董 事 长：周慕昌　　社　长：孙毓林
主　　编：孙毓林(兼)　副主编：王锡生
编 辑 部：张延淑　庄永祺　肖艳　商波
公关广告部：杨红(主任)　邢卫红　高宁　陈静
主　　办：电子工业部计算机与微电子发展研究中心
编辑出版：《软件世界》杂志社
地　　址：北京复兴路乙 20 号
通讯地址：北京 162 信箱(100036)
编辑部电话：821－2233 转 3431 或 5048
公关广告部电话：8283945
印　　刷：电子部科技情报所印刷厂
国内总发行：北京报刊发行局
订　　购：各地邮局
邮发代号：82－469
刊　　号：ISSN 1005-2348/CN11-3394
广告许可证号：京海工商广字 004 号
每期定价：2.00 元　全年定价：24 元
出版日期：1995 年 4 月 21 日

软件世界

SOFTWARE WORLD

1995 年　第 4 期　(总第 98 期)

目　录

MAIN CONTENTS

原书缺页

原书缺页

原书缺页

原书缺页

原书缺页

原书缺页

原书缺页

原书缺页

随着计算技术的发展，数字图像的压缩编码技术在数据传输和图像存储方面越来越显出其必要性。因此，图像压缩编码将成为数字图像处理领域中的一项必不可少的技术。为此，本刊自本期开始刊出"图像格式中常用的压缩方法"一文，该文分为两大部分，每部分分成两篇，每篇介绍4到5种压缩方法，同时附程序两套，最后一篇还将所有方法加以总结，给出每类方法通常运用于何种图像格式，其两部分的内容为：

1. RLE 压缩系列族(分两篇，共9种压缩方法)；

2. 高效的压缩方法(分两篇，共6种压缩方法)。

图像格式中常用的压缩方法
——RLE 压缩系列族(一)

国防科技大学 薛 滨

在通常的 PC 环境和工作站平台上，我们接触到的往往是一些常见的、标准的、用于位图格式的软件压缩方法，它们分别存在于位图图形、图像的各类典型格式中，并且每一格式一般都不止使用一种的压缩方法。本文将系统地介绍图像中常见的压缩方法并给出部分压缩与解压的程序。

行程长度压缩(Run Length Encode，简称 RLE)是压缩方法中使用范围最广，家族成员最大的系列，由它衍生的压缩方法非常多并且彼此间存在相当密切的关系，因此只要了解其中一两种，其它 RLE 压缩方法均可由此类推。

一、PCX 文件的 RLE 压缩

PCX 文件使用一种标准的 RLE 压缩，该压缩方法可以作为压缩效率和解压缩速度的一种折衷。文件数据是由一系列包含关键字节和数据字节的包组成。由这两类字节可构成两个域，即串域和字节游程域。

如果关键字节的两个高位(第6、7位)没有设置，则字节按原样写入图像，换句话说，只要串的所有字节中的图像数据能放在每个字节的低6位中，则不需要关键字节。这就是串域结构。

字节游程域有两类，第一类：如果关键字节的高两位已设置，则当前图像包的索引在6个低位中，文件的下一个字节是重复索引指定的次数。因为索引仅可以使用6位，故文件的字节游程域最大仅能有63个字节，更长的域需两个包。第二类为高位字节为1的串数据。每个这样的字节作为一个字节游程包存储，其长度为1。这样，如果有一些完全是任意数组成的数据，它的高位均设置为1，则这些数据所需的字节数比它们的原始形式要多一倍。这是很不正常的现象，并不是所有的数据都这样处理，然而这些长度游程编码系统确实经常生成非常大的文件。

