实用技术

图形图象文件格式解码实用程序*(续 8)

张明敏

8　IFF/LBM 文件

Commodore Amiga 机上的 Deluxe Paint 绘图软件包能产生精美的图形，建立了一种属于它自己的图形文件格式，称之为 IFF(interchange format files)，IFF 格式适用于 Amiga 所依赖的 68000 处理器体系结构。IFF 文件格式相当有用。它不仅作为 Deluxe Paint 的文件格式，而且也被其它软件用作其文件格式。由于 IFF 是以独立的数据块为基础进行设计的，从而很容易改写为用作特定用途的文件格式。

IFF 文件的一个主要特征是可扩充性，它进一步支持了扩充块的概念，使 IFF 文件不仅能包含图形，而且能包含动画系列、采样的声音、字处理文本和特定于 68000 应用的机器代码。与 GIF 文件一样，最基本的 IFF 文件格式允许 2 到 256 种颜色的图像，IFF 文件对所保存的图像大小没有限制。

IFF 有一种变形称为 LBM。LBM 格式实际上是 Deluxe Paint 使用的“纯”IFF 文件格式的一种变形。在处理 256 种颜色的文件时，Deluxe Paint 可以阅读纯 IFF 文件和 LBM 文件，但它总是产生 LBM 文件。从 Amiga 机器上移植过来的 IFF 文件通常带扩展名. IFF。如果使用 Deluxe Paint 绘图软件，则文件的扩展名为. LBM。

8.1　IFF 文件结构

在讨论 IFF 文件是如何组织之前，先介绍一些概念。第一个问题是微处理器的结构，Amiga 和 Mac 机都采用 Motorola 68000 系列微处理器，这种微处理器不同于驱动 PC 机的 Intel 微处理器。

Motorola 微处理器书写多字节的数(整数和长整数)是按与 Intel 芯片相反的顺序来写各个字节的，用户可以通过一个转换函数在 PC 机上读出 Motorola 的数据，转换函数如下所示：

```
long motr2intl(l)
        long l;
{
    return(((l& 0xff000000L) >>24) +
           ((l& 0x00ff0000L) >>8) +
           ((l& 0x0000ff00L) <<8) +
           ((l& 0x000000ffL) <<24));
```

IFF 文件的标题块和数据块很小，每个块定义了一些与文件有关的内容。一个块可能会告诉用户文件的大小，它的调色板是如何定义的，等等，最后才是图像数据本身。

与 Macintosh 一样，Amiga 使用 4 个字节的字符串作为类型标识符。文件类型和文件中的块用这些 4 个字节的字符串标识，一个 Amiga 文件的开始可以用如下的 C 语言结构体来定义：

```
typedef struct {
        char type[4];
        unsigned long size;
        char subtype[4];
} IFFHEADER;
```

如果用户把 IFF 文件的前十二个字节读入一个 IFFHEADER 变量中，那么 type 域包含三个字符串之一：“FROM”、“LIST”或“CAT”。

如果用户通过 motr2intl 函数转换成一个 IFF-HEADER 的 Size 域的值，其结果为文件的大小，以字节为单位。

IFF 文件的 IFFHEADER 结构变量的 Subtype 域值定义为两个字符串之一。如果文件是一个纯 IFF 文件，那么 Subtype 域存放字符串“ILBM”(interleaved bitmap)。如果文件是带有 256 种颜色的一个 Deluxe Paint LBM 文件，那么 Subtype 中存放字符串"PBM "并用一个空格把长度填充到 4 个字

* 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

节长。"PBM"表示专用的位图(proprietary bitmap)。

IFFHEADER后的第一个字节是第一个块的第一个字节。在IFF文件中,块总是由一个长整型数及其之后的4个字节的类型域组成。长整型数保存了块数据的大小。因此,如果IFF文件阅读程序遇到一个不知如何处理的块,可以简单地跳到下一个块。

一个块的大小值可能比实际块大小多一个字节。块总是被限制在偶数个字节的大小,而Size的值没有做这样的限制来反映出这一点,因此,如果l为一个块大小,那么用户可以做如下操作确保l反映出块的真正大小:

```
if(1&1L) ++l;
```

一个IFF图像文件必须包含三个块类型,它们是:

BMHD——文件的大小

CMAP——彩色映像或调色板值

BODY——图像数据本身

由Deluxe Paint产生的文件包含许多其它的块,这些块被胶片投影和动画设施使用。这些文件还包含一个类型为TINY的块,该块包含了文件图像的一个缩减版本。如果用户在Deluxe Paint环境内打开Load File框,那么每一个选中的LBM文件有一个较小的源图像的预观察图像出现在屏幕上,这个预观察图像存在TINY块中。

