

实验2、OpenGL绘制基础

姓 名： 雷昱

学 号： 22920202204666

学 院： 信息学院

专 业： 软件工程

年 级： 2020级

二〇二二年 **3**月 **23** 日

1. 实现阅读材料（1）中的Sierpinski镂垫程序，并完成如下具体功能（下面的功能要求逐条递增）
   1. 理解并实现课本程序

答：显示函数中定义了三个任意定点，并且在这个三个任意顶点形成的三角形内定义了一个任意初始点，多次循环，形成镂垫显示器上的窗口和裁剪窗口两者大小没有联系。

* 1. 尝试为不同三角形设置不同的颜色，使得看起来颜色更加美观。运行情况如图表1。
  2. 为这个镂垫生成动画，①镂垫的颜色随时间不断变化。②在①的基础上增加旋转效果。③在②的基础上增加缩放效果。

## 方法1.相对于摄像机调整被观察对象

操作流程： ①在idlefunc()函数中调用与模型变化相关的API

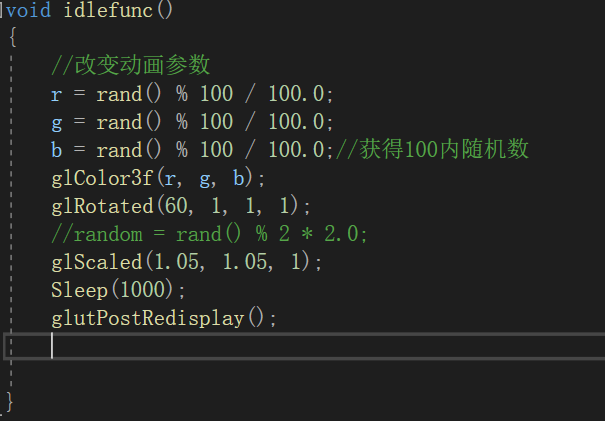
②使用void glRotate<fd> (type angle, type dx, type dy, type dz);函数调整镂垫的旋转，说明：本函数用来计算旋转矩阵，该矩阵围绕原点到点（x，y，z）的向量逆时针旋转angle角，然后用当前矩阵乘以旋转矩阵，并用结果矩阵替代当前矩阵。如果矩阵模式为GL\_MODELVIEW或GL\_PROJECTION，则在调用本函数之后绘制的所有物体均被旋转。

③使用 void glScale<fd> (type sx, type sy, type sz);函数调整镂垫的缩放，说明：本函数用当前矩阵乘以由三个轴向缩放因子指定的缩放矩阵，并用结果矩阵替代当前矩阵。如果矩阵模式为GL\_MODELVIEW或GL\_PROJECTION，则在调用本函数之后绘制的所有物体均被缩放。