标准 RLE 压缩的解包过程很简单，可描述如下：

对于每一个扫描行

对于每一位平面

取一个字节

最高两个位＝1？

如果是，本字节剩下的6位就是行程长度压缩计数值 N。下一个字节为数据，读出它并重复 N 次。

如果不是，则本字节为一个单独的数据值。

如果恢复的数值总量多于从头中读出的"每行字节数"，则复位字节计数器从下一个位平面开始。

所有位平面数据都恢复后，开始下一个扫描行。

当恢复的字节总量(所有行，所有位平面)超过计算出的图像大小时，退出。

整理图像右边界，使之与计算的 X 方向像素数目一致。

注意这种压缩方法的限制：

·数据可能不准超过每个位平面8位。

·由于6位的行程长度计数器，行程长度最大值为64。

·由于数据是在字节边界压缩的，并且每个扫描行

的字节数必须是偶数，所以在每个扫描行的末尾处常有额外的像素。

对于标准RLE打包的过程，它与解包过程相逆并且十分简单，具体的解压缩操作参见本文所附的程序。

二、紧缩位压缩(Packbit)

紧缩位是对于二级(灰度)扫描和绘图类型文件的一种简单但往往是有效的压缩方法。它是RLE压缩方法族中应用范围最广的一个压缩方式。例如，在Macintosh机上所使用的MacPaint图像格式；支持惠普打印机位图的PCL格式；Targa图像格式；TIFF图像格式等均提供了支持Packbit压缩的编码号。

紧缩位的方法是面向字节的，因此不存在字的调整问题，而且它有较好的最坏情况性能(对于每隔128个输入字节最多附加1个字节)。

解压缩的伪代码段如下所列：

循环直到你得到你希望的解压缩字节数：

将下一源字节读入n

如果n的第七位为0，则逐个复制下面的(n&0x7f)+1个字节；

否则如果n的第七位为1，且不为128则复制下一字节(n&0x7f)+1次

否则若n为128，退出循环

循环结束

在压缩(打包)过程中，3个或3个以上重复的字节总是编码为重复行程；2个字节重复的行程可编码为重复行程，也可编码为两个实行程，但一般均采用前者。

因此对于上述算法，它的一些规则如下：

·每行必须分别压缩，不压缩交叉行的边界。

·在每行末，压缩的字节数定义为(ImageWidth+7)/8，如果每行的"未压缩位"要求含有偶数的字节数，将解压缩到字边界缓冲区中。

·如果一个行程大于128字节，那么简单地把行程的剩余部分编码为一个或更多的附加的重复行程。

三、Silicon Graphics RLE压缩

Silicon Graphics RLE格式是能处理16位像素的紧缩位压缩的一种扩展，因此它也可以称为"16位的Packbit压缩"。如果像素大小是8位或更少，则这个格式与紧缩位相同。如果像素大小是16位，则描述字节仍为8位，但像素值为16位。例如：

83 1234 5678 9ABC 04 2468

表示有三个字节像素1234、5678和9ABC，后面跟着三个值为2468的像素。

四、Thunderscan RLE和增量压缩

这种压缩方法采用RLE和增量(像素到像素的区别)码一起编码4位像素。每个字节中，高2位(D7,D6)包含类型码，而低6位(3FH)则包含数据。

如果类型码是3，则数据的低4位是像素值；如果类型码是2，则数据是3位长的增量码，其中第一个码在高3位(38H)，第二个码在低3位(07H)。每一个增量码定义一个与行中前一个像素有关的像素，如表中第一排所定义。忽略值为4的增量码；如果类型码为1，数据是3个2位增量码，第一个码是在高2位(30H)，第二个码在中间2位(0CH)，第三个码在低2位(03H)。每个码定义一个与行中前一个像素相关的像素，如表中第二排所定义。忽略值为2的类型码；类型码0意味着把一行中前一行像素重复若干次，数据就是重复次数，重复次数0意味着无操作。

类型码为3			低4位
1	1	未用	低四位为像素值

类型码为2		高3位	低3位
1	0	增量码的第一码	增量码的第二码

类型码为1		高2位	中2位	低2位
0	1	增量第一码	增量第二码	增量第三码

类型码为0		低6位
0	0	将前一行像素重复的次数

Thunderscan 3位增量码

增量码	含义
0	与前一个相同
1	前一个+1
2	前一个+2
3	前一个+3
4	跳过这个码
5	前一个-3
6	前一个-2
7	前一个-1

Thunderscan 2位增量码

增量码	含义
0	与前一个相同
1	前一个+1
2	跳过这个码
3	前一个-1

五、另一种行程长度编码

对于二值(黑白)图像，还有一种行之有效的压缩方法。这样的图像数据以字节对输出。每对的第一个字节为重复计数，第二个字节为要重复的数据字节。重复次数要比重复计数多1，所以计算值0表示数据字节只用一次，255表示使用256次。

