第四章 简单 CAD 绘图系统开发实例

计算机的飞速发展，使计算机已经不再局限于解决数值计算问题，而更多更广泛地使用在非数值计算的各领域。计算机所处理的对象也由纯粹的数值数据发展到诸如字符、图像、声音、信号等各种各样具有一定结构的数据。计算机是一种信息处理装置，而计算机又只能处理数据信息，不能

识别图形。用计算机绘图，就要将图形转化为数据，通过计算机对图形数据进行加工处理，绘制图形。

一个图形系统是有软件和硬件组成的，它对于计算机辅助设计十分重要，它能帮助人们提高工作效率和质量，简化复杂的工作。Visual C++自问世以来一直是 Windows 环境下最主要的应用程序开发系统，由于 Windows 是基于 GUI 的操作系统，而 Visual C++提供了丰富的图形接口（GDI）函数，使得用 Visual C++开发的 Windows 系统下的图形应用程序很方便、简

单。

本章介绍用 Visual C++开发图形应用程序的一些知识，包括：计算机图形学绘图基础、图形的数据结构，最后介绍一个简单 CAD 绘图系统编程实例。由于篇幅所限，本章节在对简单 CAD 绘图系统编程实例的介绍中，在相应位置，有针对性地放置相关部分代码，完整程序代码请参见随本书提供的光盘。

4.1 计算机图形学绘图基础

Windows 应用程序提供一个一般应用程序所需要的全面面向对象软件组建的集成集合。这里应用程序框架是一类库的超级集合，它定义了程序的结构。一个类库是可在应用程序中使用的有关 C++类的集合。例如一个图形类可以支持一般的图形操作。使用类库可通过构建已有类库对象，或派生自己的类来实现。MFC(Microsoft Foundation Class)库是一整套简化 Windows 编程的可重用的类库，MFC 提供如 CString、CFile 等 Windows 编程常用类，封装了通用 Windows API 和数据结构的类。这些数据包括窗口、

控件和设备环境。除此之外，MFC 还提供了应用程序框架，包括组成应用程序继承结构的类，以及对各种 Windows 特性的全面支持。

当我们开发一个大型项目时，同时开发一种结构，但每个人程序的结构往往是不相同的，MFC 库应用程序框架包含自己的应用程序结构，使用 MFC 库编写 Windows 程序，就可以保持值这种一致性，有利于代码的维护和增强。使用 MFC 类库，程序可在任何时候调用 Win32 函数，可以最大程度地利用 Windows。

4.1.1 Visual C++开发系统基本绘图知识

Windows 应用程序框架的核心是文档与视的相互关系，应用程序框架中的文档和视图结构文档即应用程序处理的数据对象，如 NotePad 中的文本。视即指文档的应用程序中的表现方式或用户用于改变文档的交互窗口对象。MFC 中与绘图有关系的几个关键类如下：

（1）文档类（Document）

文档是用户处理数据的对象。文档即应用程序处理的数据对象，文档一

般从 MFC 中类 CDocument 中派生，如果支持 OLE 功能，可从 ColeDocument 或 ColeServerDoc 类中派生，由 CDocument 派生的类主要用于存储数据。CDocument 类用于相应数据文件的读取以及存储 Cview 类所需要观察和处理的信息。

（2）视类（View）

视相当于文档在应用程序中的观察窗口，它确定了用户对文档的观察方式和用户编辑文档的方式。如果需要，用户可在多个视中对文档进行操作，

即多文档结构（MDI）。对于图形来说视就好比我们进行绘图工作的画布，对图形的操作都是在视上进行的。因此掌握好视类对我们很重要。

一般情况下，应用程序的视类从 CView 中派生，对于有特殊要求的视，根据情况不同，还可从类 CScrollView、CEditView、CFormCView、CTreeView、CListView 或 CRichView 等派生。

另外，视类中有一个重要的成员函数——OnDraw()函数，应用程序中，几乎所有“画”的动作都出现在 OnDraw()中或有它来引发。该函数必须被

重载。重载的 OnDraw()函数要完成两件事，即调用相应的文档的函数获取文档数据和调用 GDI（图形设备接口）的函数在视中画出文档数据。

（3）主窗口类（Main Frame Window）

主窗口是 Windows 应用程序中限定其所有窗口范围的最外边框。应用程序中的所用其它窗口都直接或间接地为主窗口的子窗口，如标准菜单、工具条、状态条等。对于 MDI 程序，视占文档窗口的客户区，而文档窗口又是主窗口的子窗口，一个主窗口可以有多个文档窗口（即含有多个视）；对

于 SDI 应用程序，视是在主窗口中显示的，视占据了主窗口的客户区，主窗口也是文档窗口。

一个 Windows 应用程序一般具有主窗口类。SDI 应用程序的主窗口类应从 CframeWnd 中派生，MDI 程序的主窗口类应从 CMDIFrameWnd 中派生（文档窗口类应当从 CMDIFrameWnd 中派生，一种文档类型应当有一个 CMDIChildWnd 文档窗口派生类）。

（4）文档模板类（Document Template）

文档类模板用于协调文档对象、视对象、和主窗口对象的创建过程。它是从类 CDocTemplate 或其派生类中派生的。一个文档模板可以管理同一文档类型的所有文档。对于不同的类型的文档，必须使用不同的文档模板类。

对于 SDI 程序，只包含一个文档模板，它是从 CsingDocTeplate 中派生的。对于 MDI 应用程序，所有的文档应从 CmultiDocTeplate 中派生的。

（5）应用类（Application）

一个应用程序由切只有一个应用类的对象，它控制上述所有的对象，确

定程序的特点，并负责应用程序的初始化和清楚，以便创建和管理程序支持的文档模板对象，一个应用程序对象就代表一个应用程序，当用户启动应用程序，Windows 调用应用程序框架内置的 WinMain 函数，并且 WinMain 寻找一个由 CWinApp 派生的全局构造的应用程序对象，全局对象在应用程序之前构造。

（6）图形设备接口

PC 相容机种上可以连接许多种不同的视讯设备，所以，GDI 的主要目的

之一是支援与装置无关的图形。Windows 程式应该能够毫无困难地在 Windows 支援的任意一种图形输出设备上执行，GDI 通过将您的程式和不同输出设备的特性隔离开来的方法来达到这一目的。图形设备接口 GDI（Graphic Device Interface）管理 Windows 应用程序在窗口中的所有绘图操作和与此有关的诸多方面、。如图形设备的信息，坐标系和映射模式、绘图的当前状态（画笔、画刷、颜色、字体等）、绘图的具体操作（如画线、画圆等）。

一个 Windows 图形设备接口对象类型由一个 MFC 类库表示，这些类有一个共同的抽象基类：CGdiObject。一个 Windows 图形设备接口对象由 CGdiObject 派生类的 C++对象来表示，这些对象有：

