实用技术

图形图象文件格式解码实用程序*(续6)

张明敏

6 TGA图象文件格式

Targa格式(或称TGA格式)支持1位到32位彩色之间的任何大小的图象。一个象素的32位中低24位表示彩色,高8位表示其它信息。与TIFF不同的是,它的变种较少。解码TGA文件的软件可解码任何Targa文件,毫无疑问,对24位TIFF文件而言情形则不同。Targa文件可用于保存光线跟踪算法生成的真实感图象,光线跟踪是创建实体对象和光源数字模型的一种过程。计算机通过计算出光源产生的每一条光线是被模型中的物体吸收还是反射来生成一幅图象。

使用Targa文件相对比较复杂,因为它有许多任选项和模式。一个Targa文件可以支持1、8、16、24或32位彩色图象。它的扫描行可以通过简单的行程编码子程序来进行压缩,也可以用非压缩的形式进行存储。它们可以按从上到下或从左到右的方式存储,也可以按与上面的方向相反的方式进行压缩。

6.1 Targa文件结构

Targa文件由图象的文件头、任选调色板域和实际图象数据组成。8位彩色映射的Targa文件有一调色板,其它格式就没有。

Truevision的Targa说明事实上将Targa文件头分成几个较小的数据结构。为讨论方便,可以将它们合成单个部分来看待,如下所示:

```
typedef struct {
        char identsize;
        char colormaptype;
        char imagetype;
        unsigned int colormapstart;
        unsigned int colormaplength;
        char colormapbits;
        unsigned int xstart;
        unsigned int ystart;
        unsigned int width,depth;
        char bits;
        char descriptor;
        }TGAHEAD;
```

注意,Targa文件没有一标识字串,如果用扩展名".TGA"对其它文件进行重命名,解码Targa格式的程序不能立即指出这个变化。但是,当分析文件头数据时,它会发现误差。

(1) identsize域表示文件头和下面部分相隔的字节数。在绝大多数情况下都为0。但是,某些创建Targa文件的软件把一些标识放在文件头之后,这样可使用户知道文件的出处。Targa解码程序在把Targa文件第一部分读入TGAHEAD结构后,它就可以往前找到identsize字节。如果identsize为0,那么该域就无效。如果不为0,它告诉相应的解码程序不要去解释Truevision的说明信息。

(2) colormaptype域定义彩色映射类型(如果文件使用的话)。对于单色文件或者RGB彩色信息的文件它为0。如果有彩色映射的话它为1。

(3) 如果有彩色映射,colormapstart域存放调色板数据定义的第一个彩色映射索引。

(4) colormaplength域存放彩色映射包含的彩色数目。很多情况下它们分别是0和256,但是假设没有使用调色板中的其它彩色,则在一幅图象中可以定义调色板256色中的几种颜色。

(5) imagetype域定义了正在解码的图象的类型和存储量。它可以包含如下几个值中的一个:

1——非压缩调色板驱动图象
2——非压缩RGB图象
3——非压缩单色图象
9——行程编码调色板驱动图象
10——行程编码RGB图象
11——行程编码单色图象

* 注:需源代码者请与浙江大学CAD&CG国家重点实验室张明敏联系。邮编:310027,电话(0571)7951045,7990451

Targa 说明中指出值 0 到 127 被认为是图象存储的合法值(对 Truevision 而言),128 以上的值被认为是定制存储类型。事实上,以上六个值是用户可能遇到的所有值。因此,检查一下域中的这些值是保证用户的软件正在解码的是 Targa 文件的较安全的方法。

(6) xstart 和 ystart 域表示图象左上角和屏幕左上角之间的偏移量,这些域一般为 0。

(7) width 和 depth 值表示图象的尺寸大小。

(8) bits 值定义了它所用的彩色位数。这些值可能是 1、8、16、24 或 32。注意 8 位 Targa 文件有可能也没有彩色映射,如果遇到这种文件的话,可以假定它拥有与解码 24 位图象文件的程序所使用的灰度调色板一样的调色板。

(9) descriptor 域是一标志集,它表示图象数据是如何处理的。域中只有 2 位对现在讨论的情况有用。假定 tga 是类型为 TGAHEAD 的结构。如果 tga.descriptor AND 20H 为真,那么该文件的开始部分先存储源图象的最后一行,否则,它就按正常情况存储。如果 tga.descriptor AND 10H 是真,则每行以从右到左的次序存储。因此它看上去是正常图象的镜面反射图象。descriptor 域的低 4 位表示覆盖可用的位数,高 4 位现在用不到。

