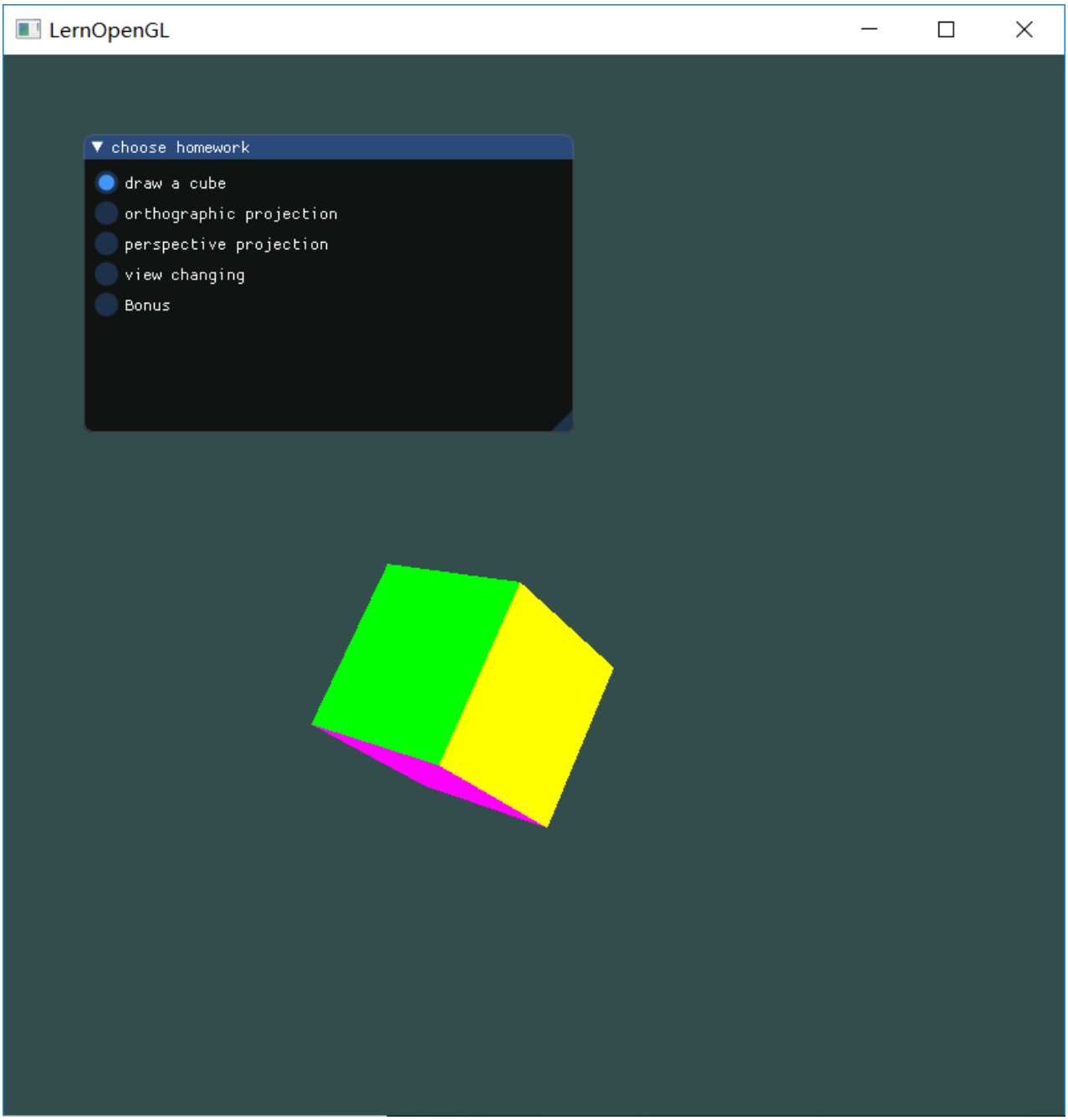
实验报告

1. Basic

1. 投影

1. 把上次的立方体放到（-1.5，0.5，-1.5）位置，要求6个面颜色不一致

1. 实验结果：



2. 实验过程：

    1. 跟上次作业一样画出立方体，其中，立方体顶点数据每一面设置的颜色要不一样

顶点数据：（分别是顶点位置、颜色）

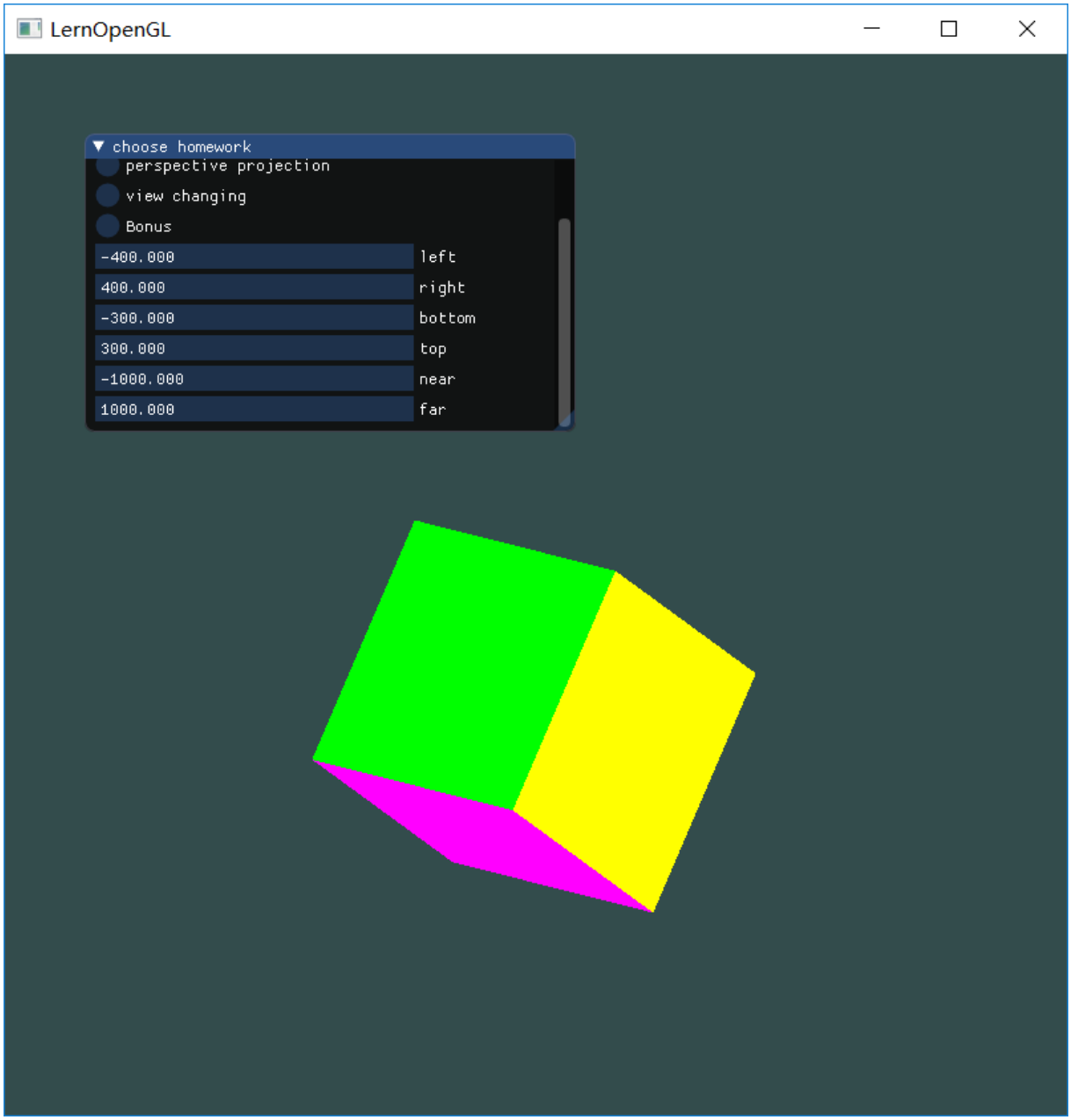
```
float vertices3[] = {
        -2.0f, -2.0f, -2.0f, 1.0f,  0.0f, 0.0f,
         2.0f, -2.0f, -2.0f, 1.0f,  0.0f, 0.0f,
         2.0f,  2.0f, -2.0f, 1.0f,  0.0f, 0.0f,
         2.0f,  2.0f, -2.0f, 1.0f,  0.0f, 0.0f,
        -2.0f,  2.0f, -2.0f, 1.0f,  0.0f, 0.0f,
        -2.0f, -2.0f, -2.0f, 1.0f,  0.0f, 0.0f,

        -2.0f, -2.0f,  2.0f, 0.0f,  1.0f, 0.0f,
         2.0f, -2.0f,  2.0f, 0.0f,  1.0f, 0.0f,
         2.0f,  2.0f,  2.0f, 0.0f,  1.0f, 0.0f,
         2.0f,  2.0f,  2.0f, 0.0f,  1.0f, 0.0f,
        -2.0f,  2.0f,  2.0f, 0.0f,  1.0f, 0.0f,
        -2.0f, -2.0f,  2.0f, 0.0f,  1.0f, 0.0f,

        -2.0f,  2.0f,  2.0f, 0.0f,  0.0f, 1.0f,
        -2.0f,  2.0f, -2.0f, 0.0f,  0.0f, 1.0f,
        -2.0f, -2.0f, -2.0f, 0.0f,  0.0f, 1.0f,
        -2.0f, -2.0f, -2.0f, 0.0f,  0.0f, 1.0f,
        -2.0f, -2.0f,  2.0f, 0.0f,  0.0f, 1.0f,
        -2.0f,  2.0f,  2.0f, 0.0f,  0.0f, 1.0f,

         2.0f,  2.0f,  2.0f, 1.0f,  1.0f, 0.0f,
         2.0f,  2.0f, -2.0f, 1.0f,  1.0f, 0.0f,
         2.0f, -2.0f, -2.0f, 1.0f,  1.0f, 0.0f,
         2.0f, -2.0f, -2.0f, 1.0f,  1.0f, 0.0f,
         2.0f, -2.0f,  2.0f, 1.0f,  1.0f, 0.0f,
         2.0f,  2.0f,  2.0f, 1.0f,  1.0f, 0.0f,

        -2.0f, -2.0f, -2.0f, 1.0f,  0.0f, 1.0f,
         2.0f, -2.0f, -2.0f, 1.0f,  0.0f, 1.0f,
         2.0f, -2.0f,  2.0f, 1.0f,  0.0f, 1.0f,
         2.0f, -2.0f,  2.0f, 1.0f,  0.0f, 1.0f,
        -2.0f, -2.0f,  2.0f, 1.0f,  0.0f, 1.0f,
        -2.0f, -2.0f, -2.0f, 1.0f,  0.0f, 1.0f,

        -2.0f,  2.0f, -2.0f, 0.0f,  1.0f, 1.0f,
         2.0f,  2.0f, -2.0f, 0.0f,  1.0f, 1.0f,
         2.0f,  2.0f,  2.0f, 0.0f,  1.0f, 1.0f,
         2.0f,  2.0f,  2.0f, 0.0f,  1.0f, 1.0f,
        -2.0f,  2.0f,  2.0f, 0.0f,  1.0f, 1.0f,
        -2.0f,  2.0f, -2.0f, 0.0f,  1.0f, 1.0f
    };
```

