PC机
C图形编程手册

徐士良 编著

清华大学出版社

PC机
C图形编程手册

徐士良 编著

清华大学出版社

内 容 简 介

本书编入了在各种常用显示卡上进行图形编程的C函数程序。内容包括HGC卡、CGA卡、EGA卡、VGA卡及长城系列微机上的有关显示卡的基本图形编程，还包括了对汉字的操作，最后一章给出了一些实用的图形函数。书中给出的所有图形函数均可直接调用，并有配套软盘。

本书可作为广大图形编程工作者的技术手册，也可作为在PC机上用C语言从事学习、工作与研究的各类人员的参考书。

（京）新登字158号

PC机
C图形编程手册

徐士良 编著
责任编辑 范素珍
☆
清华大学出版社出版
北京 清华园
北京密云胶印厂印刷
新华书店总店科技发行所发行
☆
开本：787×1092 1/16 印张：22 字数：543千字
1994年2月第1版 1994年2月第1次印刷
印数：0001—8000
ISBN 7-302-01397-7/TP·540
定价：13.90元

前　言

在微机的各种应用中，开发图形功能已越来越显得重要了。

随着PC机的不断发展以及迅速普及，其图形功能越来越强，相继出现了各种类型的图形监视器及其适配卡；图形显示速度越来越快；分辨率越来越高；颜色也越来越丰富。

当前广泛使用的图形显示适配卡有CGA，EGA与VGA。除此之外，还有单色图形卡HGC、国产微机长城系列中的显示卡系列以及功能较强的高档VGA卡。这些众多的显示适配卡都具有自己的硬件特点及其软件支持。

本书提供了各种显示适配卡的各种图形模式下的与硬件直接有关的各功能C函数，它们分别是：

- 单个像点的读写；
- 矩形域图形的清除；
- 矩形域图形的复制；
- 矩形域图形的保存；
- 矩形域图形的装载；
- 矩形填充；
- 汉字的显示。

对于后6个功能函数，不是简单的用读写像点的方法来实现，而是根据各种显示适配卡的各种图形模式的特点进行编程，以提高显示速度。

本书的最后一章还提供了一些基本图形的C函数。这些图形函数与硬件无关，只要用户根据自己所使用的计算机系统配置的显示适配卡，选定某种图形模式下的单个像点读写的函数就可以了。

由于编者水平有限，书中难免有错误及不当之处，恳请读者批评指正。

另外，本书中所有的程序有配套软盘，若读者需要可直接与清华大学出版社软件部联系购用。

作者

1993年8月

目　录

第一章　PC 机显示系统简介

PC 机的显示系统一般是指由监视器和显示卡（又称显示适配卡）两部分组成。

监视器（monitor）是独立于 PC 主机的一种外部设备。

显示卡（adapter）是插在 PC 主机上的一块电路板。但有的显示卡则与主机板设计在一起。一般的显示卡包括寄存器组、存储器及控制电路三大部分。其中存储器又包括显示 RAM 和 ROM BIOS 两部分。PC 机对显示屏幕的所有操作，都是通过显示卡来实现的。

监视器和显示卡必须配套使用，不同类型的显示系统需要不同的监视器和显示卡与之匹配，才能达到应有的显示效果。

1.1　PC 机显示系统的特性

显示系统的特性主要包括：

- 显示分辨率；
- 颜色或灰度；
- 显示速度；
- 图形显示能力。

其中最主要的是显示分辨率和颜色或灰度。

（1）显示分辨率

显示分辨率是指显示屏幕上所有基本像素点（pixel）的分布，通常用列数×行数来表示。为了获得良好的显示效果，要求监视器的分辨率与对应分辨率的显示卡相匹配。

通常，高分辨率的显示效果比低分辨率的好。但是，显示分辨率的提高对监视器与显示卡的硬、软件要求会越来越高。特别是分辨率的提高在很大程度上受到监视器的显示尺寸和扫描频率的限制，也受到显示卡存储空间的限制。