一旦已经读出了 Targa 的文件头和彩色映射信息,文件中的下一位就是图象中的第一位。用户遇到许多 Targa 文件都以非压缩的形式存储。在这种情况下,每一行的字节数可以通过图象的宽度和由它支持的彩色位数而得出,计算公式如下所示:

1 位图象——字节数/行 = (width + 7)/8
8 位图象——字节数/行 = width
16 位图象——字节数/行 = width * 2
24 位图象——字节数/行 = width * 3
32 位图象——字节数/行 = width * 4

从一非压缩文件中读出一行仅做将相应的字节放入缓冲区中的操作。但是要注意,8 位彩色的行格式对 Targa 格式而言是特殊的。当遇到 16 位 Targa 文件时,它的 2 个字节的象素可以解码成为 3 个字节的 RGB 象素。在 24 位 Targa 文件行中 3 字节的 RGB 象素的次序可以调整。在 32 位 Targa 文件行中的 4 个字节象素可以缩减成 3 个字节。因此,虽然解码全彩色 Targa 文件行比较简单,但是在用户把它与标准 RGB 彩色信息的程序代码一起使用时,需要进行一些特殊处理。

在进行了上面的处理后,还要解码行程编码的 Targa 文件。这个过程非常简单,这是因为 Targa 行程编码并不复杂。

Targa 图象的最一般的应用是:用于高级照片修描(retouch);作为彩色扫描的通用输出文件;用于复杂的计算机生成艺术(例如光线跟踪等)的输出文件。这些应用程序使用 TGA 文件的方法基本一样,图象数据过分复杂且冗余特别少。

用于存放扫描图象的 256 色图画常常可以被压缩,使用字串表压缩方法。注意,不仅这样的图象最多有 256 种可能的象素值,而且那种把它们变成 256 色的误差扩散抖动处理给它们增加了某种变化。真 24 位图象非常难以压缩,很可能在创建一 Targa 文件时没有 2 个象素恰好表示同一颜色。

Targa 格式允许使用简单类型的行程压缩,某种程度下它可以压缩绘制的 Targa 图象。当遇到一扫描图象时,可能会产生一负压缩图象(即压缩后的图象大小比原来的图象大)。由于这个原因,在 Targa 文件中常常不使用压缩。扫描全彩色图象需占据大量的磁盘空间,如果我们了解一彩色扫描仪是如何工作的话,那么就可以知道扫描图象是比较难处理的。

还有另外一种压缩算法,称为 lossy 压缩,通过抛弃颜色的较少的细节变化来寻找改进图象的压缩效果。比较流行的一种算法称为 JPEG 压缩和 MPEG 压缩。毫无疑问,当使用这种算法时,解码后的图象文件不再与压缩前的那个图像文件一样;如果效果好,解码后的图象差不多是原来的那幅图象,但是其代价是存储空间要增大。

Targa 格式倾向于在高级图象应用程序中使用(在图象质量与所需要的存储空间相比情况下图象质量更重要)。高级彩色图象处理软件常常使用较大的 Targa 文件。

在 Targa 文件中如果 Targa 行以压缩格式存储,那么 imagetype 值为 9,10 和 11。在这种情况下,一行中的第一个字节是压缩域的第一个字节,必须被认为是关键字节。该域的长度将比放在低 7 位的值多 1。如果高位为 1,该域为一象素行程(run),如高位为 0,该域为字串。

当检测 Targa 行程压缩时,必须记住除了单色 TGA 文件之外,所有域是以像素形式而不是以字节形式工作的。象素中的字节数随着文件中彩色位数的不同而不同。因此,带有 24 位彩色的文件中长度为 10 的压缩域事实上包括 30 个字节的非压缩数据——10 个 3 字节象素。

一旦行中的一个域已被解码，从文件中读出的下一字节就被认为是下一域的关键字节。该过程一直重复，直到整行都被解码为止。与大多数其它图象文件格式一样，压缩 Targa 行要在偶数行边界结束。

使得处理 Targa 文件复杂化的唯一的因素是它有较多的模式和象素格式。一般说来，编写处理 Targa 文件的代码需求远比编写处理其它图象文件的代码需求少。

如果用户忘了将相应的 RGB 字节按顺序置反的话，在测试图象中的蓝色和红色将互换，即红色用蓝色显示。

6.2 Targa 文件的解码程序

在讨论 Targa 格式的解码程序的细节之前，必须先了解一下 Targa 格式的特性。许多特性都与 Targa 文件中的颜色处理方法有关。使用 Targa 格式的变种可存储产生 Targa 图象数据的显示硬件的特性，以便以后在相应硬件上显示。

