第3章

OpenGL 编程环境介绍

3.1 Visual Studio 集成开发环境简介

集成开发环境（Integrated Development Environment，IDE）是用于提供程序开发环境的应用程序，一般包括代码编辑器、编译器、调试器和图形用户界面等工具。换言之，IDE 是集代码编写功能、分析功能、编译功能、调试功能等一体化的开发软件服务。Visual Studio 2022 是一个出色的集成开发环境，可用于构建适用于 Windows、macOS、Linux、iOS 和 Android 的丰富、精美的跨平台应用程序。Visual Studio 2022 使用诸如 WinForms、WPF、WinUI、MAUI 以及 Xamarin 等技术构建丰富的客户端应用。上述技术在 Visual Studio 2022 中均具有相应的设计器，支持用户使用丰富的工具操作和预览应用程序。基于 Visual Studio 2022 开发环境，本书采用 C 语言对计算机图形学的主要算法进行编程实现，并采用 Win32 控制台应用程序的形式展示程序运行结果。

3.2 使用 Visual Studio 2022 创建新项目

使用 Visual Studio 2022 创建新项目的操作步骤如下。

（1）新建项目，如图 3-1 所示。选择“空项目”→“空项目”，然后单击“下一步”按钮。

图 3-1 新建项目

（2）配置新项目。如图 3-2 所示，项目信息填写完毕之后单击“创建”按钮，项目名称建议用英文书写，并且遵循驼峰命名法的规则。

图 3-2　配置新项目

（3）添加新建项。配置完成后就会生成一个空的 C++ 项目，之后右键单击“源文件”→“添加”→“新建项”，如图 3-3 所示。然后选择“C++ 文件(.cpp)”创建 C 语言程序文件，输入文件名称(建议后缀名为.c，表示 C 语言程序文件)后单击“添加”按钮，这样就完成了 C 程序文件的创建，如图 3-4 所示。