(1) BMHD块

IFF文件中块的出现顺序很重要。例如,除非用户已阅读了一个BMHD块,该块定义了大小颜色深度和BODY块图像的细节,否则用户不知道如何去解码一个BODY块。BMHD块总是为第一个必需的IFF文件块。

阅读了BMHD块的类型和长度域后,块的其余部分由以下结构定义:

```
typedef struct{
    unsigned int w,h;
    int x,y;
    char nPlanes;
    char masking;
    char compression;
    char pad1;
    unsigned int transparentColor;
    char xAspect,yAspect;
    int pageW,pageH;
}BMHD;
```

BMHD结构中的w和h域以像素为单位定义图像的大小。X和Y是图像被显示时的左上角坐标。这两个值通常为0,忽略这些值则把所有图像在屏幕的左上角开始显示。

nPlanes的值定义了显示图像所需的彩色位平面的数,这个值与使用的彩色位数目相同。

masking域定义了文件中的图像数据是否用一个屏蔽进行交叉存放。masking域的值为以下四个值之一:

0——没有屏蔽

1——有一个屏蔽

2——有一个透明颜色的屏蔽

3——有一个lass0屏蔽

最简单的情形是没有屏蔽,并且大多数标准的IFF文件以这种方式建立。如果这个域的值为其它三个值之一,那么在每一行中的一个额外平面用作屏蔽。

Compression域为以下两个值之一:

0——没有压缩的图像数据

1——压缩的图像数据

如果Compression域的值为0,那么BODY块中的图像是原始数据;如果Compression域的值为1,那么图像将以行程编码方式存放。所使用的行程编码过程称为PackBits,事实上即为把MacPaint图像存放到Macintosh机上所采用的方法。

Padl域是一个过滤器——它不表示什么含义。

TransparentColor域定义彩色索引,如果masking域表示正在使用一个透明颜色,那么图像被认为是透明的,这类似于GIF文件使用的透明颜色。

最后,xAspect和yAspect域以及pageW和pageH域给出了所定义图像的源屏幕的纵横比和大小。

(2) CMAP块

一个IFF文件的另一个必需的块是CMAP。CMAP定义了用来形成一个彩色调色板的RGB值阵列。数据采用GIF文件所用的格式。每种颜色定义成三个字节。每一个字节表示红、绿、蓝光的百分比。一个256种颜色的图像具有768个字节的彩色映像信息。

注意单色IFF文件(只有2个值的颜色)通常也有一个CMAP块。它定义了带两个入口的一个调

色板。IFF 文件格式可假定所有二色文件都用黑色与白色作为两种颜色。

用户可以通过阅读 type 域来阅读 CMAP 块，检查 length 域确保它为一个合适的值，然后把块的数据读入一个缓冲区用于设置显示卡上的调色板。length 域的值不应大于 768 个字节，这个值为一个 IFF 文件所能包含的颜色数乘以每一种颜色所用的字节数(256×3)。

(3) BODY 块

存放在 BODY 块中的图像数据是没有结构的，如果用户正确地阅读了 BMHD 块的域，就应该知道如何处理图像数据。

为了读 BODY 块中的图像和数据，用户需要知道每一行中的字节数。如果所读的 IFF 文件的 Subtype 域为 ILBM，那么这个值为：

bytes = pixels2bytes (motr2intl (bmhd. m)) * bmhd→nplanes;

这里假定 bmhd 是一个类型为 BMHD 的指针变量并装入了图像的 BMHD 块的内容。

如果文件的 Subtype 域为 PBM，那么一行中的字节数是：

bytes = motr2intl(bmhd. w);

PackBits 使用的行程编码特别精致。它工作起来与我们假想的行程编码类似。然而，它使用最少的字节完成此项工作，并且通常产生相当好的压缩图像，其中包含了图像数据的可预测段(predictable runs)，还原数据图像也相当快。

这里给出如何对一行 PackBits 数据进行解压缩——这是以上的 readline 函数所要做的基本工作。

第一个读取的字节被看作是一个关键字节。如果该字节的最高位为 1——即如果该字节与 80H 作相与操作的结果为真，那么该字节后的域是一个字节段(run of bytes)。段的长度定义为一个或两个字节的值。在一个字节段域中，读随后的字节并重复域的长度所定义的次数。

如果最高位值为 0，那么该字段是一个字符串。串的长度为字节的值。在这种情况下，应该直接从源文件中读取几个字节，这里 n 为长度。

一旦读了一个字段，将重复处理过程，直到解压缩的行的长度等于所计算出的被解码图像的一个行中的字节数为止。正如以上讨论所示，如果生成的行长度太长，那么表明图像数据没有正确地被还原。

有些 PackBits 的实现通常填充压缩行使之含有偶数个字节。压缩的行可用值 80H 填充，这个值既不看作一个字符串键，也不看作一个字节段键，并且在阅读文件时可以忽略这个值。

