

计算机图形学实验

实验2、OpenGL绘制基础

姓 名：\_\_\_\_潘宇科\_\_\_\_\_ \_\_\_\_\_\_

学 号：\_\_\_ 37320222204170\_\_ \_

学 院：\_\_\_\_信息学院\_\_\_ \_\_\_\_\_\_

专 业：\_\_\_ 软件工程\_\_\_\_ \_\_\_\_\_

年 级：\_\_\_ \_2022\_ \_\_ \_\_\_\_\_\_

2024年 4月 25日

目录

[Task1：Sierpinski 3](#_Toc97636928)

[1. 为不同三角形设置不同的颜色 3](#_Toc97636929)

[2. 为这个镂垫生成动画 3](#_Toc97636930)

[Task2: 三维程序 3](#_Toc97636931)

[1. 设置*合适的相机位置和相机投影矩阵* 3](#_Toc97636932)

[2. 运行提供的示例程序(exp2-2-1.cpp)，对程序进行改写 3](#_Toc97636933)

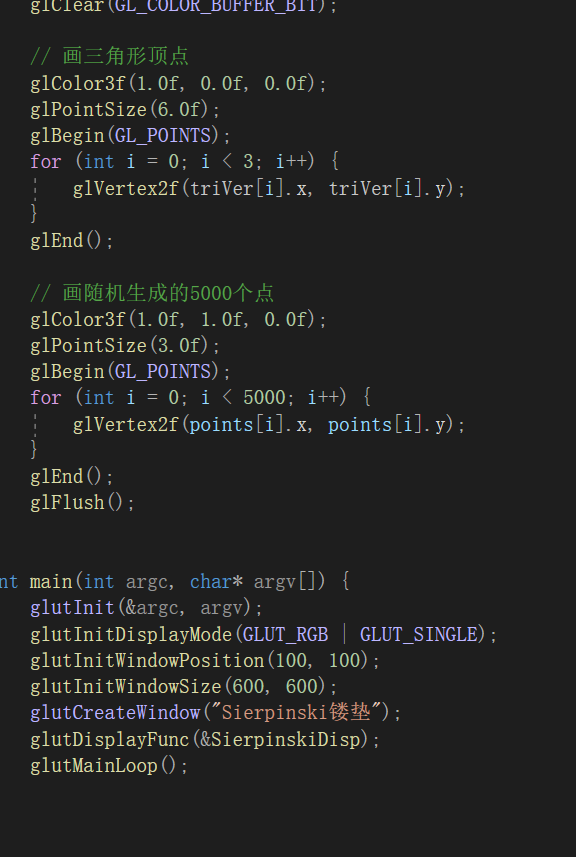
[Task3: 线框球体 3](#_Toc97636931)

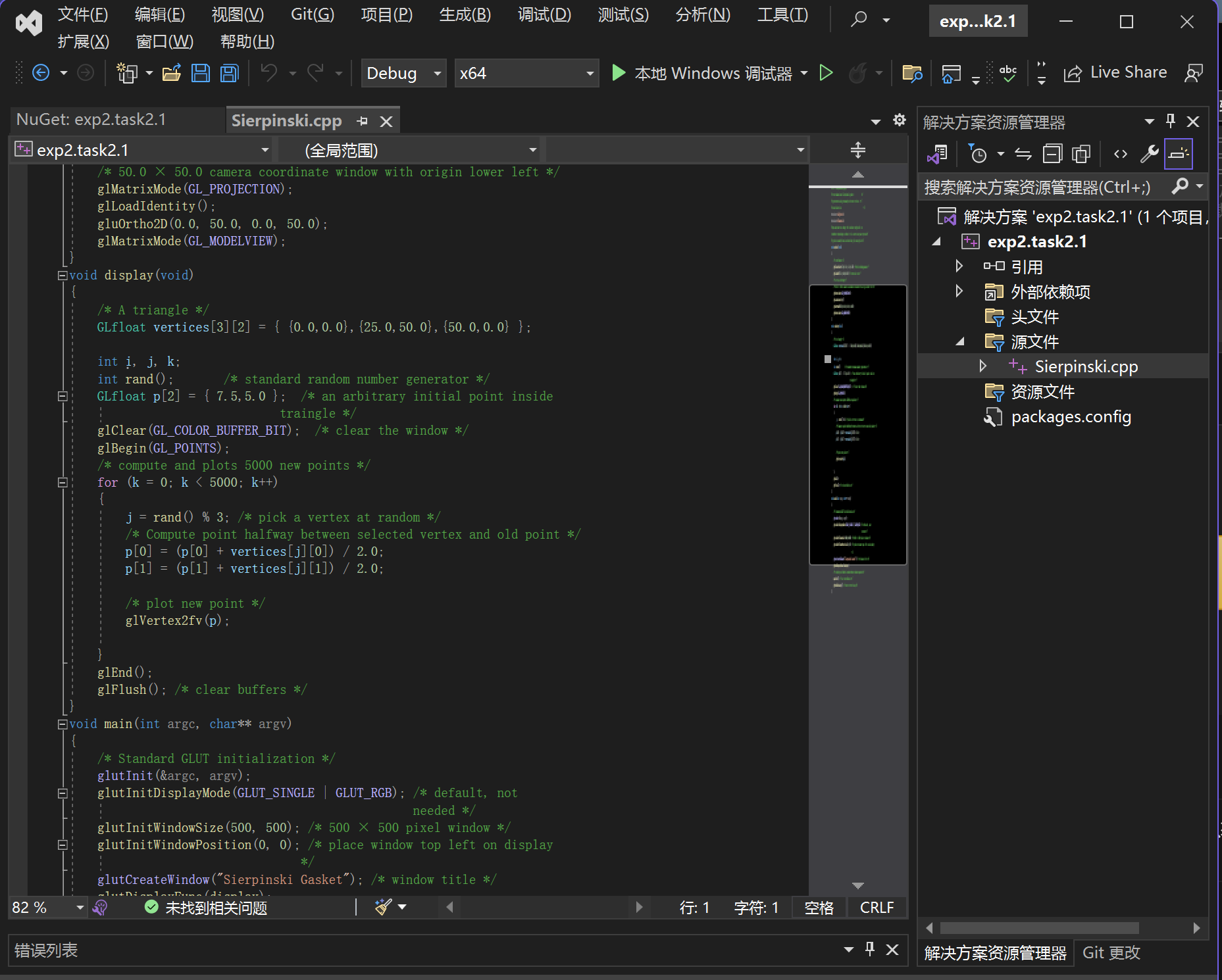
# Task1：实现阅读材料（1）中的Sierpinski镂垫程序，并完成如下具体功能（下面的功能要求逐条递增）

## 1. 1. 理解并实现课本程序

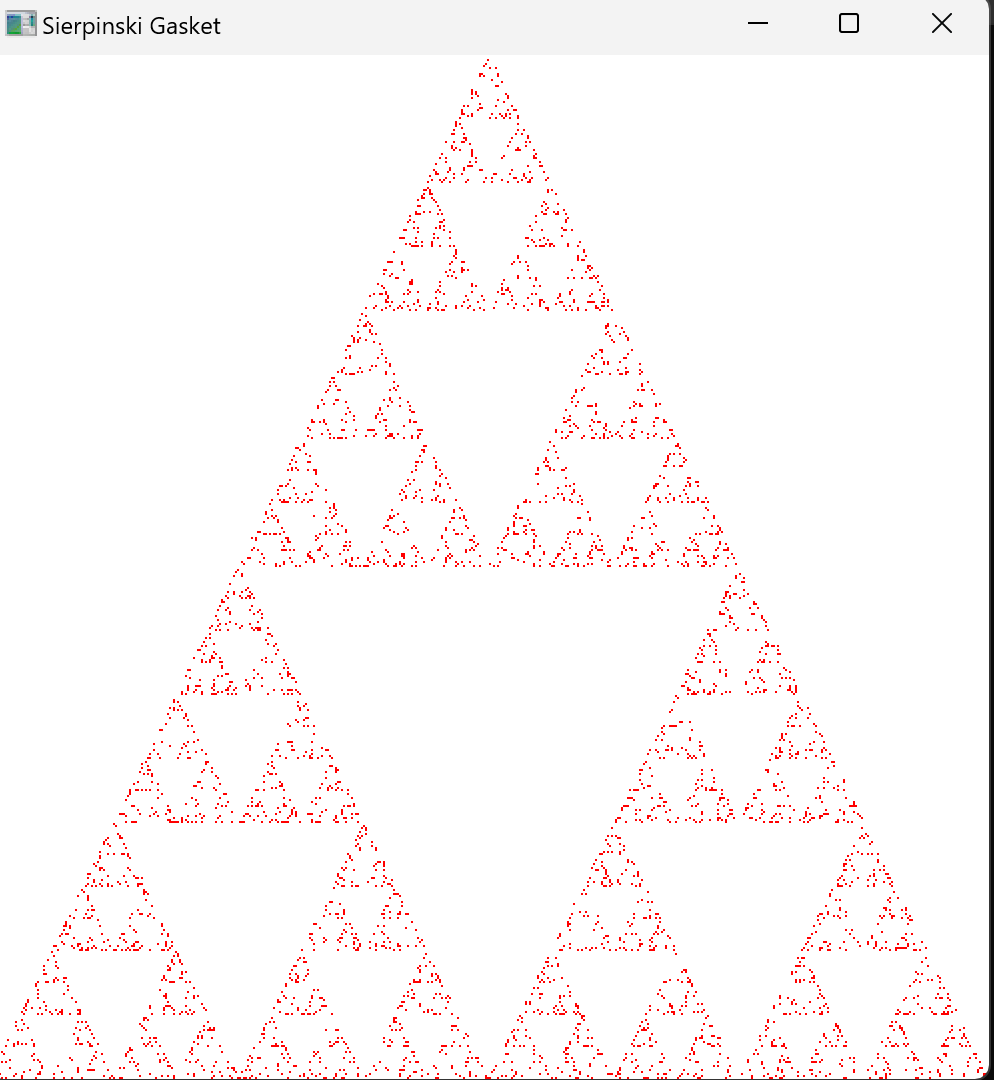
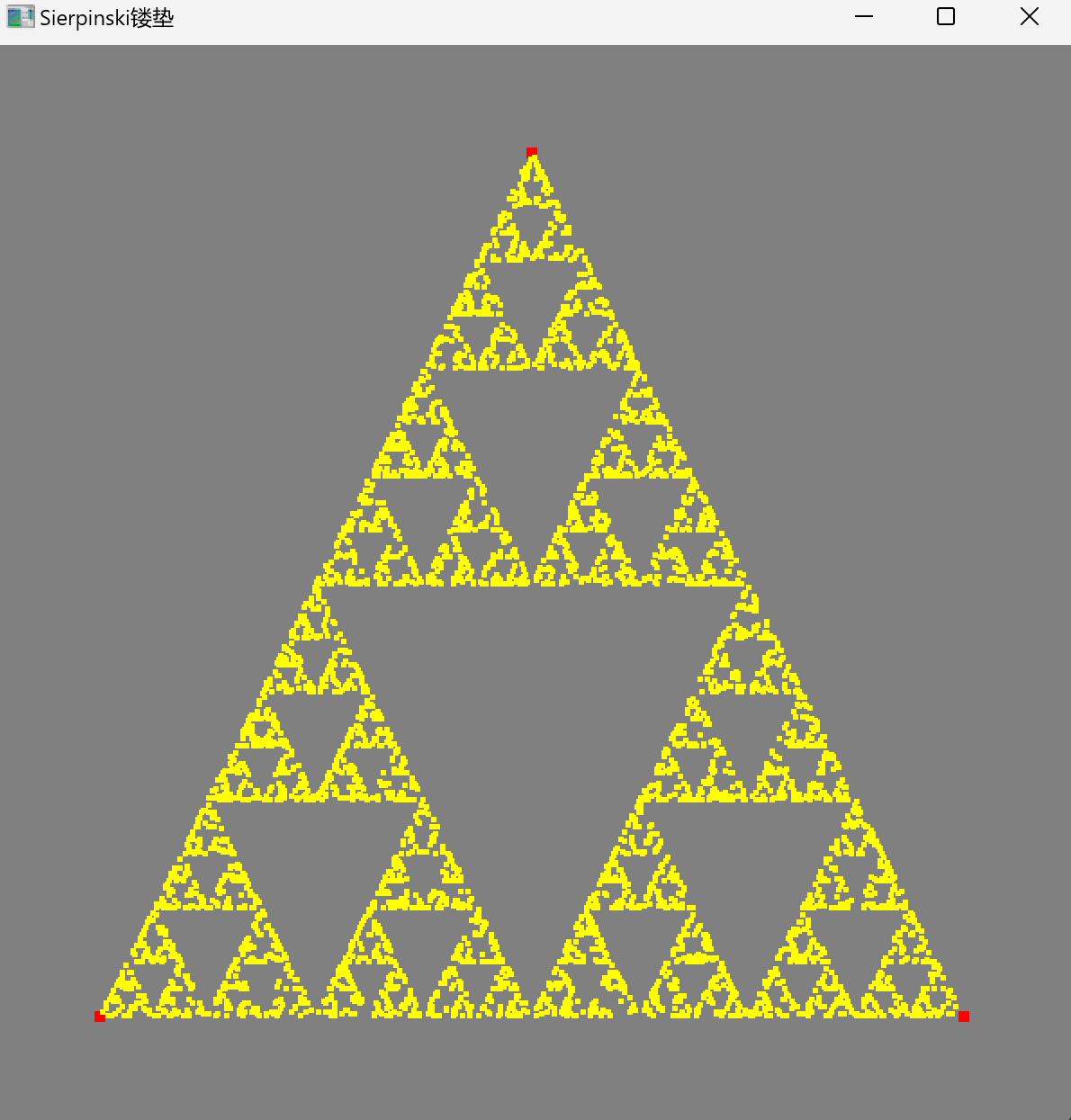
## 操作流程：从网上查找sierpinski镂垫资料/所给实例代码