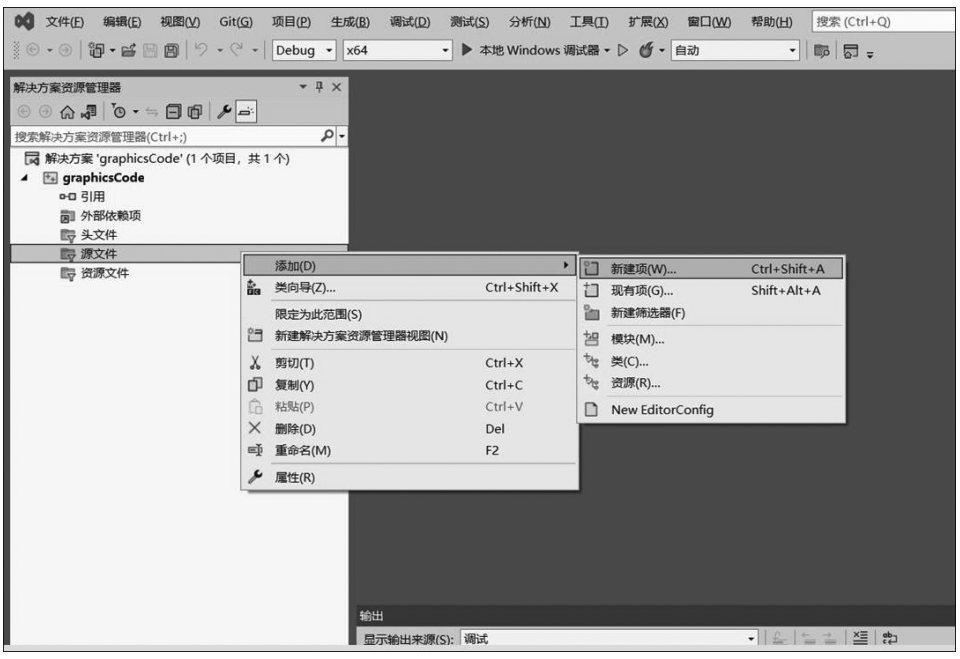

图 3-3　添加新建项

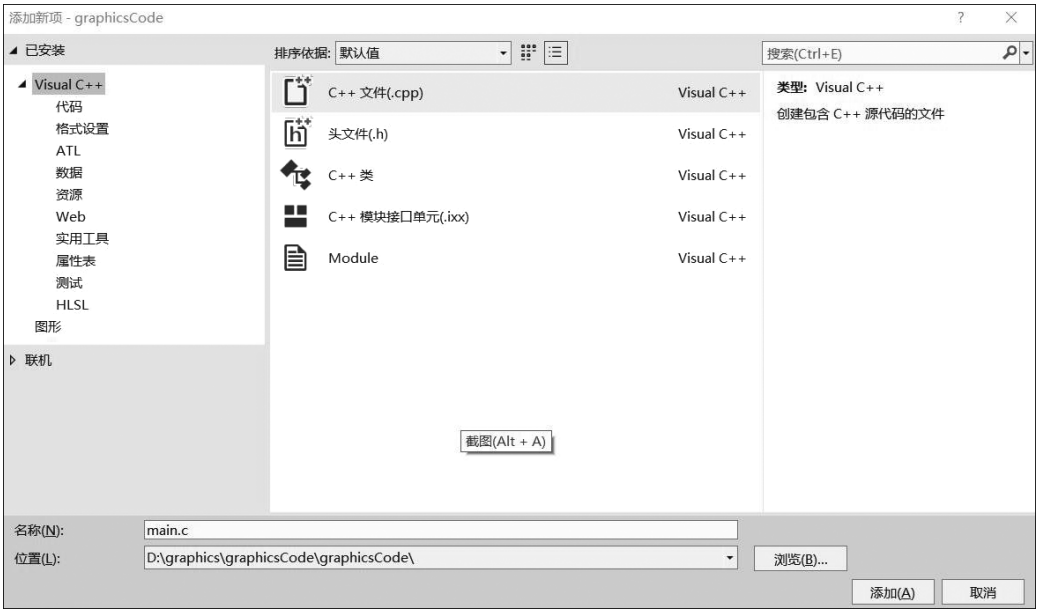

图 3-4　添加 C 语言程序文件

（4）新建完项目后需要配置项目环境。单击“项目”→“属性”→“C/C++”→“预处理器”，添加预处理器定义：_CRT_SECURE_NO_WARNINGS、_WINDOWS，最后单击“应用”按钮，如图 3-5 所示。然后选中“链接器”，将子系统更改为窗口，如图 3-6 所示。