2. 将立方体移到中心点为（-1.5，0.5，-1.5）处：使用函数`translate`

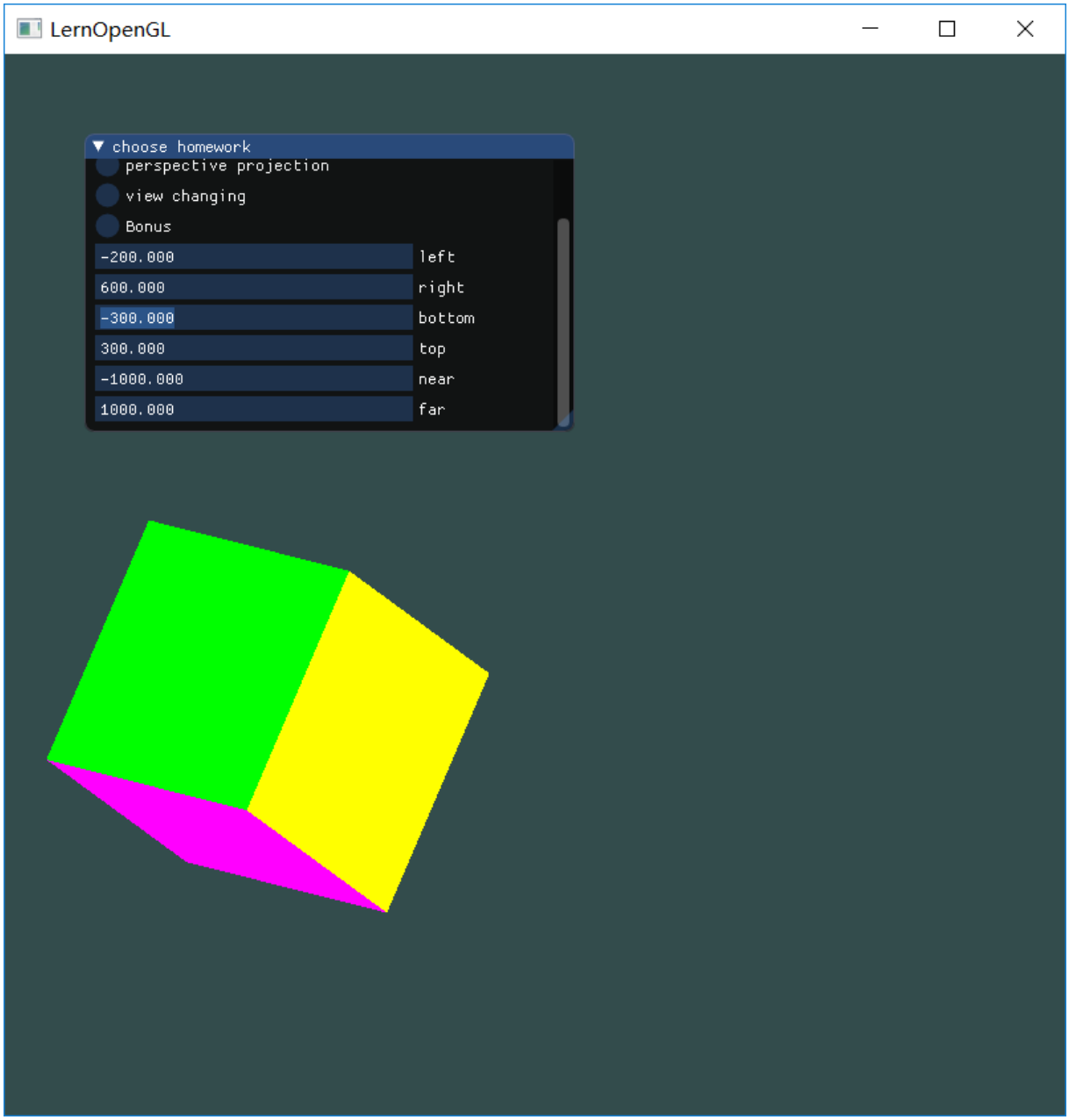
2. 正交投影

1. 要求：实现正交投影，使用多组(left, right, bottom, top, near, far)参数，比较结果差异
2. 实验结果：

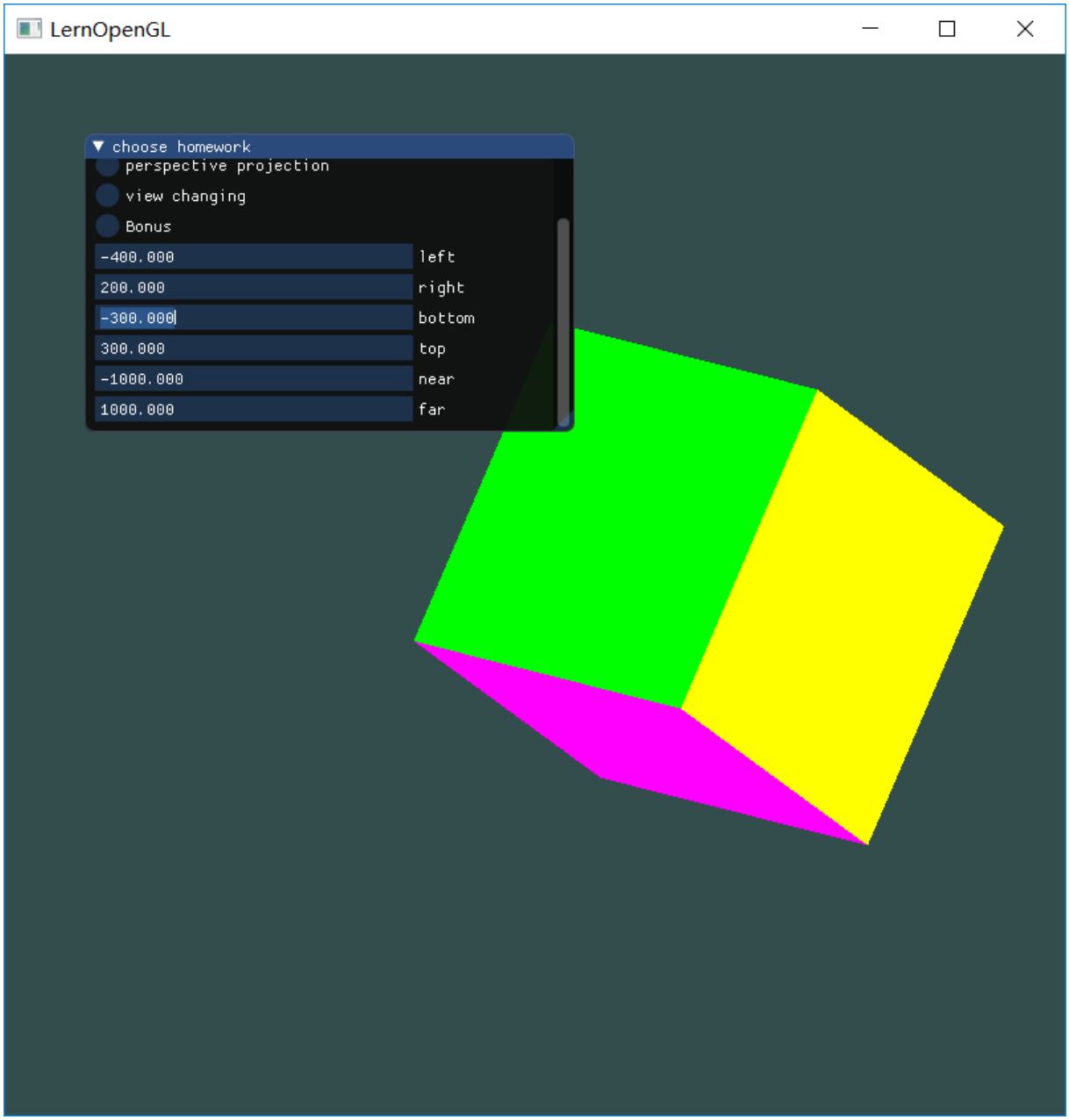
1. 第一组数据



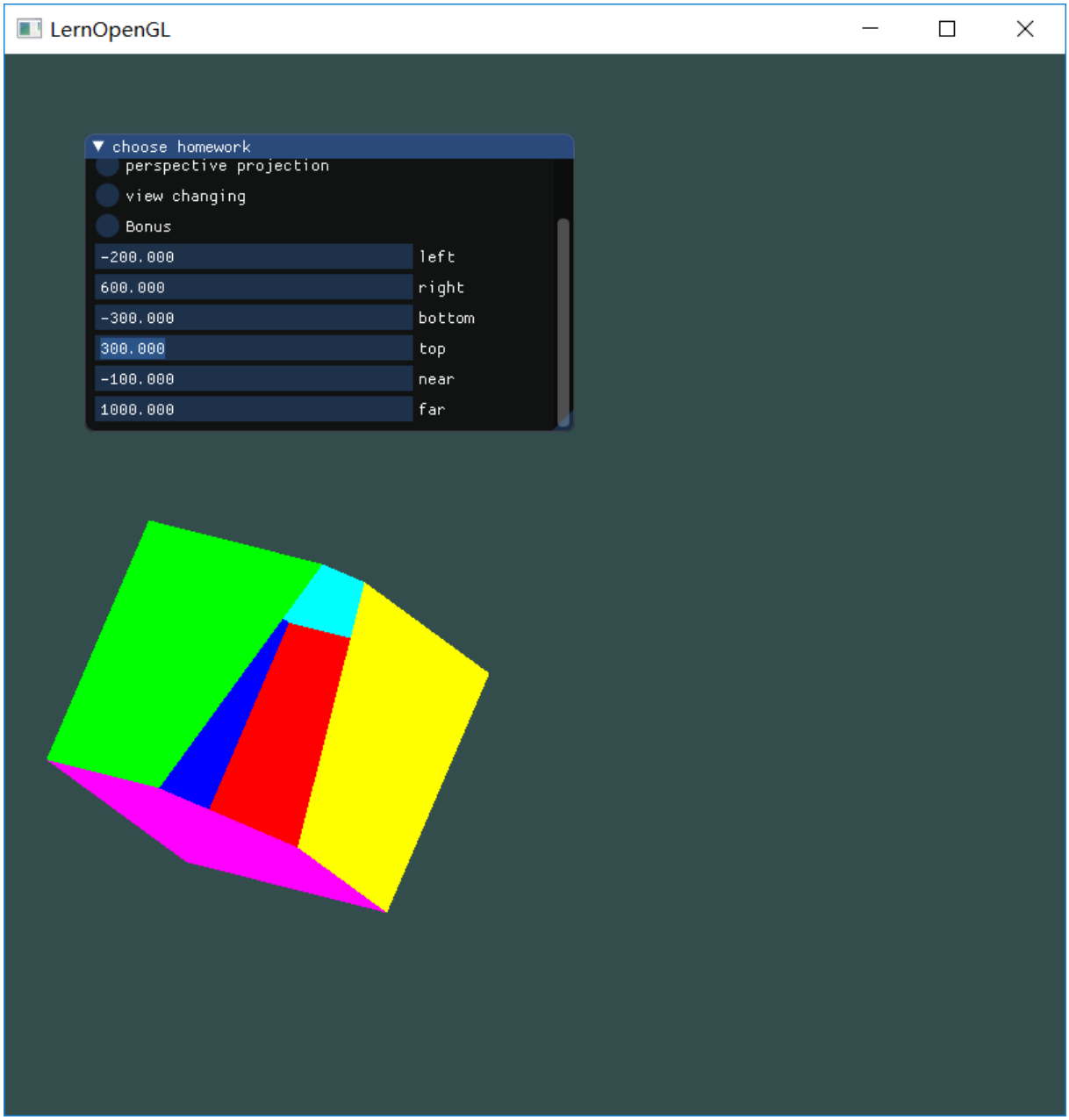
2. 第二组数据：修改了`left` `right`参数，保持 |left - right| = 800，立方体左移