关键代码截图：





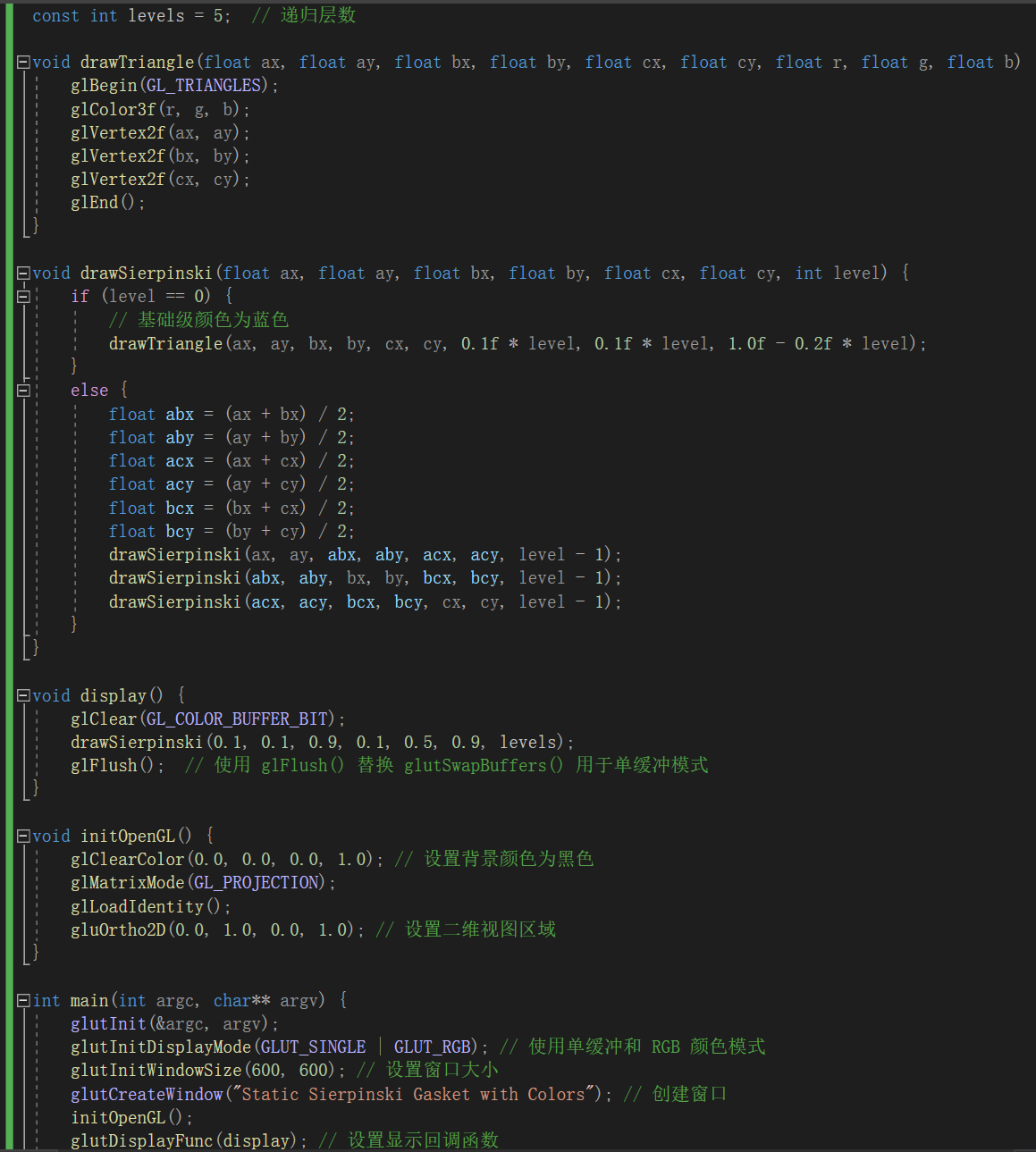
## 运行结果截图：



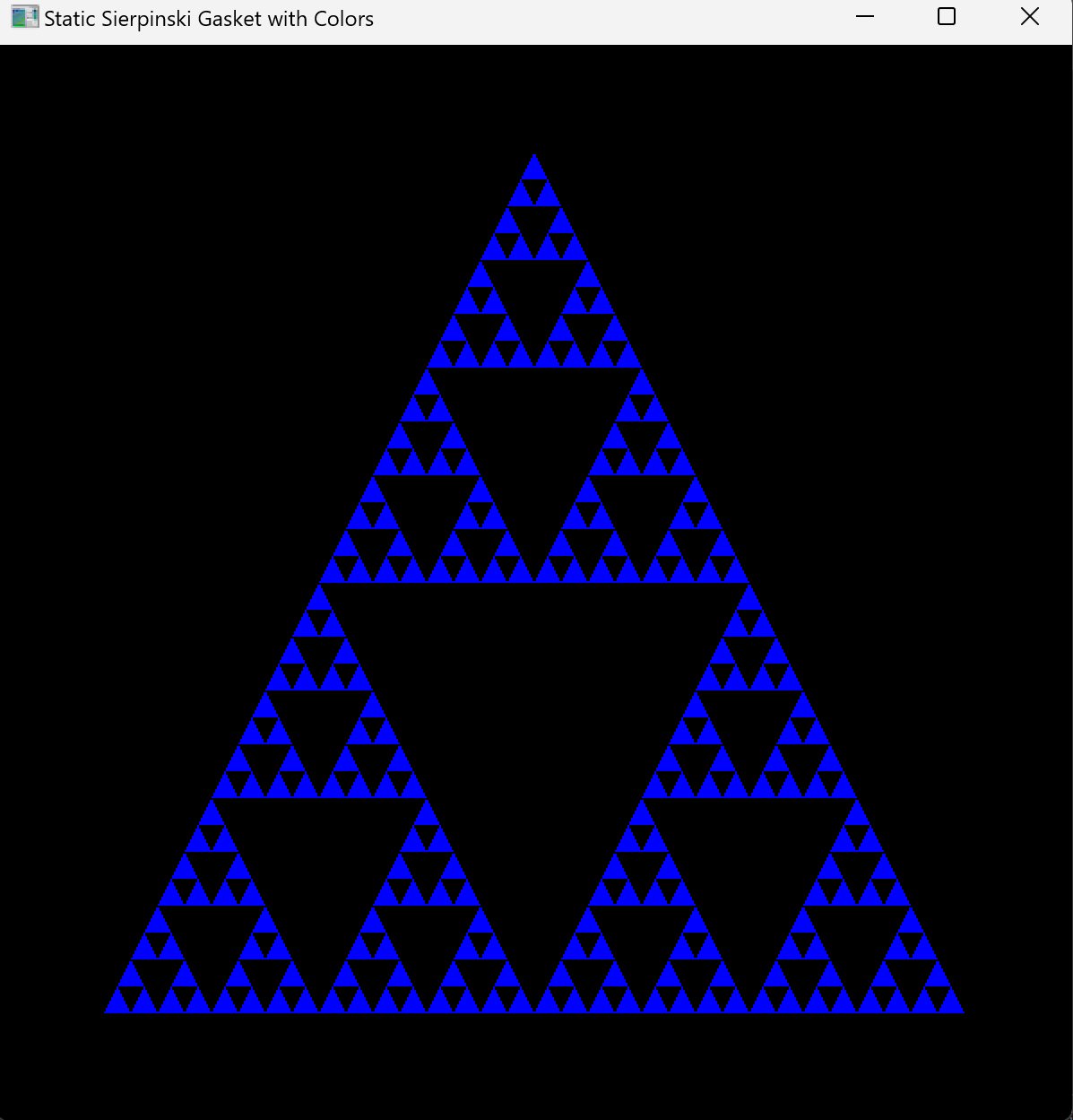
## 2. 尝试为不同三角形设置不同的颜色，使得看起来颜色更加美观。

操作流程：修改三角形的颜色参数以及背景颜色参数等

关键代码截图：



运行结果截图：



## 3. 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。

操作流程：

1.初始化OpenGL

通过 initOpenGL 函数，设置清除屏幕时使用的颜色（黑色），并配置投影矩阵为正交投影，以适应[0,1]的坐标范围。

2. 绘制Sierpinski三角形

drawTriangle 函数负责使用OpenGL的 GL\_TRIANGLES 命令绘制单个三角形。它接收三个顶点坐标，使用三角形的顶点定义三角形，并设置颜色。

drawSierpinski 是一个递归函数，用于在给定的层数（levels）基础上递归绘制更小的Sierpinski三角形。它计算每个递归步骤中新三角形的顶点，然后调用自身三次（为每个新三角形），直到达到基础层级（level 0）。

3. 动画效果

在 display 函数中，进行了一系列变换（平移、缩放、旋转），以实现动画效果。这些变换依赖于全局变量 angle 和 scale，分别控制旋转角度和缩放比例。

update 函数定时更新 angle、scale 和颜色变化的 colorShift。旋转角度持续增加以旋转Sierpinski镂垫；缩放比例在一定范围内循环变化以模拟呼吸效果；颜色的周期性变化基于正弦函数实现。