例如，一行有80个黑色位，40个黑白交替位和64个白色位，该行送出为：09 FF 04 AA 07 00

第一个计数值是09，表示10个字节(80位)的值为FFH，全黑色。第二个计数值为04，表示5个字节的0和1交替，即16进制的AAH。第三对是8个字节(64位)的00H。

该方法主要用于非常有特点的图像，该图像中的

一个字节在一个图像的较大部分中连续出现。如果字节只出现一次，也必须送一个字节对，这就有可能使“压缩后”的图像大于原图像。对于大多数二值图像，该方法能很好地压缩图像。

```
// A program to look at MAC pictures
#include<stdlib.h>
#include<stdio.h>
#include<string.h>
#include<fcntl.h>
#include<dos.h>
#include<io.h>
#include<bios.h>
#include<conio.h>

int video_mode;
int ScreenWidth;

void UnpackMacFile(FILE * fp);
int Packbit_Decode(char * p,FILE * fp);
int Packbit_Encode(char * p,FILE * fp);
void initgraph(int mode);
void color_make(int i,int R,int G,int B);
void putpixel(int x,int y,int color);
int getpixel(int x,int y);

int main(int argc,char * argv[]){
    FILE * fp;
    if(argc>1){
      if((fp=fopen(argv[1],"rb"))!=NULL)
          UnpackMacFile(fp);
      else printf("Error opening %s.\n",argv[1]);
    }else{
        puts("MAC file displayer V1.0");
        puts("Usage:MAC<filename.mac>");
    }
    return 0;
}
void UnpackMacFile(FILE * fp){
    int i,j,xi,xj;
    char far * ptr, * ptrdisp;
    unsigned char move,letter;
    video_mode=0x6Z;
    ScreenWidth=1024;
    if(fread(MACheader,1,640,fp)==640){
    if(!strncmp(MACheader+0x0041,"PNTG",4))
{
        initgraph(video_mode);
        color_make(1,0,0,0);
        color_make(0,63,63,63);
        ptr=(char * )malloc(72);
        ptrdisp=(char * )malloc(576);
        fseek(fp,640L,SEEK_SET);
        for(j=0;j<720;j++){
          if(Packbit_Decode(ptr,fp)==72){
            for(xi=0;xi<72;xi++){
              letter=ptr[xi];
              for(xj=0;xj<8;xj++){
                  move=0x80>>xj;
                  if((letter&move)!=0)
                      ptrdisp[xi*8+xj]=1;
                  else ptrdisp[xi*8+xj]=0;
              }
            }
          }
          for(i=0;i<576;i++)putpixel(i,j,
ptrdisp[i]);
        }
        getch();
        initgraph(3);
      }
      else printf("Not a MacPaint file.\n");
    }else printf("Error in reading MAC file head.\
n");
}
int Packbit_Decode(char * p,FILE * fp){
    int c,i,n=0;
    do{
      c=fgetc(fp)&0xff;
      if(c&0x80){
        i=((~c)&0xff)+2;
        c=fgetc(fp);
        while(i--)p[n++]=c;
      }else{
        i=(c&0xff)+1;
        while(i--)p[n++]=fgetc(fp);
      }
    }while(n<72);
    if(c==EOF)n=0;
    return(n);
}
int Packbit_Encode(char * p,FILE(fp){
    char b[72];
    unsigned int bdex=0,i=0,j=0,t=0;
    int destbytes=576;
    do{
      i=0;
      while((p(t+i)=p(t+i+1))&&((t+i)<
71))++i;
      if(i>0){
        if(bdex){
          fputc(((bdex-1)&0x7f),fp);
          j+=1;
          fwrite(b,1,bdex,fp);
          j+=bdex;
          bdex=0;
        }
        fputc((~i+1),fp);
        fputc(p(t+1),fp);
        j+=2;
        t+=(i+1);
      }else b(bdex++)=p(t++);
    }while(t<destbytes);
    if(bdex){
      fputc((bdex-1)&0x7f,fp);
      j+=1;
      fwrite(b,1,bdex,fp);
      j+=bdex;
      bdex=0;
    }
    return(j);
}

void initgraph(int mode){
    union REGS r;
    r.h.ah=0;
    r.h.al=mode;
    int86(0x10,&r,&r);
}
void color_make(int i,int R,int G,int B){
    union REGS r;
    r.h.ah=0x10;
    r.h.al=0x10;
    r.x.bx=i;
    r.h.dh=R;
  r.h.ch=G;
  r.h.cl=B;
  int86(0x10,&r,&r);
}

void putpixel(int x,int y,int color){
    char far * ptr;
    int vofs,vseg;
    long vadd;
    vadd=ScreenWidth * (long)y+(long)x;
    vseg=(int)(vadd>>16);
    vofs=(int)vadd;
    ptr=(char * )MK_FP(0xa000,vofs);
    outportb(0x3c4,0x0e);
    outportb(0x3c5,vseg  2);
    * ptr=color;
}
int getpixel(int x,int y){
    char far * ptr;
    int vofs,vseg;
    long vadd;
    vadd=ScreenWidth * (long)y+(long)x;
    vseg=(int)vadd>>16);
    vofs=(int)vadd;
    ptr=(char * )MK_FP(0xa000,vofs);
    outportb(0x3c4,0x0e);
    outportb(0x3c5,vseg  2);
    return(* ptr);
}
```

