实用技术

图形图象文件格式解码实用程序①(续4)

张明敏

4 Microsoft Windows 的 BMP 格式

在 Windows 下有许多种 BMP 文件的使用方法。最常见的用途是作为"墙纸"的源文件。除了作为墙纸之外,BMP 文件已成为不同的 Windows 应用程序之间的一种标准文件格式。虽然它还不可能完全替代其它文件格式,但它是处理图象的较方便、较通用的方法。

BMP 文件有两大特征使它区别于其它文件格式。首先它可以描述有多达 24 位彩色的图象。其次它的图象是以非压缩的方式存储的。因为 BMP 文件有一文件头,所以要比原始的图象数据多占一些存储空间。这看上去是比较浪费的,从某种程度上来说也确实如此。但是,对 BMP 文件格式来说,图象文件压缩有两个缺点。首先,读一压缩图象文件明显比读一原始图象文件需要更多的时间。当它们被当作墙纸使用时,从磁盘上读入 BMP 几乎不存在延误。把用作墙纸的 BMP 文件装入到系统中就不会明显减慢 Windows 的启动速度。这显然与以压缩形式存储墙纸的情况不同。第二个缺点,扫描所得的 24 位彩色图象很少有好的压缩效果,压缩后的长度常常比原有图象文件更长,所以存储非压缩的图象比存储压缩图象好。其它支持 24 位彩色图象的图象文件格式也会遇到同样问题。

BMP 格式常常不是作为档案性图象文件格式使用。用户通常并不用这种格式存储许多幅图象,它只是用来存储那些需要较短时间内检索就能获取的文件。

值得注意的是,用户所写的处理图象文件的程序不必一定要在 Windows 环境下处理 BMP 文件。BMP 文件可以在不运行 Windows 任何资源的情况下创建。基于 DOS 的应用程序有时也可能希望使用 Windows 环境下的 BMP 文件,就好像对 Windows 要使用 DOS 环境下的图形文件格式一样。

4.1 BMP 文件结构

虽然 BMP 各组成部分的所用的名字难于记忆,但是 BMP 是较容易处理的。没有涉及到压缩,不需要跟踪标签、记录或块,在使用中也没有任何秘密可言。

BMP 有很大的文件头。正如与 Windows 相关的所有数据结构那样,它的元素都由 Martian 命名,并且许多都是发不出音的。这里讨论的版本命名了事实上可使用的数据结构。剩下的许多内容都与这里讨论的内容无关。

```
typedef struct {
        char id[2];
        long filesize;
        int reserved[2];
        long headersize;
        long infoSize;
        long width;
        long depth;
        int biPlanes;
        int bits;
        long biCompression;
        long biSizeImage;
        long biXPelsPerMeter;
        long biYPelsPerMeter;
        long biClrUsed;
        long biClrImportant;
        } BMPHEAD;
```

首先,Windows 3.0 软件开发包将这个头定义

① 注:需源程序者请与浙江大学 CAD&CG 国家重点实验室张明敏联系
邮编:310027 电话:(0571)7951045,7990451

成一些较小的头。如果使用 BMP 文件，就没有理由将它们分开，因为 BMP 文件就是由较小的数据结构胶贴在一起的。

(1) id 域，总是包含 2 个字节的字符串"BM"，是"bitmap"的缩写.

(2) filesize 域定义文件的大小。注意它是长整数，在这种没有压缩存储的大图象的文件格式中，肯定常常要超出以前定义的整数范围。

(3) headersize 表示文件头中的字节数。事实上它定义了位图开始的偏移量。位图前的部分被认为是文件头。

这些域被 Windows 3.0 软件开发包当成是 BMP 文件的第一个头。它们常用名为 BITMAPFILEHEADER 的数据结构表示。这里讨论的文件头的剩余部分在 Windows 下称为 BITMAPINFOHEADER。

(4) infosize 是一个 BITMAPINFOHEADER 的字节数，是长整数，常常是 28H。

(5) width 和 depth 元素定义了文件中图象像素大小。注意，它们都是长整数；除非硬驱动器中有几千兆字节的空间，否则不能再做任何处理大 BMP 文件的操作。用户不能涉及到大小超过 65,535 像素的 BMP 文件，65,535 正是一个整数的极限。