4. 交互与事件处理

使用 glutTimerFunc 在主函数中注册 update 函数，确保以大约每秒60帧的速度调用，从而更新动画状态和触发重绘。

display 函数被设置为GLUT的显示回调函数，每次窗口需要重新绘制时被调用。

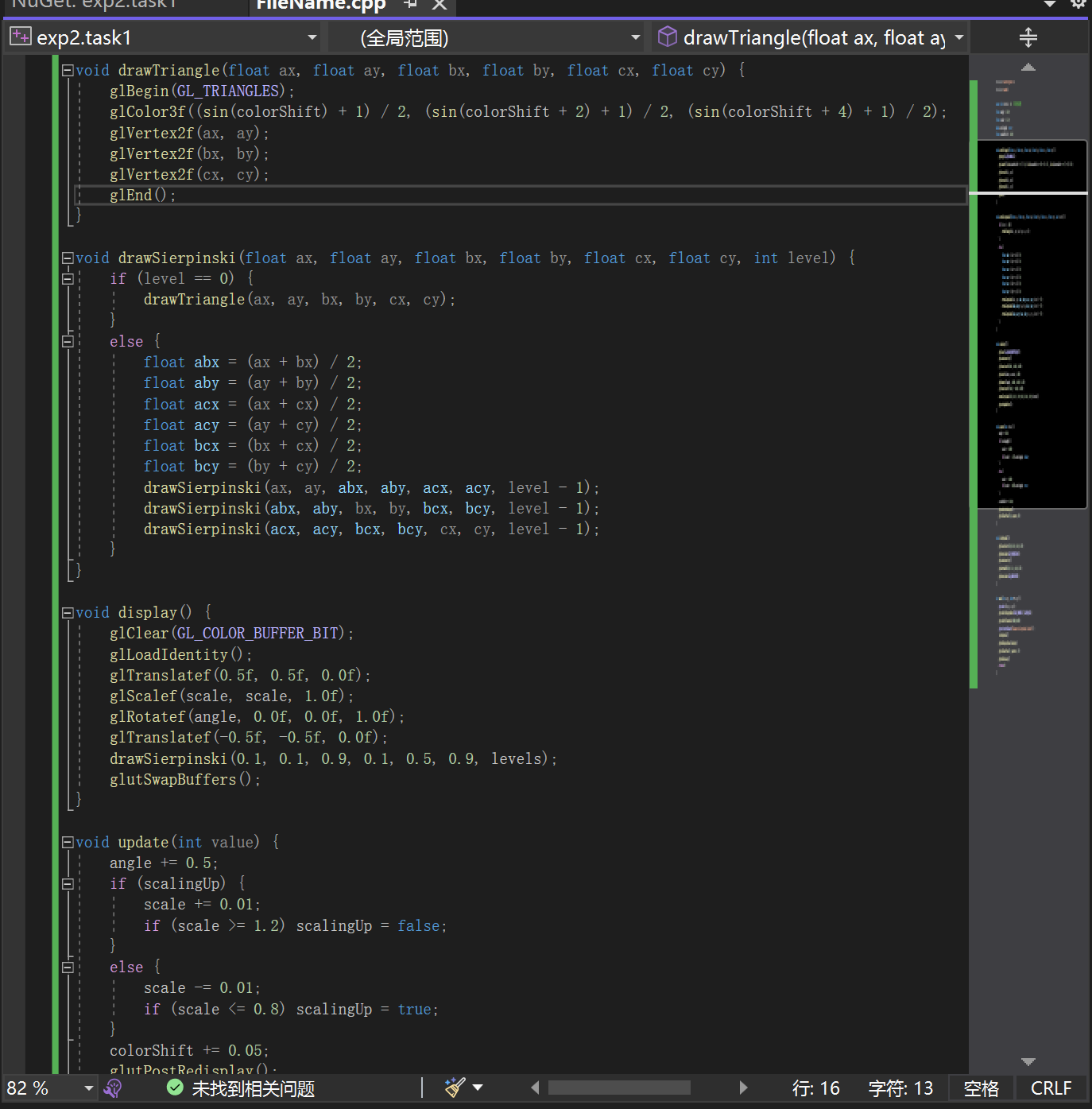
5. 主函数

初始化GLUT库和创建窗口。

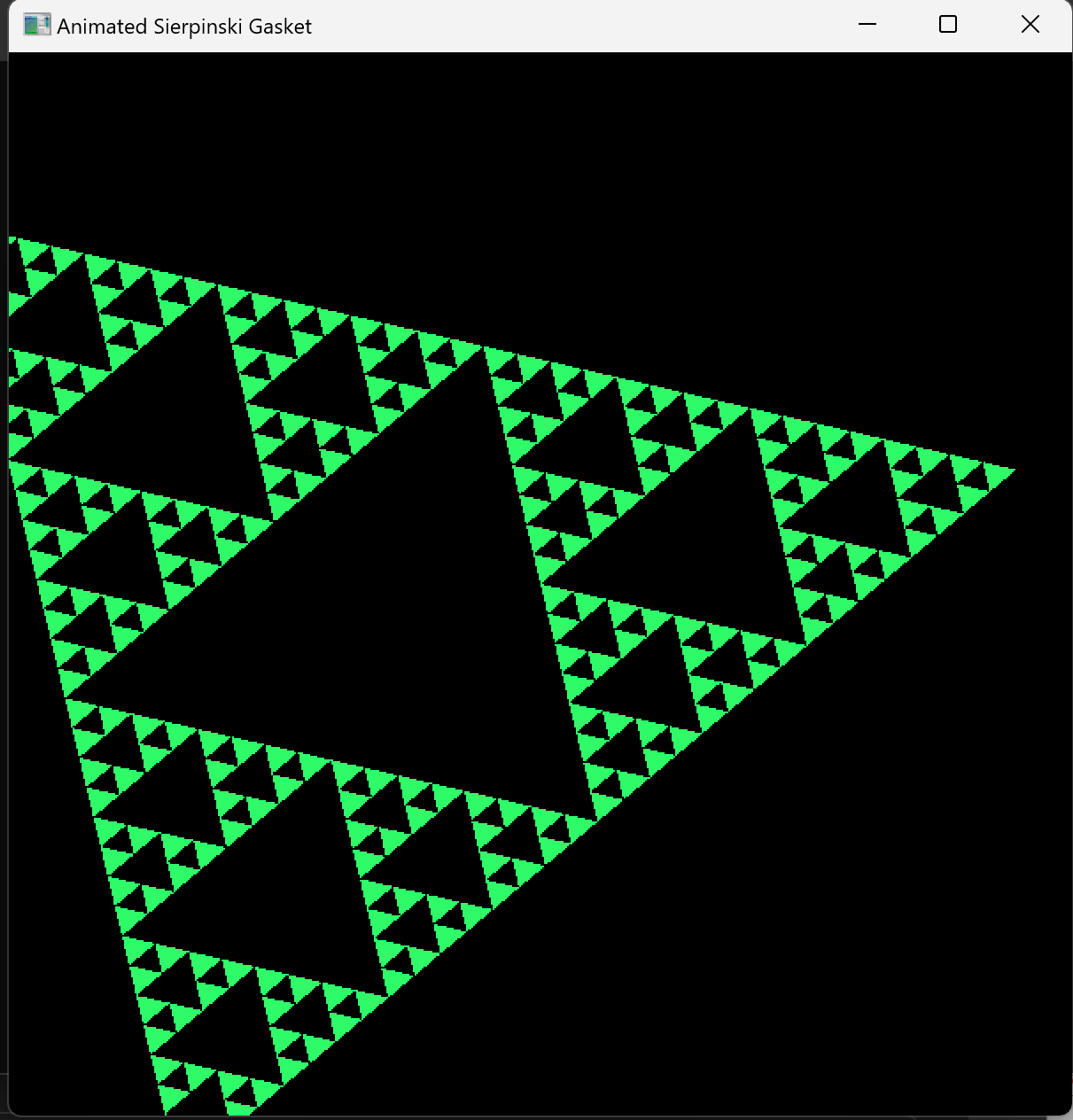
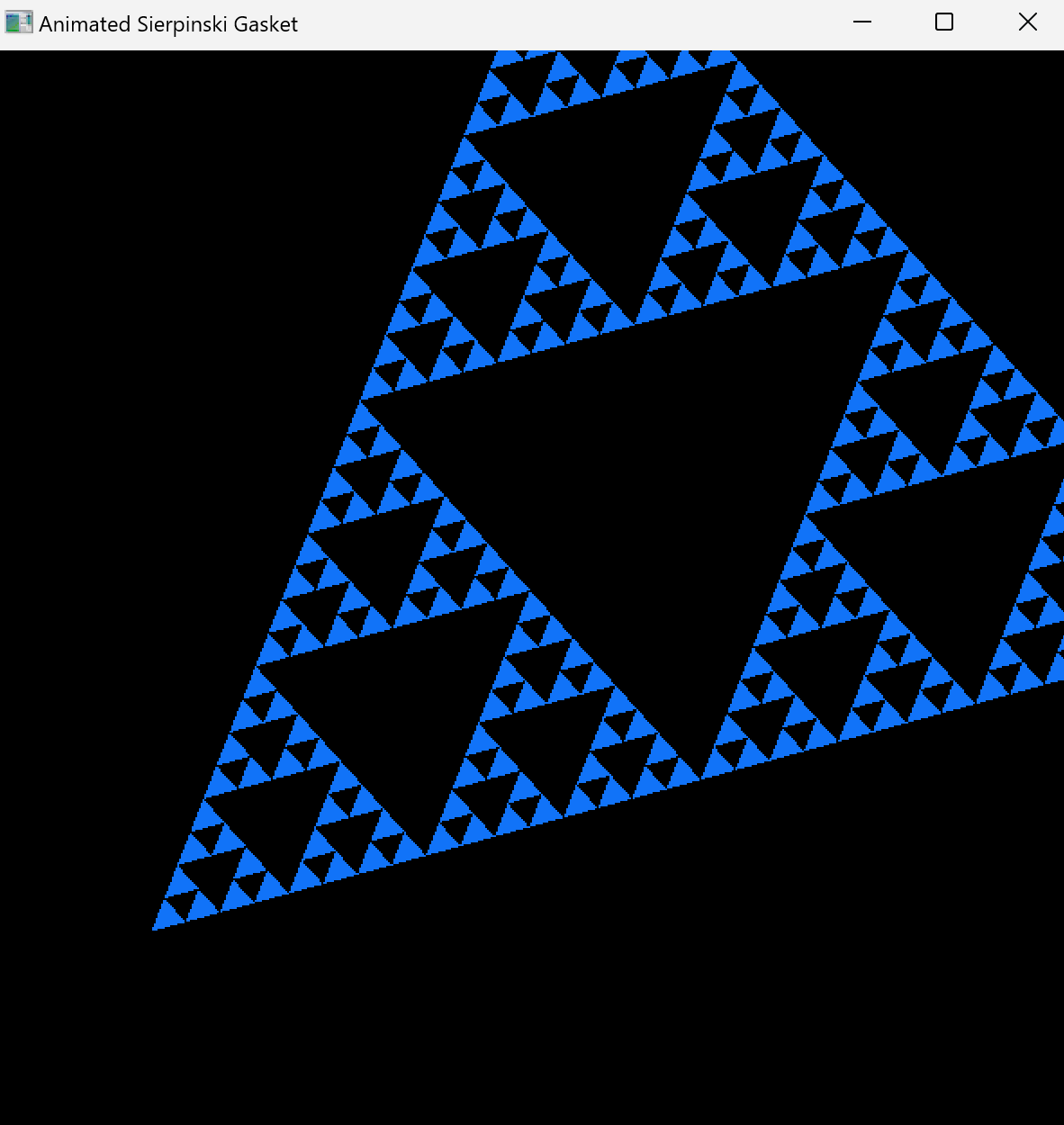
设置显示模式、窗口大小和标题。

注册显示回调函数和定时器回调，然后进入GLUT的事件处理循环。

关键代码截图：



运行结果截图：

# Task2: 完成一个三维的程序

## 1. 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，运行情况如图表2，调整glOrtho、 gluLookAt等函数的参数，了解各参数意义。

## 思考题：

## a. OpenGL中，三维空间的坐标系是怎么样的？

## b. OpenGL中，相机的方位是怎么样的？如何调整相机朝向呢？

## c. OpenGL中，相机可见范围是如何设置的？

操作流程：

打开所提供的程序尝试修改各个参数并观察改变

a. OpenGL中，三维空间的坐标系是怎么样的？

OpenGL采用的是右手坐标系。在这个坐标系中：

X轴：水平向右增加。

Y轴：垂直向上增加。

Z轴：垂直于XY平面，向屏幕外为正（向观察者为负）。

在三维图形中，这种坐标系帮助定义物体的位置、旋转和缩放。

b. OpenGL中，相机的方位是怎么样的？如何调整相机朝向呢？

在OpenGL中，通常不直接操作“相机”，而是通过变换场景来模拟相机的移动。gluLookAt 函数是设置视图矩阵来模拟相机位置和方向的常用方法。该函数需要九个参数：

相机位置的x, y, z坐标：确定相机在世界坐标系中的位置。

视点的x, y, z坐标：相机指向的点。

上向量的x, y, z分量：定义了哪个方向是上方，通常设置为(0,1,0)，表示Y轴是上方。

通过调整这些参数，可以控制相机的位置、它所看向的方向以及确定哪个方向是上方，从而调整相机的朝向。

c. OpenGL中，相机可见范围是如何设置的？

OpenGL中相机的可见范围通常通过设置投影矩阵来定义，有两种主要类型的投影：

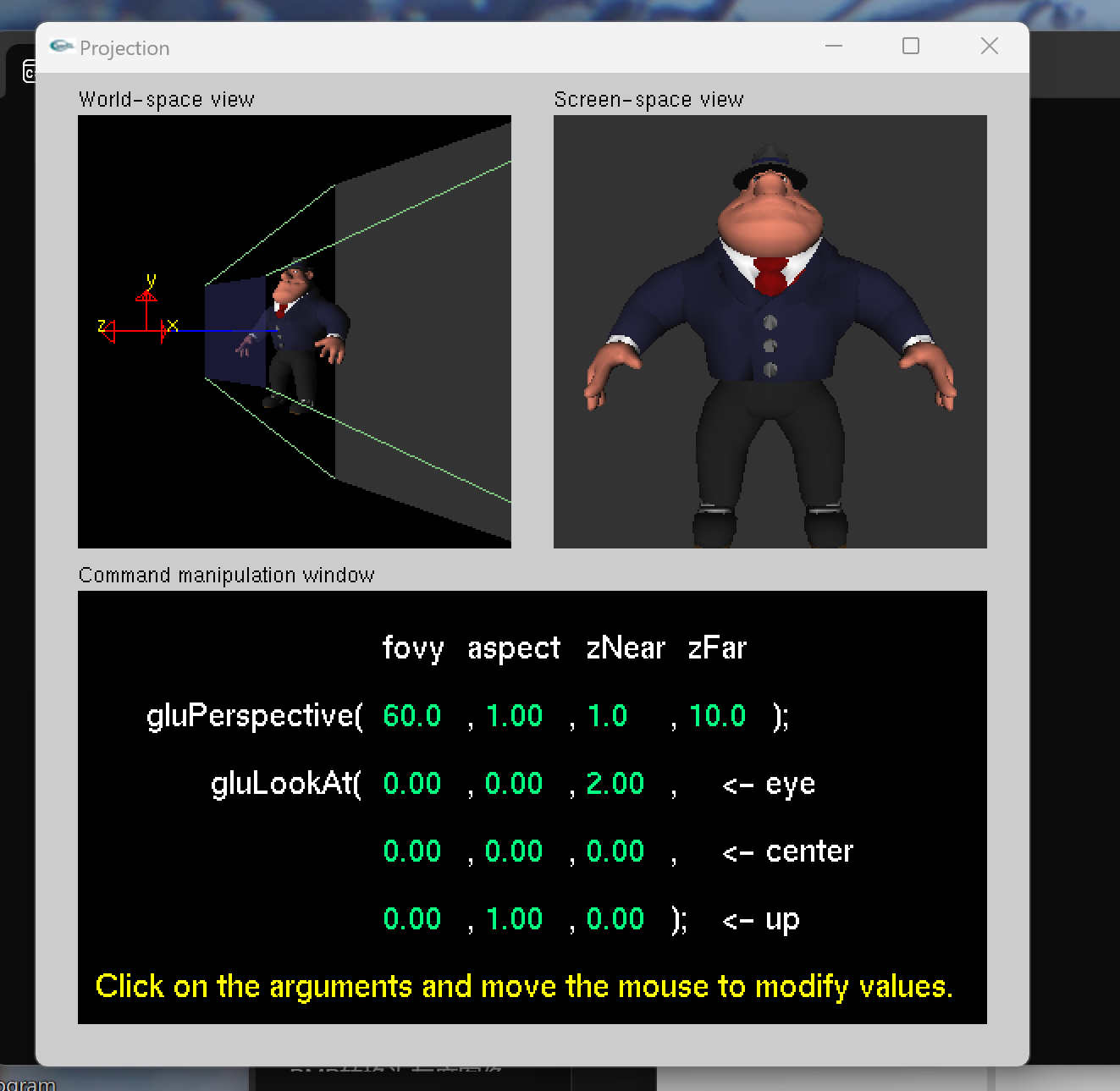
正射投影 (glOrtho)：在这种投影中，物体的大小不会随距离变化。glOrtho函数定义了一个视景体，只有在这个定义的立方体（或长方体）内的物体才可见。参数通常包括左、右、下、上、近、远平面的位置。

透视投影 (gluPerspective)：模拟人眼所见，距离越远的物体看起来越小。gluPerspective函数设置视野角度、纵横比、近平面和远平面。近平面和远平面定义了可视深度，只有在这个范围内的物体才可见。