注意 PackBits 处理过程并不真正知道压缩哪种类型的数据，它可应用于平面或面向字节的行。同样值得指出的是 PackBits 不能达到 GIF 文件中使用的字符串表压缩方法的效率。

8.2　IFF/LBM 文件的解码程序

下面是 READIFF 的源代码，它是一个简单的 IFF 文件阅读程序。正如 GIF 文件阅读程序一样，程序中使用了 VGA 的模式 13 来显示所有的图像，而不管文件实际包含多少种颜色。用户可以使用屏幕驱动技术来建立一个真正的通用的 IFF/LBM 文件阅读程序。

```
/* * * * * * * * * * * * * * * * * * * * * * * *
                IFF 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",
                                        };
        int r;

        if(argc > 1) {
            if((fp = fopen(argv[1],"rb")) != NULL)
                {
                fi.setup=dosetup;
                fi.closedown=doclosedown;
                r=unpackiff(fp,&fi);
                printf("\%s",results[r]);
                fclose(fp);
            } else printf("Error opening %s",argv[1]);
        } else puts("Argument:path to a IFF/LBM
        file");
}

/* unpack an IFF/LBM file */
unpackiff(fp,fi)
        FILE *fp;
        FILEINFO *fi;
```

```
{
    BMHD bmhd;
    unsigned long l;
    char *p, *pr,b[4];
    int i,n;

    /* get the type */
    fread(b,1,4,fp);
    if(! memcmp(b,"FORM",4) ||
      ! memcmp(b,"LIST",4) ||
      ! memcmp(b,"CAT ",4)) {

        /* ignore the size */
        fread((char *)&l,1,4,fp);

        /* get the subtype */
        fread(fi->subtype,1,4,fp);
        /* read all the chunks */
        do {
          fread(b,1,4,fp);
          fread((char *)&l,1,4,fp);
          l=motr2intl(l);
          if(l & 1L) ++l;

          /* check for a bitmap header */
          if(! memcmp(b,"BMHD",4)) {
          if(fread((char *)&bmhd,
            1,sizeof(BMHD),fp)
        ! = sizeof(BMHD)) return(BAD_FILE);
        fi->width=motr2inti(bmhd.w);
        fi->depth=motr2inti(bmhd.h);
        fi->bits=bmhd.nPlanes;
        if(! memcmp(fi->subtype,"ILBM",4))
        fi->bytes=pixels2bytes(fi->width) *
          fi->bits;
        else
        fi->bytes=fi->width;
        }

        /* check for a palette */
        else if(! memcmp(b,"CMAP",4)) {
            if((int)l <= 768) {
            if(fread(fi->palette,1,(int)l,fp)!
            =(int)l) return(BAD_READ);
            }
            else {
            if(fread(fi->palette,1,768,fp) ! =
            768)
                return(BAD_READ);
            fseek(fp,l-768L,SEEK_SET);
            }

        /* check for an image */
        else if(! memcmp(b,"BODY",4)) {
        if((fi->setup)(fi) ! = GOOD_READ)
        return(MEMORY_ERROR);

          (fi->setup)(fi);
          if((p=malloc(fi->width)) ! = NULL) {
                for(i=0;i<fi->depth;++i) {
                if(bmhd.compression)
                n=readline(p,fp,fi->bytes);
                    else
                    n=fread(p,1,fi->bytes,fp);

                if(n ! = fi->bytes) {
                    freebuffer();
                    free(p);
                    return(BAD_READ);
                }

          if(! memcmp(fi->subtype,"ILBM",4) ||
          (! memcmp(fi->subtype,"PBM ",4) &&
          fi->bits < 8)) {
          if((pr=planes2bytes(p,fi)) == NULL) {
          freebuffer();
          free(p);
          return(MEMORY_ERROR);
              }
              putline(pr,i);
              free(pr);
        } else putline(p,i);
        }
          free(p);
          (fi->closedown)(fi);
            } else return(MEMORY_ERROR);
        }

        /* skip an unknown chunk */
        else fseek(fp,l,SEEK_CUR);

    } while(! ferror(fp) && memcmp(b,"BODY",4));
    return(GOOD_READ);

    } else return(BAD_FILE);
}
```

```
/* convert a planar line to a VGA line */
char *planes2bytes(line,fi)
        char *line;
        FILEINFO *fi;
{
        char *p, *pr;
        int i,j,n;

        /* allocate a place to put the line */
        if((p=malloc(fi->width)) != NULL) {

            /* get the width of one plane */
            n=pixels2bytes(fi->width);

            /* sscan through the pixels */
            for(i=0;i<fi->width;++i) {
                pr=line;
                p[i]=0;

                /* fetch each planar pixel */
                for(j=0;j<fi->bits;++j) {
                    if(pr[i>>3] & masktable[i &
                            0x0007])
                        p[i] |= bittable[j];
                    pr+=n;
                }
                  }
                 return(p);
            }
            return(NULL);
    }

    /* read a compressed PackBits line */
    readline(p,fp,bytes)
        char *p;
        FILE *fp;
        int bytes;
    {
        int c,i,n=0;
        do {
            c=fgetc(fp) & 0xff;
            if(c & 0x80) {
                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 < bytes);
      return(n);
    }

    long motr2intl(l)
            long l;
    {
            return(((l & 0xff000000L) >> 24) +
              ((l & 0x00ff0000L) >> 8) +
              ((l & 0x0000ff00L) << 8) +
              ((l & 0x000000ffL) << 24));
    }

    motr2inti(n)
        int n;
    {
        return(((n & 0xff00) >> 8) | ((n &
        0x00ff) << 8));
    }

    /* this function is called after the BMHD and
      CMAP chunks have been read but before the
      BODY
      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,1<<fi->bits,
        fi->background);
        return(GOOD_READ);
    }

    /* This function a called after an image has been
     unpacked. 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);
}
```