CBitmap　　位图对象

CBrush　　画刷。用于表示区域填充的颜色和样式。

Cpen　　画笔用于指定线和边框的性质，如颜色、线宽、线性等

CRgn　　区域是由多个多边形和椭圆组成的组合形状，可以填充、裁剪等操作以及判断鼠标是否位于某一点。

CFont　　字体是具有定大小和风格的一套字符集。

CPalette　　调色板是一字符映射表，将逻辑颜色和设备的实际颜色相互联系。

（7）设备环境类

CDC 是 MFC 中最重要的类之一，更是绘图应用程序中最重要的类，CDC

类定义的是设备上下文对象的类。CDC 对象提供处理显示器或打印机等设备上下文的成员函数以及处理与客户区对应的显示上下文的成员，通过 CDC 类的成员函数进行所有的绘图，它封装了 GDI 的大部分内容，如坐标系、绘图方法、各种 GDI 对象的管理等。

设备环境类 CDC 的内容十分丰富，包含了和绘图有关的方方面面。CDC 类提供的成员函数可以用于对设备环境的操作、绘图工具的使用、图形设备接口（GDI）对象的选择等，但在使用 CDC 类对象视时应注意一个问题：

为使用 CDC 对象，须先构造一个 CDC 对象，然后才能调用它的成员函数。使用完成后，必须在适当的方将其删除，在 Windows 环境中可获得的设备环境的数量是有限的。如果太多的 CDC 对象没有被删除，计算机的资源将很快被耗尽，VC++也会在调试窗口中报错。

为了特殊应用，MFC 提供了几个 CDC 类的派生类。CpaintDC 类封装了对 BeginPaint 和 EndPaint 的调用，CclientDC 类管理与窗口客户区对应的显示场景。CwindowsDC 类管理与包括主框架和控件在内的设备环境与源文

件。对 CDC 中类的掌握程度直接影响我们绘图程序的质量和效率。

框架大部分的代码是应用程序自动生成的，程序设计的大部分工作是在视类的两个文件（.h 头文件和.cpp 应用文件）中进行的。

4.1.2 计算机图形学会图系统设计基本原则

设计一个图形系统是一个复杂的工作，设计工作一般分为需求分析、设计、编程、测试和运算等几个阶段，每一阶段各有明确任务。

对于一个交互式通用图形系统的设计原则大致如下：

1. 结构层次化、模块化；
2. 要通用性强，使用方便；
3. 处理速度快；
4. 程序易读、查、改以及移植和扩充。

4.1.3 图形程序设计步骤

编写绘图程序，其基本要求是可靠、正确和简单清晰。同时再满足结构程序设计模块化要求的前提下，尽可能做到程序结构合理，占用内存单元

少，程序运行速度快。另外编出的程序应该满足可读性强、修改调试方便、通用性好，移植性强等要求。编写一个绘图程序，一般需经过以下步骤：

（1）根据实际需要，确定绘图程序的功能和用途。

设计或编写绘图程序时，首先要明确所编写的绘制程序应具有的功能或说明用途是什么将有利于绘图程序设计具体工作的进行。绘图程序的功能和用途将决定具体编程中下面的一些问题，设置调用函数和参数个数、绘图方式的选择、用户与程序交互信息方式、确定数据存储结构。

（2）明确所绘制图形的几何形成过程。

在绘制一个图形时，有时需要分析清楚它的几何关系，即弄明白这个图形是怎样形成的，然后才能在此基础上得到绘图的算法，研究图形的几何关系，对于绘制复杂图形是大有帮助的，进而自己也可构造图形。

（3）根据图形绘制要求对所绘制图形进行系统的模块化分。

算法，即运算方法，它还包括对图形数据（如点的坐标等）的加减乘除四则运算，以及对数据进行的廉洁、变换、循环等操作。绘制一个图形，

一般都要写出绘制算法，最常用的是关于图形上数据的计算公式和各种变量、参数应满足的关系式。计算机通过计算式和关系式等对图形数据进行处理，最后才可绘制出图形。图形算法中体现的严格的数学表示，正确合理的算法式绘图程序能够得到正确结果的前提。

（4）针对各个模块的功能确定相应绘图算法。

程序总体上为模块化结构，即程序开始为数据说明，然后为各功能函数，最后主程序提基本上是由调用函数语句组成，而这些函数语句实际就是一

些功能模块，因此整个程序是模块化结构。

（5）编写绘图程序。

写程序时，语句要对齐。语句套语句，内套语句应该往后移 3 或 4 个字符，复合语句应与相应的列对齐。同时对变量或参数，要加注释，对函数功能，要做功能简述，做到程序的条理清晰，可读性好。编写的程序还要便于调试。程序每次调试一句，用分号分开的各程序语句，虽然 C++允许写在同一行，但不利于调试，若一个程序句一行，可对错误语句准确定位。

将程序分解成合理的模块，即分割成几个甚至很多简单函数，每一个简单函数实现一个单独功能，把各功能模块分别调试好，再组装进程序。同时要更多使用参数，少使用全局变量，

（6）上机调试、运行和绘图。

程序编号后上机调试。调试程序一般要注意下面几点：

对写好的程序，首先要在纸上进行认真仔细的检查，不要匆匆忙忙上机，程序通过认真检查后再上机调试、编译，初学者容易忽略这一点，在计算

机上占用大量时间。

编译时，可根据提示的信息，找出程序中错误的程序段的位置及错误类型并加以改更。注意有时提示的错误信息位置有误，提示的出错类型也并非绝对准确，也有可能因为错误较多而相互关联，所以要善于分析，真正找到错误所在。当遇到提示的错误信息很多(一大片)，往往使人感到问题很严重，事实上也许只是因为所使用的变量未定义，编译时会对所有使用这一变量的语句发出出错信息，这要添加相应变量的定义，所有错误可能都

会消除，另一方面也不要轻易认为程序运行输出一个正确结果就没有问题了。例如，**if** 语句有多个分支，可能在流程经过一个分支时是正确的，而经过另一个分支时可能会出错，因此要考虑全面。

4.1.4 在 Visual C++集成开发环境下程序的调试

（1）运行要调试的程序；

（2）利用功能键 F9 在需要行设置断点（同样在已设置断点的行上按住 F9 可取消断点）；

（3）可用功能键 F11 一次执行一条语句，一步步跟踪；可以跟踪进某行中调用的函数，如果不想跟踪进函数，则用功能键 F10。

打开 Watch 窗口（按 Alt+3 组合键）或 QuickWatch 窗口（按 Shift+F9 组合键），也可打开运算窗口或通过主菜单中的 Debug 菜单的选项来打开，观察已经执行过的一段程序中的变量值。在 Watch 窗口中有几个 Watch 子窗口，在每个子窗口的 Name 列任意一行键入变量名后回车后在相应的 Value 列中就得到程序中此变量的值。这样可观察程序运行中变量值的计算