这些设置对于构建一个真实感的3D场景非常关键，因为它们影响着用户视觉上对场景深度和空间关系的感知。调整这些参数可以控制场景的焦点，例如，通过调整近平面和远平面的距离来避免“剪切”问题，即视野太近或太远导致物体被错误地裁剪掉。

运行结果截图：

改变调整各个参数了解到各个参数的实际意义



## 2. 运行提供的示例程序(exp2-2-1.cpp)，对程序进行改写

**比较开启/不开启深度缓冲区 glEnable(GL\_DEPTH\_TEST) 的效果;理解深度缓冲区的作用、用法**

①操作流程：

1. a. 深度缓冲区的作用

深度缓冲区（Depth Buffer），也称作Z-buffer，存储了图像中每个像素点的深度信息，这是视点到这个像素点最近对象的距离。当多个对象在同一像素处绘制时，OpenGL会使用这些深度值来决定哪个对象是可见的（即哪个对象更接近观察者），而将其它被遮挡的部分隐藏。

b. 开启/不开启深度缓冲区的效果

开启深度测试 (glEnable(GL\_DEPTH\_TEST);)

允许OpenGL检查一个像素的新绘制深度与当前深度缓冲区的值。

如果新绘制的像素比缓冲区中存储的像素深度更接近或者相同，它会被绘制在屏幕上，并更新深度缓冲区的值。

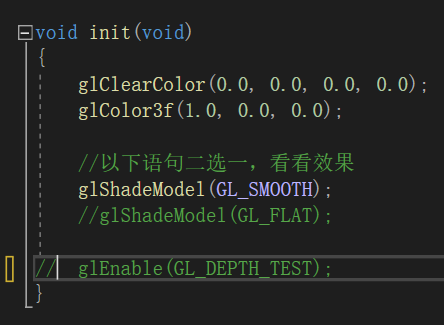
这样可以正确处理物体间的遮挡关系。

不开启深度测试 (glDisable(GL\_DEPTH\_TEST);)

OpenGL将不会检查深度信息，物体的绘制将仅依赖于它们被调用的顺序。

这可能导致远处的物体被错误地绘制在近处物体之上，从而出现视觉上的错误。

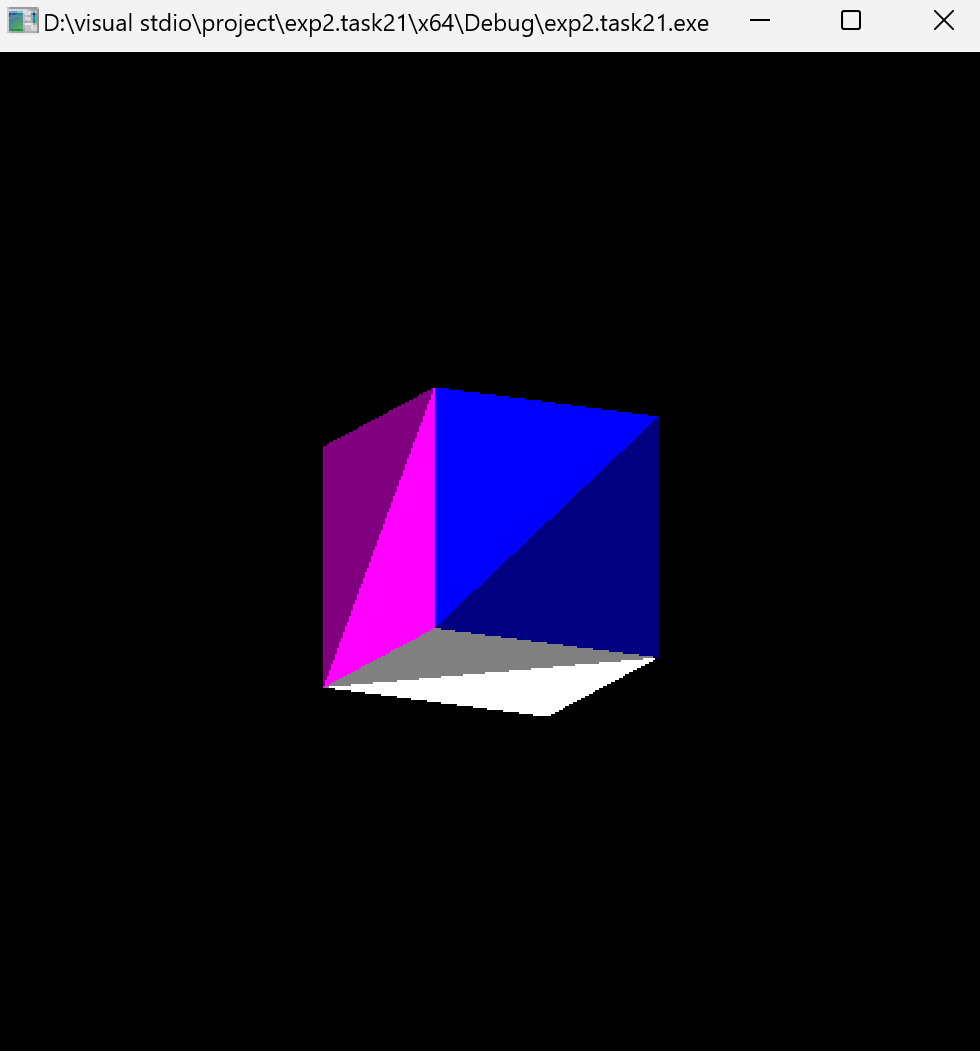
关键代码截图：



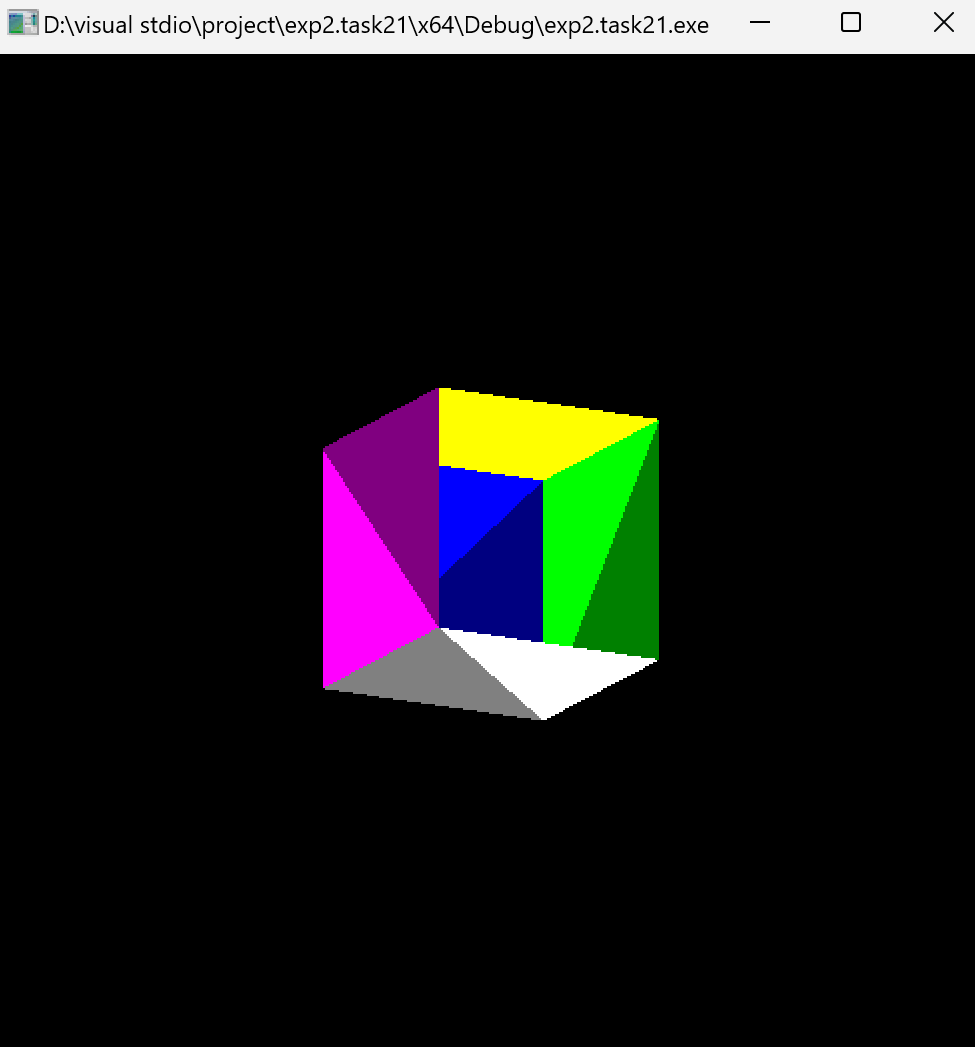
通过注释掉深度测试以完成

运行结果截图：

源程序



不开启深度测试后



**让正方体自行旋转，而不是相机旋转**

②操作流程：

初始化 (init函数):

设置3D渲染的背景色为黑色。

启用深度测试，这对于3D视图中物体的前后关系至关重要，确保远处的物体不会遮挡近处的物体。

设置着色模式为平滑，这影响了图形的颜色渐变和光滑度。

显示 (display函数):

清除颜色和深度缓冲，为新的绘制做准备。

设置模型视图矩阵，定位摄像机的位置和观察方向，确保摄像机固定看向原点，并设置其上方向。

应用旋转变换，使立方体围绕y轴旋转，旋转角度由变量 beta 控制。

使用四边形（GL\_QUADS）绘制立方体的六个面，每个面指定不同的颜色。

窗口调整 (reshape函数):

调整视口大小以匹配窗口大小，确保图形比例正确。

设置投影矩阵，使用透视投影以模拟人眼视觉，增加远小近大的视觉效果。

定时更新 (update函数):

定期更新立方体的旋转角度，创建连续旋转的动画效果。

每次更新后请求重新绘制，保持画面的连续变化。

键盘交互 (keyboard函数):

监听键盘输入，允许用户按下ESC键退出程序。

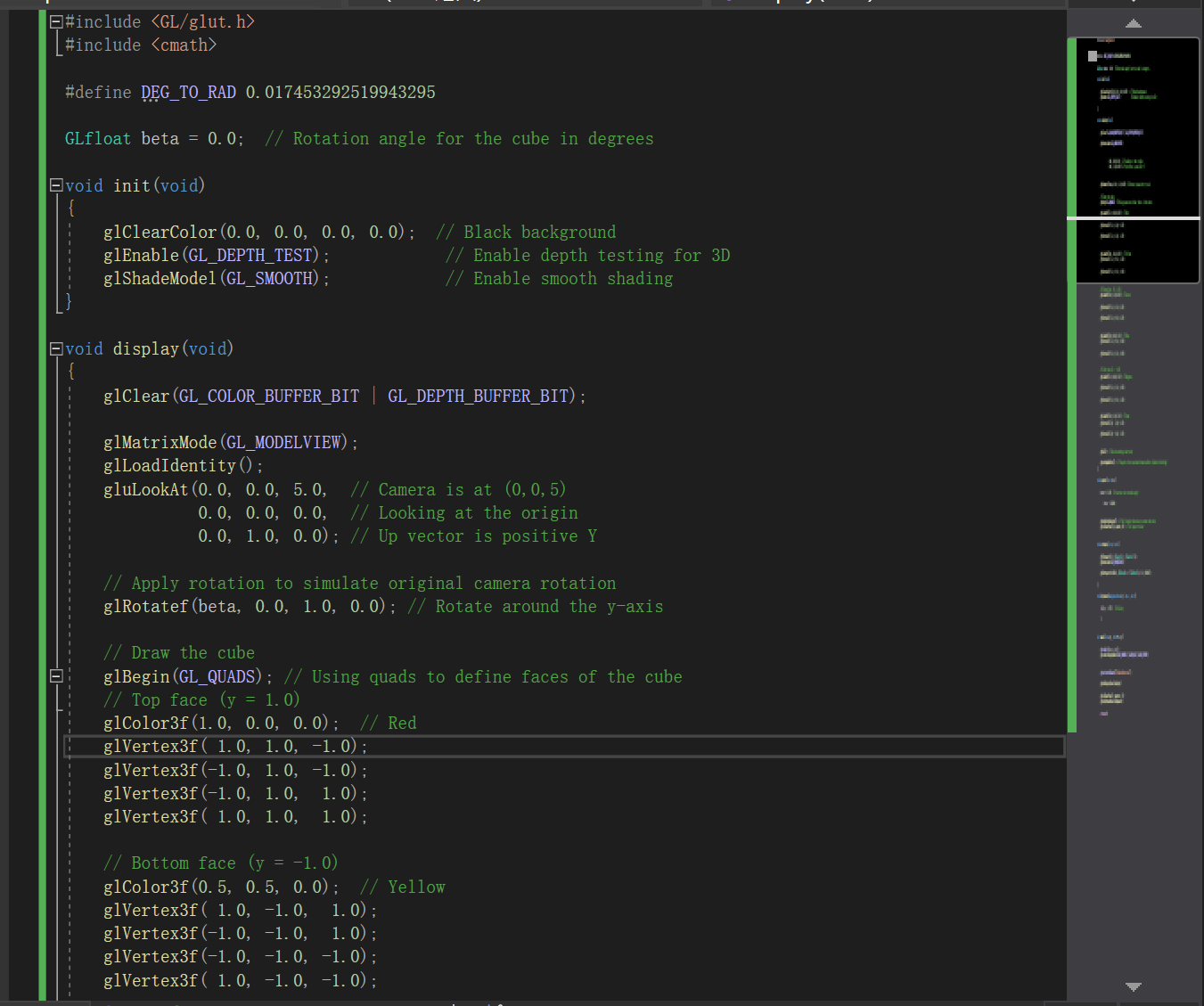
主函数 (main函数):