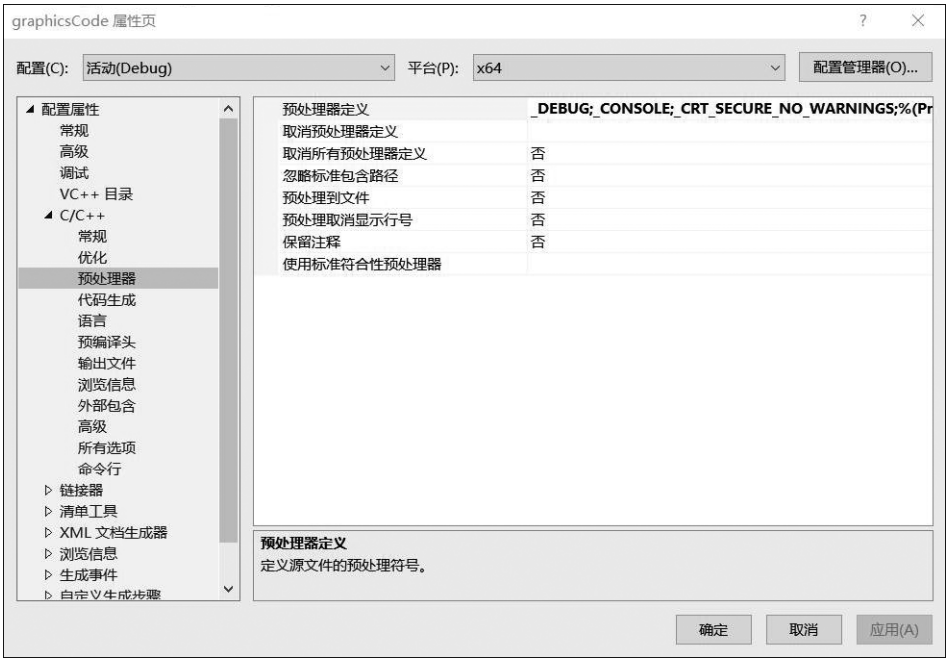

图 3-5　添加预处理器定义

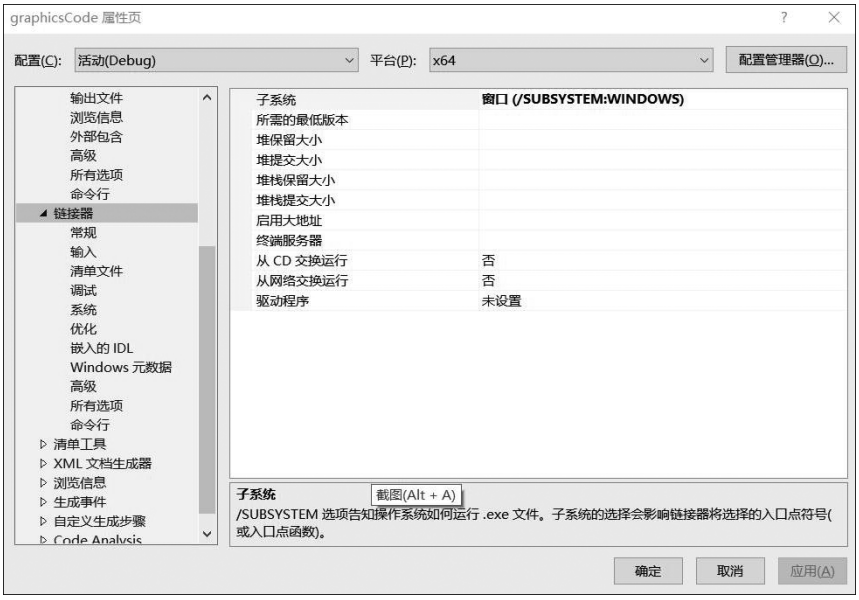

图 3-6 将子系统更改为窗口

3.3 OpenGL 简介

3.3.1 OpenGL 概述

开放图形库 OpenGL（Open Graphics Library)是一个功能强大的图形库，它实现了各种二维和三维的高级图形处理技术(如光照应用、纹理映射、透明处理等)。OpenGL 的前身是 SGI 公司为其图形工作站开发的 IRIS GL 库，由于其性能良好，该库在跨平台的移植过程中发展成为 OpenGL。由于 OpenGL 的广泛应用，它已经成为一个事实上的工业标准。

从程序开发人员的角度来看，OpenGL 是一组绘图命令的 API，利用这些 API 能够方便地描述二维和三维图形。OpenGL 的 API 集提供了物体表述、平移、旋转、缩放、光照、纹理、材质、像素、位图、文字、交互以及提高显示性能等多方面的功能，基本涵盖了开发二维、三维图形程序所需的各个方面。与一般的图形开发工具相比，OpenGL 具有如下几个突出的特点。

（1）跨平台特性。OpenGL 独立于硬件和窗口系统，在运行各种操作系统的各种计算机上都能使用，具有很高的可移植性。

（2）应用的广泛性。OpenGL 是目前最主要的二维、三维交互式图形应用程序开发环境，已经成为业界最受推荐的图形应用编程接口。OpenGL 已被广泛地应用于 CAD/CAM、三维动画、数字图像处理以及虚拟现实等领域。

（3）网络透明性。建立在客户端/服务器模式上的网络透明性是 OpenGL 的固有特性，它允许一个运行在工作站上的进程在本机或通过网络在远程工作站上显示图形。这个特点有利于均衡各图形工作站的工作负荷，共同承担图形应用任务。

（4）高质量和高性能。无论在哪个应用领域，无论是在 PC、工作站还是大型机上，

OpenGL 的高质量和高效率图形生成能力都能得到充分体现，绘制的二维、三维图形效果逼真。

（5）出色的编程特性。OpenGL 在各种平台上已经有多年的应用实践，加之严格的规范控制，因此 OpenGL 具有良好的稳定性和易使用性。

3.3.2　OpenGL 的主要功能

作为性能优越的图形应用程序设计接口（API），OpenGL 具有如下 9 大功能。

（1）模型绘制。OpenGL 能够绘制点、线和多边形。应用这些基本的形体，可以构造出大多数的三维模型。

（2）模型观察。在建立了三维景物模型后，就需要用 OpenGL 描述如何观察所建立的三维模型。观察三维模型是通过一系列的坐标变换进行的。模型的坐标变换使观察者能够在视点位置观察与视点相适应的三维模型景观。在整个三维模型的观察过程中，投影变换的类型决定观察三维模型的观察方式，不同的投影变换得到的三维模型的显示效果也是不同的。最后的视窗变换则对模型的显示结果进行裁剪缩放，即决定整个三维模型在屏幕上的图像。

（3）颜色模式的指定。OpenGL 应用了一些专门的函数来指定三维模型的颜色。程序开发者可以选择两个颜色模式，即 RGBA 模式和颜色表模式。在 RGBA 模式中，颜色直接由 RGB 值来指定；在颜色表模式中，颜色值则由颜色表中的一个颜色索引值来指定。开发者还可以选择平面着色和光滑着色两种着色方式对整个三维景观进行着色。

（4）光照应用。用 OpenGL 绘制的三维模型必须加上光照才能与客观物体更加相似。OpenGL 提供了管理 4 种光（辐射光、环境光、镜面光和漫反射光）的方法，另外还可以指定模型表面的反射特性。

（5）图像效果增强。OpenGL 提供了一系列增强三维景观的图像效果的函数，这些函数通过反走样、混合和雾化来增强图像的效果。反走样技术用于改善图像中线段图形的锯齿状现象，从而使图像更加平滑，混合用于处理模型的半透明效果，雾化使得图形从视点到远处逐渐褪色，从而使图形效果更接近于真实。

（6）位图和图像处理。OpenGL 提供了专门对位图和图像进行操作的函数。位图和图像数据均采用像素矩阵表示。

（7）纹理映射。三维景物因缺少景物的具体细节而显得不够真实，为了更加逼真地表现三维景物，OpenGL 提供了纹理映射的功能。OpenGL 提供的一系列纹理映射函数使得开发者可以十分方便地把真实图像贴到景物的多边形上，从而更加逼真地绘制三维景观。

（8）实时动画。为了获得平滑的动画效果，需要先在内存中生成下一幅图像，然后把已经生成的图像从内存复制到屏幕上，这就是 OpenGL 的双缓存技术（Double Buffer）。OpenGL 提供了双缓存技术的一系列函数。