与传递是否正确，常常能找到不容易发现的问题，还可利用窗口值来修改变量的值测试程序对各种情况的

4.1.5 计算机程序结构设计基础

应用程序结构设计，一般包括以下步骤：

1．编译预处理指令

Visual C++绘图源程序开始部分是编译预处理指令，这是 C 语言的一个独特点。编译预处理指令告诉编译系统对源程序进行编译之前应做些什么，

编译预处理指令以#号开头，并与程序语句分开，占用一单独书写行，#号前不能留有空格，指令结尾不用分号。编译预处理指令有三种，即包含文件预处理指令、宏定义预处理指令和条件编译预处理指令。编译预处理指令为程序员提供有效工具，方便编制程序，扩展编程环境，使开发的程序易于阅读、修改和移植到不同的计算机系统上去，编译预处理指令。

(1) 包含文件的预处理指令

包含文件预处理指令的一般格式：

#include <文件名>　指示编译系统按系统设定的目录搜索包含文件；

或 #include “文件名” 表示按指定的路径搜索，默认值为当前目录。

文件名为磁盘中文本文件的名字。包含文件预处理指令的功能是，在编译源程序之前，取代该预处理指令，也就是从磁盘中读取包含文件，然后把它写入源程序中预处理指令的位置上，使之成为源程序的一部分。从而避免程序员的重复性劳动，提高工作效率。例如：

(2) 宏定义预处理指令

宏定义预处理指令的一般格式：

#define 替换名　　字符串　应用程序为便于常量或变量容易记易或修改，常用替代名称代替实际常量或变量。

(3) 条件编译预处理指令：说明语句、主函数、子函数、一般语句、图形初始化语句、绘图语句、程序注释

4.1.6　绘图程序设计基本方法

任何一个图形都是由若干个简单的几何图形构成，而简单图形又是由

点、线、弧等基本的几何元素构成的。用计算机绘图就是要对根据最基本图形信息（点、线、弧、坐标值等）在图形中的几何位置以及相互之间关系进行数学描述，即构造出图形的数学关系式，设计出绘图程序，绘制出图形。图形的数学关系要准确、合理、简洁，绘图程序要求通用、方便、可移植。一般采用模块化设计方法。

4.1.6.1 图形层次结构和程序模块结构

（1）图形层次结构

一般物体均可分解成许多不同形体元素的集合，即物体可分层来表示。例如，一张床分成床头和床屉不同形体组成，这就是一个一层的结构，在工程设计中，高层与低层之间的关系实际上是物体—子物体、部件-子部件、图形—子图形的关系，这种关系也反映了它们之间的装配、调用关系，那些最底层的形体一般用基本形体来表示。而表示形体的图形是由简单的图素组成较复杂的图形，再由较复杂图形构成更复杂的图形，一层一层地构造下去，得到整个图形。

几何图形或其它形式数据构成的图形一般都有层次结构，利用图形层次结构可以方便地实现模块化的绘图程序设计。

（2）程序模块结构

程序模块指一个单独的函数，或是逻辑组合在一起的有关函数的集合，或是有关语句的集合。模块化程序设计是指在编写程序中把一个程序分成几个部分，分别编写成函数形式。由于函数的编写、测试可独立进行，从而是的程序的编写和维护更方便，同时又便于编写出优良的程序。这些部

分也就是程序中的模块。

Visual C++绘图程序中主要模块可分为两种：主函数模块和函数模块。一个简单的程序中没有函数，就只有模块，这一模块就是主函数模块。程序中的函数就叫函数模块，它们都是独立的代码，可通过参数接收数据。程序中的每条语句都属于一个特定的模块，C++语言能很好地支持模块化程序。

模块化编程设计有助于计算机绘图程序生成各种各样的图形。图形中一

层子图形可用程序中一个模块生成，整个图形由程序全部模块生成。例如，现有一画花瓣的模块，将方向、位置各不相同的花瓣组合起来画出一朵花，它是通过调用花瓣模型同时加上方向，位置参数的变化来生成，这朵花又称为另一模块，将方向、位置各不相同的花组合成为花花丛，进一步组成花园。这里按层次构成图形（图像）即花园，其程序模块结构描述为：一个花园图的程序可分为多次调用画花丛的模块；画花丛又可多次调用画花朵的模块；画花朵的程序由可多次调用画花瓣的模块，一幅画花园的图形

就这样通过调用模块化出来了。

4.1.6.2 面向对象程序设计

面向对象程序设计与传统的结构程序设计方法不同，代表了新的程序设计方式，使计算机问题更符合人类自身的思维方式和方法，提高了软件的重用性和扩充性，并能有效控制软件的复杂性和软件维护的开销。

图形软件的设计中，将一些基本的图素，如点、线、面、弧等定义成基本的图素类，图素的操作则可分别通过定义这些相应的数据成员和成员函

数　　来　　提　　供　　。

MFC类库中也提供了许多基本图素类，面向对象程序设计更好地支持图形的层次结构，即问题在不同层次上的抽象。在面向对象的程序设计中，是通过类的继承机制来实现。例如将图素类的直线定义为一个基类，即可从这一基类导出折线类、多边形类等第一级导出类，由第一级导出类又可产生圆类、椭圆类、圆弧类等第二级导出类。再往下还可导出 B 样条曲线类、NURBS 曲线类等等。这样就可产生一个非常清楚的类的体系结构，有利于

用户的掌握和使用，同时也可很方便地从其中的某一个类中导出特殊需要的类。所以在进行程序设计，搞清楚所涉及的问题以及体系结构的逻辑关系和层次关系，便于将程序中多次出现的、具有共性的部分提出来成为类，通过提取类的方法来解决复杂问题。

4.1.6.3 绘图子程序和主程序

1．绘图子程序

编写绘图程序时，首先需要编制一定数量的绘图子程序，根据其功能不

同，绘图子程序由以下几类：

（1）基本子程序

在编写绘图程序时，把绘制点、直线、圆弧及各种线型、字符等绘图程序成为子程序，称为基本子程序。必要时可利用 MFC 类库提供的基本子程序。

（2）功能子程序

把绘图过程中经常要用到的绘制三角形、四边形、圆、椭圆等预先编写

成子程序，在编制绘图程序时调用这些子程序进行绘制，这些子程序又是通过调用基本子程序编写的，称为功能子程序，又称二级子程序。必要时也可利用 MFC 类库提供的功能子程序。

（3）应用子程序

利用上面两种子程序开发编写一些比较复杂的程序，能满足绘图需要、通用性相对较小的程序，称为应用子程序，又称三级子程序，可为某一类程序设计所引用。

因此，在编写大型绘图程序中，一般需按照模块化程序设计方法的要求，把复杂程序分成几个较易调试的子程序即模块来编写。对于更大更复杂的程序，可画出绘图程序的模块结构图，明确程序由那些模块组成然后编写每个模块的程序，一个模块就是一个子程序，程序设计中尽可能采用模块子程序，以至最后的程序由调用各个模块子程序构成。