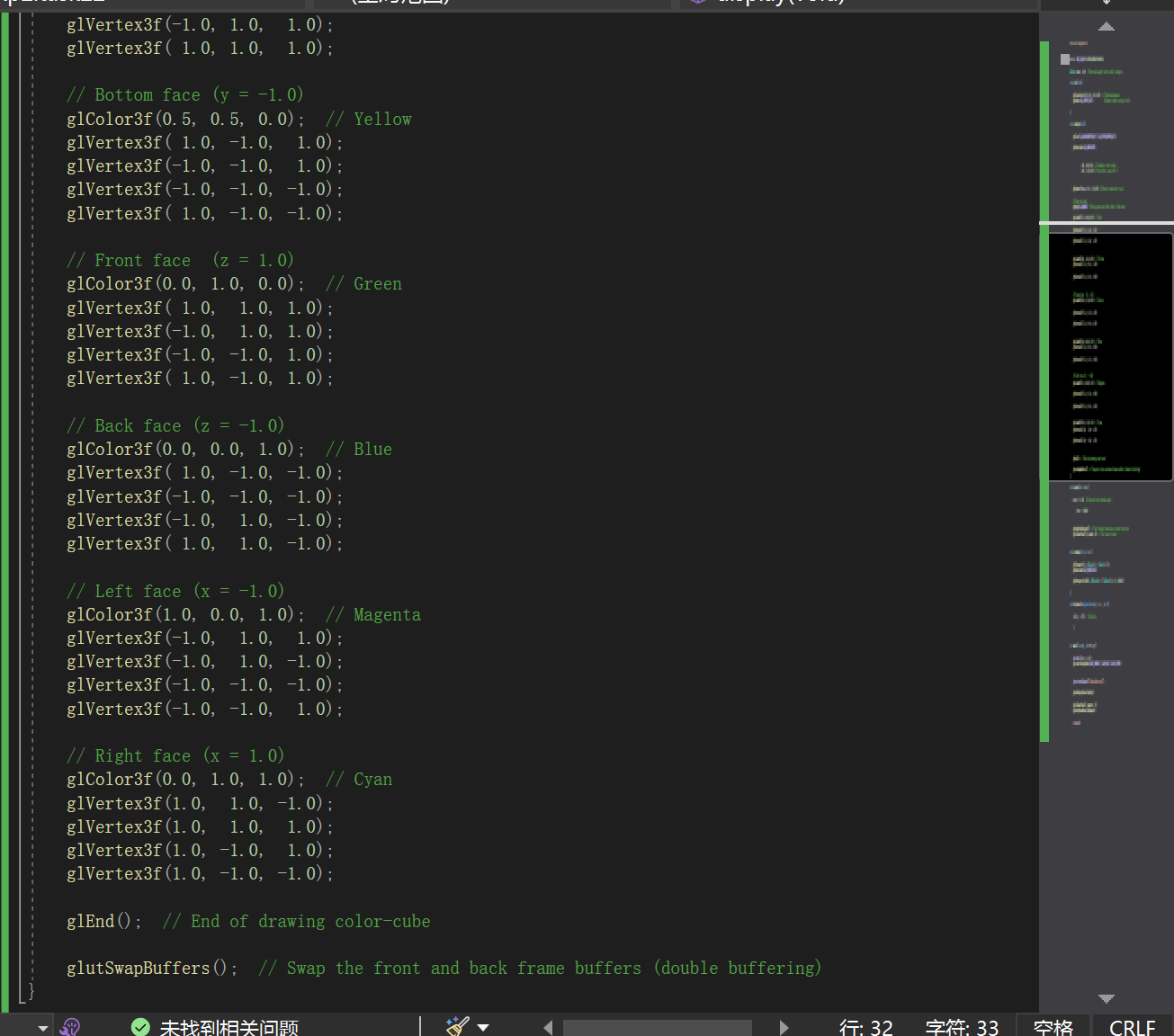
初始化GLUT库和创建窗口。

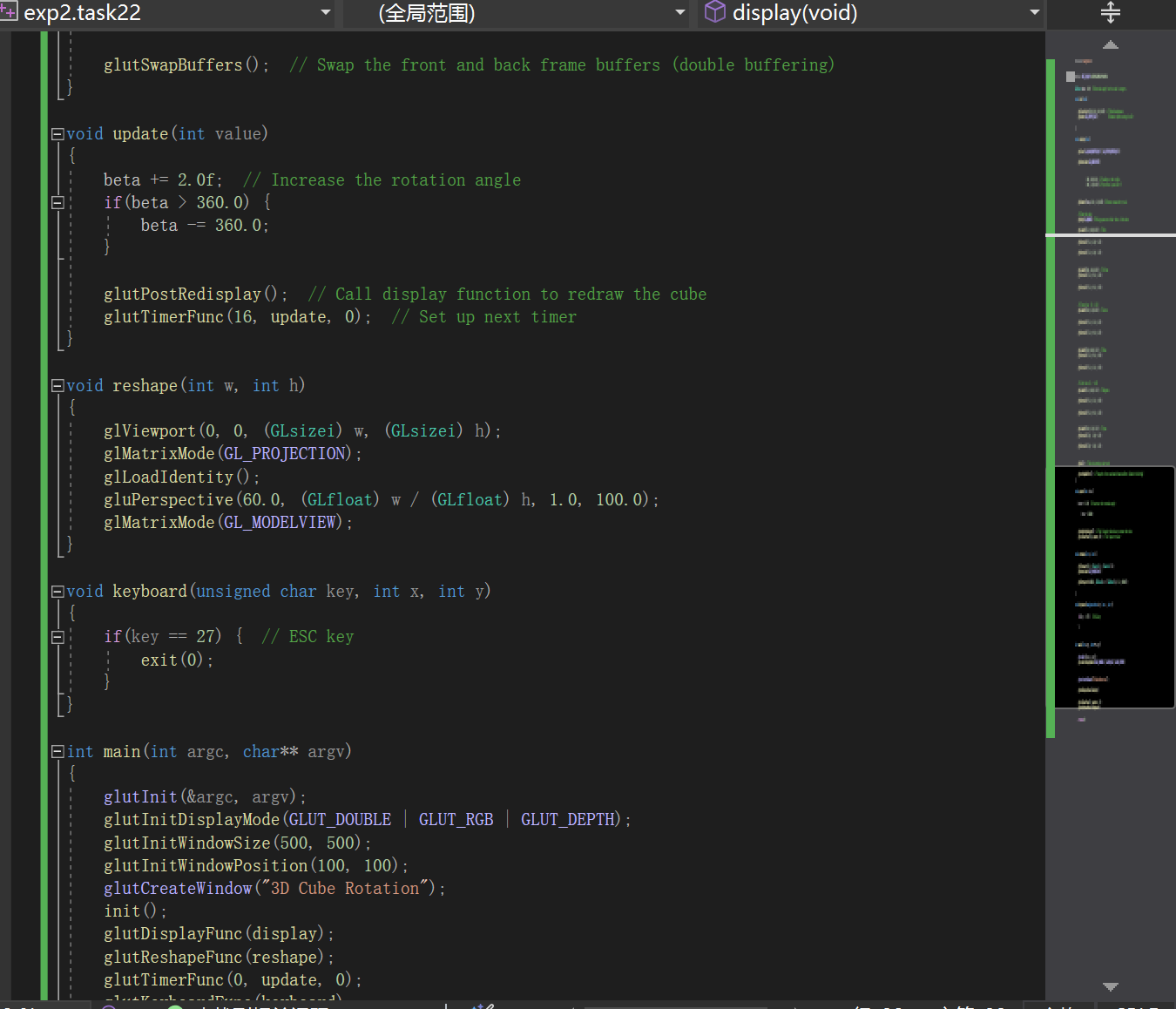
注册各种回调函数（如显示、窗口调整、键盘交互等）。

进入GLUT的主循环，开始事件驱动的图形显示

关键代码截图：

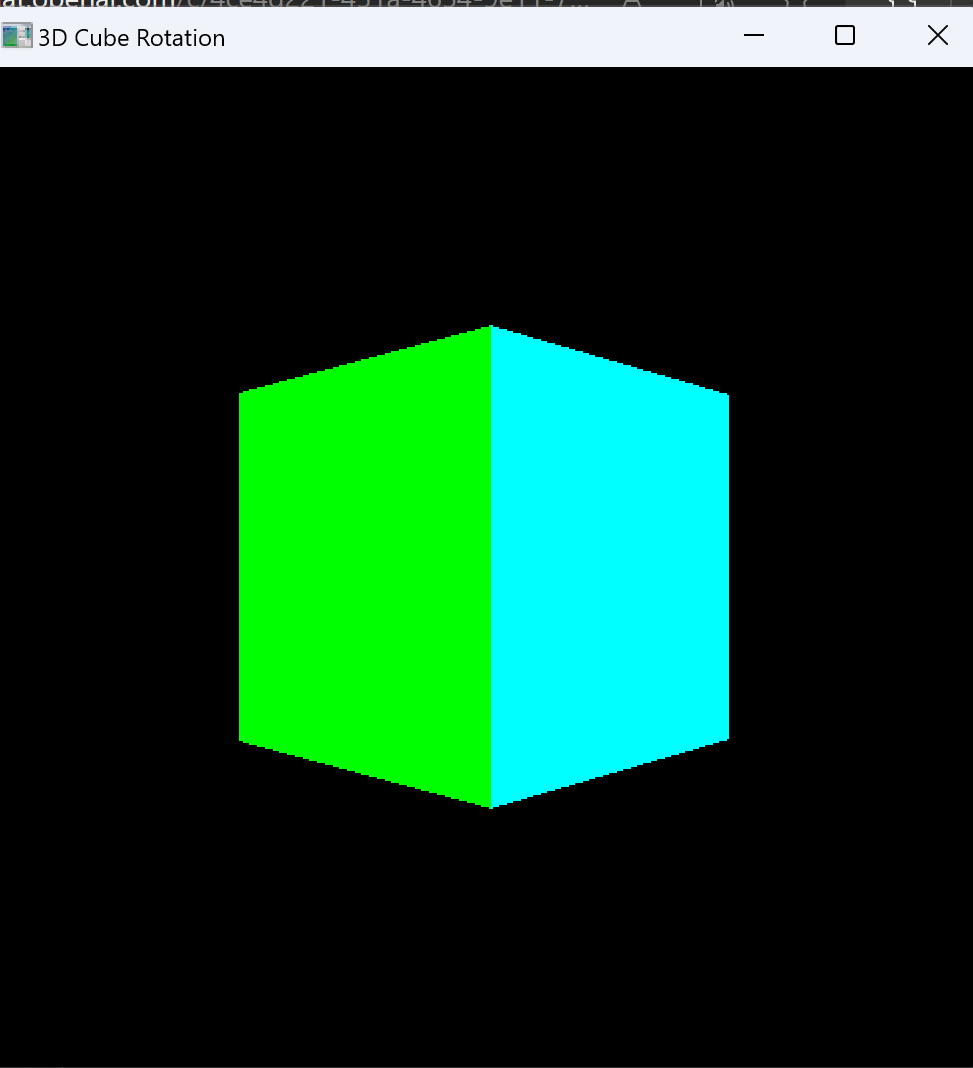


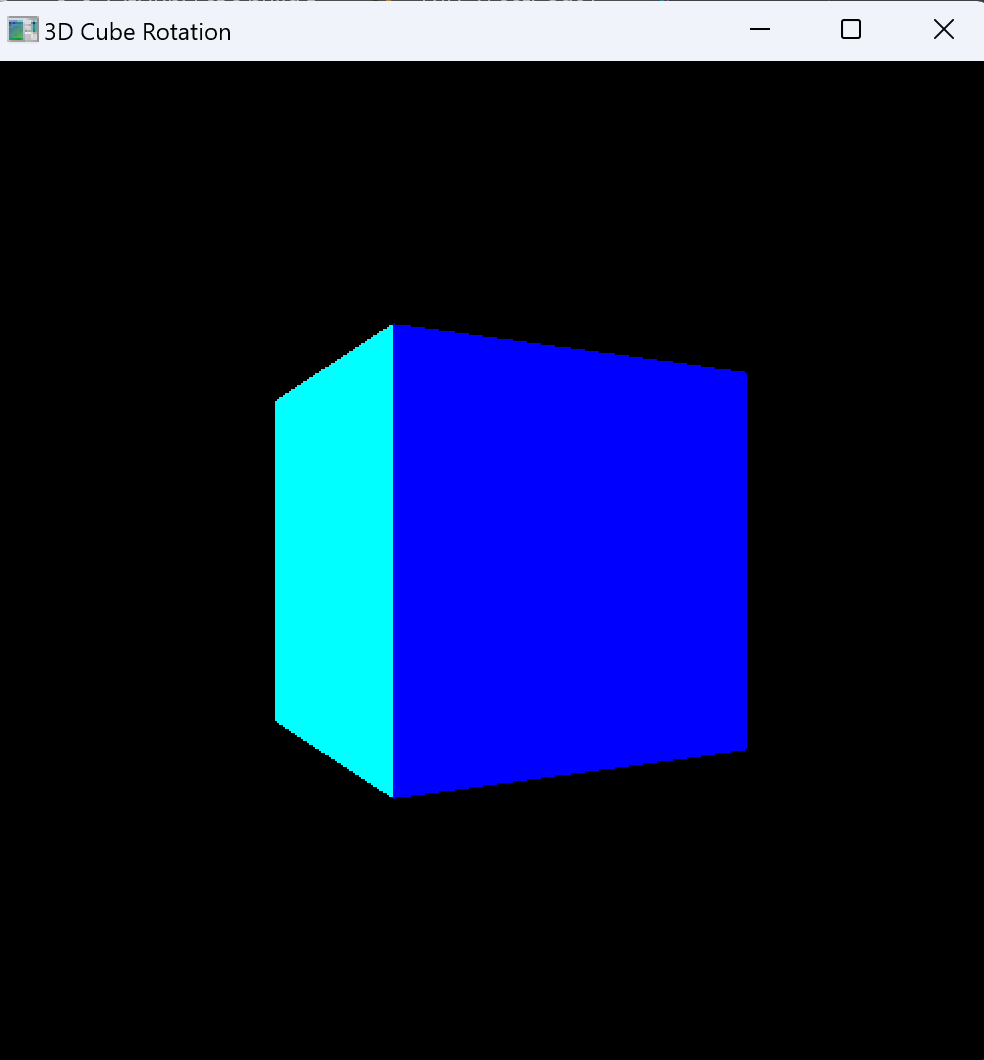




运行结果截图：

固定相机，使得立方体绕y轴旋转





**在正方体自行旋转的前提下，实现交互式的相机控制（wasd控制相机的前进后退左右移动，qe实现相机的升降，使用鼠标调整相机的朝向，L锁定相机的移动和旋转）**

③操作流程：

初始化（init 函数）：

设置3D渲染环境，包括背景色和启用深度测试，确保3D场景中物体的前后能正确渲染。

设置着色模式为平滑，这样物体表面的颜色过渡会更自然。

显示（display 函数）：

清除颜色和深度缓冲区，为新一帧的渲染做准备。

设置摄像机的位置和朝向，根据全局变量 camX, camY, camZ（摄像机位置）和 camPitch, camYaw（摄像机朝向）计算得到。

根据全局变量 beta 控制立方体绕y轴的旋转。