④由于旋转会使部分镂垫被裁剪，因此调整视图矩阵的大小，以保证在旋转过程中镂垫会全部显示。



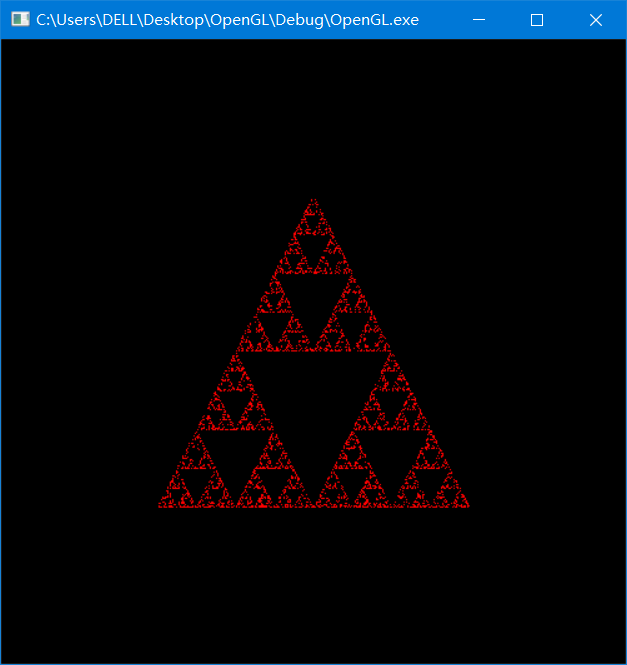
关键代码截图：

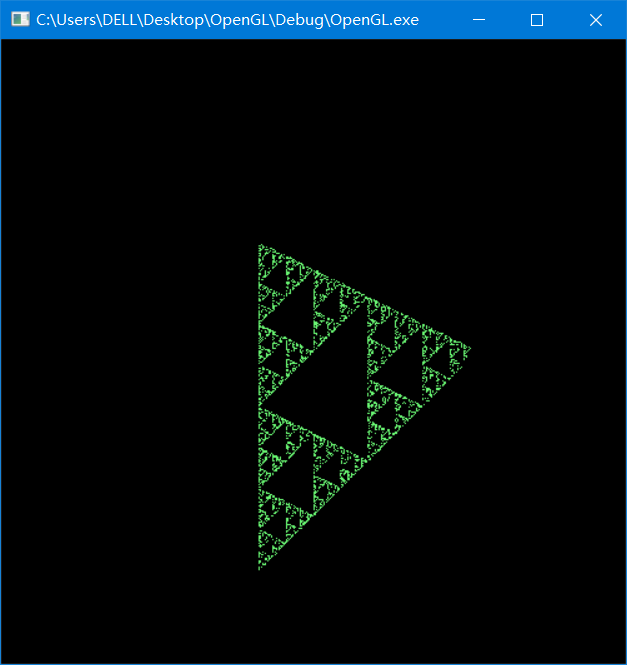


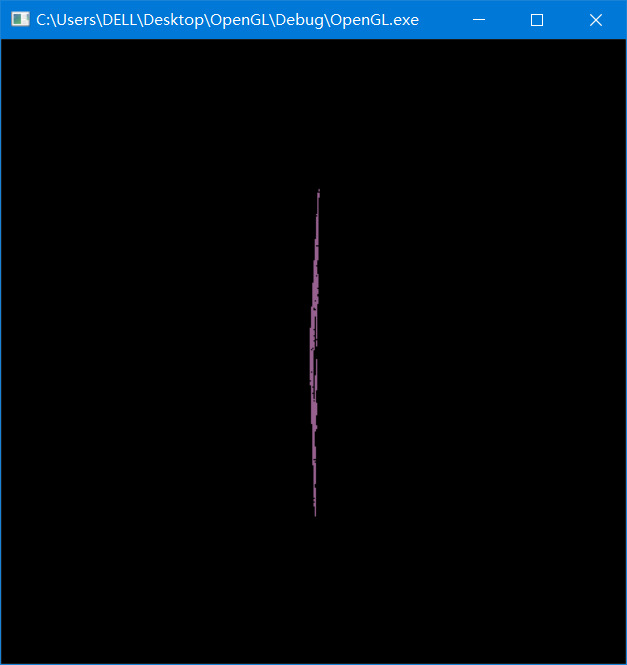
或者进行随机数缩放

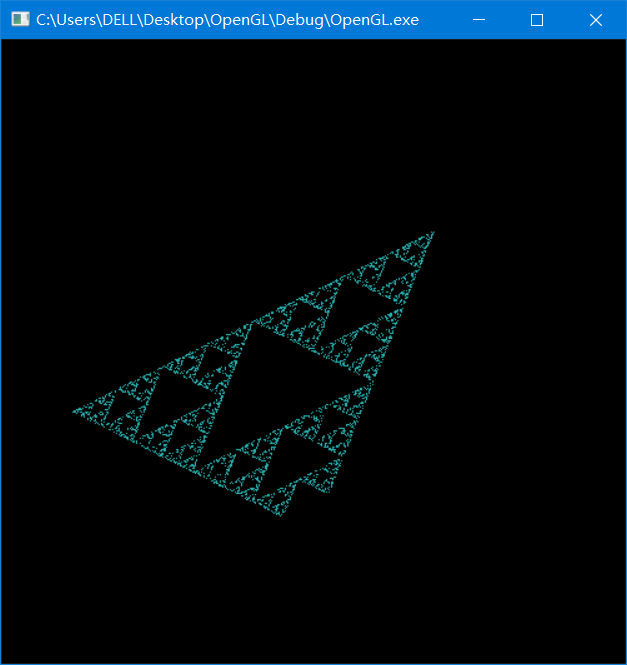


运行结果截图：

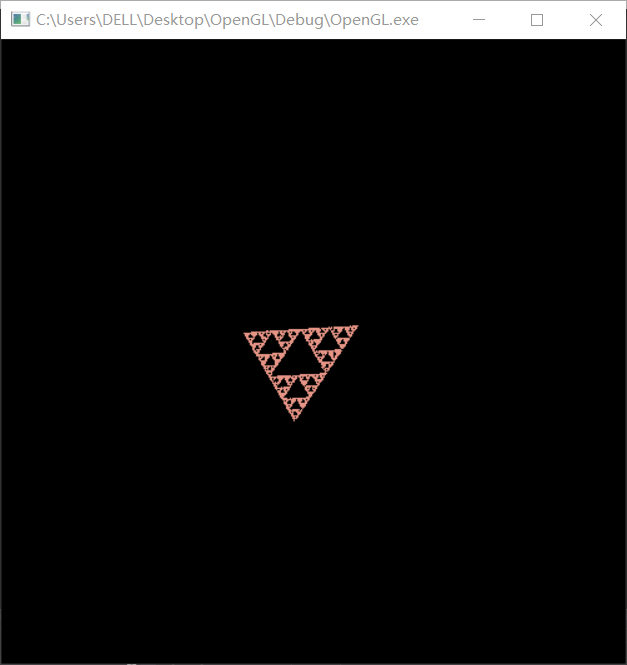


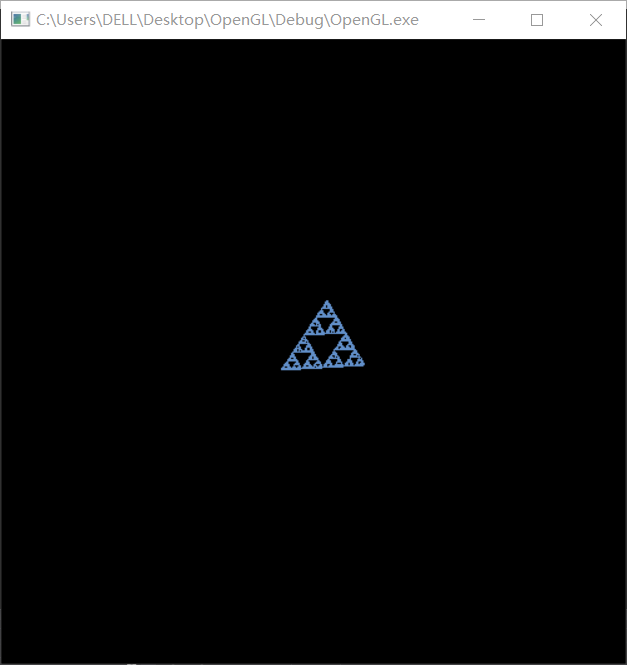


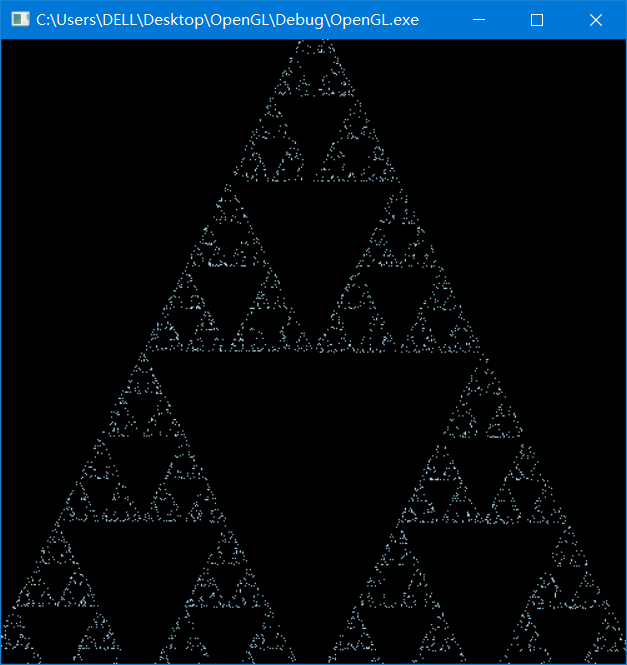




随机数缩放截图：







## 方法2.相对于被观察对象调整摄像机

操作流程： ①使用gluLookAt()函数来对定位和定向，可以用来调整旋转