2．绘图主程序

绘图主程序即程序执行体。在模块子程序（函数）编写好的基础上，编

写主程序（主函数）较容易，有时只需调用各个模块子程序即可构成主程序，这是典型的模块结构程序。主程序中还可采用人机对话形式，即根据运行时提示图形的几何参数或结构参数由用户一次输入，这是编写较为通用的绘图程序常用的方法编写绘图程序要采用模块子程序和主程序。如果一个几百条语句以上的程序，逻辑关系复杂，整个程序又没有用模块子程序（即函数），这样的程序可能会运行很长时间也不出结果，千万不要以为系统出现问题。若将同样程序划分为一定功能模块的几个子模块然后通过

主程序调用来运行，这样将很快得到结果，这是采用主程序和子程序带来的效率。

4.1.6.4 绘图方法

在编程绘图时，根据所绘图形的复杂程度不同可采用边计算边画图的方法或采用先计算后画图的方法，一般绘制平面图采用前者，绘制复杂曲面图形采用后者。计算机绘制图形又可分为以下几种：解析法、样条法、变形法、拼合法、和创造法。

1. 解析法

根据图形的解析表达式或参数表达式，编写绘图程序时，先计算出图形中各点的坐标值等，然后用绘制函数绘出其图形。这种方法的关键是要将图形用解析式表示，同时如何用解析式取图形上的点、区多少个点都很重

要。这将影响绘图质量，主要是图形的光滑程度。

解析式是常用方法，特别是绘制一般几何图形时。

2. 样条法

当一个物体或图形不是用解析式表示，或者不能用解析精确表示时，往往是用物体或图形的一些实际数据值亦称型值点，构造曲面或曲面拟合它，或者用样条曲线来逼近它，同时通过不断调整、修正样条曲线或曲面，绘制出这些图形。用样条法绘制图形，是工程中绘制自由曲线和曲面的实用方法。

3. 变形法

变形法是对基本图形或称单元图形施行各种几何变换（比如平移、对称、旋转等）从而形成新的或更复杂的图形，如对基本图形矩形进行多次比例、平移变换即可绘制一座小屋。变形法是计算机绘制图形的重要方法之一，

它可使绘制者不必逐笔、逐线、逐个形体进行绘制，而只需找出图形之间的内在关系，对基本图形施行各种几何变换，重新组合排列便可获得所需图形。

4.　拼合法

即将图形分解成若干个基本图形元素（简称图素），把相同部分的图素编写称通用的子程序，绘制图形是可根据实际需要有所不同，但一般要考虑独立性、常用性和可组合性。

5.　创造法

一般绘制图形，是按照一个物体（或景物）绘制出它的图形或是根据原图形绘制出图形。创造法绘图与此不同，实现没有图形或物体，而是设计出绘图算法创造性地绘制图形。例如分型图，预先不能想象出算法所绘制出的图形样子，只有程序运行完后才能知道图形
的全貌。

创造法绘制图形为图形领域开辟了另一个美妙的新世界。

一般说图形处理程序要比一般非图形程序复杂，但只要在开始时在头脑中就建立起完好的构思结构及清晰的有关概念，便可克服绘图程序设计中可能出现的混乱现象。

（1）坚持到底

4.2 图形的数据结构

图形是由点、线、面、体各种几何元素组成的，如何用数据描述图形，图形数据又如何存入计算机，使计算机处理时更准确、方便、完整、迅速地描述图形，这就需要研究图形数据结构。有以上讨论已知一般图形都有

层次结构，任何复杂的图形均可用简单图素来组织描述。而 C++语言具有指针、结构等丰富的数据类型，同时它的面向对象的程序设计方法使图素模块（或绘图模块）之间的关系变得更清晰便于对图形进行修改、删除、插入等操作。

用计算机绘图，就要将图形转化为数据，通过计算机对图形数据进行加工处理，绘制图形。这样用计算机绘图首先要考虑如何在计算机中建立恰

当的模型表示不同图形对象？如何组织图形对象的描述数据以使存储这些数据所要的空间最省，检索、处理这些数据的速度更快。

4.2.1 图形信息的分类

在组织绘制图形过程中会遇到大量信息，这些信息通常被分为：图形信息和非图形信息。

图形信息是图形对象及构成它的点、线、面的位置、相互间关系和几何尺寸等；非图形信息是表示图形对象的线型、颜色、亮度以及供模拟、分析用的质量、比重、体积等数据。

对大量信息的管理，常用概念还有：基本图形元素与段

基本图形元素：图素或图元、体素。用数据来描述，其中图素指可以用一定的几何参数和属性参数描述的最基本的图形输出元素。体素是三维空间中可以用有限个尺寸参数定位和定形的体，常有三种定义形式：

(1)从实际形体中选择出来，可用一些确定的尺寸参数控制其最终位置和形状的一组单元实体

(2)由参数定义的一条(或一组)轮廓线沿一条(或一组)空间参数曲线作扫描运动而产生的形体。

(3)用代数半空间定义的形体。段（也称图段）:图形学软件在输出基元和画面之间设置一个中间数据结构，称为图段。图段用规则来描述。图段是由一组输出基元和一组性质构成的集合，图段可以嵌套，图段的常见性质 ：

可见性、优先度、突出性等。信息中的各个数据元素之间有着一定的结构关系，数据结构就是研究数据的特征以及数据之间存在的关系。

4.2.2 图形数据结构

1．常用图形数据结构有：线性表、数据和树等逻辑结构以及顺序存储和链表存储等存储结构。

线性表是一种最简单、最基本的线性数据结构，线性表具有相同类型的 n 个（n≥0）数据元素的有限序列，即在线性表中，数据元素之间的相对

位置是线性的，除最后一个数据外，表中其余元素都有且只用一个后继；除第一个外，都有且只有一个前趋。

线性表存储方式有：顺序存储和链表存储。顺序存储是使用一组连续的存储单元一次存储线性表中的各元素。链表是也是一种常见的。链表一般有一个头指针，存放一个地址，该地址指向一个元素，称为节点。一个链表节点包括：数据域和链域，分别用来存放数据和下一个节点的地址指针。链表的最后一个节点不再指向其它元素，称为“表尾”。

数组是线性表的推广，如线性表中的元素是一个单个数据时的数据结构为数组，称为一维数组。当线性表中包含有多个相同描述的数据记录的线性表为多维数组，线性表的长度是可变的，但大小固定。此外还有字符数组等，详细的内容请参考数据结构相关书籍。