使用OpenGL的四边形（GL\_QUADS）绘制一个颜色立方体，每个面使用不同的颜色标识。

相机和立方体的动态更新（update 函数）：

定时更新立方体的旋转角度 beta，实现连续旋转效果。

触发重绘事件，以持续更新立方体的视觉显示。

窗口调整（reshape 函数）：

当窗口大小发生变化时，更新视口和投影矩阵，确保3D场景的比例和视角正确。

键盘和鼠标交互（keyboard 和 mouseMove 函数）：

键盘输入控制相机的移动（WASD键）、升降（QE键）和锁定（L键）。

鼠标移动用于调整相机的俯仰（camPitch）和偏航（camYaw）角度。

相机位置和朝向的改变影响 gluLookAt 函数的参数，从而改变观察的视角。

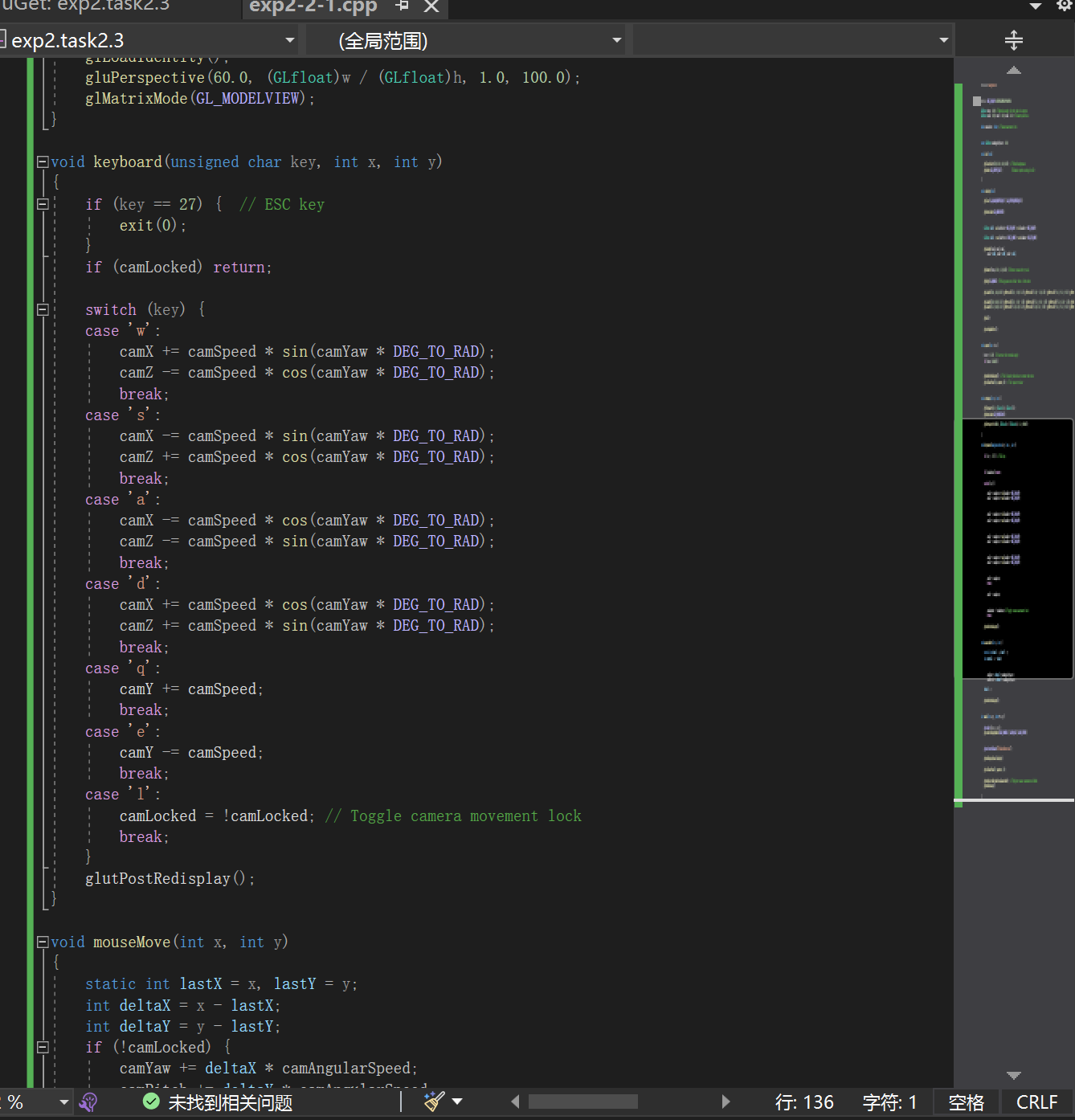
主函数（main 函数）：

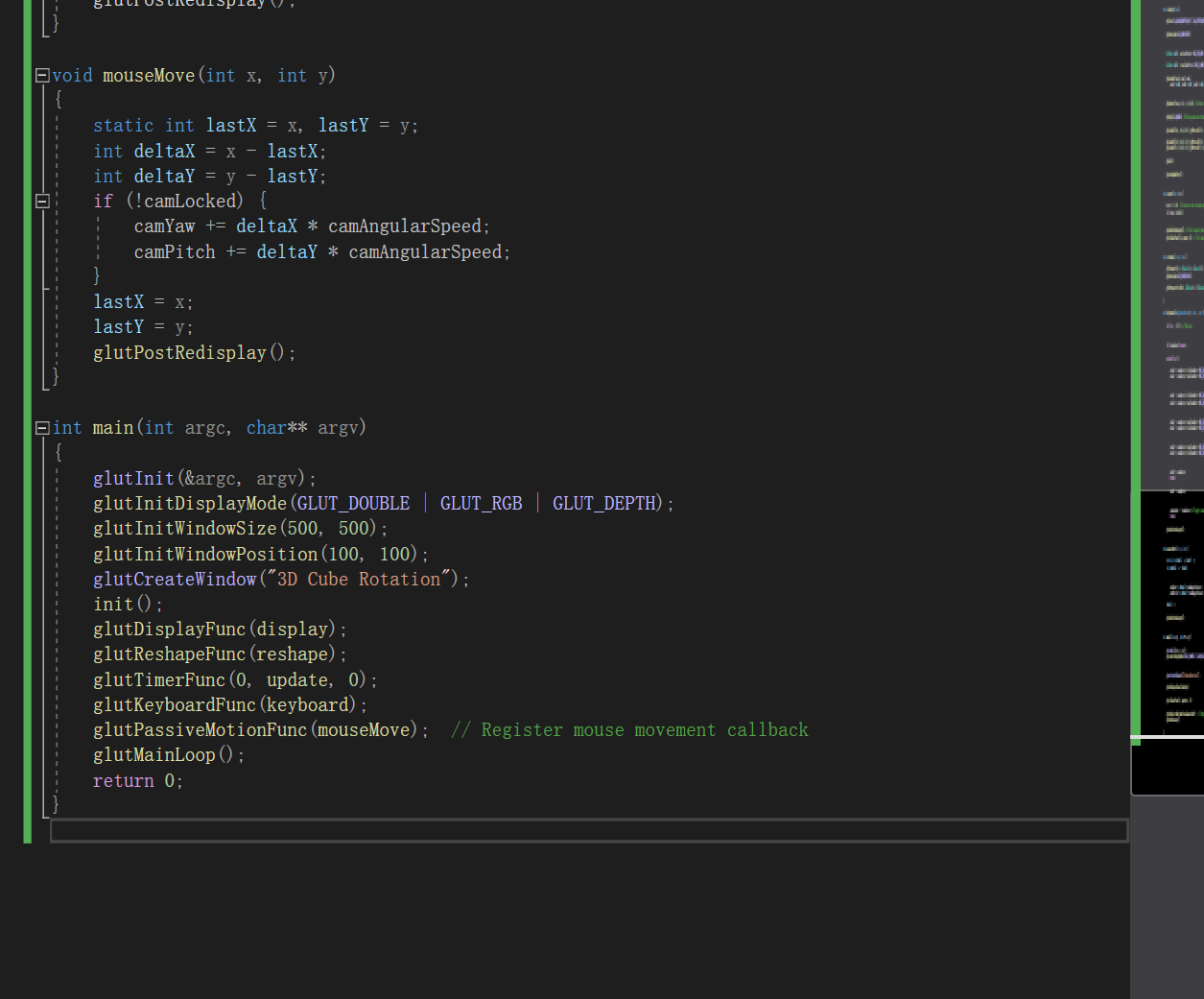
初始化 GLUT 和 OpenGL 环境。

注册回调函数，包括显示、窗口调整、键盘输入、鼠标移动等。

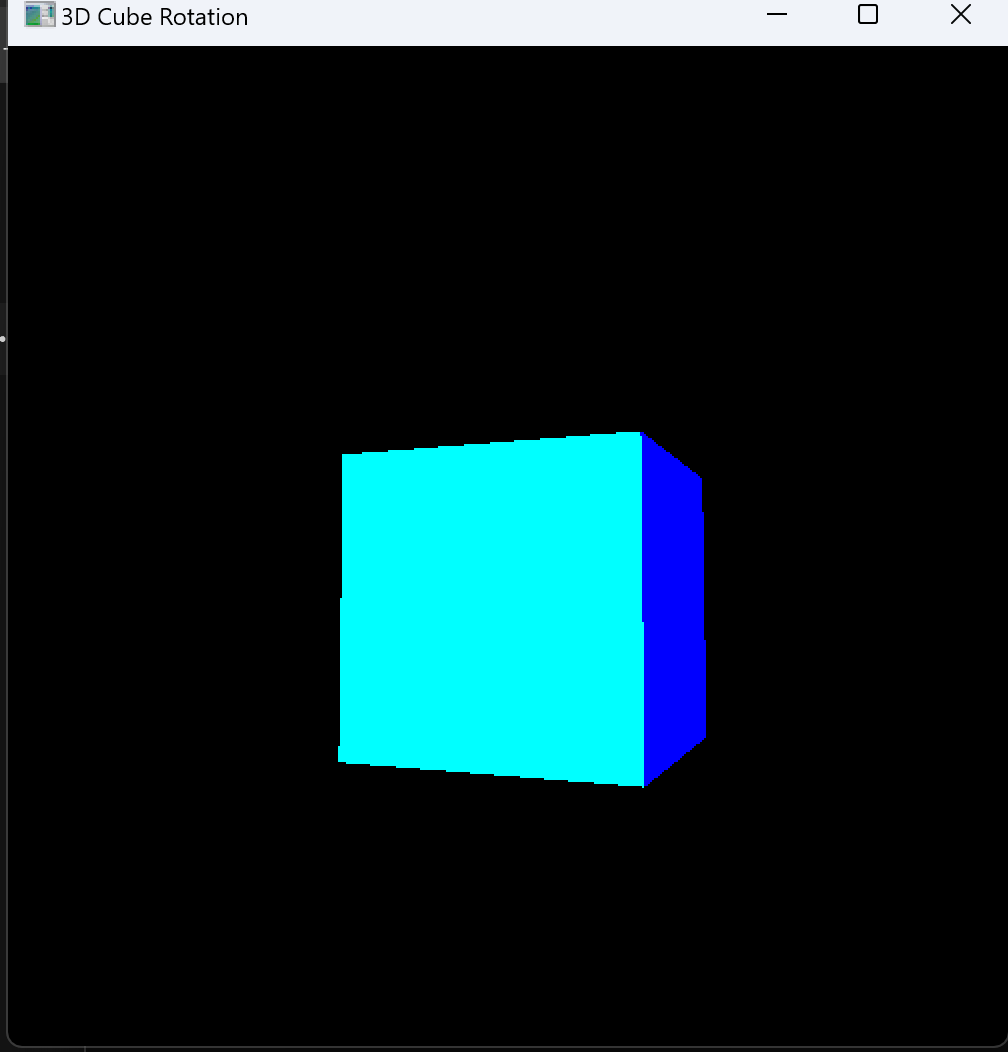
开始事件处理循环，响应用户交互和系统事件。

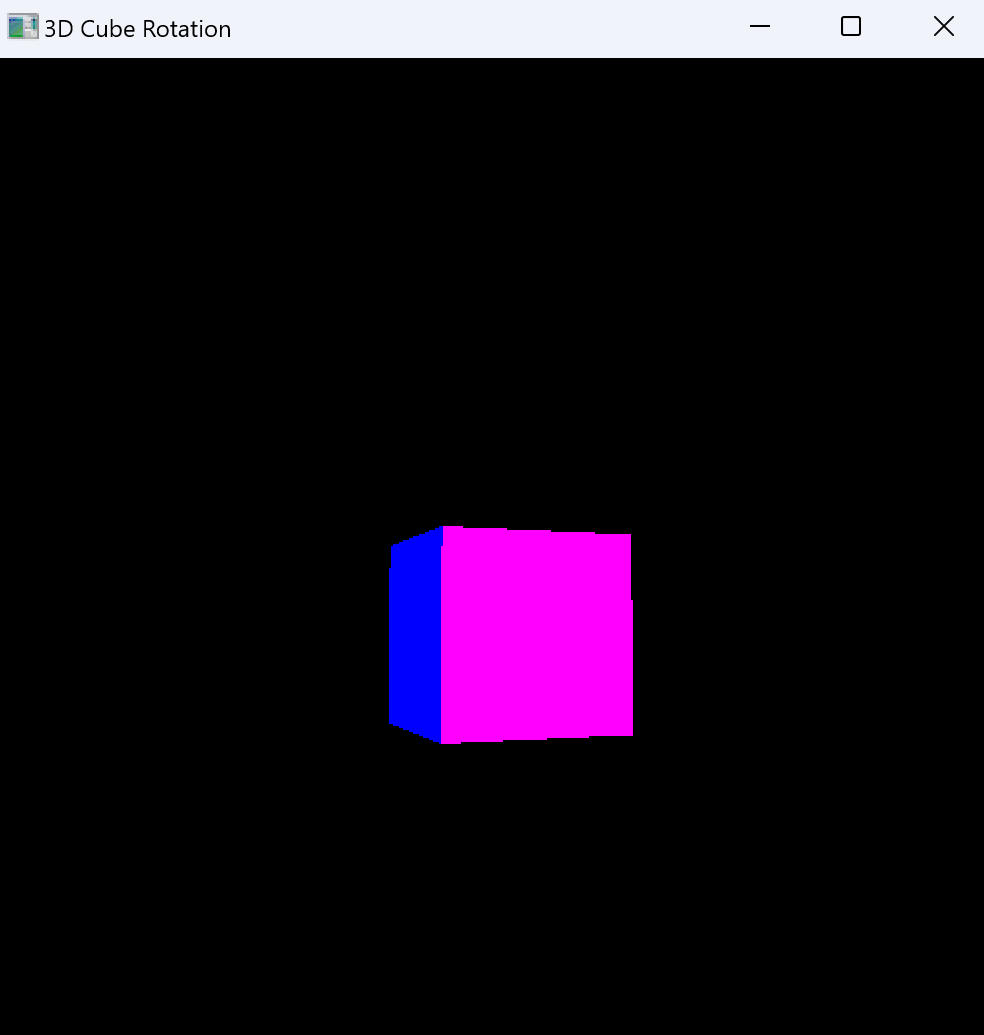