（9）交互技术。目前有许多图形应用需要人机交互，OpenGL 提供了方便的三维图形人机交互接口，用户可以选择修改三维景观中的物体。

3.3.3 OpenGL 状态机简介

状态机(State Machine)也称为有限状态自动机,是一种计算模型,用于表示特定类型的计算过程。状态可以理解为系统所处的不同状态,状态之间可以通过指定的条件进行转移。当状态转移满足一定条件时,状态机将从一个状态转移到另一个状态,并且执行一些相关的操作。状态机具有以下几个基本元素。

(1) 状态集:有限个状态的集合,用来描述状态机可能处于的所有状态。

(2) 输入字母表:状态机接受的输入字符集合(也称为“输入事件”)。

(3) 转移函数:指定状态如何转移到下一个状态的规则及条件。

(4) 初始状态:在状态机开始处理输入前所处的特殊状态。

(5) 终止状态:满足某些条件后状态机停止处理输入并输出结果的特殊状态。

状态机被广泛应用于各种计算机领域,例如编译器、语言解析器、控制器等。开发人员可以使用状态机作为一种结构化编码方法,从而更容易地解决复杂的问题。OpenGL 状态机指一系列用于控制 OpenGL 行为的变量集合。程序员可以通过改变这些状态机变量的值来改变 OpenGL 的行为,例如设置投影矩阵、模型视图矩阵、材质属性、光照等。使用状态机的优点在于,它可以方便地应用一组参数到整个场景中的多个对象,而不必每次都手动设置每个对象的参数,从而提高了编程效率。OpenGL 状态机具有以下功能特点。

(1) 提供高度可定制性:OpenGL 状态机可以根据具体应用场景进行配置,开发人员可以通过修改各种状态变量和状态转换函数来调整渲染效果和图形输出,实现高度自定义化。

(2) 实现精细化控制:OpenGL 状态机提供了对于图形处理的精细化控制技术,这使得开发人员可以更加准确地对图形进行操作,实现更加精细化的效果展示。

(3) 强大的计算能力:OpenGL 状态机通过使用各种状态变量和状态转换操作,可以在处理大量数据时提供强大的计算能力,支持高速、大规模的图形处理操作。

(4) 支持多种编程语言:OpenGL 状态机支持多种编程语言,主流的有 C++ 、Java 等。开发人员可以使用最熟悉的编程语言进行 OpenGL 图形处理的相关操作。

(5) 简单易用:OpenGL 状态机为开发人员提供了一种简单直接的编程模型,通过使用可编程着色器和运行时参数,开发人员可以灵活控制 OpenGL 的物理实现,使其更加简单易用。

(6) 支持多种平台:OpenGL 状态机可以在不同的操作系统和硬件上运行,并且支持跨平台使用。无论在 Windows、macOS 还是 Linux 环境下,开发人员都能够方便地使用 OpenGL 状态机。

3.3.4 OpenGL 状态机原型简介

Canvas.h 头文件中主要包含颜色、绘图环境、二维坐标点、三维坐标点结构体的定义,以及各种计算机图形学函数原型的声明,函数的具体实现均在 Canvas.c 文件中完成。

在 Canvas.h 头文件中(如源程序 3-1 所示),#ifndef 全称为 if not defined,意思是如果没有定义宏 _CANVAS_H_,则执行下面的代码;如果已经定义了 _CANVAS_H_ 这个宏,则跳过这段代码。#define _CANVAS_H_ 是对宏 _CANVAS_H_ 的定义操作。一般情况

下，这种用法可以防止同一个头文件被多次包含，以避免重复定义变量、函数等问题。最后一行代码的 #endif 表示条件编译的结束。#include"Stack.h"语句引入了自定义的栈代码，相关代码将在第 6 章的源程序 6-12 中说明。