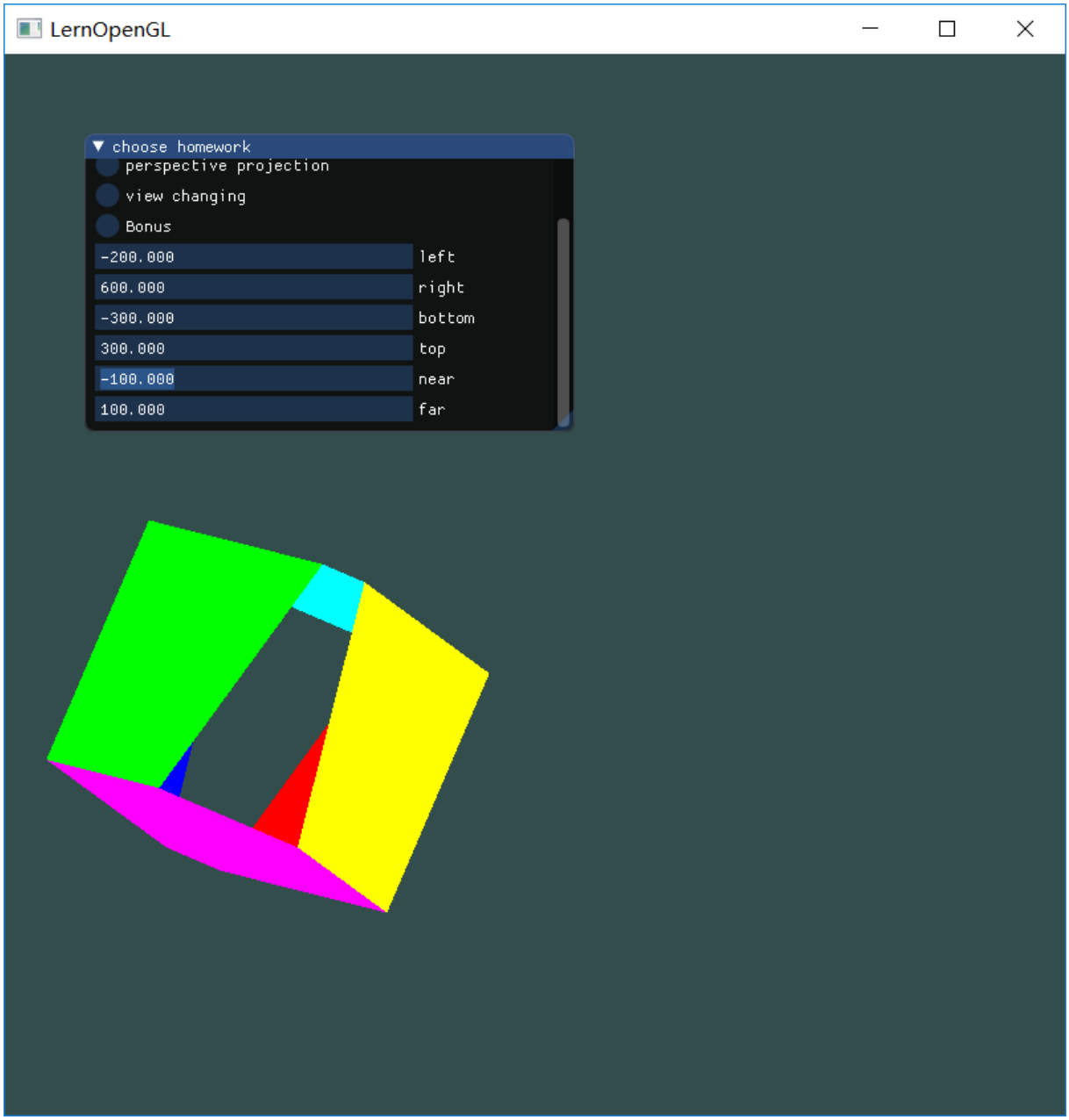
3. 第三组数据：修改了 left 参数，立方体拉伸



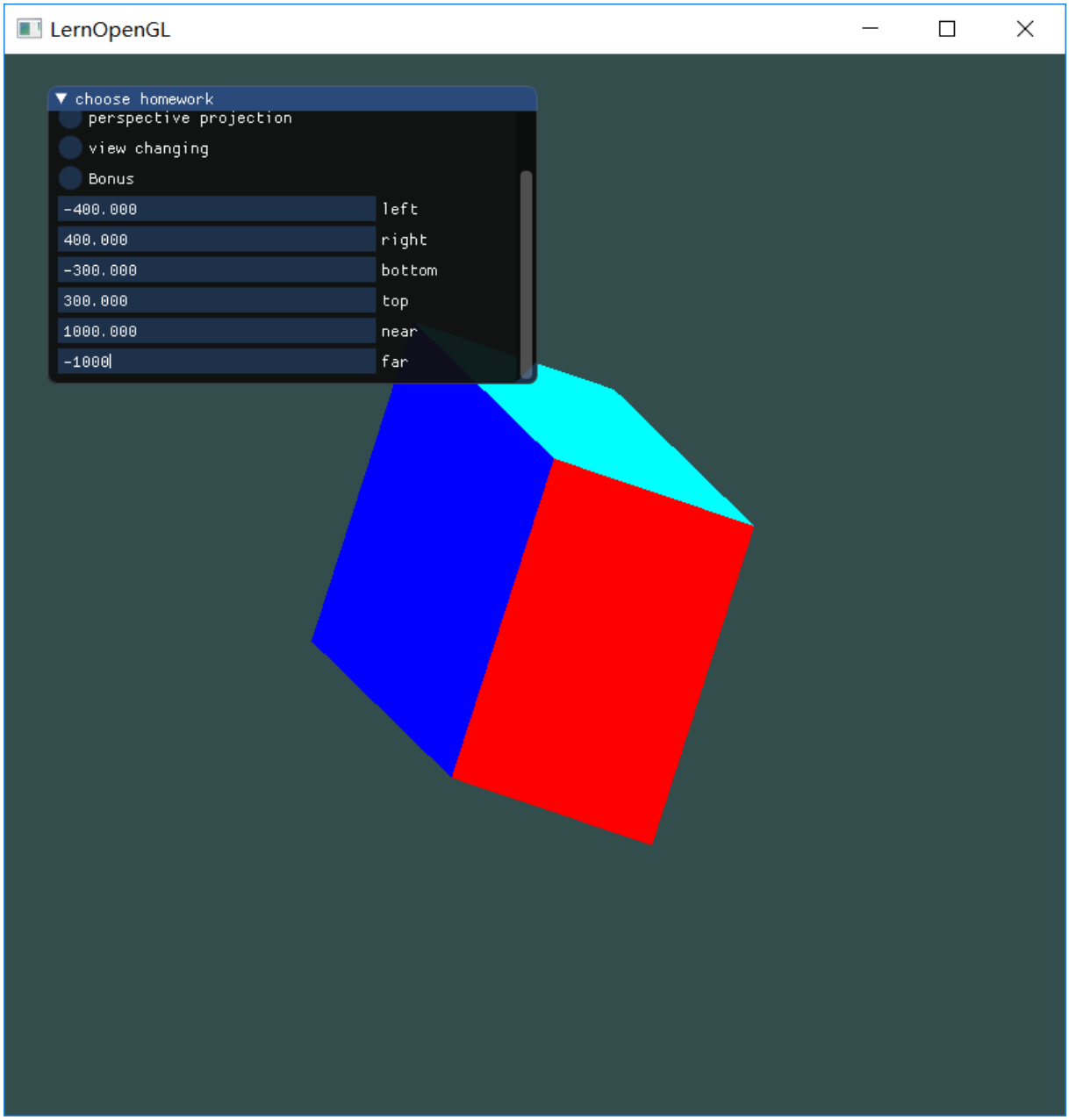
4. 第四组数据：修改near数据，保持near为负数



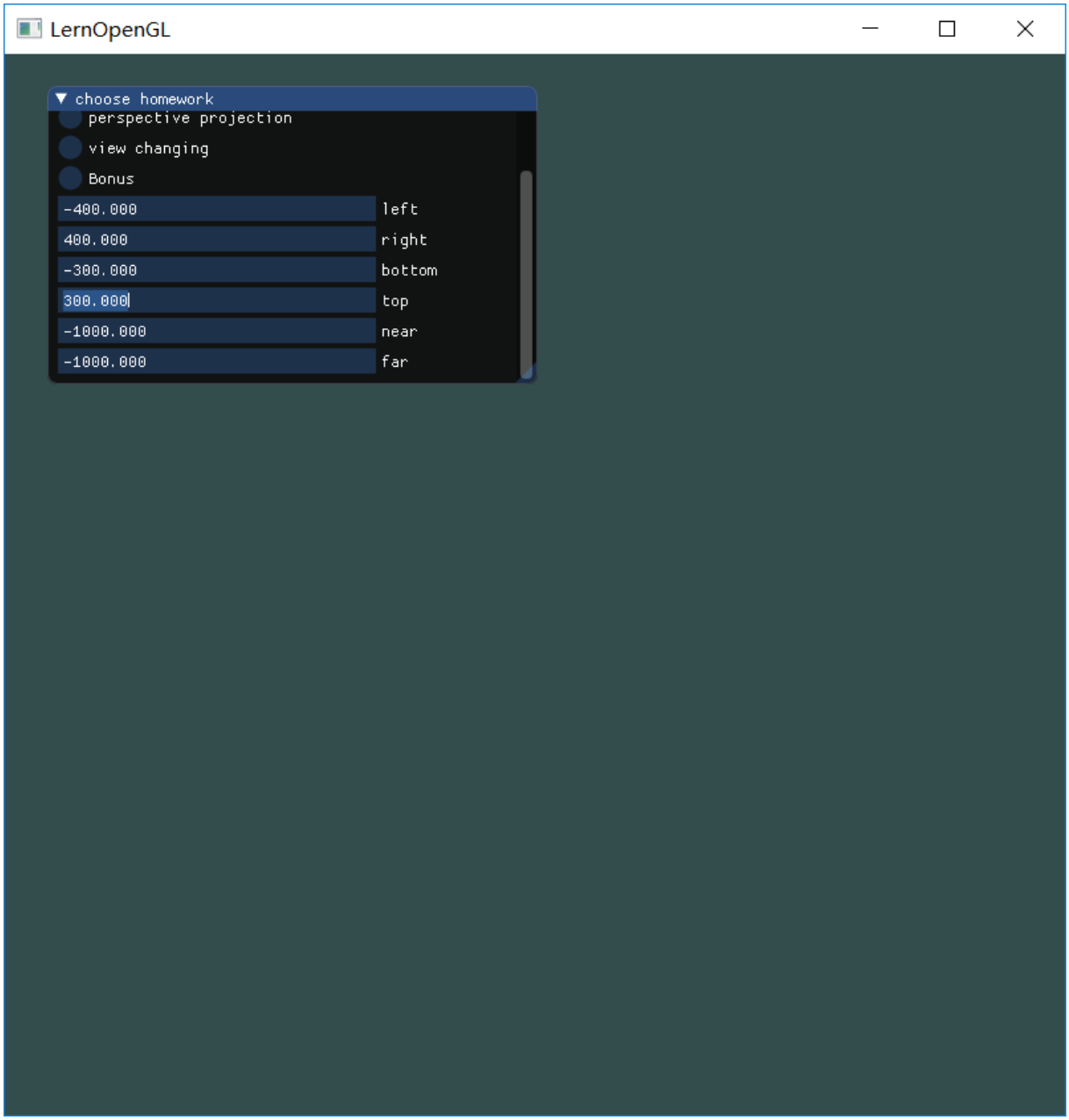
5. 第五组数据：修改far数据



6. 第六组数据：保持near和far数值不变，交换正负



7. 第七组数据：保持near和far数值不变，使得两个参数同正负



3. 结果差异

   left、right、bottom、top四个参数影响立方体出现在屏幕的什么地方，单位是像素