(6) biPlanes 域常常为 1，它表示最终解码位图图象的目标显示设备的位数。它通常是不起作用的，但是它是 Microsoft 所需要的。

(7)bits 元素放源图象的颜色位数。它可以是 1,4,8 或 24。

(8)biCompression 域常常是 0L，它表示文件是非压缩的。BMP 文件格式的文档讨论了一种 BMP 图象数据的行程压缩方法。到 1992 年为止，这种方法还不能在 Windows 下使用。

(9)biSizeImage 域定义图象的字节数。可以想到这可由文件头中所有其它域计算出。

(10) bitPelsPerMeter 和 bitPelsPerMeter 域说明图象的分辨率，它与真正讨论的内容无关。

(11)biClrUsed 和 biClrImportant 域说明真正用在图象中的颜色数，颜色数是很重要的。例如，用户可以用 256 种可能的颜色创建 8 位 BMP 文件而只使用这 256 种颜色中的 200 种。如果 Windows 真正使用这些信息的话，在同一时刻，这些内容是不清楚的。如果将该域置为 0L，就要考虑使用所有有效的颜色。

接着是 BMP 文件头的固定部分，如果文件有一个调色板，那么就可以找到调色板，否则即为图象数据的开头。BMP 文件的彩色调色板存储方式与其它文件格式的调色板不同，但它总是以 RGB 的形式表示，每个入口都是 4 个字节长。这是因为调色板的入口是 4 的倍数时访问的速度较快——即只要左移两位即可——用真正的整数乘法时就较慢。更麻烦的是，每一入口 3 个彩色入口的顺序是相反的。

下面是 BMP 文件中一个颜色的入口：

```
typedef struct {
        char blue;
        char green;
        char red;
        char filler;
        } RGBQUAD;
```

BMP 文件中位图化的图象数据是以连续行的形式存储的。要显示的图象以相反的顺序存储，在这种情况下，文件中读出的第一行是图象最后一行。

打开一个单色的 BMP 文件是很简单的。假设已读了文件头，就很容易找到图象数据的开始(由 headersize 元素说明)，然后就可以一行一行地读出图象数据了。这里假设文件头是在一个称为 bmp 的 BMPHEAD 的对象中，文件指针 fp 开始指向图象数据的开头，还有一足够存放一图象行数据的缓冲区 b。

```
int i;
for(i=0;i<(int)bmp.depth;++i){
fread(b,1,pixels2bytes((int)bmp.width)fp);
putline(b,(bmp.depth-1)-i);
}
```

事实上，上面的描述过于简单化了，实际读入的字节数往往四舍五入为偶数(长整数)。单色图象行是以单个位平面位图的形式存放的，八位图象是以字节数组的形式存放，这样每个图象行的长度为 bmp. width 字节，24 位图像行长为 bmp. widthbmp. width * 3 字节。每一像素都是一个 RGB 值。注意，这些值不是存在 BMP 文件调色板使用的 4 字节 RGBQUAD 结构中。

4 位图象行的存储方式与 WPG 文件中所用的存储方式类似，即每一存储字节其低 4 位存一像素，高 4 位再存一像素。

BMP 文件内部结构的简单特性使得读者很容易了解 BMP 的工作过程。下面介绍处理 24 位 BMP 文件的方法。24 位彩色 BMP 文件可以支持的彩色比 VGA 卡 0BH 模式能显示的彩色多得多。为了获得较简单的解码和显示 24 位 BMP 文件的过程，则需要使用下面介绍的技巧。

本节介绍的 BMP 文件解码程序将 24 位的文件作为灰度图象显示。可以通过对 3 种成分颜色加权的方法将一个 RGB 的彩色值简化为灰度。结果将是一幅灰度图画,大致上就是彩色照片投射到黑白底片上所显示的颜色。其灰度计算公式如下:

gray=.30 * red+ * .59 * green+.11 * blue

虽然上面是用 C 语言中的浮点运算进行处理,但也可以使用整数运算,从而使计算一样有效。grayvalue 宏将一 RGB 颜色值转换为一灰度。

```
#define grayvalue(r,g,b)
(((r * 30)/100)+((g * 59)/100)+((b * 11)/100))
```