2．数据结构研究主要内容：

（1）研究数据元素之间的关系，即逻辑结构。

（2）研究数据在计算机内部的存储方式。

（3）研究如何在数据的各种结构上进行处理，即算法。

3．设计图形程序时，同一图形可以设计不同的数据结构，但不论采取什么数据结构，图形数据结构应满足以下要求：

（1）能够记录图形的几何和拓扑信息；

（2）该数据结构能完全、唯一地描述某一图形；

（3）适应对图形进行运算和处理，边与管理、查找、检索和修改。

（4）节省内存。

4.2.3 计算机对数据的管理—数据文件

文件为程序设计中的重要内容，计算机中的数据是以文件形式存在的，文件一般是指存储在外部介质（如磁盘）上数据的集合。一批数据以文件的形式存放在外部介质上，操作系统以文件为单位对数据进行管理，若想找到存储在外部磁盘介质上的数据，必须先按文件名指定文件，然后再从文件中读取数据。要想向外部磁盘介质上存储数据，也必须先建立一个文

件，才能向外部介质输出数据。

在程序运行时，常常需要将一些数据输出到磁盘上存放起来，需要时再从磁盘输入到计算机内存，即对数据文件进行操作。

Visual C++中把文件看作是一个字符的序列，由一个一个字符的数据按顺序组织起来。根据数据形式，文件可分为 ASCII 代码和二进制文件，ASCII 文件又称为文本（text）文件，其中每一个字节放一个 ASCII 代码，代表一个字符。二进制文件是把内存中的数据阿呢日至数据的各是存储的文件。

用 ASCII 输出的数据与字符一一对应，一个字节代表一个字符，便于对字符进行处理，但占用内存较多，速度慢。用二进制形式输出数值，可节省外存空间和转换时间，但一个字节并不对应一个字符，不能直接输出字符形式。

Visual C++对文件的存取是一字符为单位，输入或输出的数据流的开始和结束仅受程序控制而不受物理符号（如回车换行符）控制。在处理文件过程中，系统自动在内存区为一个正使用的文件开辟一个缓冲区，从内存

向磁盘输出数据必须先送到内存中的缓冲区，装满缓冲区后才可送到磁盘，从磁盘向内存读入数据为一次性将一批数据输入到内存缓冲区，在从缓冲区逐个将数据送到程序数据区。

4.2.4 图形数据的存储状态

在编制一个图形系统时，其中最基本也是最重要的是如何组织数据结构和存储方式。数据的存储方式有两类：静态和动态。

静态：即数据不被系统处理时的状态，此时全部数据资料都以外部文件

的形式存储。任何一个成熟的系统都有其自身的文件结构。不同应用程序中的静态文件结构有所不同，一般来说，静态的文件结构有以下两部分：

（1）头文件：包含文件中数据的数量、存放格式等所有主控信息；

（2）数据部分：数据主体。

一般来说，静态时的数据存储组织相对比较简单，按照确定的组织顺序把各种数据信息放到一个二进制文件中即可。

动态：即数据资料被系统处理时的状态。系统可对数据资料进行增加、

删除、修改、查询等操作。一般静态的文件结构不能满足动态的操作需要。此时系统要求对数据可进行任意使用和重新组织，同时在速度上，要求快速的对数据进行操作。

4.2.5 动态文件数据结构的组织原则

动态文件数据结构的组织原则为足够和可扩容的容量、良好的适应性、较快的速度等，根据系统要求，这些因素共同协调，形成一个有机整体，实现既定目标，而不能厚此薄彼。

下面我们将利用 AppWizard 生成的 Draw 画图应用程序的程序编写过程，讲述在图形系统的设计中，如何组织一般图形元素的数据结构和存储方式。

4.2.6 简单 CAD 绘图系统编程实例中的数据结构

利用 Visual C++中面向对象的程序设计和 C++的组织方法，组织建立一个基本图形系统的图形元素类。

4.2.6.1 图形元素基类的组织

复杂的图形系统，都是由一些最基本的图形元素组成的。对各种图形元素进行分析，把各类图形元素具有一些相同的属性和操作功能组织存放在一个图形元素基类中，程序中其它一些图形元素再由此基类派生。简单CAD绘图系统编程实例中，在头文件 Draw.Doc.h 中，定义一个图形元素基类CDraw。代码如下：

class CDraw:public CObject //基本图形类，用来存储图形的颜色、线型、层等信息

```
{
protected:
    CDraw(){}     //构造函数
    short m_ColorPen;     //笔色
    short m_ColorBrush;   //填充刷颜色
    short m_LineWide;     //线宽（像素）
    short m_LineType;     //线型（像素）
```

```
    short m_Layer;          //所处层
    int m_id_only;      //图形元素唯一的识别号
    BOOL b_Delete;              //是否处于删除状态
public:
    CDraw(short ColorPen,short ColorBrush,short LineWide,
        short LineType,short Layer,int id_only,BOOL Delete)    //构造函数
        {
```

```
m_ColorPen=ColorPen;

m_ColorBrush=ColorBrush;

m_LineWide=LineWide;

m_LineType=LineType;

m_Layer=Layer;

b_Delete=Delete;

m_id_only=id_only;
```

```
        }

}
;
```

说明：在此图形基类中，有两个构造函数，第一个不带参数，第二个构造函数由七个参数，用来初始化类中的成员变量。

1．直线类 Cline

有了图形元素的基类 CDraw，在此基类的基础上派生一直线类 Cline，直线类的基本参数（线型、线宽、颜色、删出标志等）由基类 CDraw 类中继承，除此之外直线类中的特殊属性（如直线的起点和终点坐标等）则在直线类中定义。在头文件 Draw.Doc.h 中定义直线类 CLine：

```
class CLine:public CDraw      //直线类
{
```

代码见纸书　}

};

自此直线类中也重载了两个构造函数，第一没有参数，用来定义一个不带参数的 CLine 直线类对象，另一个带有十一个参数的构造函数中利用前面 CDraw 基类中的七个参数的调用对基类中的成员变量进行初始化，并对自己 CLine 类中的四个参数（直线的起点和中的坐标）进行初始化。

2．连续直线或封闭多边形类组织

连续直线除了具有基类 CDraw 图形元素所有属性外，从图形的几何特

征上连续直线由许多顶点组成，顶点的数目是不确定的，对一条连续直线来说可能只有两个，也可能有几千个顶点，在头文件 Draw.Doc.h 中定义连续直线类 CPline 派生数据类结构来存储连续直线的一个顶点坐标：

```
typedef struct
{
    float x;
    float y;
```

```
    float z;
}PointStruct;     //存储每个顶点坐标的结构
```