用于 PCX 文件浏览和存储

```
#include<stdlib.h>
#include<stdio.h>
#include<string.h>
#include<fcntl.h>
#include<dos.h>
#include<io.h>
#include<conio.h>

typedef struct {
    char manufacturer;
    char version;
    char encoding;
    char bits_per_pixel;
    int xmin,ymin;
    int xmax,ymax;
    int hres;
    int vres;
    char palette[48];
    char reserved;
    char colour_planes;
    int byte_per_line;
    int palette_type;
    char filler[58];
}PCXHEAD;
PCXHEAD header;
int width,depth;
```

```
int bytes,
int video_mode
int ScreenWidth;
char palette[768];

void UnpackPcxFile(FILE *fp);
void PackPcxFile(FILE *fp);
int RLE_PcxDecode(char *p,FILE *fp);
int RLE_PcxEncode(char *p,FILE *fp);
void initgraph(int mode);
void putpixel(int x,int y,int color);
int getpixel(int x,int y);
void setVGApalette(char *p);
void getVGApalette(char *p);

int main(int argc,char *argv[])
{
    FILE *fp, *fo;
    if(argc>1){
      fp=fopen(argv[1],"rb");
      fo=fopen(argv[2],"wb");
      if(fp!=NULL)&&(fo!=NULL)){
        UnpackPcxFile(fp);
        PackPcxFile(fo);
        getch();
        initgraph(3);
      }
      else printf("Error in opening %s or
        creating %s.\n",argv[1],argv[2]);
    }else{
      puts("PCX file displayer V1.0");
      puts("Usage:PCX<filename.pcx>
        <savefilename.pcx>");
    }
    return 0;
}

void UnpackPcxFile(FILE *fp){
  register int i,j;
  char *ptr;
  video_mode=0x5d;
  ScreenWidth=640;
  if(fread((char *)&header,1,sizeof(PCXHEAD),
    fp)==sizeof(PCXHEAD)){
    if(header.manufacturer==0x0a&&
      header.version==5){
      if(!fseek(fp,-769L,SEEK_END)){
        if(fgetc(fp)==0x0c &&
          fread(palette,1,768,fp)==768){
          fseek(fp,128L,SEEK_SET);
          width=(header.xmax-
            header.xmin)+1;
          depth=(header.ymax-
            header.ymin)+1;
          bytes=header.byte_per_line;
          initgraph(video_mode);
          setVGApalette(palette);
          ptr=(char *)malloc(width);
          for(j=0;j<depth;++j){
            RLE_PcxDecode(ptr,fp);
            for(i=0;i<width;++i)
              putpixel(i,j,ptr[i]);
          }
          free(ptr);
        }else puts("Error reading palette");
      }else puts("Error seeking to palette");
    }else printf("Not a 256 color PCX file.\n");
  }else printf("Error reading PCX file)
        header.\n");
    fclose(fp);
}

void PackPcxFile(FILE *fp)