【源程序 3-1】　Canvas.h 文件源代码。

```
#ifndef _CANVAS_H_
#define _CANVAS_H_
#include<stdbool.h>
#include"Stack.h"
typedef unsigned char byte;
//定义一个结构体来存储图像中一个像素点的颜色信息
//其中颜色由 3 个分量 m_r、m_g、m_b 组成，分别表示红、绿和蓝的灰度级，范围均为 0~255，而 m_a 值则表示像素点的不透明度或者透明度
struct RGBA {
    byte m_b;
    byte m_g;
    byte m_r;
    byte m_a;
};
typedef struct RGBA RGBA;
//定义一个结构体来封装画布的各种信息
//宽度和高度属性分别记录画布的大小，绘制缓冲区则用于保存该画布渲染后的图像数据
struct Canvas {
    int m_width;
    int m_height;
    RGBA * m_buffer;
};
typedef struct Canvas Canvas;
//二维坐标点，定义一个结构体来封装二维坐标系中一个点的坐标和颜色信息
//x 和 y 属性分别记录该点在二维平面上的位置，而 RGBA 类型属性 color 则用于描述该点的颜色信息
struct Point2D {
    float x;
    float y;
    RGBA color;
};
typedef struct Point2D Point2D;
//三维坐标点，定义一个结构体来封装三维坐标系中一个点的坐标和颜色信息
//x、y、z 属性分别记录该点在三维平面上的位置，而 RGBA 类型属性 color 则用于描述该点的颜色信息
struct Point3D {
    float x;
    float y;
    float z;
    RGBA color;
};
typedef struct Point3D Point3D;
//线段边表结构体
//jx 和 dx，用于记录线段与扫描线的交点；ymin 和 ymax，记录线段在扫描线下方和上方的最小和最大纵坐标；sp，用于标记该边在所扫描的像素点中是销毁还是加入 Fill 区域
struct Bian_list {
    float jx, dx;
```

```
    int ymin, ymax;
    bool sp;
};
typedef struct Bian_list Bian_list;
//线段表中元素信息结构体
//定义长度为 10 的浮点型数组 x,用于存储线段在扫描线与线段交点处的 x 坐标;整型属性 num,表示该元素中存储的线段个数
struct Xb {
    float x[10];
    int num;
};
typedef struct Xb Xb;
//线段表结构体
//整型属性 num,记录线段数据的条数;指针数组 next,用于存储该条线段所包含的边表元素
struct Huo_list {
    int num;
    Bian_list * next[10];
};
typedef struct Huo_list Huo_list;
//枚举类型来定义一个包含两个常量的集合,用于指示状态机中当前操作的矩阵模式
//GT_MODELVIEW 和 GT_PROJECTION 分别代表"模型视图矩阵"和"投影矩阵"
enum MATRIX_MODE {
    GT_MODELVIEW = 0,
    GT_PROJECTION = 1
};
//三维变换状态机
struct Statement {
    enum MATRIX_MODE m_matrixMode;        //矩阵模式
    float m_modelViewMatrix[4][4];        //模型视图矩阵
    float m_projMatrix[4][4];             //投影矩阵
};
typedef struct Statement Statement;
//设置矩阵模式 modelView 或者 projection(当前操作的矩阵模式,三维变换状态机)
void glMatrixMode(enum MATRIX_MODE mode, Statement * m_state);
//用传进来的矩阵代替当前的矩阵(4×4 矩阵,三维变换状态机)
void glLoadMatrix(float matrix[4][4], Statement * m_state);
//将系统设置的矩阵更改为单位阵(三维变换状态机)
void glLoadIdentity(Statement * m_state);
//右乘矩阵(4×4 矩阵,三维变换状态机)
void glMultiMatrix(float matrix[4][4], Statement * m_state);
//向量和矩阵相乘(顶点数组,顶点数,三维变换状态机)
void glMultiMatrixVec(Point3D tp3[], int npt, Statement * m_state);
//初始化给定的链表 H(线段表链表指针)
void InitiateHuo_list(Huo_list * H);
//将给定的变量链表节点 b_list 插入链表 H 的末尾(线段表指针,线段边表指针)
void InsertHuo_list(Huo_list * H, Bian_list * b_list);
//删除链表 H 中第 j 个节点(线段表指针,整型 j)
void Deleteb_list(Huo_list * H, int j);
```

```
//对给定的结构体指针 xb 指向的数据中,按照元素值从小到大的顺序进行冒泡排序(线段表中元素信息结构体指针)
void pai_xuHuo_list(Xb * xb);
//画点操作(点坐标,画布),该函数用于在画布上绘制一个点,接收两个参数: Point2D 结构体和 Canvas 结构体
void drawPoint2D(Point2D _pt, Canvas _canvas);
//画线操作(DDA 算法)(起点坐标,终点坐标,颜色,Canvas 结构体),使用 DDA 算法在给定的画布_canvas 上以颜色_color 绘制从点 pt1 到点 pt2 的直线
void drawLineDDA(Point2D pt1, Point2D pt2, RGBA _color, Canvas _canvas);
//在给定的画布_canvas 上以颜色_color 和水平线粗度 nwidth 在点 pt 处绘制一条横向线段(点坐标,线宽,颜色,Canvas 结构体)
void horilineBrushLine(Point2D pt, int nwidth, RGBA _color, Canvas _canvas);
//在给定的画布_canvas 上以颜色_color 和竖直线粗度 nwidth 在点 pt 处绘制一条纵向线段(点坐标,线宽,颜色,Canvas 结构体)
void vertlineBrushLine(Point2D pt, int nwidth, RGBA _color, Canvas _canvas);
//指定线宽画线操作(起点坐标,终点坐标,颜色,Canvas 结构体),在给定的画布_canvas 上以颜色_color 绘制一个线宽为 nwidth 的直线,起点为 pt1,终点为 pt2
void drawLineByWidth(Point2D pt1, Point2D pt2, int nwidth, RGBA _color, Canvas _canvas);
//画线操作(中点画线算法)(起点坐标,终点坐标,颜色,画布),使用中点画线算法绘制直线
```