②gluPerspective()定义相机的内在镜头参数，可以用来调整缩放

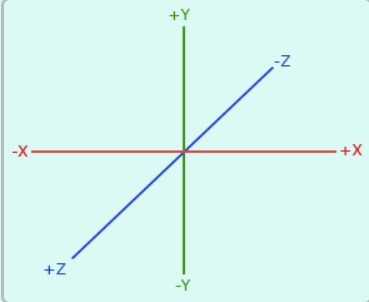
1. 完成一个三维的程序：
   1. 设置合适的相机位置和相机投影矩阵，使得OpenGL相机能够“看到”需要绘制的物体。

说明：运行提供给大家的程序 projection/projection.exe，运行情况如图1，调整glOrtho、 gluLookAt等函数的参数，了解各参数意义。

思考题：

* + 1. OpenGL中，三维空间的坐标系是怎么样的？

答：右手坐标系，坐标系画起来如图，z轴向外，x轴向右，y轴向上



* + 1. OpenGL中，相机的方位是怎么样的？如何调整相机朝向呢？

答：相机初始方位位于世界坐标系原点，指向z轴负方向，朝上向量为（0，1，0），摄像机位置就是世界空间中代表摄像机位置的向量。相机朝向由gluLookAt函数调整，六个参数，分别调整视点、被观察点和向上向量，对摄像机进行对位和定向。

