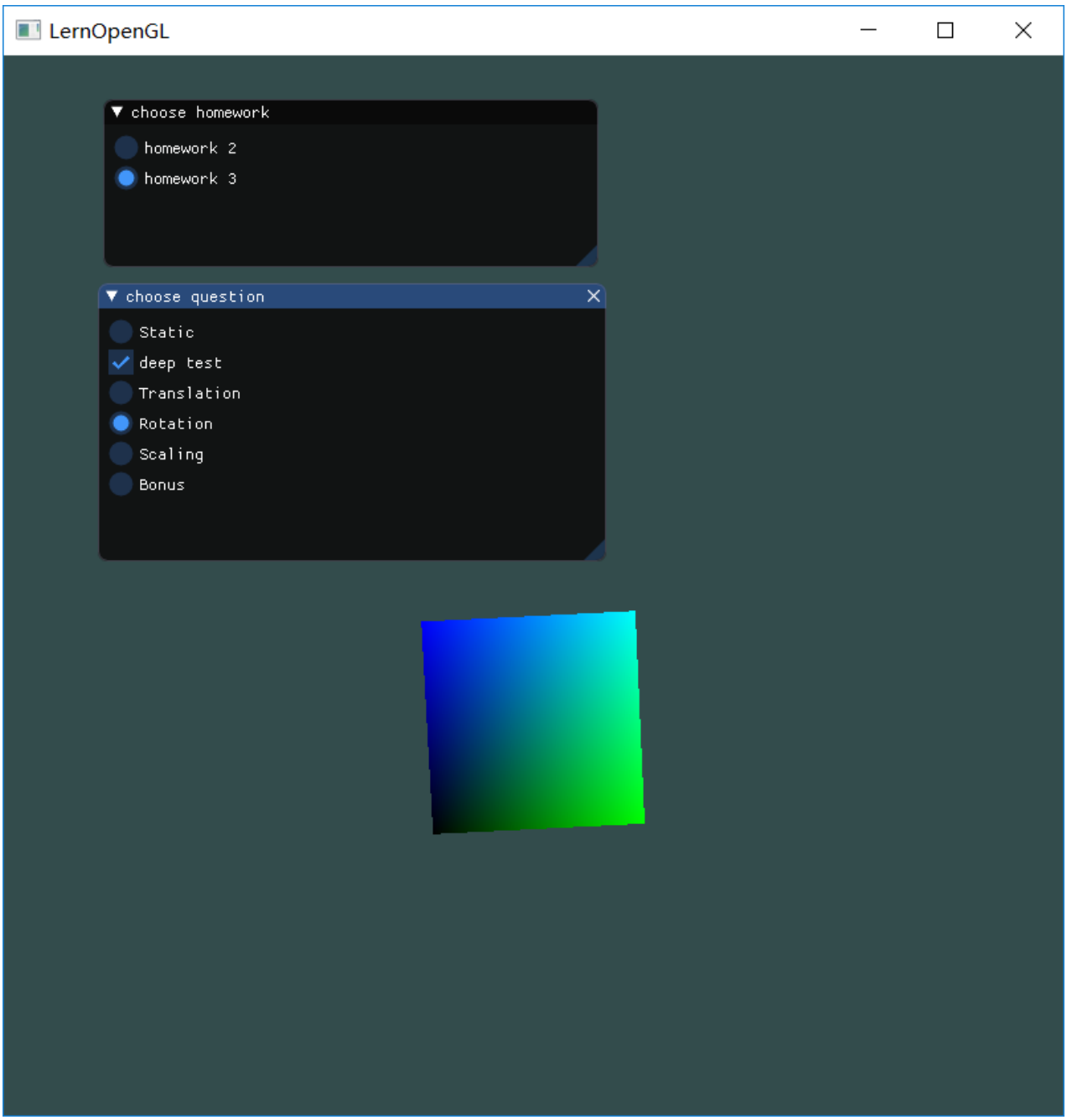
        1. 使用函数 `glm::translate`，传入的vec3的x部分的值为：5*sin(5*glfwGetTime())，使得该处的值在[-5,5]上变化实现左右移动的效果