   near、far影响立方体的z轴的什么部分将展现在屏幕上，哪一部分属于前，哪一部分属于后，单位也是像素

   因此修改前四个参数，会出现平移、拉伸、裁剪等效果

   修改后两个参数会出现裁剪深度的效果

4. 实现思路

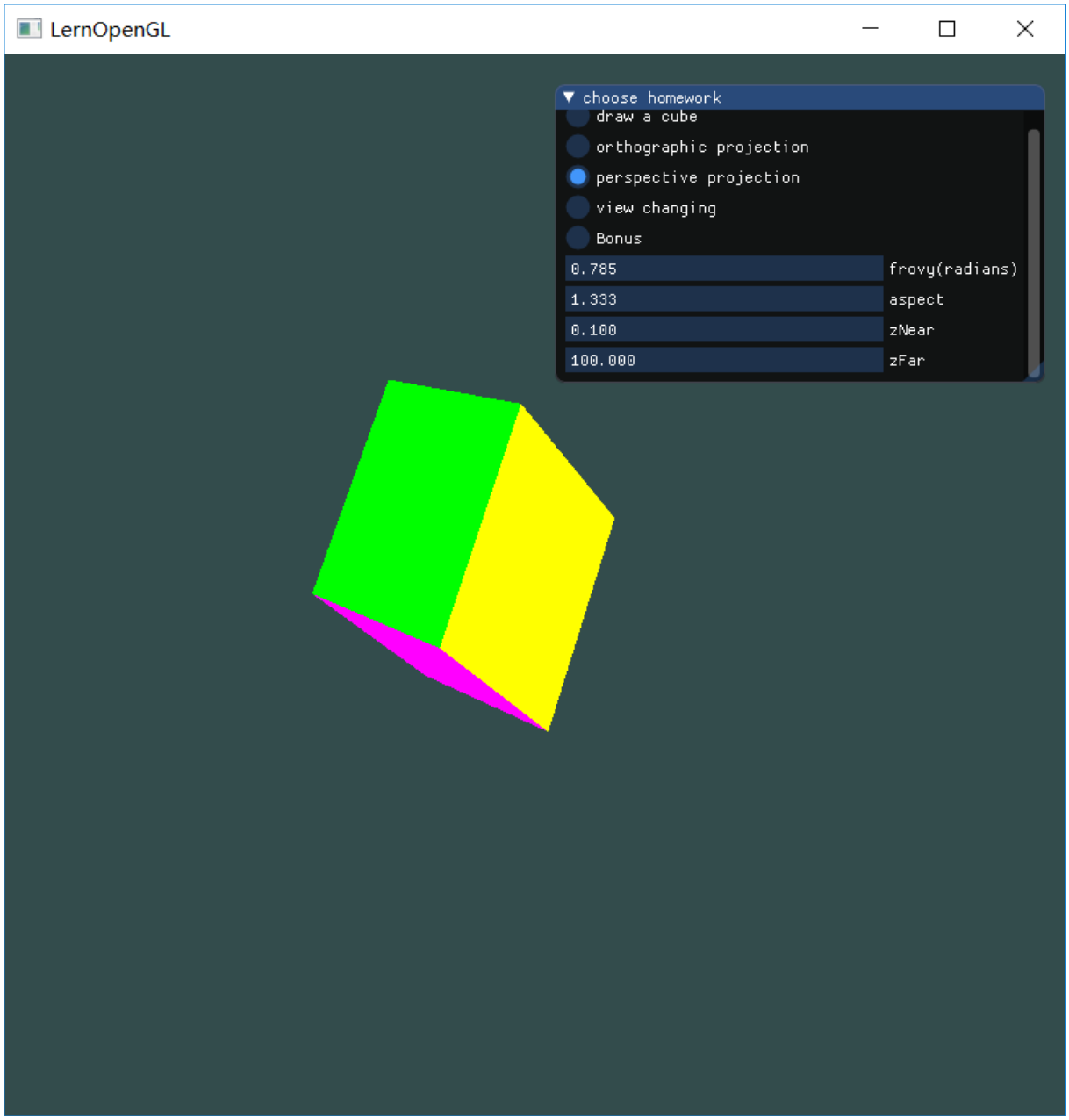
   1. 绘制立方体整体都相同
   2. 添加ImGui的输入框，输入影响正交投影的六个参数
   3. 由于单位是像素，而立方体输入的数值是相对坐标，因此需要先将立方体放大，使用函数`scale`将立方体放大
   4. 使用函数`ortho`实施正交投影，输入参数为前面输入的六个参数