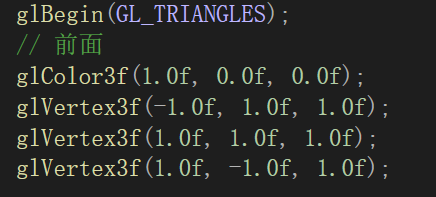
* + 1. OpenGL中，相机可见范围是如何设置的？

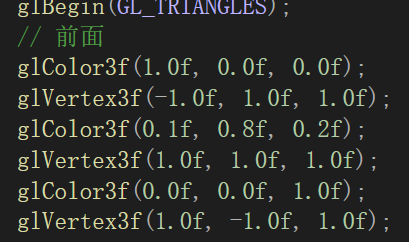
答：设置视域体的大小，即相机的可见范围。OpenGL提供两种投影取景方式——正交投影和透视投影， 正交投影glOrtho六个参数left、right、bottom、top、near、far形成长方体共同确定相机可见范围；透视投影glFrustum六个参数left、right、bottom、top、near、far形成四棱台共同确定相机可见范围；透视投影gluPersprctive更特殊，使用四个参数fovy、aspect、zNear、zFar形成四棱台共同确定相机可见范围，其中flvy,定义可视角的大小；aspect，定义物体显示在画板上的x和y方向上的比例，aspect小于1，则物体显示出来比实际更高。