根据应用情况的不同，在不超过监视器最高分辨率的条件下，可以通过对显示卡的不同设置而使用不同的分辨率。

（2）显示速度

显示速度是指在屏幕上显示图形和字符的速度。显示速度与显示分辨率和监视器的扫描频率密切相关。

显示分辨率越高，整个显示屏幕的像素点也就越多，显示速度就越慢。在这种情况下，为了提高显示速度，就需要提高扫描频率。

如果监视器只有一种扫描频率，则它只能与一种显示卡相匹配使用。目前，随着显示技术的发展，出现了可变频率的监视器，它采用自动跟踪技术，使监视器的扫描频率自动与显示卡的输出同步，从而具有较宽的适用范围。同一监视器可以适应多种分辨率与显示速度的显示卡。

(3) 颜色与灰度

颜色和灰度是衡量显示系统性能的重要参数。

单色监视器只有亮和暗两种灰度。

配接彩色监视器的显示系统CGA，在字符方式下可达到16种颜色，在图形方式下可有4种颜色。

对于显示系统EGA，在图形方式下最多可以达到16种颜色。而显示系统进入VGA时，最多可以使用256种颜色。

通过对显示卡的改进，出现了彩色图形显示卡适配单色监视器的显示系统，它用灰度代替了颜色。

颜色和灰度的增多要受显示内存容量的限制。分辨率越高，颜色越丰富，所需要的显示内存就越多。另外，在PC机中，显示内存（RAM）的扩大还受到留作显示用的地址空间的限制，一般不超过64K。当显示系统使用大于64K地址空间时，访问显示缓冲区的控制也就变得比较复杂了。

(4) 图形显示能力

图形显示能力是显示系统对屏幕上的每一个像素点都可以设置成不同值的能力。

通常，图形显示对硬件的要求比字符显示高得多，同时，图形显示对显示缓冲区的要求也要比字符显示时高得多。

1.2 PC机显示卡的种类

常用的显示卡有以下几种。

(1) MDA卡

MDA（Monochrome Display Adapter——单色字符监视器适配卡）与单色字符监视器配接，它只支持字符显示功能，无图形功能。MDA支持80列、25行字符显示，含有4K的显示缓冲区，用于显示字符及属性。

(2) HGC卡

HGC（Hercules Graphics Card——单色图形显示卡）卡不仅支持80列、25行的字符显示，而且支持单色图形功能。在图形方式下，其图形显示分辨率最高可达720×348点阵，并且，通过软件的设置，还可支持640×400单色图形显示以及模拟CGA卡图形方式。

(3) CGA卡

CGA（Color Graphics Adapter——彩色图形显示卡）卡支持字符/图形两种方式。

在字符方式下，它支持80列、25行及40列、25行的方式，颜色可选16种。但字符质量比较差，只有8×8点阵。

在图形方式下，它支持最大为640×200分辨率，但只有黑、白两种颜色。此外，它还支持320×200的中分辨率，每个像素点可以有4种颜色。

CGA卡有16K的显示缓冲区，用于显示字符及属性或图形方式下的图形数据。

(4) EGA卡

EGA卡（Enhanced Graphics Adapter——增强型图形显示卡）的字符显示能力和图形显示能力都比CGA卡有了较大的提高，显示分辨率达到640×350，最高分辨率图形方式的

颜色达到16种。EGA卡的显示模式也比CGA卡丰富，并且兼容CGA卡与MDA卡的显示模式。

(5) VGA卡

VGA卡（Video Graphics Array——视频图形阵列）是一种功能十分强大、颜色丰富的显示卡。

VGA的标准分辨率可达到640×480，并具有16种颜色。一些兼容性VGA卡（如TVGA卡、EVGA卡）的分辨率最高可达640×480（256颜色）、800×600（16颜色）或1024×768（16颜色）。