参数说明如下。

pt1、pt2：直线的两个端点坐标,结构体类型为 Point2D。

_color：绘制直线的颜色,结构体类型为 RGBA。

_canvas：绘制直线的画布,结构体类型为 Canvas。

```
void drawLineMid(Point2D pt1, Point2D pt2, RGBA _color, Canvas _canvas);
//画线操作(Bresenham 算法)(起点坐标,终点坐标,颜色,Canvas 结构体),把给定的两个点(pt1 和 pt2)之间的直线画在画布上,使用 Bresenham 画线算法。
void drawLineBresenham(Point2D pt1, Point2D pt2, RGBA _color, Canvas _canvas);
//画圆算法: 中点画圆(圆心坐标,半径,颜色,Canvas 结构体)
void drawCircleMid(Point2D pt, int r, RGBA _color, Canvas _canvas);
//画圆线宽函数(点坐标,线宽,颜色,Canvas 结构体): 在给定点 pt 附近画一个宽度为 nwidth 的垂直于坐标轴的圆
void vertlineBrushCircle(Point2D pt, int nwidth, RGBA _color, Canvas _canvas);
//画圆算法: 指定线宽中点画圆(圆心坐标,半径,线宽,颜色,Canvas 结构体)
void drawCircleMidByWidth(Point2D pt, int r, int nwidth, RGBA _color, Canvas _canvas);
//Bresenham 绘制圆形(圆心坐标,半径,颜色,Canvas 结构体)
void drawCircleBresenham(Point2D pt, int r, RGBA _color, Canvas _canvas);
//画二维直角坐标系(Canvas 结构体),坐标原点在(300,300)
void drawCCS(Canvas _canvas);
//画三维直角坐标系(Canvas 结构体),坐标原点在(300,300)
void draw3D(Canvas _canvas);
//绘制画单个三角形(三角形的顶点数组,Canvas 结构体)
void drawTriangleSingle(Point2D shape[3], Canvas _canvas);
//绘制单个四边形(四边形的顶点数组,Canvas 结构体)
void drawRectangleSingle(Point2D shape[4], Canvas _canvas);
//绘制三维图形(顶点数组,顶点数,Canvas 结构体)
void draw3DGraphics(Point3D tp3[], int npt, Canvas _canvas);
```

```
//3×3 矩阵相乘(数组 1,数组 2,数组 3,参与矩阵计算的点数)
void multiplyMatrix2D(float a[3][3], float b[3][3], float c[3][3], int npt);
//4×4 矩阵相乘(数组 1,数组 2,数组 3,参与矩阵计算的点数)
void multiplyMatrix3D(float a[4][4], float b[4][4], float c[4][4], int npt);
//二维三角形的平移(三角形各顶点的坐标数组,数组长度,x 轴平移距离,y 轴平移距离,Canvas 结构体)
void translate2D(Point2D ptArray[], int arrayLength, float xs, float ys, Canvas _canvas);
//三维图形的平移(三维图形各顶点的坐标数组,数组长度,x 轴平移距离,y 轴平移距离,z 轴平移距离)
void translate3D(Point3D tp3[], int npt, float deltax, float deltay, float deltaz);
//二维三角形的旋转(三角形各顶点的坐标数组,数组长度,角度,Canvas 结构体)
void rotate2D(Point2D ptArray[], int arrayLength, float jd, Canvas _canvas);
//三维绕 x 轴旋转调用(顶点数组,数组长度,角度)
void rotate3DByX(Point3D tp3[], int npt, float jd);
//三维绕 y 轴旋转调用(顶点数组,数组长度,角度)
void rotate3DByY(Point3D tp3[], int npt, float jd);
//三维绕 z 轴旋转调用(顶点数组,数组长度,角度)
void rotate3DByZ(Point3D tp3[], int npt, float jd);
//二维三角形的缩放(三角形各顶点的坐标数组,数组长度,x 轴缩放,y 轴缩放,Canvas 结构体)
void scale2D(Point2D ptArray[], int arrayLength, float xs, float ys, Canvas _canvas);
//三维变比调用(顶点数组,数组长度,x 轴缩放,y 轴缩放,z 轴缩放)
void scale3D(Point3D tp3[], int npt, float deltax, float deltay, float deltaz);
//二维三角形的对称(三角形各顶点的坐标数组,数组长度,a,d,b,e,Canvas 结构体)
void reflect2D(Point2D ptArray[], int arrayLength, float a, float d, float b, float e, Canvas
_canvas);
//三维反射调用(顶点数组,数组长度,x 轴反射、y 轴反射、z 轴反射)
void reflect3D(Point3D tp3[], int npt, float deltax, float deltay, float deltaz);
//二维三角形的错切(三角形各顶点的坐标数组,数组长度,x 轴错切因子,y 轴错切因子,Canvas 结构体)
void shear2D(Point2D ptArray[], int arrayLength, float deltax, float deltay, Canvas _canvas);
//三维错切调用(沿 x 轴方向(x1,x2),沿 y 轴方向(y1,y2),沿 z 轴方向(z1,z2))
void shear3D(Point3D tp3[], int npt, float x1, float x2, float y1, float y2, float z1, float
z2);
/* 使用 Cohen-Sutherland 算法对点(x,y)的区域编码做二进制表示(待编码点的 x 坐标,待编码点的 y 坐
标,表示裁剪窗口的左右下上 4 个边界值(XL、XR、YB、YT),用于表示裁剪区域的左侧面,用于表示裁剪区域的右
侧面,用于表示裁剪区域的底部面,用于表示裁剪区域的顶部面,存储编码结果的指针变量) */
void encode(int x, int y, int XL, int XR, int YB, int YT, int LEFT, int RIGHT, int BOTTOM, int
TOP, int *code);
/* Cohen-Sutherland 裁剪算法,其中 x1, y1,x2, y2 分别为线段的起点、终点的坐标值;XL、XR、YB、YT 表示
裁剪窗口的左右下上 4 个边界值;LEFT 表示裁剪区域的左侧面;RIGHT 表示裁剪区域的右侧面;BOTTOM 表示裁
剪区域的底部面;TOP 表示裁剪区域的顶部面 */
void CSline(int *x1, int *y1, int *x2, int *y2, int XL, int XR, int YB, int YT, int LEFT, int
RIGHT, int BOTTOM, int TOP);
/* Cohen-Sutherland 裁剪绘画,对线段进行裁剪之后绘制在画布上,并保留视口边界需要保留的部分,其中
x1, y1,x2, y2 分别为线段的起点、终点的坐标值;XL、XR、YB、YT 表示裁剪窗口的左右下上 4 个边界值,LEFT 表
示裁剪区域的左侧面;RIGHT 表示裁剪区域的右侧面;BOTTOM 表示裁剪区域的底部面;TOP 表示裁剪区域的顶
部面 */
void CSlineDraw(int *x1, int *y1, int *x2, int *y2, int XL, int XR, int YB, int YT, int
LEFT, int RIGHT, int BOTTOM, int TOP, Canvas _canvas);
//利用顶点坐标数组绘制多边形(图形数组,数组长度,Canvas 结构体)
void drawPoly(Point2D vlist[], int n, Canvas _canvas);
```

```
//计算点位置关系码 (Sutherland-Hodgeman 和 Cohen-Sutherland 共用) ,x 和 y 分别表示当前待计算点的横纵坐标,border 用于存储 4 条边界线的横纵坐标信息
int calculateCode(float x, float y, float border[]);
//Sutherland-Hodgeman 多边形裁剪算法(边缘数组,待裁剪的顶点数组,数组长度,Canvas 结构体)
void sutherlandHodgeman(float border[], Point2D sp[], int spLength, Canvas _canvas);
//LiangBarsky 算法(点 1,点 2 ,x 左限制,x 右限制,y 上限制,y 下限制,颜色,Canvas 结构体指针)
void LiangBarsky(Point2D pt1, Point2D pt2, int XL, int XR, int YT, int YB, RGBA _color, Canvas * _canvas);
//使用扫描线算法在画布上绘制一个多边形,接收 3 个参数: Point3D 结构体数组指针、点的数量 num_point、Canvas 结构体
void ScanLine(Point3D * Polygon_point, int num_point, Canvas _canvas);
#endif
```