* 1. 简单地绘制一个正方体，要求:
     1. 总共6个面，每个面用2个三角形来表示
     2. 为每个顶点指定颜色
     3. 渲染模式分别采用smooth模式和flat模式 （请查glShadeMode函数）
     4. 理解深度缓冲区的作用、用法，比较开启/不开启深度缓冲区（ glEnable(GL\_DEPTHTEST) ）的效果

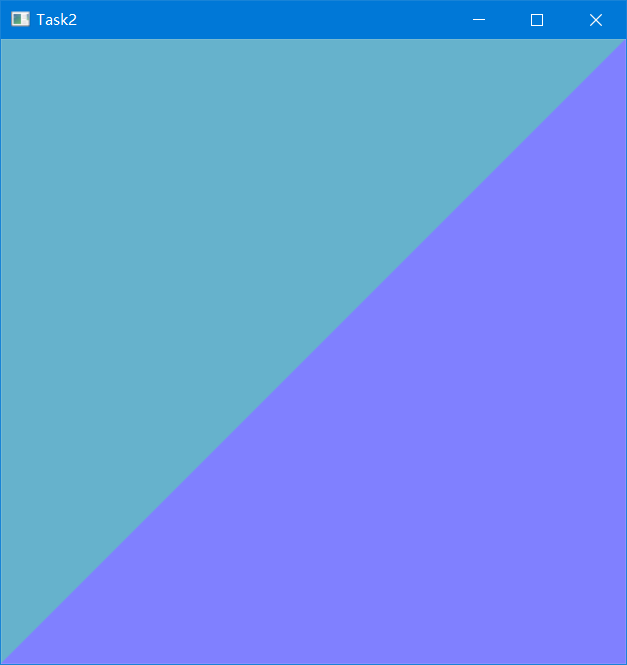
操作流程：

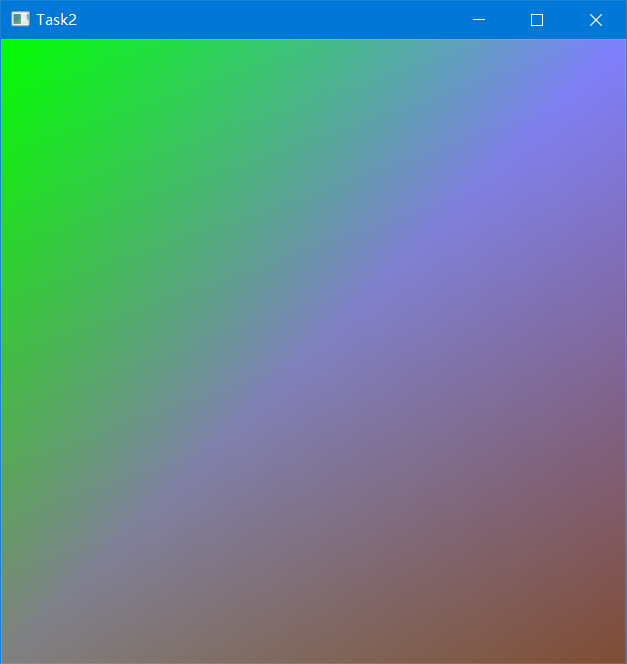
①画出依次正方体六个面，每个面使用两个三角形

（FLAT模式下 未为每一个点指定颜色）

（SMOOTH模式下 为每一个点指定颜色）

②未调整摄像机视角之前效果图

（使用FLAT模式渲染）

（使用SMOOTH模式渲染）

③调整摄像机的视角



④启用OpenGL深度缓存功能

三步： 在main()申请深度缓存

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB | GLUT\_DEPTH);