VGA卡兼容MDA卡、CGA卡、EGA卡的所有显示模式。

除了上述各种通用的显示卡以外，我国长城系列微机配置的显示卡有长城0520CH卡、CMGA卡等。

1.3 视频BIOS

视频BIOS（Basic I/O System 基本输入输出系统）是与显示卡配套的一个重要组成部分。用户通过调用它可以完成一些与显示有关的控制功能。对于要用到显示系统的一般程序设计都可以通过调用视频BIOS或更高的视频函数库来完成。但要想获得更高性能的显示程序，还必须要对显示卡的寄存器和显示内存直接进行编程。在本书中的所有图形显示函数，都采用对显示卡中的寄存器和显示内存直接进行编程，而不用调用视频BIOS中读写像点的功能。本书唯一调用视频BIOS功能的是设置显示模式。

1.4 显示模式

显示模式按功能可以分为字符模式和图形模式两大类。

字符模式也称字母数字模式，即A/N模式（Alpha Number mode）。在这种模式下，显示缓冲区中存放的是显示字符的代码和属性，而显示屏幕被划分为若干字符显示行和列。

图形模式（Graphics modes）也称APA模式（All Points Addressable mode）。在这种模式下，显示缓冲区中存放的是监视器屏幕上的每个像素点的颜色或灰度值，而显示屏幕按分辨率被划分成像素行和列。

由于显示卡的种类很多，因而出现了各种显示模式用于支持各种用途（字符、图形）。其中有些显示模式在不同类型显示卡及不同厂家的显示卡之间是通用的，这类显示模式称为标准模式。还有些显示模式是专用的，称为非标准模式。通常，显示模式号小于14H的是标准模式，其它则为非标准模式。表1.1列出了标准图形模式的分辨率、颜色以及适用的显示卡。表1.2列出了一些常见的非标准图形模式。

表1.1 标准图形模式

模式号	分辨率	颜色	显示卡
4，5	320×200	4	CGA，EGA， VGA，CGE400， 长城CH，长城CMGA

续表

模式号	分辨率	颜色	显示卡
6	640×200	2	CGA，EGA，VGA，CGE400，长城CH，长城CMGA
0DH	320×200	16	EGA，VGA
0EH	640×200	16	EGA，VGA
0FH	640×350	2	EGA，VGA
10H	640×350	16	EGA，VGA
11H	640×480	2	VGA
12H	640×480	16	VGA
13H	320×200	256	VGA

表 1.2　非标准图形模式

模式号	分辨率	颜色	显示卡
无	720×348	单色	HGC
22H	640×504	2	CMGA
29H	800×600	16	EVGA
2DH	640×350	256	EVGA
2EH	640×480	256	EVGA
30H	800×600	256	EVGA
37H	1024×768	16	EVGA
42H	640×400	16	CGE400
5BH	800×600	16	TVGA
5CH	640×400	256	TVGA
5DH	640×480	256	TVGA
5EH	800×600	256	TVGA
5FH	1024×768	16	TVGA

第二章　汉字库操作

2.1　汉字库的重建

在许多具有汉字显示功能的应用系统中，往往只用到很少一部分汉字，此时，为了节省存储空间，提高显示汉字的速度，没有必要将全部汉字库装入系统。

本节中的几个程序将告诉你如何在现有汉字库中挑出你所需要的汉字，从而构成特殊系统所要求的汉字库。

目前，常用的汉字库有16×16点阵、24×24点阵、32×32点阵等。而根据汉字库中各汉字点阵的信息结构划分，可以分为以行为主及以列为主两种形式。

所谓以行为主，是指在汉字库中，每个汉字的点阵信息以行为顺序连续存放（即一行接一行地顺序存放），一行上连续的8个点阵信息放在一个字节内，且左边的点对应字节的高位。在这种结构下，对于16×16点阵的一个汉字，一行有16点，分别存放在两个字节内，共有16行，因此共占32个字节；对于24×24点阵的一个汉字，一行有24点，分别存放在3个字节内，共有24行，因此共占72字节；对于32×32点阵的一个汉字，一行中的32个点分别用四个字节存放，32行点阵信息共占128个字节。对于其它规格的汉字点阵信息的存放，可以此类推。