根据连续直线的顶点数目，对于连续直线的顶点坐标，采用动态存储空间的方法，即在 CPline 对象中分配连续直线的存储空间。在头文件 Draw.Doc.h 中定义连续直线类或多边形区域类 CPline 如下：

```
class CPline:public CDraw     //连续直线或多边形区域类
{
```

```
protected:
        int m_Numble;      //连续直线或多边形区域的顶点数
        BOOL b_Fill;  //是否为连续直线
    public:
        PointStruct* m_PointList; //存储顶点的数组指针
        CPline()    //不带任何参数的构造函数
        { m_Numble=0;}
```

```
CPline(short ColorPen,short ColorBrush,
    short LineWide,short LineType,short Layer,int id_only,
    BOOL Delete,int Numble,PointStruct* PointList,BOOL Fill)
:
CDraw(ColorPen,ColorBrush,LineWide,LineType,Layer,id_only,Delete)
{
    m_Numble=Numble;
```

```
b_Fill=Fill;
m_PointList=new PointStruct[Numble+1];    //分配空间
if(Numble>0)
{
    for(int i=0;i<Numble;i++)
        m_PointList[i]=PointList[i];
}
```

```
        }
        ~CPline()  //析够函数
        {
            if(m_Numble>0)
                delete m_PointList;
        }
};
```

从 CPline 类的定义中可看出，连续直线或多边形区域的顶点坐标存储在一个结构数组 m_PointList 中，连续直线类中也重载了两个构造函数，第一没有参数，用来定义一个不带参数的 CPline 连续直线类对象，另一个带有多个参数的构造函数中利用前面 CDraw 基类中的七个参数的调用对基类中的成员变量进行初始化，在第二个结构函数中，对顶点数大于 0 的连续直线或多边形动态分配了存储顶点坐标的结构数组。b_Fill 成员变量用来表示图形元素为区分连续直线还是多边形，b_Fill 为真时，表示图形元素为多边

形区域，相反为假时，这个图形为连续直线。

3. 圆类

圆类的定义方法除了具有基类 CDraw 图形元素所有属性外，从图形的几何特征上看，还可以用圆心和半径作为特征参数表示圆的几何特征，在头文件 Draw.Doc.h 中定义圆类 CCircle 派生数据类结构如下：

```
class CCircle:public CDraw   //圆及圆形区域类
{
```

代码见纸书　　}

};

在圆类中定义了三个成员变量 m_CircleX、m_CircleY、m_CircleR 来记录圆心和半径，同样，利用 b_Fill 成员变量用来表示图形元素为区分圆还是圆形区域，b_Fill 为真时，表示图形元素为圆形区域，相反为假时，表示图形元素为圆。

此类中也定义了两个构造函数，第一没有参数，另一个带有多个参数的

构造函数中利用前面 CDraw 基类中的七个参数的调用对基类中的成员变量进行初始化。上面定义的圆类中用圆心和半径的方法表示圆的几何特征，实际中的图形设计中常采用边界矩形方法表示一个椭圆，而圆只是被看作椭圆的一个特例。

4．圆弧类组织

圆是圆弧的弧度为 360 度时的特例。圆弧可从圆类派生，在头文件 Draw.Doc.h 中定义圆弧类 CArc 派生数据类结构如下：

```
class CArc:public CCircle      //圆弧类
    {
    protected:
        float m_Angle1,m_Angle2;    //圆弧的起点和终点角度
```

代码见纸书

```
        }
};
```

说明：在圆弧类中定义了两个成员变量 m_Angle1、m_Angle1 来分别表示一个圆弧的起始和结束弧度。同样，类中也定义了两个构造函数，第一没有参数，另一个带有多个参数的构造函数中利用前面 CCircle 基类中的构造函数的调用对基类圆类和基本图形元素基类中的成员变量进行初始化。

5．标注文本类

文本标注在图形系统中也为一常用工具，标注文本类除了具有基类 CDraw 图形元素所有属性外，还具有位置、字体及标注内容等信息，在头

文件 Draw.Doc.h 中定义标注文本类 CText 派生数据类结构如下：

```
class CText:public CDraw      //标注文本类
{
代码见纸书      }
};
```

说明：CText 类中也定义了两个构造函数，第二个带有多个参数的构造函数中利用前面 CDraw 基类中的构造函数的调用对基本图形元素 CDraw

类中的成员变量进行初始化。

4.2.6.2 组织图形类系统文档

下面讲述利用 MFC 应用程序的文档类管理体系组织文档，实现基本图形系统的文档管理功能。

1. 组织面向对象的文档存储管理机制

面向对象的系统中，数据的管理，即通过文档组织机制，可利用图形元素类创建很多图形元素对象，其中每一图形元素对象作为一个整体来组织

存储空间的分配、保存和读取等功能。通过建立一种存储机制来管理指向所有元素对象的指针，来管理所有图形元素对象。这种文档管理机制具有组织简单、结构化和移植性好等特点，缺点为将占用很大空间。

2. 利用 MFC 模板管理图形元素对象指针的对象

管理一个图形系统文档的思路为：每一个图形元素是图形元素类创建的一个对象，在创建这个对象时得到指向这个对象的指针，通过建立一个对

象指针数组来管理这些指针，达到管理所有图形元素对象的目的。

在 MFC 类中利用类模板 CTypePtrArray 来定义一个管理类指针的对象,，例如，定义一个管理 CLine 类指针的对象：

其中对象 m_LineArray 由第一个参数指向的类(CObArray)创建并管理第二个参数指定的类指针，以上代码含义为：对象 m_LineArray 是由 CObArray 类创建的用来存放 CLine 类指针。

针对我们所创建的图形程序在文档类 CDrawDoc 中创建的管理各类图

形元素对象指针的 CObArray 对象为：

```
private:
CTypedPtrArray<CObArray,CLine*>m_LineArray; //管理直线对象指针的对象
CTypedPtrArray<CObArray,CPline*>m_PLineArray;  //管理连续直线对象指针的对象
CTypedPtrArray<CObArray,CCircle*>m_CircleArray;//管理圆对象指针的
```

对象

CTypedPtrArray<CObArray,CArc*>m_ArcArray;　　//管理圆弧对象指针的对象

CTypedPtrArray<CObArray,CText*>m_TextArray;　//管理标注文字对象指针的对象

实现文档的管理功能

4.2.6.3 增加图形元素

在上述应用程序中增加一个图形元素（如一条直线），需要进行以下步骤：

（1）创建一个图形对象，并用图像元素的实际数据初始化这个图形元素对象，

（2）把指向新创建的图形元素对象的指针，增加到文档类管理图形元素对象指针的对象中

（3）为实现上述功能，在文档类 CDrawDoc 中定义几个函数，分别来

完成增加各类图形圆的操作功能：

（4）最后函数返回指向新增图形元素对象的指针。

```
CLine* AddLine(short ColorPen,short ColorBrush,short
        LineWide,short LineType,short Layer,int id_only,float X1,float
Y1,
        float X2,float Y2);      //增加一条直线
    CPline* AddPLine(short ColorPen,short ColorBrush,
```

```
        short LineWide,short LineType,short Layer,int id_only,int
Numble,
        PointStruct* PointList,BOOL b_Fill);
    CCircle* AddCircle(short ColorPen,short ColorBrush,
        short LineWide,short LineType,short Layer,int id_only,
        float CircleX,float CircleY,float CircleR,BOOL Fill);
    CArc* AddArc(short ColorPen,short ColorBrush,short
```