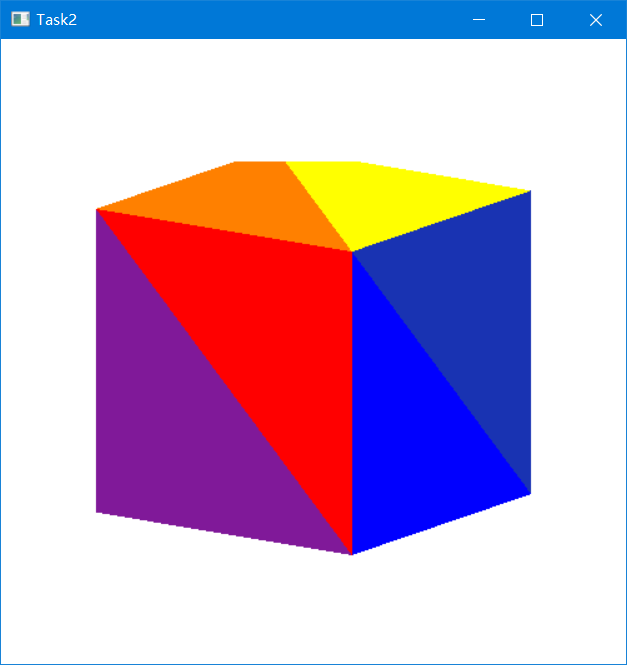
在init()启用深度缓存功能

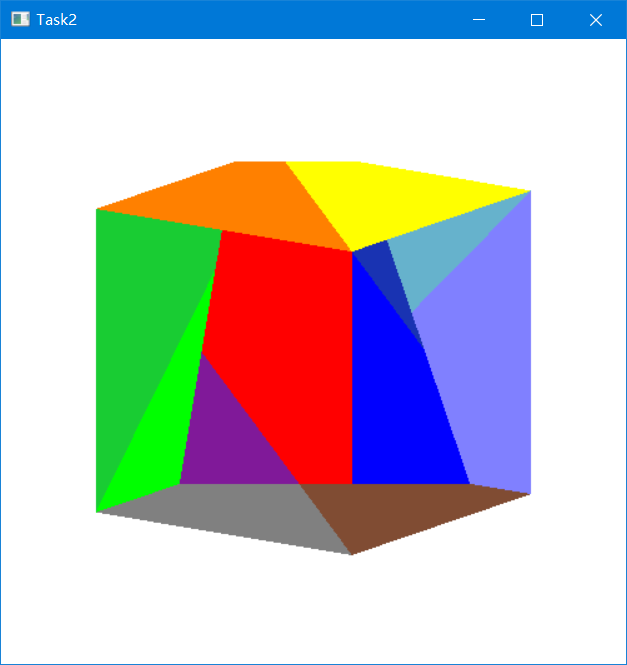
glEnable(GL\_DEPTH\_TEST);