Targa 文件使用 3 个字节来表示 RGB 颜色，3 个字节的存储顺序与其它常用软件保存颜色所用的顺序相反。正常的象素或调色板入口将以红、绿、蓝的形式存储，在 Targa 文件中以蓝、绿、红的方式存储。

在讨论图象文件时，由于受微机上显示设备的限制，一般采用把 24 位图象压缩为灰度图象的方法。彩色误差并不常常在灰度图象中显现出来，如果使用光线跟踪生成的 Targa 文件作为测试图象（因为它们很容易获得），那么即使在彩色模式下观察它也不容易发现彩色值有什么误差。

8 位 Targa 文件以与其它 8 位图象格式文件同样的方法存储，即把彩色当成调色板中的索引，这是很容易处理的。多于 8 位的文件是以 RGB 数组的形式存放的。然而，与 24 位 BMP 文件不同的是，每个象素不需要 3 个字节。在 Targa 文档资料中，带 RGB 彩色（或真彩色）的 Targa 文件可以用 16、24 或 32 位存储彩色信息。一个 16 位 Targa 文件以 5 位索引值存储每种颜色，总共用 15 位，它可以压入 2 个字节中，剩下一位不用。开始 5 位表示蓝色的强度，接着 5 位表示绿色的强度，最后 5 位表示红色的强度。如果当前讨论的图象是用来覆盖在已存在的图象上时，就使用 16 位 Targa 像素的高位（high-order bit），这时定义的图象是透明的。在这种情况下，如果已设置了最高位，则象素写入显示器中。如果没有设置，则不显示象素，在当前位置显示原有的象素。除了在每个象素中 3 个字节的次序是相反的之外，24 位彩色 Targa 文件与 24 位 BMP 文件工作原理是相同的。因此，每个象素中 3 个字节分别表示蓝、绿、红的强度。24 位 Targa 图象的每一行的字节长度是图象宽度的 3 倍。除了每个象素由 4 个字节的彩色信息而不是 3 个字节的彩色信息组成之外，32 位彩色 Targa 文件的结构与 24 位 Targa 文件是相似的。最前面的那个字节称为混合值。如果一个象素的最高位为 0，则象素由 3 个 RGB 值组成，第四个字节忽略掉；如果最高位为 1，其低 7 位表示象素颜色与要覆盖的图象相混合的程度。这使得每个象素可以在完全透明与完全不透明之间变化。

与 24 位 BMP 文件一样，下面这个简单的解码 Targa 格式的程序以灰度图象的形式处理多于 8 位彩色文件。当然这样会使得用户观察到的图象不如原有的彩色图象那么有趣。

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * *
        TGA 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],"TGA");
          strupr(path);
          if((fp = fopen(path,"rb")) ! = NULL) {
                  fi.setup=dosetup;
                  fi.closedown=doclosedown;
                  r=unpacktga(fp,&fi);
                  printf("\%s",results[r]);
                  fclose(fp);
    } else printf("Error opening %s",path);
        } else puts("Argument:  path to a TGA file");
}
/* unpack a TGA file */
unpacktga(fp,fi)
  FILE *fp;
  FILEINFO *fi;
{ TGAHEAD tga;
  char *p,*pr;
```