- - -

3. 旋转

使画好的cube沿着XoZ平面的x=z轴持续旋转。

1. 实验结果

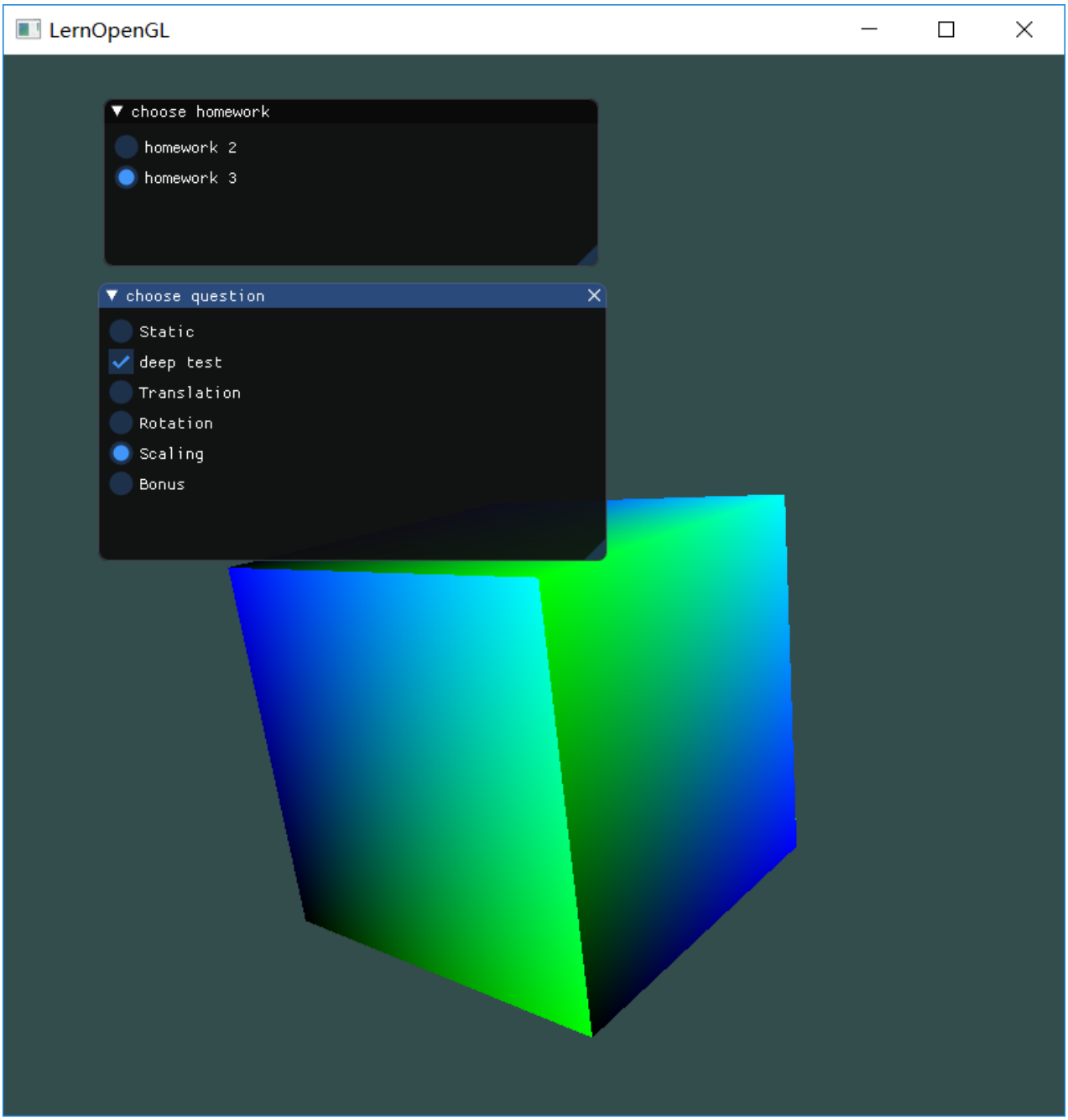


2. 实现思路
    1. 其他与前面的一样，只是修改模型矩阵
        1. 使用函数 `glm::rotate`，传入的旋转角度值为：50.0f*(float)glfwGetTime()，vec3传入x和z都是1.0f，使得该立方体绕x=z旋转