3. 透视投影

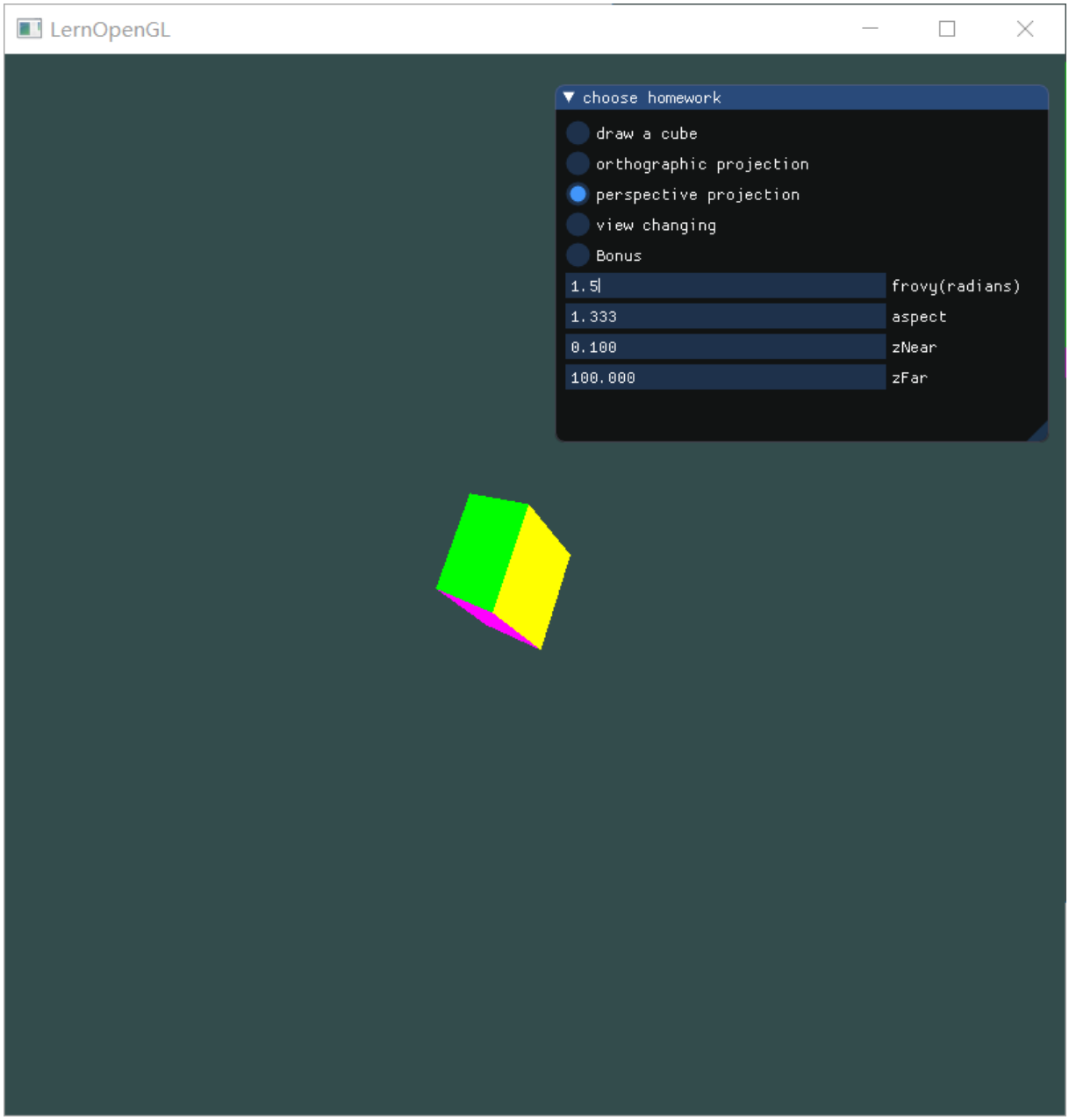
1. 要求：实现透视投影，使用多组参数，比较结果差异

2. 实验结果

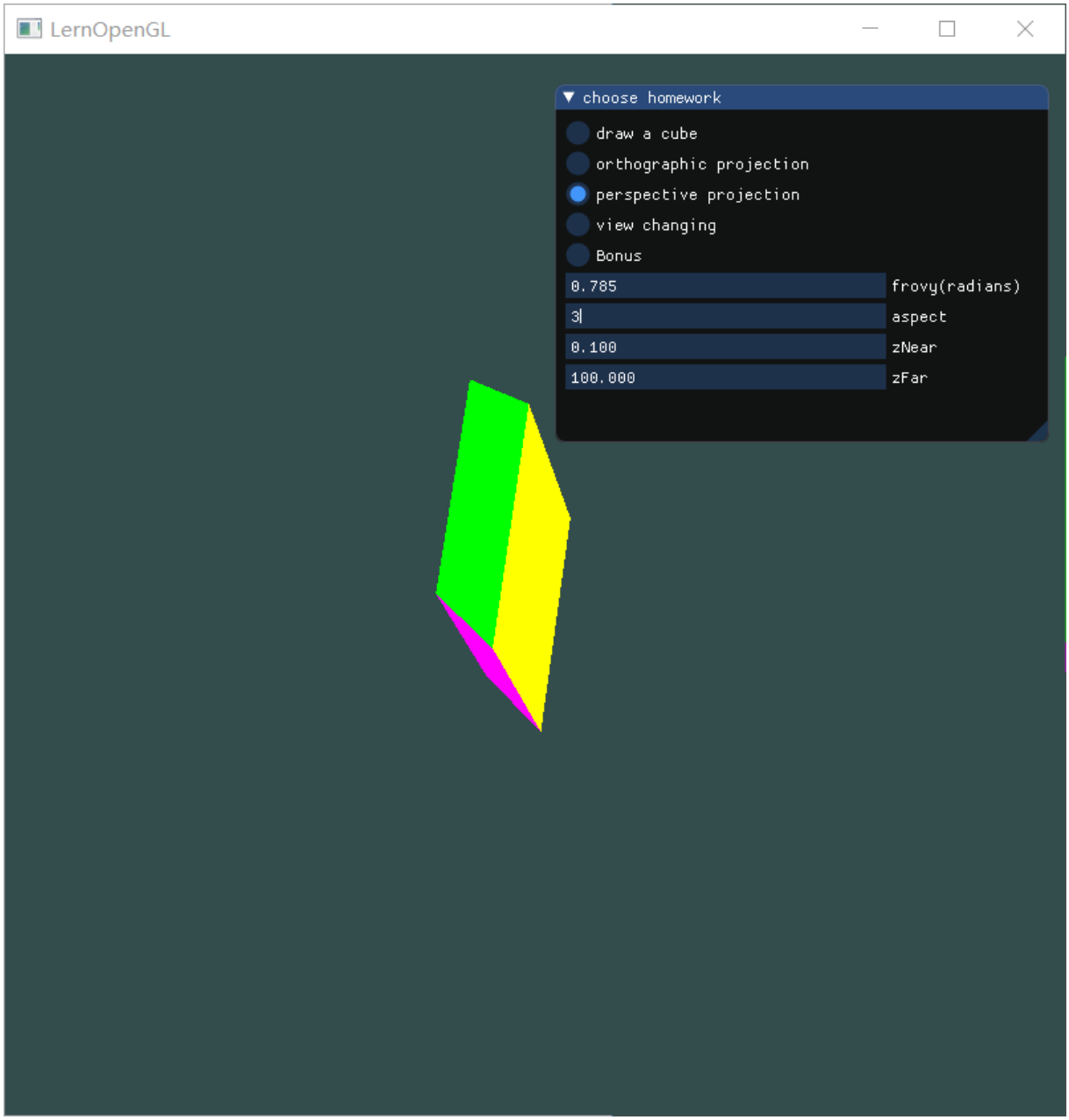
1. 第一组参数



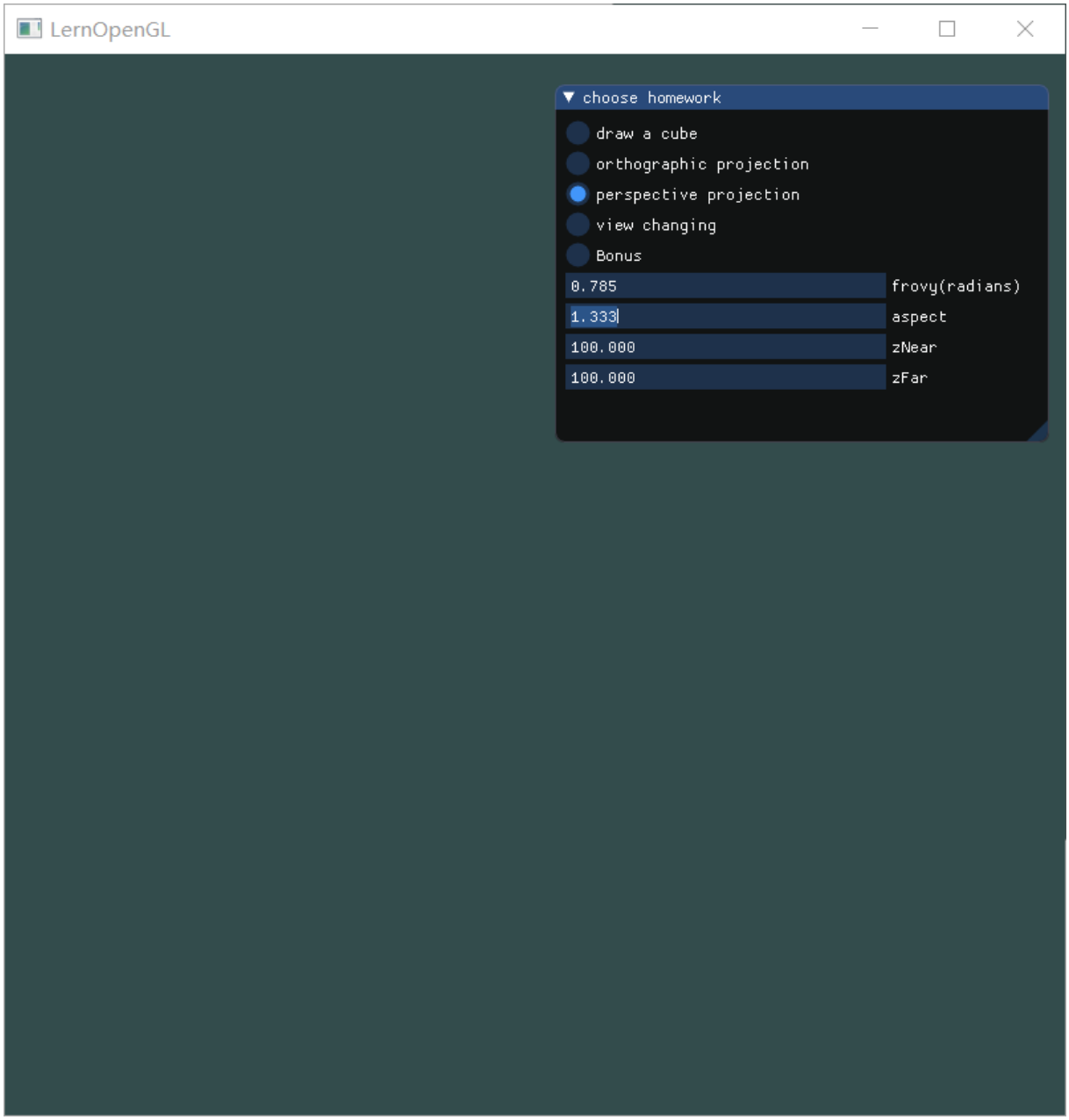
2. 第二组参数：修改了 frovy 参数

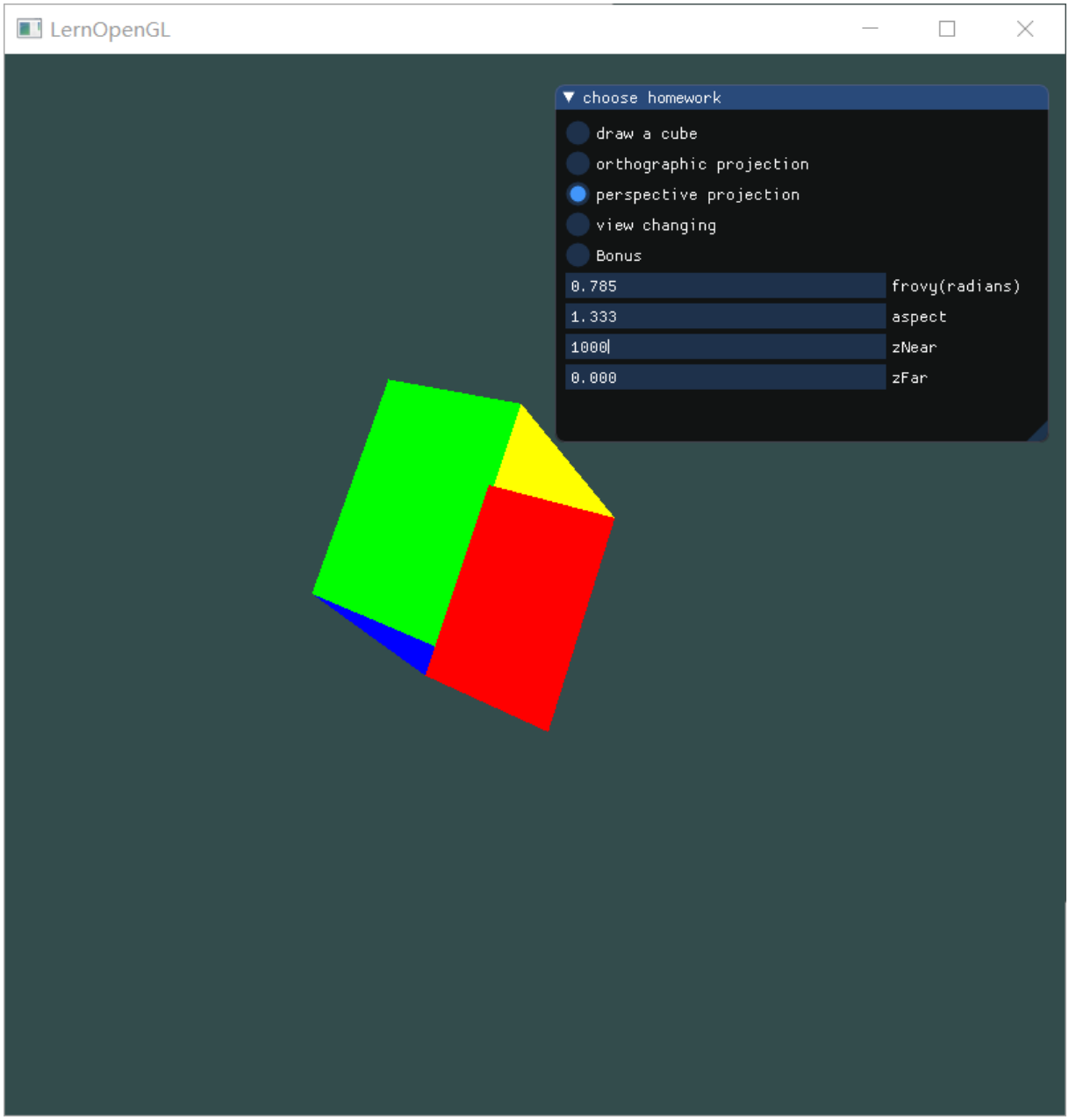