所谓以列为主，是指在汉字库中，每个汉字的点阵信息是以列为顺序连续存放的，即一列接一列地顺序存放，在一列上连续的8个点阵信息放在一个字节内，且上边的点对应字节的高位。在这种结构下，对于16×16点阵的一个汉字，一列有16个点，分别存放在2个字节内，共有16列，因此共占32个字节；对于24×24点阵的一个汉字，一列中的24个点分别用3个字节存放，24列点阵信息共占72个字节。对于32×32点阵及其它规格的汉字点阵信息的存放，可以此类推。

在本章程序中，不管原来汉字库中的汉字点阵存放的顺序如何，重建后的汉字库中的汉字点阵信息总是以行为主连续存放的。而且，在重建后的汉字库中，其汉字代码即为该汉字的点阵信息在汉字库中的位置序号。

另外，在有的汉字库中，由于区位码从第7行到第12行之间为空白，因此在汉字库中没有它们的点阵信息；而在另外一些汉字库中，这些空白区的点阵信息也存放在汉字库中。在本章的程序中将分别予以考虑。

读者在使用本章程序时，先要弄清楚你所具有的汉字库的点阵规格及汉字库的结构，然后再决定选用哪个程序。如果事先还不清楚你的汉字库的点阵规格或汉字库的结构，则只能通过试验后才能决定采用哪一个程序。

下面分四种情况给出建立用户汉字库的函数程序。但必须注意，重建后的汉字库中，其汉字点阵信息总是以行为主顺序存放的。

一、原汉字库的汉字点阵信息以行为主存放且6行空白区未被压缩

在这种情况下，只需根据给定的汉字区位码，从原汉字库中取出汉字点阵信息，依次存放在新建的汉字库中。

其中区位码为 code 的汉字的点阵信息在原汉字库中的序号为

$$m=((code/100)-1)*94+(code\%100)-1^{*}$$

而汉字的点阵信息的第一个字节在原汉字库中的偏移量为

$$k=m*kk$$

其中 kk 为一个汉字点阵信息所占的字节数。

函数中的形参意义如下：

num——整型变量。原汉字库中汉字点阵的规格。当 num=1 时，表示 16×16 点阵，一个汉字点阵信息占 32 个字节；当 num=2 时，表示 24×24 点阵，一个汉字点阵信息占 72 字节；当 num=3 时，表示 32×32 点阵，一个汉字点阵信息占 128 字节。其它情况以此类推。

fn1——字符串指针。指向原汉字库的文件名。

fn2——字符串指针。指向用户新建的汉字库的文件名。

fn3——字符串指针。指向由各汉字区位码组成的文本文件名。在此文件中，存放用户所选定的部分汉字的区位码，各区位码之间用空格分隔（参见后面的例）。汉字区位码在此文件中的序号即为该汉字在新建后的汉字库中的代码。

函数程序如下(文件名为 hzlibr1.c)：

```
#include "stdlib.h"
#include "stdio.h"
void hzlibr1(num,fn1,fn2,fn3)
char *fn1,*fn2,*fn3;
int num;
{ FILE *fp1,*fp2,*fp3;
  int code,i,j,m,x,y,kk;
  long int k;
  char *p;
  kk=(num+1)*(num+1)*8;
  p=malloc(kk*sizeof(char));
  if ((fp1=fopen(fn1,"r+b"))==NULL)
    { printf("cannot open fn1 ! \n");
      exit(0);
    }
  if ((fp2=fopen(fn2,"w+b"))==NULL)
    { printf("cannot open fn2 ! \n");
      exit(0);
    }
  if ((fp3=fopen(fn3,"r+t"))==NULL)
    { printf("cannot open fn3 ! \n");
      exit(0);
    }
```

---

* 注：本书为使文字叙述与程序保持一致，英文字母均用正体，故公式中的量的符号也用正体。

```
    i=0;
    while (fscanf(fp3,"%d",&code)! =EOF)
      { j=code;
        y=j/100;
        x=j-y*100;
        m=(y-1)*94+(x-1);
        k=(long)m*(long)kk;
        fseek(fp1,k*sizeof(char),SEEK_SET);
        fread(p,sizeof(char),kk,fp1);
        k=(long)i*(long)kk;
        fseek(fp2,k*sizeof(char),SEEK_SET);
        fwrite(p,sizeof(char),kk,fp2);
        i=i+1;
        }
    fclose(fp1); fclose(fp2); fclose(fp3);
    free(p);
    return;
}
```