```
int i,n,startline,endline,incline;
memcpy(fi->palette,"\000\000\000
        \377\377\377",6);
if(fread((char *)&tga,1,sizeof(TGAHEAD),fp)==
   sizeof(TGAHEAD)) {
if(tga.imagetype==0x01 ||tga.imagetype==0x02 ||
   tga.imagetype==0x03 ||tga.imagetype==0x09 ||
   tga.imagetype==0x0a ||tga.imagetype==0x0b) {
    fi->width=tga.width;
    fi->depth=tga.depth;
    fi->bits=tga.bits;
    if(fi->bits==1) fi->bytes=
      pixels2bytes(fi->width);
    else if(fi->bits==8) fi->bytes=fi->width;
    else fi->bytes=fi->width*3;
    fseek(fp,(long)tga.identsize,SEEK_CUR);
    if(tga.colourmaptype) {
      switch(tga.colourmapbits) {
        case 24:
          for(i=tga.colourmapstart;i<tga.
              colourmaplength;++i) {
          if(i >= 768) break;
          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);
        }
        break;
  case 16:
        for(i=tga.colourmapstart;i<tga.
            colourmaplength;++i) {
          if(i >= 768) break;
          n=fgetword(fp);
          fi->palette[i*RGB_SIZE+
              RGB_BLUE]=
              ((n >> 10) & 0x1f) << 3;
          fi->palette[i*RGB_SIZE+
              RGB_GREEN]=
              ((n >> 5) & 0x1f) << 3;
          fi->palette[i*RGB_SIZE+RGB_RED]=
              (n & 0x1f) << 3;
    }
    break;
  }
}
if(fi->bits > 8) {
    for(i=0;i<256;++i)
        memset(fi->palette+(i*RGB_SIZE),i,
RGB_SIZE);
}
else if(fi->bits==1) memcpy(fi->palette,"\000\000\
    000\377\377\377",6);
if((p=malloc(fi->bytes)) != NULL) {
if((fi->setup)(fi) != GOOD_READ) {
        free(p);
        return(MEMORY_ERROR);
}
if(!(tga.descriptor & 0x20)) {
        startline=fi->depth-1;
        endline=-1;
        incline=-1;
}
else {startline=0;
        endline=fi->depth;
        incline=1;
}
for(i=startline;i != endline;i+=incline) {
  if(readtgaline(p,fp,&tga)) {
        free(p);
        freebuffer();
        return(BAD_READ);
  }
  switch(fi->bits) {
        case 1:pr=mono2vga(p,fi->width);
            if(pr != NULL) {
                reverse(pr,&tga);
                putline(pr,i);
                free(pr);
            }
            else {free(p);
                return(MEMORY_ERROR);
            }
            break;
        case 8:reverse(p,&tga);
            putline(p,i);
            break;
      case 16:case 24:case 32:
            pr=rgb2vga(p,fi->width);
            if(pr != NULL) {
                reverse(pr,&tga);
                putline(pr,i);
                free(pr);
            }
            else {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);
}
/* reverse one line left for right needs be */
reverse(p,tga)
    char *p;
    TGAHEAD *tga;
{char *pr;
    int i;
    if(! (tga->descriptor & 0x10)) return;
    if((pr=malloc(tga->width)) != NULL) {
      for(i=0;i<tga->width;++i) pr[i]=
          p[tga->width-1-i];
      memcpy(p,pr,tga->width);
      free(pr);
    }
}
/* read one line in the appropriate mode */
readtgaline(p,fp,tga)
    char *p;
    FILE *fp;
    TGAHEAD *tga;
{
    int c,i,n=0,size;
    int r,g,b,linesize;
    if(tga->bits==1) linesize=
      pixels2bytes(tga->width);
    else linesize=tga->width;
    if(tga->imagetype==0x01 ||
      tga->imagetype==0x02 ||
      tga->imagetype==0x03) {
      switch(tga->bits) {
          case 1:fread(p,1,pixels2bytes
                    (tga->width),fp);
                  break;
          case 8:fread(p,1,tga->width,fp);
                  break;
          case 16:for(i=0;i<tga->width;++i) {
                    c=fgetword(fp);
                    r=((c >> 10) & 0x1f) << 3;
                    g=((c >> 5) & 0x1f) << 3;
                    b=(c & 0x1f) << 3;
                    *p++=r;
                    *p++=g;
                    *p++=b;
                  }
                  break;
          case 24:for(i=0;i<tga->width;++i) {
                    b=fgetc(fp);
                    g=fgetc(fp);
                    r=fgetc(fp);
                    *p++=r;
                    *p++=g;
                    *p++=b;
                  }
                  break;
          case 32:for(i=0;i<tga->width;++i) {
                    b=fgetc(fp);
                    g=fgetc(fp);
                    r=fgetc(fp);
                    fgetc(fp);
                    *p++=r;
                    *p++=g;
                    *p++=b;
                  }
                  break;
      }
    /* handle compressed lines */
    else {
      do {
        c=fgetc(fp);
        size=(c & 0x7f)+1;
        n+=size;
        if(c & 0x80) {
            switch(tga->bits) {
                case 1:case 8:
                    c=fgetc(fp);
                    for(i=0;i<size;++i) *p++=c;
                    break;
                case 16:c=fgetword(fp);
                    r=((c >> 10) & 0x1f) << 3;
                    g=((c >> 5) & 0x1f) << 3;
                    b=(c & 0x1f) << 3;
                    for(i=0;i<size;++i) {
                        *p++=r;
                        *p++=g;
                        *p++=b;
```