3. 第三组数据：修改了aspect数据

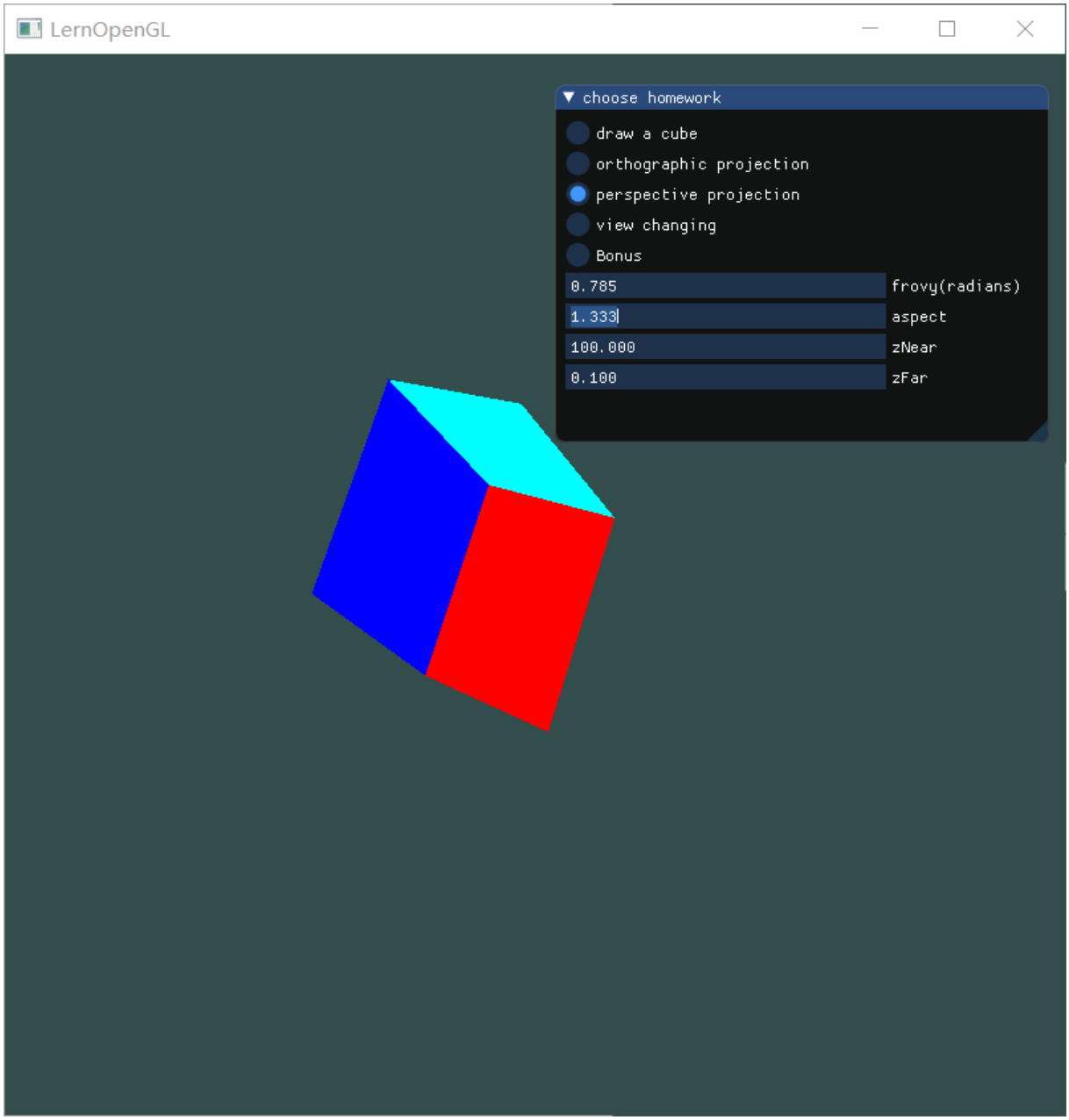


4. 第四组数据：修改zNear和zFar的比值


LernOpenGL
choose homework
draw a cube
orthographic projection
perspective projection
view changing
Bonus
0.785