二、原汉字库的汉字点阵信息以行为主存放且6行空白区被压缩

在这种情况下，只需根据给定的汉字区位码，从原汉字库中取出汉字点阵信息，依次存放在新建的汉字库中。

其中区位码为code的汉字点阵信息在原汉字库中的序号为：

当(code/100)<7时，

$$m=((code/100)-1)*94+(code\%100)-1$$

当(code/100)≥7时，

$$m=((code/100)-7)*94+(code\%100)-1$$

而汉字点阵信息的第一个字节在原汉字库中的偏移量为

$$k=m*kk$$

其中kk为一个汉字点阵信息所占的字节数。

函数中的形参意义如下：

num——整型变量。原汉字库中汉字点阵的规格。当num=1时，表示16×16点阵，一个汉字点阵信息占32个字节；当num=2时，表示24×24点阵，一个汉字点阵信息占72个字节；当num=3时，表示32×32点阵，一个汉字点阵信息占128个字节。其它情况以此类推。

fn1——字符串指针。指向原汉字库的文件名。

fn2——字符串指针。指向用户新建汉字库的文件名。

fn3——字符串指针。指向由各汉字区位码组成的文本文件名。在此文件中，存放用户所选定的部分汉字的区位码，各区位码之间用空格分隔（参看后面的例）。汉字区位码在此文件中的序号即为该汉字在新建后的汉字库中的代码。

函数程序如下(文件名为hzlibr2.c)：

```
#include "stdlib.h"
#include "stdio.h"
void hzlibr2(num,fn1,fn2,fn3)
```

```
char *fn1,*fn2,*fn3;
int num;
{ FILE *fp1,*fp2,*fp3;
  int code,i,j,m,x,y,kk;
  long int k;
  char *p;
  kk=(num+1)*(num+1)*8;
  p=malloc(kk*sizeof(char));
  if ((fp1=fopen(fn1,"r+b"))==NULL)
    { printf("cannot open fn1 ! \n");
      exit(0);
    }
  if ((fp2=fopen(fn2,"w+b"))==NULL)
    { printf("cannot open fn2 ! \n");
      exit(0);
      }
  if ((fp3=fopen(fn3,"r+t"))==NULL)
    { printf("cannot open fn3 ! \n");
      exit(0);
      }
  i=0;
  while (fscanf(fp3,"%d",&code)! =EOF)
    { j=code;
      y=j/100;
      x=j-y*100;
      if (y<7) m=(y-1)*94+(x-1);
      else m=(y-7)*94+(x-1);
      k=(long)m*(long)kk;
      fseek(fp1,k*sizeof(char),SEEK_SET);
      fread(p,sizeof(char),kk,fp1);
      k=(long)i*(long)kk;
      fseek(fp2,k*sizeof(char),SEEK_SET);
       fwrite(p,sizeof(char),kk,fp2);
       i=i+1;
      }
  fclose(fp1); fclose(fp2); fclose(fp3);
  free(p);
  return;
  }
```

三、原汉字库的汉字点阵信息以列为主存放且6行空白区未被压缩

在这种情况下，根据给定的汉字区位码，从原汉字库中取出汉字点阵信息，然后将这些点阵信息转换为以行为主的存放形式，最后再依次存放在新建的汉字库中。

其中区位码为 code 的汉字的点阵信息在原汉字库中的序号为

$$m=((code/100)-1)*94+(code\%100)-1$$

而汉字点阵信息的第一个字节在原汉字库中的偏移量为

$$k=m*kk$$

其中 kk 为一个汉字点阵信息所占的字节数。

本函数要调用函数 pch()，它的功能是将以列为主存放的汉字点阵信息转换为以行为

主的点阵结构。

函数 hzlibc1()中的形参意义如下：

num——整型变量。原汉字库中汉字点阵的规格。当 num=1 时，表示 16×16 点阵，一个汉字点阵信息占 32 个字节；当 num=2 时，表示 24×24 点阵，一个汉字点阵信息占 72 个字节；当 num=3 时，表示 32×32 点阵，一个汉字点阵信息占 128 个字节。其它情况可以此类推。

fn1——字符串指针。指向原汉字库的文件名。

fn2——字符串指针。指向用户新建汉字库的文件名。

fn3——字符串指针。指向由各汉字区位码组成的文本文件名。在此文件中，存放用户所选定的部分汉字的区位码，各区位码之间用空格分隔（参看后面的例）。汉字区位码在此文件中的序号即为该汉字在新建后的汉字库中的代码。