如果仅给出合法的 RGB 值,grayvalue 只能产生 256 种可用的灰度。因此,如果 VGA 卡的调色板是以 256 种不同的灰度进行设置的,那么对 24 位图象的每个三字节像素,使用 grayvalue 函数所得的结果将是灰度调色板的一个索引值数组。事实上,这种处理将使 24 位 RGB 描述的图象行转换成 8 位调色板驱动的图象行。

下面是用户创建调色板的方法:

```
int i;
for(i=0;i<256;++i)memset(palette+
(i * 3),i,3);
```

这里假设 palette 是一至少 768 字节长的缓冲区。注意,这里用到的处理 RGB 调色板或像素值的源代码使用常量来定义调色板入口大小以及 3 个颜色值的入口偏移量,如下所示:

```
#define    RGB_RED      0
#define    RGB_GREEN    1
#define    RGB_BLUE     2
#define    RGB_SIZE     3
```

使用这些常数事实上不便改变用户代码工作的方式,但它可使用户的程序更加可读。另外,用户可以将一幅使用调色板的彩色图象快速转换为灰度图象。在转换时,通过 grayvalue 传送调色板。如果 palette 是 768 字节的缓冲区,包含有图象的彩色查找表,使用以下方式可使整幅图象重映射成灰度图象。

```
int i,n;
for(i=0;i<256;++i)
  memset(palette+(i * RGB_SIZE),
    grayvalue(palette[i * RGB_SIZE+RGB_RED],
              palette[i * RGB_SIZE+RGB_GREEN],
              palette[i * RGB_SIZE+RGB_BLUE]),
    RGB_SIZE);
```

事实上图象中的每个实际值像素都没有改变。它们仍旧指向一个调色板表入口,但是调色板入口本身定义的彩色已经改为了灰度值。

正如在前面提到的那样,VGA 卡并不能真正显示 24 位真彩色图象。它们的调色板入口是以 6 位数字组成,而不是由以前讨论过的在 RGB 彩色相联系的 8 位数字组成。

由于 VGA 中的彩色寄存器只涉及到 6 位数字,所以实际上最多只能支持 64 种不同颜色强度,而不是在一真 RGB 颜色中所提到的 256 级。遇到一 256 种不同灰度的灰度调色板时,VGA 卡显示调色板中最前面的 4 个级别用屏幕上一种灰度显示,接下去的四级灰度也作为一个级别,等等。也就是说,VGA 卡把原有的彩色强度 I 变为 I/4(整除运算)。

因为 VGA 卡中的调色板只有 64 种不同的灰度,所以用户只需要提供 64 种灰度,剩下的 192 种彩色值可用在其它地方。用户可以通过简单地把灰度值右移 grayvalue 位来完成压缩灰度的工作。缺省的情况下是假设有一 256 级灰度调色板那样提供灰度。右移一位可以将调色板缩小为 128 级,右移 2 位可以将调色板缩小为 64 级,等等。

注意,每英寸 300 点的 PostScript 激光打印机通过半色调(halftoning)的方法来重定义灰度,它可以处理大约 32 种不同的灰度。

如果在安装了 VGA 卡的屏幕驱动程序机器上使用 Ventura Publisher 的 GEM 版本,并且设置 Define Color 框来产生灰度,输入灰度 TIFF 文件在屏幕上作为有较高清晰度的灰度照片显示时,这些图象只有 16 种灰度。

4.2 BMP 文件的解码程序

程序 READBMP 是一个简单的 BMP 文件解码程序,它与前面提到的解码其它文件格式的程序类似,很容易理解。unpackbmp 函数阅读所处理文件的第一部分(该部分是送给 BMPHEAD 对象的),并且从它里面抽出相关的数值。如果有调色板的话,它也阅读调色板信息。注意,它将 BMP 文件调色板入口的 RGBQUAD 结构转换为正常的 3 字节的调色板入口。然后将文件指针指到图像数据的开始并处理它。