frovy(radians)
1.333
aspect
100.000
zNear
100.000
zFar



5. 第五组数据：zNear和zFar互换数据



3. 比较差异

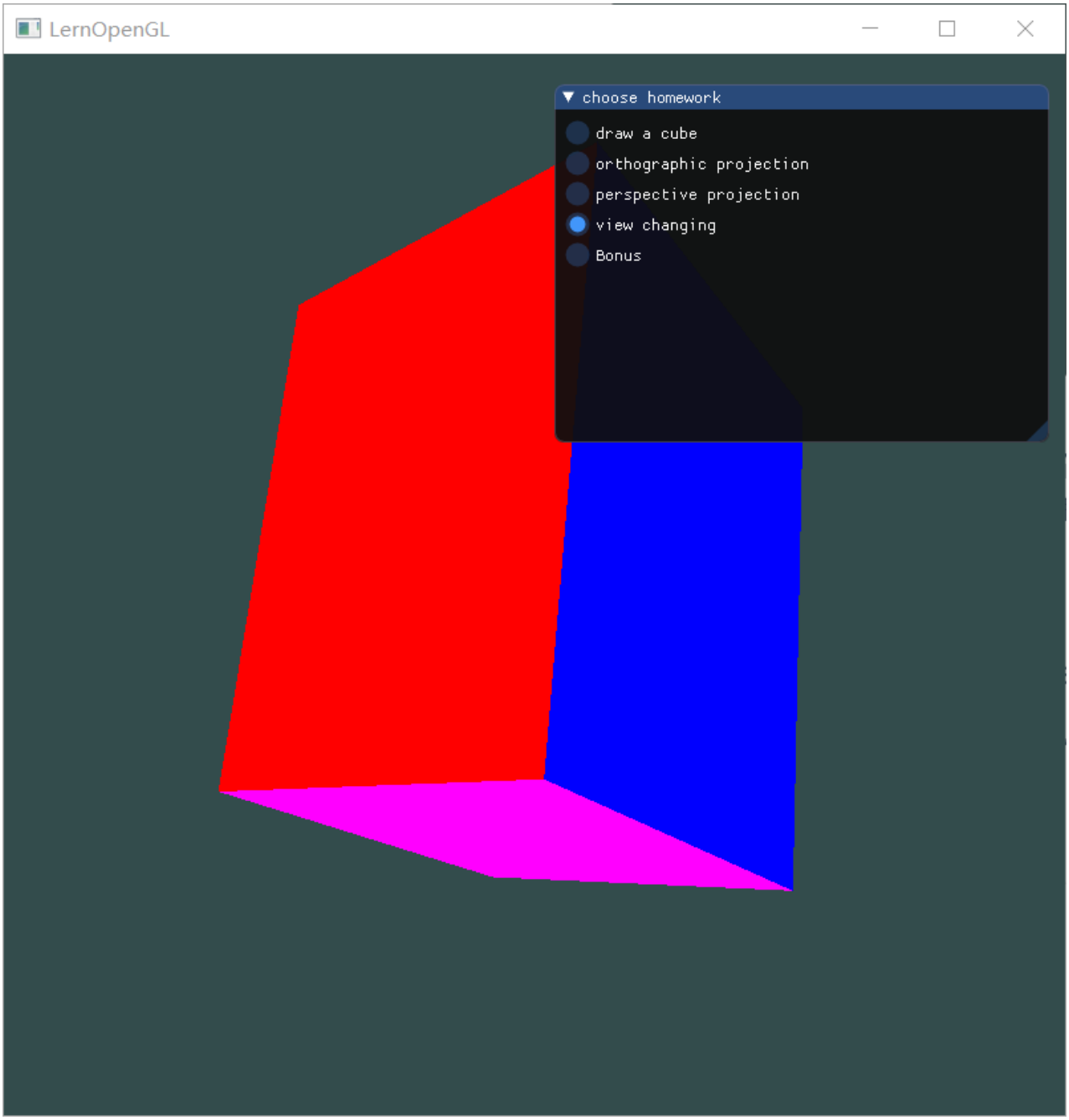
   变换 frovy 会变换视野的大小，因此变换这个会导致物体等比例放大或缩小；aspect设置宽高比，比例不对就会显示立方体的比例不是设置的比例，zNear和zFar设置的是近平面和远平面