IFF 文件阅读程序的功能主要通过 unpackiff 函数来完成，该函数也完成了 LISTIFF 程序的大部分功能，如果它遇到一个 BMHD 块，它就把

BMHD 块的内容读入一个 BMHD 结构中。该函数以同样的方式处理 CMAP 块中的调色板信息。最后它从 BODY 块读入图像行数据。

readline 函数可以完成两个功能中的一个：或是对已被编码成 PackBits 的行程数据行进行解压缩，或是读入原始的字节。在实际读 BODY 块的 readiff 部分，代码或者调用 readline 对一个 PackBits 行进行解码，或者用 fread 读入一个非压缩的行。究竟完成哪一种功能由 bmhd. compression 的内容决定。

通常，一个 IFF 文件包含单个图像，而 BODY 块为文件的最后一个块。因此，readiff 反复循环读 IFF 文件的块直到 BODY 块被解码为止。

全新 AP 系列首款产品——康柏新型专业工作站 AP400

作为 Windows NT®工作站产品的领先供应商——美国康柏电脑公司近日在全球发布了康柏专业工作站 AP400。这是康柏第一次直接面向寻求低价位、双处理器系统的用户而推出的工作站产品。AP400 是 AP 系列中的第一款产品，该系列专门为那些需要价格和性能最佳组合的客户而设计，将包括广泛的产品。

全新 AP 系列的首款产品——康柏专业工作站 AP400 将英特尔最新的Pentium®处理器与高性能的 2D 与 3D 图形解决方案完美地结合起来，以全新的价格水平提供了工作站级的性能。它可以理想地与主流计算机辅助设计(CAD)、建筑设计及建设(AEC)、金融交易以及数字内容创作(DCC)等应用软件相匹配。该产品采用小巧的台式设计，可以方便地安装至空间有限的环境中，如金融交易场所。

康柏专业工作站 AP400 可以支持多达 2 个 350 或 400MHz 英特尔 Pentiumll 处理器，每个处理器具有 512KB 集成的二级高速缓存以及 100MHz 外部总线。该系统的内存可以扩展至 1GB 100MHz ECC SDRAM。它提供 5 个托架，可扩展拥有 27. 3GB 的内部磁盘空间，此外，还包括 6 个扩展插槽(PCI、ISA 和 AGP)。康柏专业工作站 AP400 同时还标配一个集成的驱动器控制器，可支持从 4. 3GB 至 9. 1GB 的高速 Ultra ATA 和 Wide-U SCSI 硬盘。所有型号的工作站均包括一个可以迅速、方便地集成至网络环境中的 10/100 自适应网络控制器、一个 32 倍速 MAX 光驱以及预装的 Microsoft Windows NT Workstation 4. 0 操作系统软件。

康柏专业工作站 AP400 支持加速图形端口(AGP)，可以对庞大的计算密集型图形文件进行更快的渲染。Matrox Millenniumll 图形控制器可以为诸如金融分析、电子设计以及软件开发这样的 2D 应用提供出众的性能。对于那些需要多显示器支持的 2D 应用，STB 系统公司的 MVP Pro-128 图形控制器可以在一个单一平台上支持多达 8 台显示器。对于那些运行诸如主流计算机辅助设计或 Web 创作这样需要快速 2D 性能和强劲 3D 支持的应用，客户可以选择支持 AGP 的 ELSA GLoria Synergy 图形控制器。对于那些在动画或者三维立体建模应用中需要高性能 3D 能力的客户，他们还可以选择 Diamond Fire GL 4000 图形控制器。(汪虹)