关键代码截图：





运行结果截图：





**TASK3：绘制一个线框球体**

操作流程：

1. 初始化 OpenGL 环境

使用 init 函数设置OpenGL的初始配置，包括背景色、深度测试的启用以及投影模式的设置。这确保了3D图形可以正确渲染。

2. 绘制球体

drawSphere 函数负责绘制球体的线框。它通过遍历纬度和经度来计算球体上每个点的坐标，使用GL\_LINE\_LOOP和GL\_LINES等OpenGL基本图元进行绘制。

3. 动画效果

使用 update 函数实现球体的动画效果。这个函数周期性地更新球体的旋转角度 beta，并调用 glutPostRedisplay 触发重绘，使球体连续旋转。

4. 调整视窗和保持比例

reshape 函数响应窗口大小的变化，更新视口和投影矩阵，确保不论窗口尺寸如何变化，场景的比例都保持不变，防止图形变形。

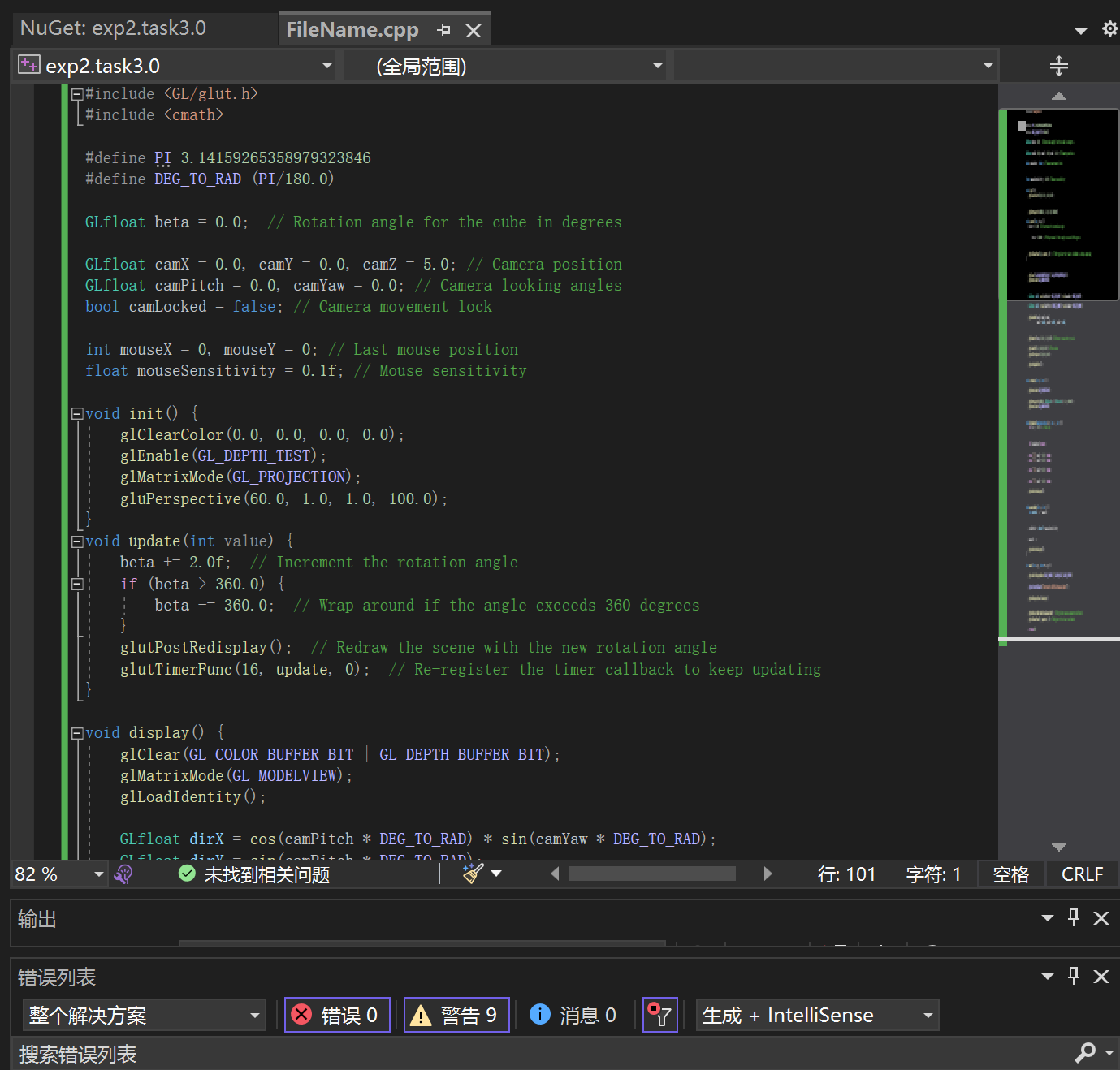
5. 交互式相机控制

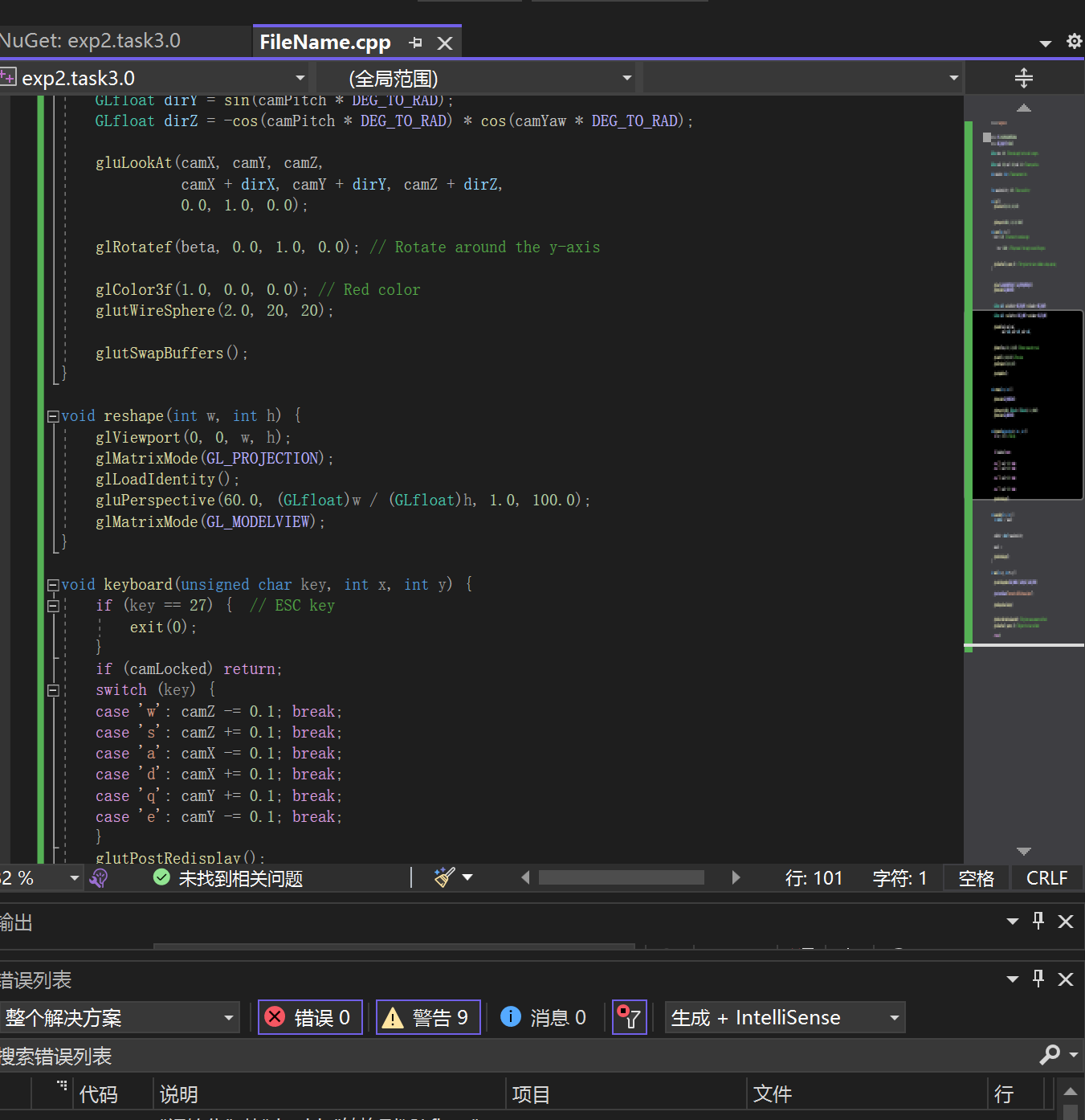
keyboard 和 mouseMove 函数允许用户使用键盘和鼠标控制相机。键盘控制包括相机的移动（WASDQE键）和锁定（L键），而鼠标移动则用来调整相机的俯仰和偏航角度，实现灵活的视角转换。