4. 实现思路

   1. 立方体画法和前面的一样
   2. 添加ImGui的输入框，输入影响透视投影的四个参数
   3. 使用函数`perspective`设置透视投影，传入参数就用第二步输入的参数。

2. 视角变换

1. 要求：把cube放置在(0, 0, 0)处，做透视投影，使摄像机围绕cube旋转，并且时刻看着cube中心
2. 实验结果



3. 实现思路
    1. 立方体画法和前面一样
    2. 设置 cameraPos 为 $(sin(glfwGetTime()) * Radius, 0.0f, cos(glfwGetTime()) * Radius)$，cameraTarget 为 (0.0f, 0.0f, 0.0f)，设置 worldUp 为 (0.0f, 1.0f, 0.0f)
    3. cameraDirection 通过式子 $cameraPosition - cameraTarget$ 获得，再通过函数`normalize`标准化；cameraRight 通过式子 $up \times cameraDirection$ 获得，再进行标准化；cameraUp 通过式子 $cameraDirection \times cameraRight$ 获得，再标准化
    4. 使用函数`lookAt`获得 view 矩阵，其中传入参数分别是 cameraPos cameraTarget cameraUp

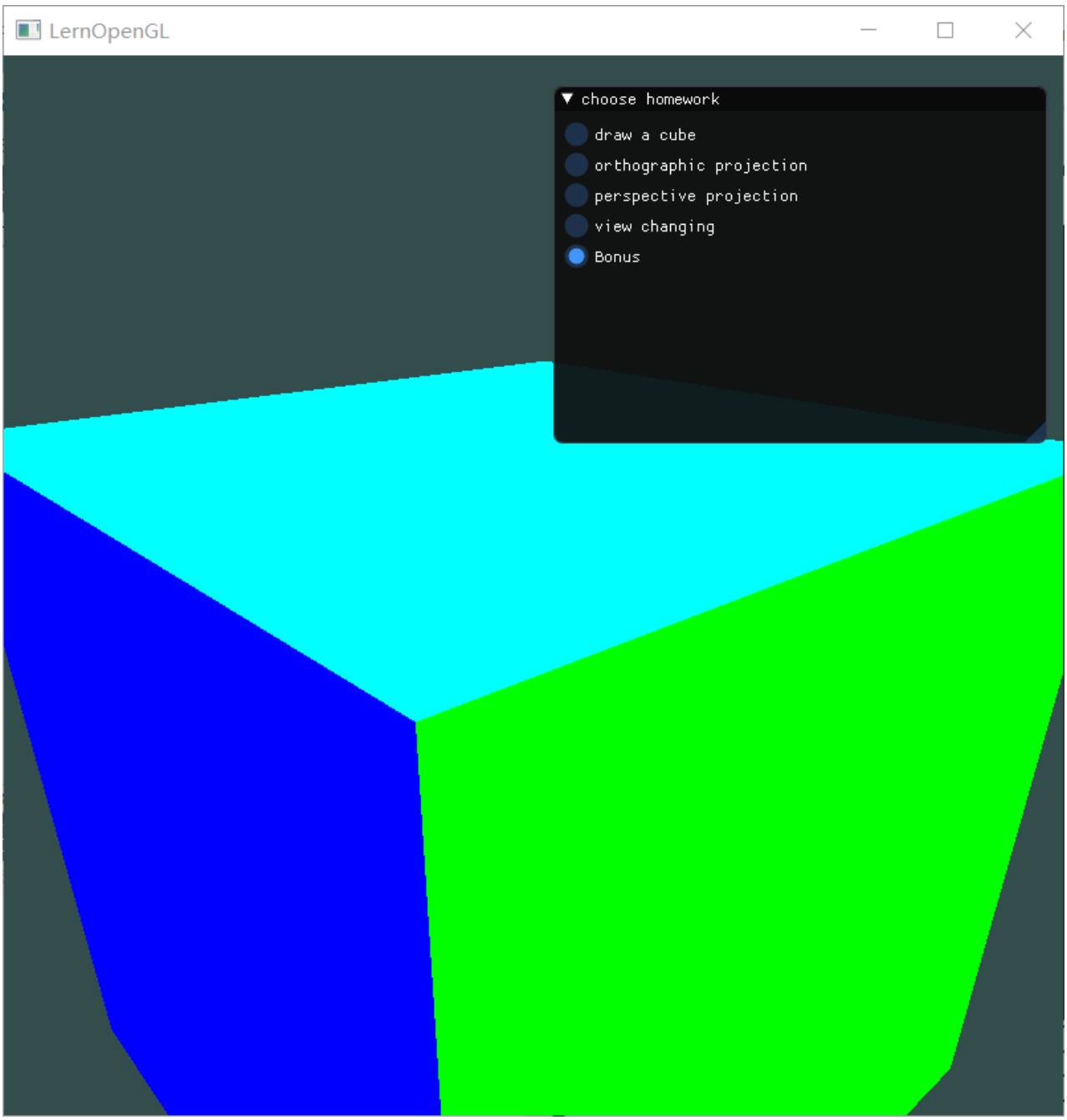
4. 问答题

题目：在现实生活中，我们一般将摄像机摆放的空间View matrix和被拍摄的物体摆设的空间Model matrix分开，但是在OpenGL中却将两个合二为一设为ModelView matrix，通过上面的作业启发，你认为是为什么呢？在报告中写入。（Hints：你可能有不止一个摄像机

回答：在目前的可编程管线中，是将 view 和 model 两个矩阵分开的，但是在以前的固定管线中是用合二为一的 ModelView matrix ，原因是通常 view 矩阵都非常固定，除非有用到摄像机变换等情况，否则基本都是一样的，因此为了编程的方便，会将两个矩阵合并成一个使用。

2. Bonus

1. 要求：实现一个camera类，当键盘输入w,a,s,d ，能够前后左右移动；当移动鼠标，能够视角移动("look around")，即类似FPS(First Person Shooting)的游戏场景
2. 实验结果：



3. 实现过程
    1. 和前面的方法一样画出立方体

2. 新建 camera 类，里面包含私有变量：cameraPosition、cameraFront、cameraRight、cameraUp、worldUp、pitch、yaw，和函数 Camera、getViewMatrix、moveForward、moveBackward、moveLeft、moveRight、rotate
    1. 构造函数里面，cameraPosition 初始为 (0.0f, 0.0f, 0.0f)，cameraFront 初始为 (0.0f, 0.0f, -1.0f)，worldUp 初始为 (0.0f, 1.0f, 0.0f)。cameraRight 通过式子 $cameraFront \times worldUp$ 获得，再进行标准化；cameraUp 通过式子 $cameraRight \times cameraFront$ 获得
    2. 向前后左右走函数通过变换 cameraPosition 来实现
    3. 旋转：传入x变化量和y变化量，分别加到 yaw 和 pitch 中，再将 yaw 和 pitch 转换成 cameraFront，再获得 cameraRight 和 cameraUp

    ```
    //转换成cameraFront的方法
    cameraFront =
    glm::normalize(glm::vec3(cos(glm::radians(yaw))*cos(glm::radians(pitch)),
    sin(glm::radians(pitch)), sin(glm::radians(yaw))*cos(glm::radians(pitch))));
    ```

3. 在原来就有的 `processInput` 函数中，添加按下 wsad 的处理
4. 添加鼠标移动回调函数，通过获得此时的鼠标位置，记录上次的位置，获得x和y的偏移量。再通过函数 `glfwSetCursorPosCallback` 添加
5. 获得的 view 矩阵通过 `camera.getViewMatrix` 获得