READBMP 程序可显示可任何 Windows BMP 文件。值得注意的是,它不支持 BMP 格式的行程压缩特征,不能显示行程编码的 BMP 文件,它同样不

能处理 OS/2 PM (Presentation Manager) 内容的 BMP 文件。虽然它们都以 BMP 为扩充名,但它们的结构却不同。

```
/* * * * * * * * * * * * * * * * * * * * * * * * *
      BMP 解码程序
* * * * * * * * * * * * * * * * * * * * * * * * */
 main(argc,argv)
     int argc;
     char *argv[];
 {
     FILE *fp;
     static char results[8][16]={"Ok",
                                 "Bad file",
                                 "Bad read",
                                 "Memory error",
                                 };
   char path[81];
   int r;
     if(argc > 1) {
       strmfe(path,argv[1],"BMP");
       strupr(path);
       if((fp = fopen(path,"rb")) ! = NULL) {
         fi.setup=dosetup;
         fi.closedown=doclosedown;
         r=unpackbmp(fp,&fi);
         printf("\%s",results[r]);
         fclose(fp);
     } else printf("Error opening %s",path);
   } else puts("Argument:          path to a BMP file");
}
/* unpack a BMP file */
unpackbmp(fp,fi)
   FILE *fp;
   FILEINFO *fi;
{
   BMPHEAD bmp;
   char *p,*pr;
   int i,n;
   /* set a default monochrome palette */
   memcpy(fi->palette,"\000\000\000\377\377\377",
            6);
   /* get the header */
   if(fread((char *)&bmp,1,sizeof(BMPHEAD),fp)=
     =sizeof(BMPHEAD)) {
   /* check the header */
   if(! memcmp(bmp.id,"BM",2)) {
     /* set the details */
     fi->width=(int)bmp.width;
     fi->depth=(int)bmp.depth;
     fi->bits=bmp.bits;
     /* work out the line width */
     if(fi->bits==1) fi->bytes=pixels2bytes
               (fi->width);
     else if(fi->bits==4) fi->bytes=pixels2bytes
                (fi->width)<<2;
     else if(fi->bits==8) fi->bytes=fi->width;
     else if(fi->bits==24) fi->bytes=fi->
                  width * 3;
     /* round up to an even dword boundary */
     if(fi->bytes & 0x0003) {
       fi->bytes |= 0x0003;
       ++fi->bytes;
     }
     /* get the palette */
     if(fi->bits > 1 && fi->bits <=8) {.
       n=1<<fi->bits;
       for(i=0;i<n;++i) {
         fi->palette[(i * RGB_SIZE)+RGB_BLUE]
              =fgetc(fp);
         fi->palette[(i * RGB_SIZE)+RGB_
              GREEN]=fgetc(fp);
         fi->palette[(i * RGB_SIZE)+RGB_RED]=
              fgetc(fp);
         fgetc(fp);
       }
}
   else if(fi->bits==24) {
     for(i=0;i<256;++i)
       memset(fi->palette+(i * RGB_SIZE),i,
                    RGB_SIZE);
}
/* allocate a line buffer */
if((p=malloc(fi->bytes)) ! = NULL) {
  if((fi->setup)(fi) ! = GOOD_READ) {
    free(p);
      return(MEMORY_ERROR);
  }
  /* find the start of the image data */
  fseek(fp,bmp.headersize,SEEK_SET);
  /* read all the lines */
  for(i=0;i<fi->depth;++i) {
      if(fread(p,1,fi->bytes,fp) ! = fi->bytes) {
          freebuffer();
          free(p);
          return(BAD_READ);
      }
```

```
/* translate the line types into VGA */
switch(fi->bits) {
  case 1:
    pr=mono2vga(p,fi->width);
    if(pr != NULL) {
      putline(pr,fi->depth-1-i);
      free(pr);
    }
    else {
      freebuffer();
      free(p);
      return(MEMORY_ERROR);
    }
    break;
  case 4:
    pr=ega2vga(p,fi->width);
    if(pr != NULL) {
      putline(pr,fi->depth-1-i);
      free(pr);
    }
    else {
      freebuffer();
      free(p);
      return(MEMORY_ERROR);
    }
    break;
  case 8:
    putline(p,fi->depth-1-i);
    break;
  case 24:
    pr=rgb2vga(p,fi->width);
    if(pr != NULL) {
      putline(pr,fi->depth-1-i);
      free(pr);
    }
    else {
  freebuffer();
  free(p);
  return(MEMORY_ERROR);
  }
  break;
}
    }
  (fi->closedown)(fi);
  free(p);
  return(GOOD_READ);
  } else return(MEMORY_ERROR);