{
  char *ptr;
  register int i,j;
  memset((char *)&header,0,sizeof(PCXHEAD));
  header.manufacturer=0x0a;
  header.version=5;
  header.encoding=1;
  header.bits_per_pixel=8;
  header.xmin=0;
  header.ymin=0;
  header.xmax=width-1;
  header.ymax=depth-1;
  header.colour_planes=1;
  header.byte_per_line=bytes;
  header.palette_type=2;
  fwrite((char *)&header,1,
    sizeof(PCXHEAD),fp);
  ptr=(char *)malloc(width);
  width=640;
  depth=480;
  bytes=640;
  for(j=0;j<depth;j++){
    for(i=0;i<width;i++)
      ptr[i]=getpixel(i,j);
      RLE_PcxEncode(ptr,fp);
  }
  free(ptr);
  getVGApalette(palette);
  fputc(0x0c,fp);
  fwrite(palette,1,768,fp);
  fclose(fp);
}

int RLE_PcxDecode(char *p,FILE *fp){
  int n=0,c,i;
  memset(p,0,bytes);
  do{
    c=getc(fp)&0xff;
    if((c&0xc0)==0xc0){
      i=c&0x3f;
      c=fgetc(fp);
      while(i--)p[n++]=c;
    }
    else p[n++]=c;
  }while(n<bytes);
  return(n);
}

int RLE_PcxEncode(char *p,FILE *fp){
  char b[64];
  unsigned int j=0,i=0,t=0;
  do{
    i=0;
    while((p[t+i]==p[t+i+1])&&
      ((t+i)<bytes)&&(i<63))i++;
    if(i>0){
      fputc(i|0xc0,fp);
      fputc(p[t],fp);
      t+=i;
      j+=2;
    }else{
      if((p[t]&0xc0)==0xc0){
        fputc(0xc1,fp);
        j++;
      }
      fputc(p[t++],fp);
      j++;
    }
  }while(t<bytes);
  return(j);
}

void initgraph(int mode){
  union REGS r;
  r.h.ah=0;
  r.h.al=mode;
  int86(0x10,&r,&r);
}

void putpixel(int x,int y,int color){
  char far *ptr;
  int vofs,vseg;
  long vadd;
  vadd=ScreenWidth*(long)y+(long)x;
  vseg=(int)(vadd>>16);
  vofs=(int)vadd;
  ptr=(char *)MK_FP(0xa000,vofs);
  outportb(0x3c4,0x0e);
  outportb(0x3c5,vseg^2);
  *ptr=color;
}

int getpixel(int x,int y){
  char far *ptr;
  int vofs,vseg;
  long vadd;
  vadd=ScreenWidth*(long)y+(long)x;
  vseg=(int)(vadd>>16);
  vofs=(int)vadd;
  ptr=(char *)MK_FP(0xa000,vofs);
  outportb(0x3c4,0x0e);
  outportb(0x3c5,vseg^2);
  return(*ptr);
}

void setVGApalette(char *p){
  union REGS r;
  struct SREGS sr;
  int i;
  for(i=0;i<768;i++)p[i]=p[i]>>2;
  r.x.ax=0x1012;
  r.x.bx=0;
  r.x.cx=256;
  r.x.dx=FP_OFF(p);
  sr.es=FP_SEG(p);
  int86x(0x10,&r,&r,&sr);
}

void getVGApalette(char *P){
  union REGS r;
  struct SREGS sr;
  int i;
  r.x.ax=0x1017;
  r.x.bx=0;
  r.x.cx=256;
  r.x.dx=FP_OFF(p);
  sr.es=FP_SEG(p);
  int86x(0x10,&r,&r,&sr);
  for(i=0;i<768;++i)p[i]=p[i]<<2;
}
```