LineWide,short LineType,short Layer,int id_only,float CircleX,

float CircleY,float CircleR,BOOL Fill,float Angle1,float Angle2);

CText* AddText(short ColorPen,short ColorBrush,short

LineWide,short LineType,short Layer,int id_only,float StartX,

float StartY,float Angle1,float Angle2,float TextHeight,float TextWide,

float OffWide,unsigned char TextFont,int TextLong,CString);

在实现文件 DrawDoc.cpp 中加入函数的实现代码：

//添加一个直线对象

代码见纸书}

4.2.6.4 实现各类图形的绘制

利用虚函数来实现各种图形元素的绘制功能，在图形元素的基类 CDraw

中，抽象定义一个进行绘制操作的虚函数。因为在应用程序中，不用 CDraw 类直接定义对象，所以可以将虚函数定义成纯虚函数：

virtual void Draw(CDC *pDC,int m_DrawMode,int m_DrawMode1,short BackColor)=0;

各参数的含义：

pDC：指向当前绘图对象的指针；

m_DrawMode：绘制模式。1—R2_COPYPEN,2—R2_NOT。根据需要可

加入其它的模式。

m_DrawMode1：直线绘制模式。1—正常显示，2—特殊显示（如鼠标选中时的现实）。根据需要可加入其它的模式。

BackColor：当参数 m_DrawMode1 取 2 时，此参数指定颜色号；

在各种图形元素类中重载虚函数 Draw（对于用来创建对象的类，必须有基类纯函数的重载）

Public:

virtual void Draw(CDC *pDC,int m_DrawMode,int m_DrawMode1,short BackColor)

以下将根据图形元素的绘制方法，在各个图形元素类中加入 Draw 函数的实现代码：

//完成直线的绘制功能

void CLine::Draw(CDC *pDC,int m_DrawMode,int m_DrawMode1,short

BackColor)

{

代码见纸书

}

说明：

1.对于连续直线和封闭区域的绘制，主要用到绘图 CDC 类中的 Polygon 成员函数：

Polygon 函数绘制一个多边形，必要时系统会自动把第一个和最后一个顶点连成封闭多边形，用 SetPolyFillMode 设置的填充模式填充。

2.在绘制函数中，根据 CPline 类中的 b_Fill 成员变量的不同，选择不同的操作。

b_Fill 为 TRUE 当前图形元素为一多边形区域，若正常显示（m_DrawMode==0 || int m_DrawMode1==2）调用 Polygon 函数绘制一个多边形区域，若为特殊显示（m_DrawMode1= =1），用连续直线的顶点坐标初

始化一个区域 rgn，用 CDC 类的成员函数 InvertRgn 反色显示这个区域。b_Fill 为 FALSE，调用 LineTo 函数分段绘制连续的各个直线段。

```
//圆和填充圆的绘制
void CCircle::Draw(CDC *pDC,int m_DrawMode,int m_DrawMode1,short BackColor)
{
    if(b_Delete)    //如果已经处于删除状态
```

代码见纸书

```
    pDC->SelectObject(cjcbakf);//恢复字模
}
```

4.2.6.5 保存图形数据到文档

用鼠标在屏幕上交互绘制的图形元素，要创建一个图形元素对象并将指

向这个图形元素对象的指针保存起来。即在用鼠标在视图中交互绘制图形元素时，在屏幕上完成图形元素绘制的同时，还需把这一图形元素保存到文档中，否则，重画视图时交互绘制的图形元素就消失了。

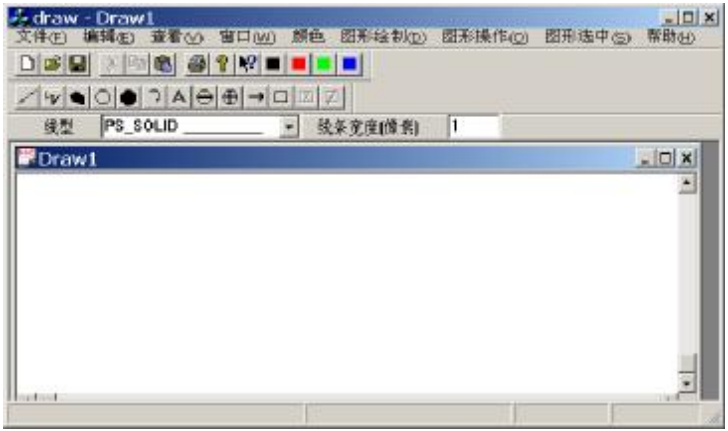
用鼠标交互绘制一个图形元素后，要在屏幕上马上显示着以图形元素的实际形态，必须将图形元素以实际的形态重画，因为 VC++的 R2_NOT 绘图模式下，线条是彩色的，拓东的图形都不是图形元素的实际颜色，必须对图形元素以实际形态进行重画。重画这个新绘制的图形元素，可通过发

送一个窗口消息使屏幕全部或部分区域重新来实现，但这以方法效果不好，因为要达到使新绘制的图形元素重画的目的，重画的最小区域也包括这一个图形元素的边界矩形（或许为一任意多边形区域），在有大量图形元素的情况下，重画的速度是很慢的，所以一般不采用这种方式。

4.3 简单 CAD 绘图系统功能简介

4.3.1 简单 CAD 绘图系统运行界面

运行简单 CAD 绘图系统得到如图 1-1 的运行主界面。



4.3.2 简单 CAD 绘图系统功能

1. 直线绘制

选择绘制直线操作后，首先用鼠标左键点中直线的起点，此时鼠标移动时就会拖动一条橡皮筋线，再按下鼠标左键就会选中直线的终点而完成直线的交互绘制；按中鼠标右键就会擦除橡皮线放弃直线的绘制。

Visual C++ 中采用 R2_NOT 绘制模式(SetROP2(R2_NOT);)作用首先将原来的橡皮线擦除，然后绘制直线起点到鼠标点的连线，并将当前鼠标点记为上一个移动点 mPointEnd。当鼠标移动时，这一过程一直进行下去，从

而实现拖动橡皮线的功能。

2．连续直线绘制

选择绘制连续直线操作后，首先用鼠标左键点中连续直线的起点，此时鼠标移动时就会拖动一条橡皮筋，以后继续按下鼠标左键就会增加连续直线的顶点，按下鼠标右键就会完成连续直线的绘制（当顶点数小于 **2** 时，不能绘制成功）。