} else return(BAD_FILE);
} else return(BAD_READ);
}
/* convert a monochrome line into an eight bit line */
char *mono2vga(p,width)
      char *p;
      int width;
{
      char *pr;
      int i;
      if((pr=malloc(width)) != NULL) {
        for(i=0;i<width;++i) {
          if(p[i >> 3] & masktable[i & 0x0007])
            pr[i]=1;
          else
            pr[i]=0;
        }
        return(pr);
      } else return(NULL);
}
/* convert a four bit line into an eight bit line */
char *ega2vga(p,width)
      char *p;
      int width;
{
      char *pr;
      int i,j=0;
      if((pr=malloc(width)) != NULL) {
        for(i=0;i<width;) {
          pr[i++]=(p[j] >> 4) & 0x0f;
          pr[i++]=p[j] & 0x0f;
          ++j;
        }
        return(pr);
      } else return(NULL);
}
/* convert an RGB line into an eight bit line */
char *rgb2vga(p,width)
      char *p;
      int width;
{
      char *pr;
      int i;
      if((pr=malloc(width)) != NULL) {
        for(i=0;i<width;++i) {
          pr[i]=greyvalue(p[RGB_RED],p[
                          RGB_GREEN],p[RGB_
                BLUE]);
          p+=RGB_SIZE;
```

```
        }
    return(pr);
    } else return(NULL);
}
/* this function is called before an image is unpacked */
    dosetup(fi)
    FILEINFO *fi;
{
    union REGS r;
    if(! getbuffer((long)fi->width * (long)fi->
        depth,fi->width,fi->depth))return
            (MEMORY_ERROR);
    r.x.ax=0x0013;
    int86(0x10,&r,&r);
    setvgapalette(fi->palette,256,fi->background);
    return(GOOD_READ);
}
/* This function a called after an image has been un-
packed. It must display the image and deallocate memory.
*/
doclosedown(fi)
    FILEINFO *fi;
{
    union REGS r;
    int c,i,n,x=0,y=0;
    if(fi->width > SCREENWIDE) n=
    SCREENWIDE;
    else n=fi->width;
    do {
      for(i=0;i<SCREENDEEP;++i) {
        c=y+i;
        if(c>=fi->depth) break;
        memcpy(MK_FP(0xa000,SCREENWIDE*i),
                getline(c)+x,n);
      }
    c=GetKey();
    switch(c) {
      case CURSOR_LEFT:
        if((x-STEP) > 0) x-=STEP;
        else x=0;
        break;
      case CURSOR_RIGHT:
          if((x+STEP+SCREENWIDE) < fi->
            width) x+=STEP;
        else if(fi->width > SCREENWIDE)
            x=fi->width-SCREENWIDE;
        else x=0;
        break;
      case CURSOR_UP:
        if((y-STEP) > 0) y-=STEP;
        else y=0;
        break;
      case CURSOR_DOWN:
      if((y+STEP+SCREENDEEP) < fi->depth)
        y+=STEP;
        else if(fi->depth > SCREENDEEP)
          y=fi->depth-SCREENDEEP;
        else y=0;
        break;
      case HOME:
        x=y=0;
        break;
      case END:
        if(fi->width > SCREENWIDE)
          x=fi->width-SCREENWIDE;
        else x=0;
        if(fi->depth > SCREENDEEP)
          y=fi->depth-SCREENDEEP;
        else y=0;
        break;
    }
  } while(c ! = 27);
    freebuffer();
    r.x.ax=0x0003;
    int86(0x10,&r,&r);
    return(GOOD_READ);
}
/* get one extended key code */
GetKey()
{
    int c;
    c = getch();
    if(! (c & 0x00ff)) c = getch() << 8;
    return(c);
}
/* set the VGA palette and background */
setvgapalette(p,n,b)
    char *p;
    int n,b;
{
    union REGS r;
    int i;
    outp(0x3c6,0xff);
    for(i=0;i<n;++i) {
      outp(0x3c8,i);
```

```
        outp(0x3c9,(*p++)>>2);
        outp(0x3c9,(*p++)>>2);
        outp(0x3c9,(*p++)>>2);
    }
    r.x.ax=0x1001;
    r.h.bh=b;
    int86(0x10,&r,&r);
}
/* make file name with specific extension */
strmfe(new,old,ext)
    char *new,*old,*ext;
{
    while(*old != 0 && *old != '.') *new++=
     *old++;
     *new++='.';
     while(*ext) *new++=*ext++;
     *new=0;
}
```