VGA 256色作图技术及实现方法分析

山东工程学院　周升锋　周长城　张　建　李振路

摘要　本文对256色图形模式(13H)作了简要分析，对该模式下作图的常用编程方法作了较详细的讨论，并以画点为例通过实际运行比较了执行效率。

视频图形阵列 VGA 是 IBM 公司于1987年推出的产品，现已成为 PS/2系列微机的标准。由于它兼容了 CGA 及 EGA 的显示模式，并且可以以更高的分辨率或更多的颜色显示图形，因此它在图形显示、图像处理等方面获得了广泛的应用。VGA 使用模拟显示器，能够方便地实现256种颜色同时显示。分辨率为320×200。虽然分辨率降低，但因同时能够显示的颜色数量增多，所以在图形图像处理及编写界面程序方面时仍经常用到。诸多软件产品(如 Turbo 系列)具有 EGA/VGA 各模式下的图形库函数，但由于 VGA 256色模式编程的特殊性，还没有为这种模式专门开发图形库函数和过程，而 VGA 256色模式作图是经常需要的。为此，本文以画点为例对该模式作图的常用方法作了讨论。由于 C 语言高效、易读，所以文中的子函数均按 Turbo C 调用格式编写。

一、256色视频模式

1. 显示存储器

VGA 256色图形模式即是模式13H，屏幕分辨率为320×200个像素。13H 模式要能同时显示256种颜色，需一个整字节(8位)表示一个像素值，图1是该模式的显示存储映象，其基地址为 A000:0000H，屏幕与显示存储器是线性映射关系，屏幕上一个像素对应于显示存储器一个字节，屏幕上像素点(x,y)与显示存储器中对应字节的地址关系如：

字节地址=x+320*y

其中：x:0－319(水平)，y:0－199(垂直)模式13H时的存储映象如下：

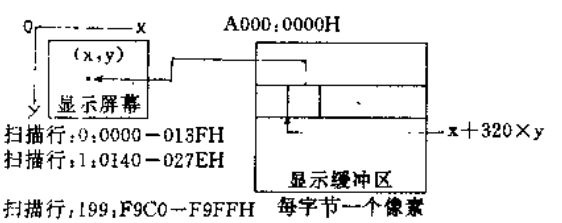

图1　VGA 256色模式(13H)显示存储映象

2. 视频 DAC

VGA 系统另外增加了一个数字模拟变换器 DAC (Digital Analog Converter)，它把数字彩色信息转换成模拟的红、绿、兰三色信号，送给显示器。DAC 提供一张彩色查询表，查询表把8位的彩色值，通过三路 DAC 变成三路模拟彩色信号，每路有6位分辨率，即64级，VGA 有256个色彩寄存器，每个有18位长，红、绿、兰分量各6位，因红绿兰每色可取64种不同值所以色彩总数共有：64×64×64＝262144种，256个寄存器均有初值，通过修改的三色分量，可改变其所代表的颜色。

二、作图的编程方法分析

1. 调用 BIOS 中断

在 IBM 及其兼容机中，基本输入输出系统(BIOS)是一组低层固化程序，视频 BIOS 为视频显示功能提供服务。VGA 的 BIOS 为16K 字节长，放在处理器存储空间的 C000:0000H～C000:3FFFH 之间。VGA BIOS 的0CH 号功能用来向指定页面中指定位置画点，入口参数为：

AH＝0CH，BH＝页面号(通常取0页)

AL＝像素值，CX＝横坐标，

DX＝纵坐标

调用 BIOS 中断编程简便，但速度稍慢(见表1)。

(1)　用 C 语言直接调用 BIOS 中断

```
PutPt1(int x,int y,char Color){
  union REGS Regs;                    /*  定义联合类型  */
  Regs.h.ah=0x0c;
  Regs.h.al=Color;                    /*  入口参数送寄存器  */
  Regs.x.cx=x;
  Regs.x.dx=y;
  Regs.h.bh=0;
  int86(0x10,&Regs,&Regs);            /*  执行10H 中断  */
}
```

在 Turbo C 的集成环境下可直接编译。

(2)　C 语言中嵌入汇编

```
PutPt2(int x,int y,char Color){
  asm mov ah,0ch;
  asm mov al,Color;
  asm mov cx,x;
  asm mov dx,y;
  asm mov bh,0;
```

```
asm int 10h;
}
```