3.3.5　Windows 编程简介

Windows 桌面编程指开发针对 Microsoft Windows 操作系统的桌面应用程序的过程。这类运行在 Windows 平台上的程序可以用于满足用户个人或工作需求。常见的 Windows 桌面应用程序包括文本编辑器、音乐播放器、图像处理软件、办公套件等。主要的 Windows 桌面编程技术包括以下几种。

(1) **Win32 API**。Win32 API 是一组 API 或函数库,可用于编写 Windows 桌面应用程序。它提供了许多基础控件如对话框、窗口和菜单等,以及处理消息和事件相关函数。Win32 API 提供了通过 C/C++ 编写 Windows 桌面应用程序的底层方法。

(2) **.NET 框架**。.NET 是一个跨平台的应用程序框架。它支持多种编程语言(如 C#、VB.NET 等),为 Windows 桌面应用程序提供了一种高级且快速的开发方式。.NET 框架包含了广泛的类库和工具平台,可以大大简化 Windows 应用程序的开发过程。

(3) **WPF**。WPF 是由 Microsoft 公司开发的一种新型 Windows 桌面应用程序开发平台。WPF 使用 XAML(扩展应用程序标记语言)作为用户界面的描述语言,包含了许多基础控件和各种动画效果,可以实现非常炫酷的用户交互体验。

(4) **UWP**。UWP 是一种新型的 Windows 平台应用程序开发平台。它支持多种设备(如 PC、手机等),并且具有自适应布局和优化性能等特点。UWP 通过使用 XAML 进行界面设计,并结合 C#、C++ 、VB.NET 等代码实现逻辑控制。

总之,Windows 桌面编程技术非常多样化,开发人员可以根据自己的需求选择最适合自己的开发方式。无论是 Win32 API 还是 WPF,都提供了强大的功能和便利的编程模型,使得 Windows 桌面应用程序的开发变得更加容易。

在 main.c 文件中引入 windows.h 头文件,使用 Win32 API 进行 Windows 窗口编程,创建窗口程序,相关计算机图形学代码都将在 Render() 函数中进行调用。源程序 3-2 为 main.c 文件中初始的代码。