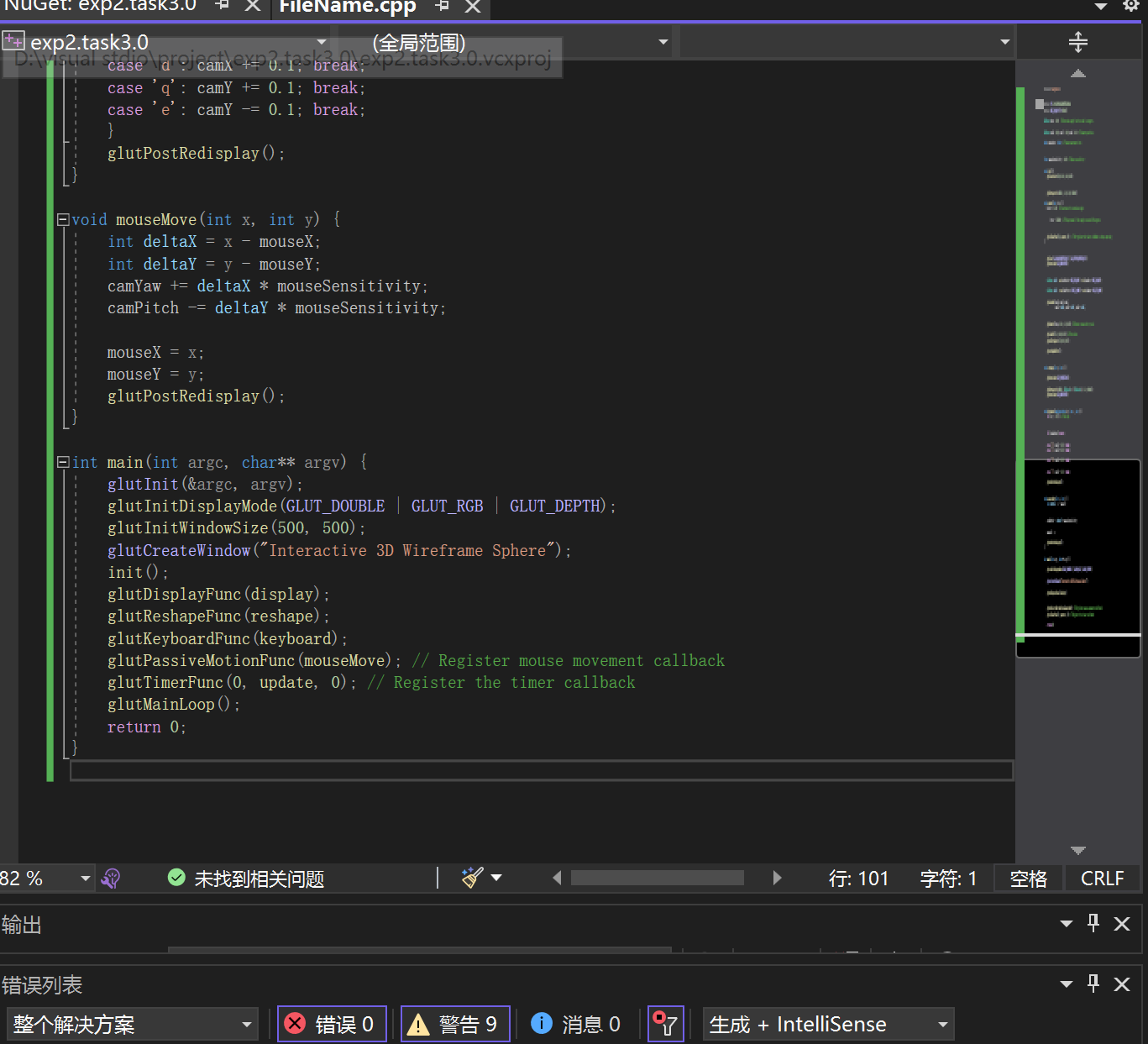
6. 主函数设置和事件循环

main 函数中，设置OpenGL显示模式、窗口大小和标题，初始化所有回调函数，并开始GLUT的事件处理循环。这个循环响应用户输入和定时事件，持续更新和渲染3D场景。

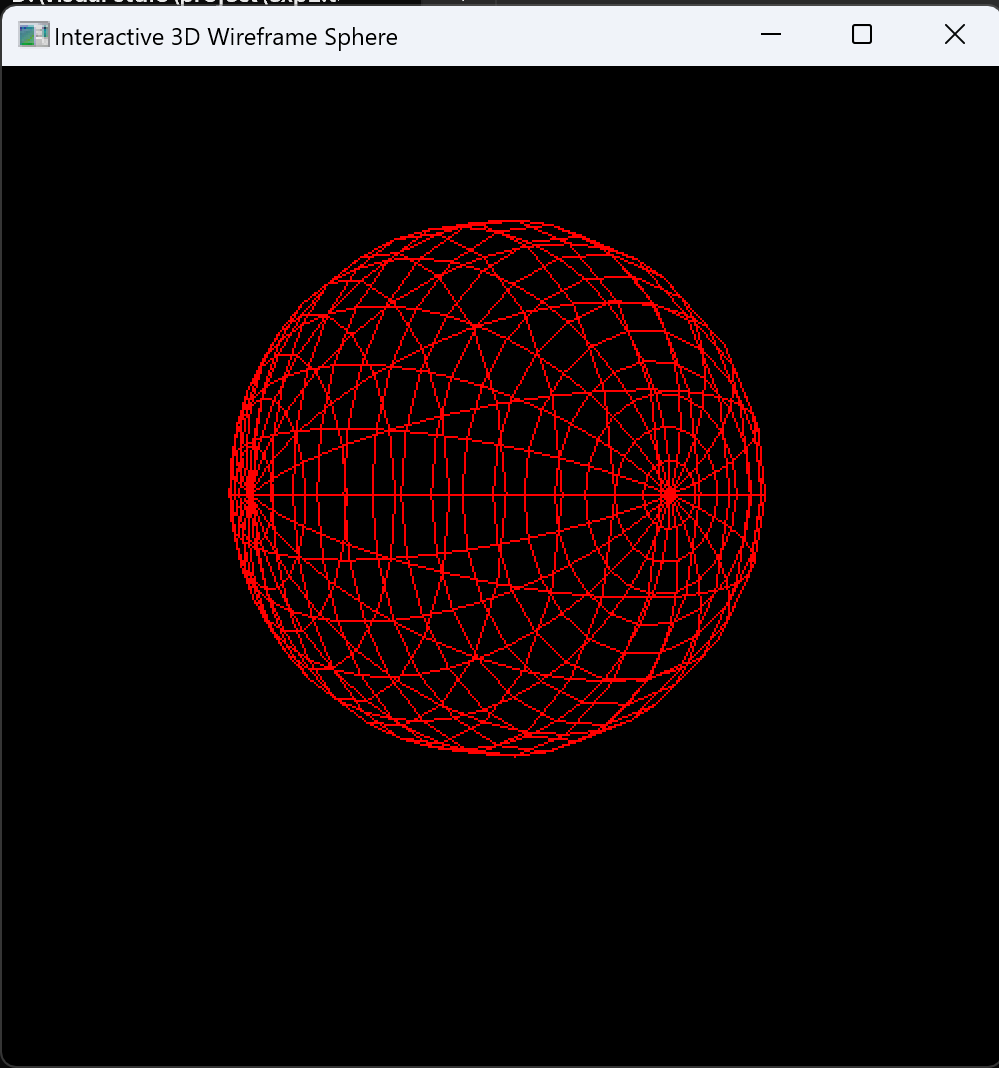
关键代码截图：



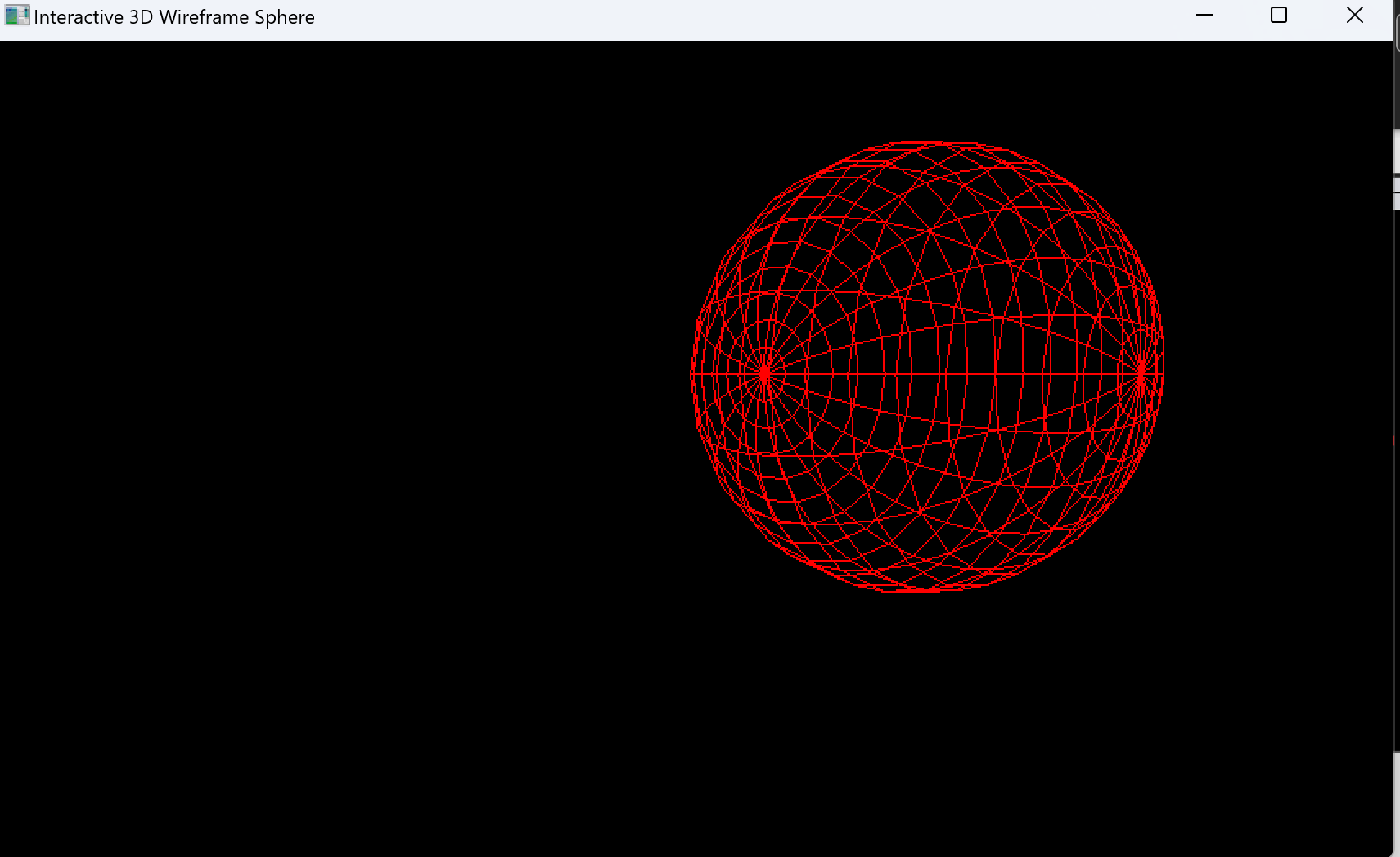




运行结果截图：



更改窗口大小不发生形变



**思考题**

**a. OpenGL中，三维空间的坐标系是怎么样的？**

OpenGL采用的是右手坐标系。在这个坐标系中：

X轴：水平向右增加。

Y轴：垂直向上增加。

Z轴：垂直于XY平面，向屏幕外为正（向观察者为负）。

在三维图形中，这种坐标系帮助定义物体的位置、旋转和缩放。

**b. OpenGL中，相机的方位是怎么样的？如何调整相机朝向呢？**

在OpenGL中，通常不直接操作“相机”，而是通过变换场景来模拟相机的移动。gluLookAt 函数是设置视图矩阵来模拟相机位置和方向的常用方法。该函数需要九个参数：

相机位置的x, y, z坐标：确定相机在世界坐标系中的位置。

视点的x, y, z坐标：相机指向的点。

上向量的x, y, z分量：定义了哪个方向是上方，通常设置为(0,1,0)，表示Y轴是上方。

通过调整这些参数，可以控制相机的位置、它所看向的方向以及确定哪个方向是上方，从而调整相机的朝向。

**c. OpenGL中，相机可见范围是如何设置的？**

OpenGL中相机的可见范围通常通过设置投影矩阵来定义，有两种主要类型的投影：

正射投影 (glOrtho)：在这种投影中，物体的大小不会随距离变化。glOrtho函数定义了一个视景体，只有在这个定义的立方体（或长方体）内的物体才可见。参数通常包括左、右、下、上、近、远平面的位置。

透视投影 (gluPerspective)：模拟人眼所见，距离越远的物体看起来越小。gluPerspective函数设置视野角度、纵横比、近平面和远平面。近平面和远平面定义了可视深度，只有在这个范围内的物体才可见。

这些设置对于构建一个真实感的3D场景非常关键，因为它们影响着用户视觉上对场景深度和空间关系的感知。调整这些参数可以控制场景的焦点，例如，通过调整近平面和远平面的距离来避免“剪切”问题，即视野太近或太远导致物体被错误地裁剪掉。

**a. 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。）**

在OpenGL中，前后面的遮挡关系是通过深度测试（Depth Test）来管理的，这涉及到了深度缓冲区（Depth Buffer，也称为Z-buffer）。深度测试是3D图形渲染中的一个核心功能，用于决定一个像素的颜色和深度值是否应该覆盖在另一个像素上面。

深度缓冲区的工作原理

深度缓冲区在每个像素点存储了一个深度值，这个值表示从相机到该像素所表示物体表面的距离。当新的几何体（如三角形）被渲染时，它的每个像素的深度值会与深度缓冲区中对应位置的值进行比较：

如果新像素的深度值小于缓冲区中存储的值（表示更靠近观察者），这个像素就会被绘制在屏幕上，同时更新缓冲区中的深度值。

如果新像素的深度值大于或等于缓冲区中的值，这个像素就会被丢弃，不会对当前视图产生影响。

这种机制确保了只有最近的表面会被绘制，从而实现了物体之间正确的遮挡关系。

开启和使用深度测试

启用深度测试：

在OpenGL中，你需要显式启用深度测试，这可以通过调用 glEnable 函数并传入 GL\_DEPTH\_TEST 常量来完成：

glEnable(GL\_DEPTH\_TEST);

清除深度缓冲区：

每次渲染新的一帧时，你需要清除深度缓冲区，以避免旧的深度信息影响新的渲染。这通常在清除颜色缓冲区时一同进行：

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

配置深度测试函数：

OpenGL允许你指定深度测试的具体行为，即决定何种情况下像素应该被覆盖。这通过 glDepthFunc 函数设置：

glDepthFunc(GL\_LESS); // 默认值，只有当新像素的深度值小于当前值时，才覆盖。

其他常用的参数包括 GL\_LEQUAL（小于或等于）、GL\_GREATER（大于）等**。**