在C语言中嵌入汇编后不能在集成环境下直接编译，必须用TCC命令行编译程序：TCC -B 文件名 〔库文件名〕

(3) C调用汇编语言(混和语言编程)

Turbo C调用汇编程序的参数传递是通过栈进行的，一个简单的汇编语言模式如下，先将该汇编语言子程序段编译成OBJ文件，然后同调用它的函数通过建立工程文件(PRJ)编译连接。假定该程序段名为41.ASM，调用它的函数名为40.C，则可建立如下工程文件(DRAWP.PRJ)：40.C,41.OBJ

然后在Turbo C的集成环境下编译：

```
        PUBLIC _PutPt3
_TEXT   SEGMENT BYTE PUBLIC 'CODE'
        ASSUME CS:_TEXT,DS:_TEXT
_PutPt3 PROC    NEAR
        PUSH    BP
        MOV     BP, SP
        MOV     CX, 〔BP+4〕;入口参数送寄存器
        MOV     DX, 〔BP+6〕
        MOV     AL, 〔BP+8〕
        MOV     AH, 0CH
        MOV     BH, 0
        INT     10H
        POP     BP
        RET
_PutPt3 ENDP
_TEXT   ENDS
        END
```

2. 硬件编程(直接写显示缓冲区VRAM)

通过指定段地址和偏移量指向VRAM的地址，即可直接向显示缓冲区写数据，下列画点函数以嵌入C的汇编形式写出：

```
PutPt4(int x,int y,char Color){
  asm mov ax, 0a000h;
  asm mov es, ax    ;
  asm mov as, 320   ;
  asm mov bx, y     ;
  asm mul bx        ;
  asm add ax, x     ;
  asm mov di, ax    ;
  asm mov al, Color;
  asm mov es:〔di〕, al;
}
```

从附表的执行效率可见，直接写显示缓冲区画点的速度是很快的，如果单纯用C语言编写，可用movedata()函数来实现：

```
PutPt5(int x,int y,char Color){
  unsigned char Colorp〔1〕;
  unsigned offaddr=0;
  Colorp〔0〕=Color;
  offaddr=320*y+x;
  movedata(FP_SEG(Colorp),FP_OFF(Colorp),
      0x0a000,offaddr,1);
}
```

如将段地址和偏移量写成绝对地址，则执行效率提高，同汇编语言速度差不多(PutPt5中Colorp为指针变量，对其赋以多个值，可画水平线，从而提高速度)：

```
PutPt6(int x,int y,char Color){
  char far *p;
  p=(char far *)(0x0a0000000L);
  *(x+y*320+p)=Color;
}
```

本文只是对画点问题进行了讨论，旨在对编程方法上提供一点参考，要开发真正的图形，还必须与良好的算法相结合，以达到高效率、高质量。程序是在386SX下运行的，软件环境为：MS-DOS 3.31，Turbo C 2.01，MASM 5.10。

表1 各函数执行效率表

函数名	EXE文件长度B (含调用程序)	循环调用32000次时间(秒)	COM文件长度
PutPt1	24559	3.6264	……
PutPt2	20388	1.8681	……
PutPt3	24416	1.8681	……
PutPt4	20404	0.3296	……
PutPt5	24651	0.7143	……
PutPt6	24529	0.3296	……

(上接第62页)

的功能，用户可不再为寻找电脑中的某个文件发愁。

当我听到联想OFFICE这个名字的时候，我的第一个反映就是它与Microsoft的OFFICE有何不同呢？汉字系统事业部的张勇先生告诉我，联想OFFICE与Mircosoft的OFFICE在功能上差不多，如微软的Word、Excel、PowerPoint等功能联想OFFICE中都有。但是二者的最大的区别是微软的OFFICE只是内核上的中文化，而没有考虑到国人的习惯，还带着"洋"味。从这个意义上来讲，方正盘古组件是联想OFFICE的主要竞争对手。谈到方正盘古组件的特点时，张勇先生很客观地承认盘古组件的双域电子表是其一大特点，与之相比，联想OFFICE的电子表功能就显得相对薄弱一些了。在软件盗版的今天，联想OFFICE并不想通过"加密"软件这条路来保护自己的利益。而是通过服务来控制版权，为每套软件分配一个登记号，用户可通过一条专门的服务热线来咨询，但得到解答的正确前提是报出登记号经核对正确的。