4.放缩

使画好的cube持续放大缩小。

1. 实验结果



2. 实现思路
    1. 其他与前面的一样，只是修改模型矩阵
        1. 使用函数 `glm::rotate` ，和静止的状态一样，再使用 `glm::scale` ，传入的数均为abs(3*sin(glfwGetTime()))

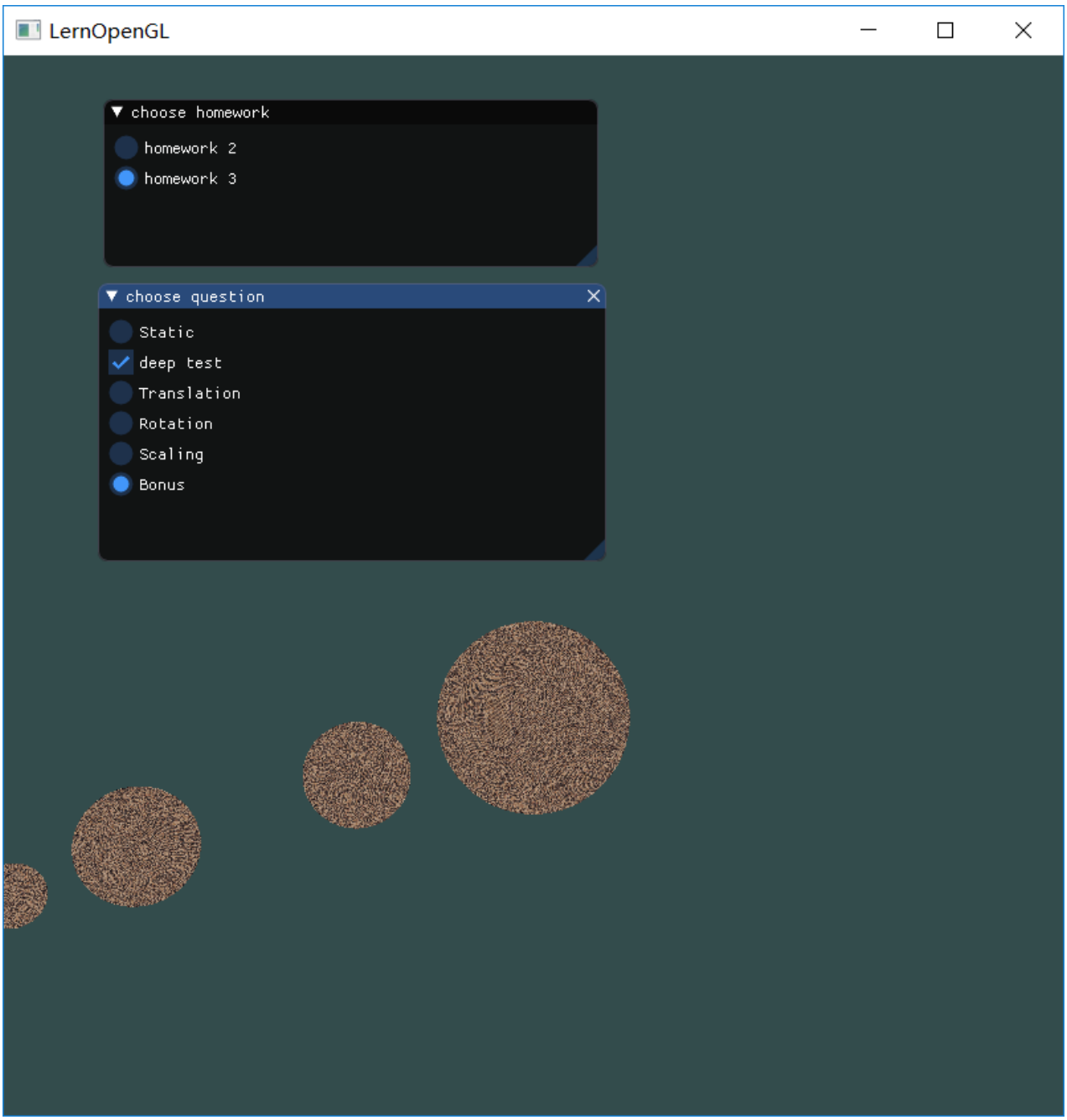
5. 结合Shader谈谈对渲染管线的理解

shader分成顶点着色器和片段着色器，顶点着色器负责将3D坐标转换成屏幕坐标，并处理输入的传入的顶点颜色数据，片段着色器负责处理传入的颜色或者纹理图案，着色器负责将这些组合起来然后输出。