```
                }
                break;
                case 24:b=fgetc(fp);
                g=fgetc(fp);
                r=fgetc(fp);
                for(i=0;i<size;++i) {
                    *p++=r;
                    *p++=g;
                    *p++=b;
                }
                break;
            case 32:b=fgetc(fp);
                g=fgetc(fp);
                r=fgetc(fp);
                fgetc(fp);
                for(i=0;i<size;++i) {
                    *p++=r;
                    *p++=g;
                    *p++=b;
                }
                break;
        }
    }
    else {
        switch(tga->bits) {
            case 1:case 8:
                for(i=0;i<size;++i)
                    *p++=fgetc(fp);
                break;
            case 16:for(i=0;i<size;++i) {
                    c=fgetword(fp);
                    r=((c >> 10) & 0x1f) << 3;
                    g=((c >> 5) & 0x1f) << 3;
                    b=(c & 0x1f) << 3;
                    *p++=r;
                    *p++=g;
                    *p++=b;
                }
                break;
                case 24:
                for(i=0;i<size;++i) {
                    b=fgetc(fp);
                    g=fgetc(fp);
                    r=fgetc(fp);
                    *p++=r;
                    *p++=g;
                    *p++=b;
                }
                break;
            case 32:
                for(i=0;i<size;++i) {
                    b=fgetc(fp);
                    g=fgetc(fp);
                    r=fgetc(fp);
                    fgetc(fp);
                    *p++=r;
                    *p++=g;
                    *p++=b;
                }
                break;
        }
    }
} while(n < linesize);
}
return(ferror(fp));
}
/* 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 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);
}
fgetword(fp)
    FILE *fp;
{return((fgetc(fp) & 0xff) +
  ((fgetc(fp) & 0xff) << 8));
}
```

程序 READTGA 中的 unpacktga 函数包含许多 Targa 专用代码，它先将一 Targa 文件头读入 TGAHEAD 结构，然后检查一下文件是否为它所认识的一种图象类型。然后确定文件的大小并计算出它需要多大的行缓冲区。接着找出附在文件头标尾部的内容。

接下来解码彩色映射，在这个程序中，因为多于 8 位彩色的图象必须以灰度的形式显示，如果没有读入合适的调色板的话，就创建一人工合成灰度彩色映射。

通过调用 readtgaline 从源文件中读入每一行，readtgaline 负责读入所需要的原始字节或非压缩行。同时把 16、24 和 32 位象素表示的行转换成标准 24 位 RGB 格式。注意，根据 TGAHEAD 描述字节的内容不同，每行的行号可以增加或减少。同样，非压缩行可以按相反的顺序放置。

在创建 READTGA 时，Targa 文件中的特征信息都被忽略了。非常明显的是，这些代码抛弃了所有 Targa 覆盖和透明的信息，这是因为它对于显示 Targa 图象是不必要的。如果读者在实际工作中确实要这些信息，可以扩大读 Targa 格式文件的程序和功能，将这些功能包括进来。

AVER 可视电话

有了 AVER 可视电话您就能在家庭、企业、娱乐中遨游。AVER 可视电话能广泛用于家庭、企业、娱乐和媒体的应用。

AVER TV CAPTURE 可集视讯会议、电话接收、视频捕捉、电脑游戏、放像机五大功能。良好的家庭影院界面，方便的红外遥控操作，PCI 总线设计，在 WINDOWS95 操作系统下运行是您 1998 年首选之多媒体环境。它支持图象分辨率 1024×768，遵守 H323 协议。在 INTERNET 或电话线上进行电视电话。

特性：

(1) INTERNET/点对点可视电话应用
- 遵照 H. 324 协议
- 要求配声卡，CAMCORDER 或台式相机

(2) 在计算机屏上看电视
- 电视画面大小可从 ICON 到满屏随意改变
- 支持 VGA 分辨率直到 1024×768
- 为 TV 协调功能而准备的 181 频道电缆
- 关闭字幕
- 切断频道源
- 16 频道的预示
- 自动搜索
- SAP/立体声(只在美国 NTSC 条件下)，静音
- 电视调度器

(3) 图象采集
- 与 AVI 兼容
- RGB32，24，15 规格的 YUV4:2:2 品质
- 支持 COAX，COMPOSITE 及 S-VIDEO 输入

(4) 不需要特殊的连接器及 VGA 闭环电缆

(5) 遵照 PCI2. 1

(6) 支持颜色及图象调节

(7) 选择图象源

(8) 图象始终显示在屏幕上