利用结构数组 **PointXyz** 来存储已经按下的连续直线顶点的实际坐标。

3．多边形区域绘制

多边形区域的绘制方法与连续直线相同，不同的是当顶点数目小于3时，不能绘制。

4．圆和圆形区域绘制

圆的绘制用的是圆心半径法。即开始绘制圆的操作后，第一次按下鼠标左键选中圆的圆心，此时鼠标移动时会拖动一个圆，再按下鼠标左键会选中圆上的一点而形成一个圆，按下鼠标右键时会擦除橡皮圆放弃圆的绘制。

5．圆弧绘制

圆弧绘制用的是三点法。即用起点、经过点、终点三点形成一个圆弧。首先按下鼠标左键选中圆弧的起点，第二次按下鼠标左键选中圆弧的经过点，此时鼠标移动时就会拖动一个圆弧，第三次按下鼠标左键选中圆弧的终点而完成一个圆弧的交互绘制。在绘制过程中，按下鼠标右键会放弃圆弧的绘制操作。

6．文本标注

当选择进行文本标注操作后，首先用鼠标左键在屏幕上点中要标注文本的左下角起点，点中后会出现如图 **2-1** 所示的对话框，将要标注的文本输入到对话框中的编辑框中，按“确定”退出后，就完成了文本的标注。按“放弃”就会放弃本次标注。

进行标注时，字体的参数靠按下“字体参数”按钮得到。当按下“参数”按钮时，会得到如图 **2-2** 所示的对话框。在对话框中填入字体的信息，“字体高度”和“字体宽度”指的是标注字体的高度和宽度，“字体间距”是指

字体间的距离，它们都是以图形中的实际单位为单位。“标注角度”指的是标注行与 X 轴正方向的夹角，以角度表示，逆时针方向为正。“字体旋转角度”指的是标注字体的旋转角度，以角度表示，正常时为 0，以逆时针方向为正方向。用户填入相应的参数值，按“确定”将字体参数设置为现在的参数退出，按“放弃”按钮放弃所做的修改退出。

IsOpen 函数用来检查对话框窗口是否建立，IsVisual 函数用来检查当前对话框窗口在屏幕上是否可见，Init 函数用来初始化对话框类的成员变

量 m_Text。

消息处理函数 DrawText 由消息 ID_TEXT_MESSAGE 激发运行，用来完成在视图中绘制正在标注的文本，当在文本标柱对话框的编辑框中修改标柱文本内容时，这一消息处理函数被调用，将标柱文本的内容会知道屏幕上。

图 2-1　标注信息窗口

图 2-2　标注字体参数设置窗口

7．图形放大

从“图形操作”菜单中选择执行“图形放大”菜单项，或在工具条中点中“放”按钮，就进入图形放大操作状态，此时需要用鼠标在屏幕上画出一个窗口，窗口中的内容就会放大到整个屏幕上。第一次按下鼠标左键，选择放大窗口的一个顶点，鼠标移动时就会拖动一个矩形移动；第二次按下鼠标左键点中窗口的另一个顶点，系统就会进行图形放大操作，把窗口内的图形放大到整个屏幕中。在放大操作过程中，按下鼠标右键，会放弃

放大操作。

对于图形元素，本系统提供了点选、窗口选择、圆选择、多边形区域选择等多种选择方式。

8．选中图形元素的特殊显示；

当用鼠标在屏幕上点选图形元素时，通过计算可以判断是否选中了图形元素，同时告诉用户已选中图形元素，这需要把选中的图形用特殊的方式

显示，就用不同的形式重新绘制这一图形元素，在我们的系统中，用虚线显示选中的图形元素。

9．保留选中的图形元素

在图形元素操作过程中，要进行图形放大、重画等操作，需要保留选中的图形元素，即保留图形元素的数据和视图屏幕。**GraphSelectStruct** 为保留图形元素建立的数据结构。

DrawGraph 函数为屏幕视图重画时，视图重画选中的图形元素。图形元素

的选中操作方法有多种，可以点选，也可以用区域选中。点选图形元素的是当鼠标在视图屏幕上按下鼠标就可得到按下点的屏幕像素坐标，并通过与其它图形元素比较判断这点所在的图形元素，即为选中的图形元素。区域选中使通过判断各种图形元素是否与一个区域相交或包含在一个区域内来实现。区域选中最常用的时窗口选中（矩形区域选中）。本绘制图形系统采用点选的方式选中图形元素

运行菜单“图形选中”下的“点中图形元素”菜单项，或直接点中工具

条的“选”按钮，系统就进入到了点选图形元素状态，此时，用鼠标左键点中图形元素时，就能够选中图形元素。选中的图形元素会用虚线或反色显示。

判断一个点是否在直线上关键是判断一个点与此直线之间的距离是否在有效距离范围之内。

Cline 类的 **IsPoint** 函数的主要操作步骤：

首先判断按中点(**x,y**)是否在直线的边界矩形内，如果在函数返回 **TURE**；

如果按中点在直线的边界矩形内，调用 **PointLine** 函数计算按中点到直线的距离 **xx**，如果距离 **xx** 小于有效距离，即选中了直线，函数返回 **TURE**，否则返回 **FALSE**。

10. 放弃选中

运行菜单“鼠标选中”菜单下的“放弃选中”菜单项，或直接点中工具条上的“清”按钮，系统会放弃对图形元素所做的选中（选中的图形元素恢复正常演示），使选中的图形元素数为 **0**。

放弃选中与选中图形圆素相似，方法包括：一是修改数据，使选中的图形元素为 **0**。二要在视图中把选中的图形元素恢复到正常显示状态。

11．从屏幕上删除图形元素

首先用鼠标选中要删除的图形元素，然后选择运行“编辑”菜单下的“删除图形”菜单项，或直接点中工具条上的“删”按钮，系统会把选中的图形元素从图形中删除。

当把选中的图形元素进行删除时，要求在屏幕上实时地把药删除的图形

元素去掉。最简单的办法就是把图形再进行重新绘制，此时作了删除标记的图形元素不再绘制，即从屏幕上消失。利用。**UpdateAllView** 函数进行全屏幕重画；局部重画是只重新绘制要删除的图形元素的边界矩形内的区域，从而达到从视图屏幕上删除图形元素的目的。利用**InvalidaeRect(rr,TURE)**函数。

12. 图形移动

首先用鼠标选中要移动位置的图形元素，然后选择运行“编辑”菜单下

的“图形移动”菜单项，或直接点中工具条上的“移”按钮，系统就进入到移动图形元素操作状态，第一次按下鼠标左键选择一个基点，此时移动鼠标时，会拖动选中的图形元素移动，第二次按下鼠标左键选中目标点，选中的图形元素就会按着这两个点的相对位置进行移动。

以上简单介绍计算机图形学绘图基础、图形的数据结构，和简单 CAD 绘图系统的功能和设计过程，详细介绍请参见有关书籍和随本书提供的光盘中“简单 CAD 绘图系统”应用程序。