渲染管线的渲染过程是模型数据→顶点着色器→图元装配→几何着色器→光栅化→片段着色器→测试和混合，其中顶点着色器、几何着色器、片段着色器均可以自行编写，顶点数据就是一个顶点的集合，通过顶点着色器将集合里每个点的3D坐标从局部坐标乘模型矩阵转换成世界坐标，再从世界坐标乘观察矩阵转换成观察空间坐标，再乘投影矩阵获得裁剪空间坐标，最后通过视口变换变成屏幕坐标。图元装配是将顶点着色器的输出作为顶点输入，将所有点装配成指定的图元的形状。上一步的结果传递到几何着色器，，通过新产生的顶点构造出新的图元生成其他形状。然后进入光栅化，即将区域进行分块，把图元映射为屏幕上相应的像素，生成供片段着色器使用的片段。片段着色器用于计算每块显示的颜色、纹理等。测试和混合阶段则是将前一步计算好的数据通过深度测试判断遮挡关系，决定是否丢弃，然后混合处理。

Bonus：太阳系部分星球环绕运动

1. 实验结果：



2. 实现思路

1. 创建窗口、初始化ImGui等（和上一次作业一样）
2. 设置纹理坐标的数组，根据后面一步会提到的模拟球体的方法，将左上、左下、右上、左下、右上、右下的值传入数组中。
3. 创建球体的顶点数组
    1. 将球体进行横向切割纵向切割成很多个小块（列要是偶数且不能被4整除），每个小块模拟成矩形，然后拆分成两个三角形进行模拟
    2. 具体实现
        1. 遍历球体的每层，计算每层上下两条边的y值，和每层的俯视图的那个圆形的半径
        2. 遍历某一层两次，计算每列在该层上下两条边交点的x和z坐标，加入vector中，遍历两次的目的：遍历第一次每一个矩形只传入了一个三角形
        3. 设置纹理坐标：记录一个index，从0开始将第二步创建的数组依次传入vector里面
4. 创建shader类，其中，顶点着色器的position为：

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

    shader类包含函数：设置shader、glUseProgram、setMat4
5. 创建纹理：使用函数 glGenTextures glBindTexture glTexParameteri 创建、绑定等操作，再通过 stbi_load 读取图片，glTexImage2D 将图片传入纹理
6. 和之前方法一样，将数组的内容添加到缓存里
7. 设置model（模型矩阵）、view（观察矩阵）、projection（投影矩阵）
    1. 其中，模型矩阵绕y轴旋转45°，绕x轴旋转22.5°
    2. 观察矩阵在z轴平移-20，即在z轴远离物体20的地方观察它
    3. 投影使用透视投影
8. 通过函数 glGetUniformLocation 和 glUniformMatrix4fv 函数将矩阵传入着色器
9. 通过函数glBindVertexArray 和 glDrawArrays 函数将点画出
10. 创建一个其他球体的位置数组
11. 这里模拟了地球绕太阳转，金星绕太阳转，月球绕地球转，地球、金星、月球自转，下面分别阐述实现过程
    1. 前面绘出的是静止的太阳
    2. 模型矩阵使用 `glm::scale()` 函数将球体缩小成原来的一半，再对这个矩阵用 `glm::rotate` 设置球体绕（0，0，0）点随时间旋转
    3. 遍历9创建出来的数组
        1. 如果index不为2，即金星和太阳
            1. 让其根据9的数组平移到指定位置，并通过 `glm::rotate` 设置自转（这里传出的矩阵不再是原来的model，防止自转对下一个球体产生影响）
            2. glUniformMatrix4fv 函数处理模型矩阵，glDrawArrays 绘出图形
        2. 如果是月亮，先设置它旋转（旋转速度更快），此时由于未移动，圆心为地球的球心，并设置缩放为地球的1/2，后面步骤和index为2的一样