美国副总统 AI Gore 盛赞 VTEL 提出数字视讯联接全美目标

美国副总统 AI Gore 在华盛顿会见 VTEL 总裁兼首席执行官 Jerry Benson

(美国德州奥斯汀 1998 年 2 月 27 日讯)全球数字视讯的领导者——VTEL 美国视讯公司开始逐步向美国乡村及贫困地区提供先进的视讯产品,以建立起当地与发达地区的联络。在美国商务部、国家通讯信息处(NTIA)及公共设施法律项目组织(PULP)联合举行的招待会上,VTEL 美国视讯公司总裁兼首席执行官 Jerry Benson 宣布,公司将积极主动地支持"共享—通讯—技术"活动(Community/Communications/Technology,C/C/T)。活动在昨日闭幕的"为 21 世纪建立全美通讯联络"大会上拉开序幕。

美国副总统 AI Gore 在会议上发表重要讲话,高度赞扬了 VTEL 公司与政府,企业间进行合作,提供了操作方便、品质超群的高科技通讯方案。Gore 提到:"我很感谢公共设施法律项目组织(PULP)与 VTEL 公司建立合作关系,帮助乡村及贫困地区建立通讯接入中心。这个共享通讯技术项目会帮助美国乡村及贫困地区的学校、图书馆、儿童保健及急救中心,建设和使用这些技术网络将切实地改善当地人民的生活。"

VTEL 与 PULP 进行"共享—通讯—技术"活动的合作将有助于大型通讯中心及企业在计划、投资、开发及应用高技术方面的统一。VTEL 数字视讯技术将成为"共享—通讯—技术"活动的的主要组成部分。

Benson 在副总统讲话后说:"由于 PULP 一直不断地服务于乡村及贫困地区的通讯客户,VTEL 非常愿意向他们提供最先进的系统。除此之外,VTEL 还决心提供资金帮助并可随时进行技术咨询和指导。我们会一如继往地为美国人民打开世界远程教学、远程医疗及交互式视讯之门而不懈努力,并确保最需要它的人们就可以得到它。"

参加"为 21 世纪建立全美联络"大会的还有商务秘书 William Daley,FCC 专员 Gloria Tristani,FCC 主席 William E. Kennard,内布拉斯加州议员 Robert Kerry,蒙大拿州议员 Gonrad Burns,德克萨斯州代表 Sheila Jackson Lee,新泽西州代表 Robert Menedez 以及纽约代表 Major Owens。

Benson 还在他的讲话中提到加州诺沃克-拉米拉达学区是使用 VTEL 视讯系统实现各界人士与全美联络的范例。他说:"诺沃克-拉米拉达利用我们的系统组建了数字通讯网,把分布于不同地区的不同种族及不同宗教信仰的学生联系在一起,使他们更好地了解了本区域及全球范围内各种文化的相同点及差别。"

"为 21 世纪建立全美联络"大会在公共网上,还进行了一系列交互式视读讲座和演示,其中包括了著名小提琴家 Pinchas Zuckerman 利用 VTEL 会议电视设备从加拿大渥太华为美国纽约、明尼苏达及华盛顿的学生进行表演并就其表演艺术与学生进行讨论的远程教学示范。