Windows 软件的加密技术

北京飞天瑞格软件研究所 刘传憬

摘要 本文从系统内部出发，介绍了几种流行的 Windows 软件的加密技术，并着重介绍了瑞格软件研究所的最新加密工具 Winlock5.0的核心技术与实现原理。

一、概 述

现在，由 DOS 转向 Windows 平台的程序员愈来愈多，随之而来，运行在 Windows 下的应用软件也飞速增多，相应地，Windows 应用软件的加密问题也提了出来。可是，Windows 对于 DOS 来说，几乎是一个全新的系统，以往的 DOS 软件加密工具全部无法适用于 Windows 软件。为了保护软件开发者的合法权益，北京瑞格软件研究所率先涉足 Windows 软件加密领域，从1993年11月起，陆续推出了5个版本的 Windows 软件加密工具：从第1版的加密卡到第2版的加密狗，以及第3、4版的增强型加密狗和第5版的密纹磁盘加密工具。经过数千用户的使用与测试，现已完全成熟，成为 Windows 软件加密工具中的佼佼者，笔者通过近一年的开发历程，系统分析了 Windows 的核心，摸清了 Windows 的运行机制，并吸收了不少国内外软件加密专家的经验，在此详细向各位读者介绍 Windows 软件加密的原理与实现技术。

二、Windows 执行程序加密的思想

大体来讲，与 DOS 可执行程序加密的思想相类似，一般的 Windows 可执行文件加密工具都采用下面的几种思路：

1. **变形法**：用预处理程序先对待加密软件进行变形，使之成为不可运行的变形程序，成为被"加密"的软件；想运行这个软件必须由专用的解码程序来还原变形，并调用执行，这种软件加密方法实现起来比较容易，但一般使加密后的软件的操作发生改变，而且解密起来也相对容易，保密性不佳。

2. **外壳法**：直接处理待加密软件，在原软件的外面罩上一层"外壳"，这层外壳在原软件运行前先得到执行，通过探测 KEY 的存在等各种用户合法性检查，判断继续运用否，达到软件加密的目的，这种方法实现起来相对困难，需要清晰了解 Windows 可执行文件的格式，而且还要解决外壳与原软件之间的连接问题，但如果采用了高级的反跟踪与变形技术，经加密后的软件不仅操作丝毫不用改变，而且还有很好的保密性。

3. **内嵌法**：这是最难实现的一种方法，加密工具需要扫描整个待加密软件，在其内部一个入口处嵌入检测 key 的模块，判断用户的合法性，以决定是否正常运行。其难点在于这个嵌入点的定位，既要保证该点必须在软件调入后就能执行到，还要保证从该点插入代码后，不影响原软件的功能，这往往需要一种集统计决策和经验数据于一体的算法。

现在市面上的几种 Windows 加密工具，以采用变形法和外壳法的居多。而瑞格软件研究所的 Winlock4.0与5.0版，都同时使用了外壳法与内嵌法两种方法，即在原软件外面罩上一层加密外壳，在其内部还嵌入了一个检测模块，配以高效的反动态、静态跟踪技术，以及巧妙的代码变形，可以达到万无一失的效果。

下面就具体介绍一下 Winlock5.0的原理与实现技术

三、Winlock5.0的运行原理与实现技术

要想成功地对一个 Windows 软件同时进行外壳和内嵌加密，必须达到下面的要求：

1. 将一段探测 key 的程序罩在原软件的外面，或是嵌入其内部，并与原软件融为一体，成为一个新的 Windows 可执行文件。

2. 保证加密后的软件被调入时执行外壳程序及嵌入程序检测 key 的存在，以确认用户的合法性。

3. 采用反动态跟踪技术，使解密者使用动态调试工具跟踪不能顺利进行。

4. 采用代码与数据变形技术，使被加密软件和外壳(内嵌)中的测 key 程序不能被正常反汇编，以防止解密者的静态分析。

5. 检 key 程序要与原被加密软件完全融合在一起，不能对原软件的功能、操作及运行速度有影响。