函数 pch()中的形参意义如下：

num——整型变量。原汉字库中汉字点阵的规格，意义同函数 hzlibc1()中的形参 num。

p——字符数组。按字节存放以列为主存放的一个汉字点阵信息。

q——字符数组。按字节存放经转换后的一个汉字点阵信息（即以行为主存放）。

函数程序如下(文件名为 hzlibc1.c)：

```
#include "stdlib.h"
#include "stdio.h"
void hzlibc1(num,fn1,fn2,fn3)
char *fn1,*fn2,*fn3;
int num;
{ FILE *fp1,*fp2,*fp3;
  int code,i,j,m,x,y,kk;
  long int k;
  char *p,*q;
  void pch();
  kk=(num+1)*(num+1)*8;
  p=malloc(kk*sizeof(char));
  q=malloc(kk*sizeof(char));
  if ((fp1=fopen(fn1,"r+b"))==NULL)
    { printf("cannot open fn1 ! \n");
      exit(0);
    }
  if ((fp2=fopen(fn2,"w+b"))==NULL)
    { printf("cannot open fn2 ! \n");
      exit(0);
    }
  if ((fp3=fopen(fn3,"r+t"))==NULL)
    { printf("cannot open fn3 ! \n");
      exit(0);
    }
  i=0;
  while (fscanf(fp3,"%d",&code)! =EOF)
    { j=code;
      y=j/100;
      x=j-y.*100;
      m=(y-1)*94+(x-1);
```

```
        k=(long)m * (long)kk;
        fseek(fp1,k * sizeof(char),SEEK_SET);
        fread(p,sizeof(char),kk,fp1);
        pch(num,p,q);
        k=(long)i * (long)kk;
        fseek(fp2,k * sizeof(char),SEEK_SET);
        fwrite(q,sizeof(char),kk,fp2);
        i=i+1;
      }
  fclose(fp1); fclose(fp2); fclose(fp3);
  free(p); free(q);
  return;
}

#include "math.h"
static void pch(num,p,q)
char p[],q[];
int num;
{ int  i,j,k,m,n,l,ii,jj;
  char  tt,d[8]={128,64,32,16,8,4,2,1};
  jj=(num+1) * 8;
  ii=jj * (num+1);
  for (i=0;  i<=ii-1;  i++)  q[i]=(char)0;
  for (l=0;  l<=num; l++)
    { n=0; k=0;
      for  (i=0; i<=jj-1; i++)
        { for (j=0; j<=7;  j++)
            { m=n+(num+1) * j+l; tt=d[j];
              if ((p[i+jj * l]&tt)! =(char)0)
                { tt=d[k]; q[m]=q[m]|tt;}
            }
          n=n+jj;
          if  (n==ii) n=0;
          if  (((i+1)%(num+1))==0) k=k+1;
          if  (k==8) k=0;
        }
    }
  return;
}
```

四、原汉字库的汉字点阵信息以列为主存放且6行空白区被压缩

在这种情况下，根据给定的汉字区位码，从原汉字库中取出汉字点阵信息，然后将这些点阵信息转换为以行为主的存放形式，最后再依次存放在新建的汉字库中。

区位码为code的汉字的点阵信息在原汉字库中的序号为：

当(code/100)<7时，

$$m=((code/100)-1)*94+(code\%100)-1$$

当(code/100)≥7时，

$$m=((code/100)-7)*94+(code\%100)-1$$

而汉字点阵信息的第一个字节在原汉字库中的偏移量为