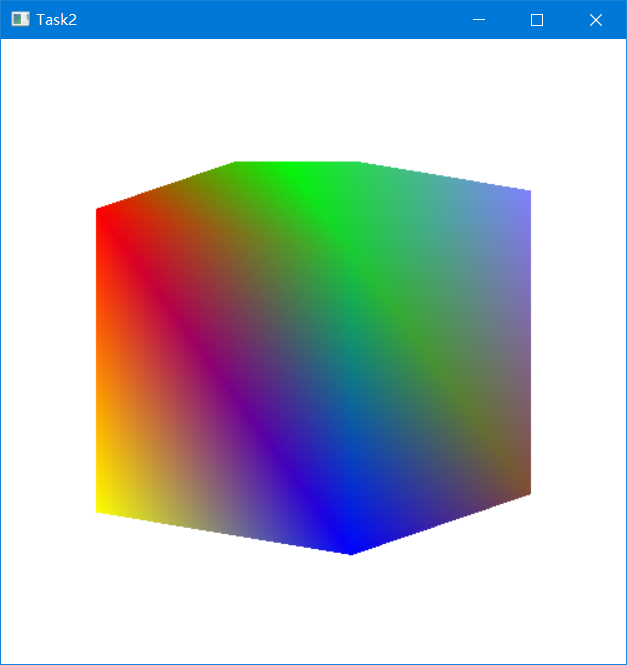
在display()绘制前，清空颜色缓存，并清空深度缓存

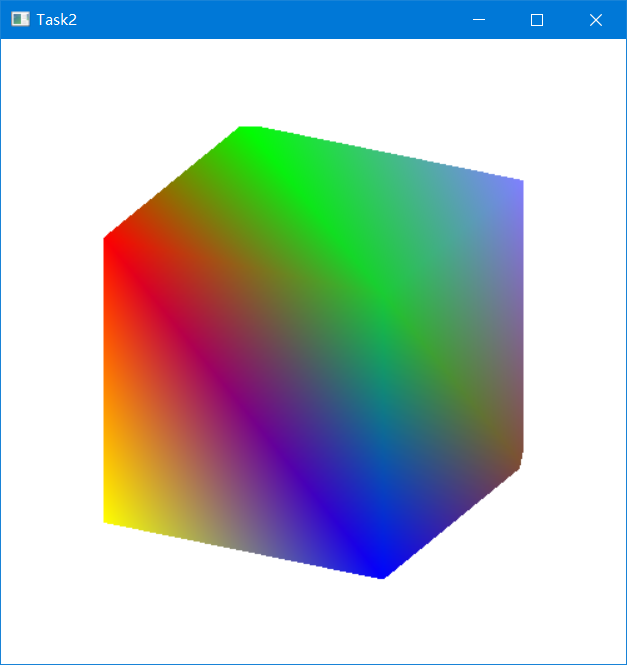
glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

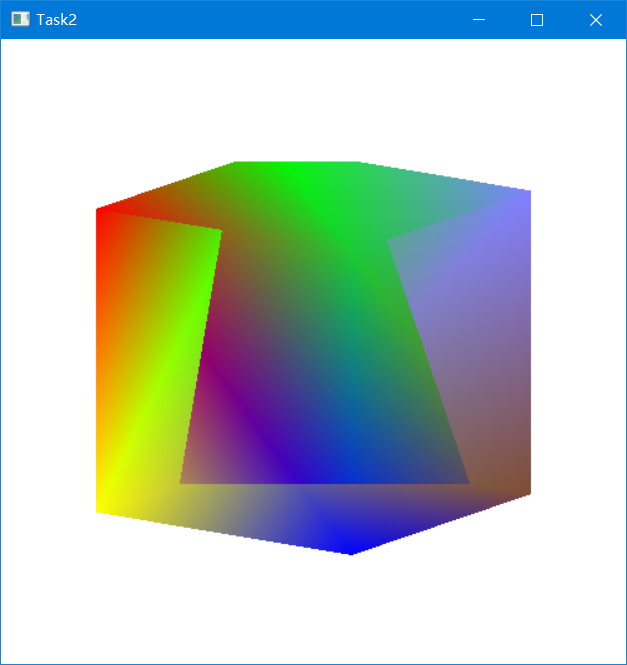
⑤最终效果展示

（开启深度缓存，FLAT）

（不开启深度缓存,FLAT）



（开启深度缓存，SMOOTH）

（未开启深度缓存，SMOOTH）

思考题：

* + 1. 如何实现前后面的遮挡？（你需要先自行了解一下深度缓冲区和深度测试的作用。搜索：GL\_DEPTH\_TEST关键字。）

答：深度缓存算法：①增加额外的深度缓存空间来保存在绘制过程中绘制器已经绘制图元的深度信息；②依据深度值比较结果，判断当前图元的颜色信息是否更新颜色缓存中对应像素的颜色值，若图元离投影机更近则更新颜色缓存，否则图元颜色被丢弃。

glEnable(GL\_DEPTH\_TEST);

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB | GLUT\_DEPTH);