【源程序 3-2】 main.c 文件代码。

```
#include <windows.h>                    //窗口相关的头文件
#include"Canvas.h"                      //自定义画布结构体的头文件
#include <math.h>                       //数学计算相关的头文件
#define pi 3.1415926535898              //预定义 π 的值
```

```
#define DEC (pi/180)                    //弧度转换为角度的系数
LRESULT CALLBACK MyProc(HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam);
                                        //声明消息回调函数

HWND hWnd;                              //定义句柄用来保存成功创建窗口后返回的句柄
int wWidth = 800;                       //窗口宽度
int wHeight = 600;                      //窗口高度
HDC hDC;                                //设备描述上下文,显示器取数据的地方,一块内存
HDC hMem;                               //绘图的 DC,再复制到 hDC 中,双缓存
void * buffer = 0;
struct Canvas _canvas;                  //定义画布结构体

void Render();                          //绘图函数声明

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int
nShowCmd)
{
    MSG msg;                            //定义消息结构体变量
    static TCHAR szAppName[] = TEXT("计算机图形学");    //定义窗口类名称
    WNDCLASS wndclass;                                  //指定窗体类
    wndclass.style = CS_HREDRAW | CS_VREDRAW;           //指定窗口风格
    wndclass.lpfnWndProc = MyProc;                      //指向处理窗口消息的函数入口的函数指针
    wndclass.hInstance = hInstance;                     //设置窗口进程的实例句柄
    wndclass.hIcon = LoadIcon(NULL, IDI_APPLICATION); //设置窗口的图标为 NULL
    wndclass.hCursor = LoadCursor(NULL, IDC_ARROW);   //设置窗口的光标为 NULL
    wndclass.hbrBackground = (HBRUSH) GetStockObject(WHITE_BRUSH);  //设置窗口的背景为白色
    wndclass.lpszMenuName = NULL;           //不需要设置菜单名称,将其设置为空
    wndclass.lpszClassName = szAppName;  //将自定义的固定字符数组 szAppName 作为新窗口的类名
    //注册窗体类,如果失败直接返回 0 结束程序
    if(!RegisterClass(&wndclass))
    {
        MessageBox(NULL, TEXT("error"), TEXT("title"), MB_ICONERROR);
        return 0;
    }
    hWnd = CreateWindow(szAppName,
                        TEXT("计算机图形学"),
                        WS_POPUP,
                        CW_USEDEFAULT,
                        CW_USEDEFAULT,
                        wWidth,
                        wHeight,
                        NULL,
                        NULL,
                        hInstance,
                        NULL
    ); //创建窗体
    ShowWindow(hWnd, nShowCmd);             //显示窗体的 API 传入需要显示的窗体句柄和显示方式
    UpdateWindow(hWnd);                     //刷新窗体的 API
```

```
    //创建绘图所需的位图
    hDC = GetDC(hWnd);
    hMem = CreateCompatibleDC(hDC);

    BITMAPINFO bmpInfo;
    bmpInfo.bmiHeader.biSize = sizeof(BITMAPINFOHEADER); //指定结构长度
    bmpInfo.bmiHeader.biWidth = wWidth;                  //指定图形的宽度
    bmpInfo.bmiHeader.biHeight = wHeight;                //指定图像的高度
    bmpInfo.bmiHeader.biPlanes = 1;                      //指定目标设备的平面数
    bmpInfo.bmiHeader.biBitCount = 32;            //指定表示颜色时要用到的位数
    bmpInfo.bmiHeader.biCompression = BI_RGB;     //指定位图是否压缩,实际存储方式为 bgr

    //创建画布
    HBITMAP hBmp = CreateDIBSection(hDC, &bmpInfo, DIB_RGB_COLORS, (void **) &buffer, 0, 0);
                                                  //在这里设置 buffer 的大小
    SelectObject(hMem, hBmp);
    memset(buffer, 0, wWidth * wHeight * 4);      //清空 buffer 为 0

    //初始化画布
    _canvas.m_width = wWidth;
    _canvas.m_height = wHeight;
    _canvas.m_buffer = buffer;

    Render();                                     //绘图函数
    while(GetMessage(&msg, hWnd, 0, 0))
    {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }
    return msg.wParam;
}

LRESULT CALLBACK MyProc(HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{

    switch (message)
    {
        case WM_DESTROY:
            PostQuitMessage(0);
            return 0;
    }
    return DefWindowProc(hwnd, message, wParam, lParam);
}

//绘图函数
void Render()
{
    //每次绘图前做清理
    if(_canvas.m_buffer != NULL)
```

```
        {
            memset(_canvas.m_buffer, 0, sizeof(struct RGBA) * _canvas.m_width * _canvas.m_height);
        }

        //在这里调用后续各种的图形学相关算法函数

        BitBlt(hDC, 0, 0, wWidth, wHeight, hMem, 0, 0, SRCCOPY); //将绘制结果复制到设备上下文
    }
```

习　　题

3-1　OpenGL 有什么特点？

3-2　OpenGL 有什么功能？