实用技术

图形图象文件格式解码实用程序(续5)

张明敏

5 TIFF文件格式

TIFF文件是一种常用的图象文件格式。各种扫描仪(如手持扫描仪、黑白灰度扫描仪、彩色扫描仪、高精度扫描仪等)所配备的软件都直接或间接地用到TIFF图象文件格式,一些图象应用软件都能把TIFF文件作为输入/输出格式。TIFF是Tagged Image File Format的缩写,它是由美国Aldus Developer's Desk和Microsoft Windows Marketing Group公司联合创造发展的。它是一种国际上非常流行的适于各种计算机和操作系统的图象文件格式,目前国际上流行的很多软件都支持TIFF格式。

TIFF是一种很好的图象文件格式,它是弹性的、可扩展的,能支持各种专用特征和应用说明的信息。TIFF的弹性使得许多解码TIFF文件的开发工具包的作用受到限制:有100多种构造TIFF文件的方法;TIFF标准支持6个或更多的图象压缩;1位颜色的图象可以以倒置或正常方式存储;多于1位颜色的图象可以以位平面、字节或RGB三色等各种形式存储;像图象范围和纵横比率这样的参数可以用长整数、短整数、有理数、正文字串的形式存放。只要不违反TIFF标准,任选项表内容很多。一般情况是保证要存储的数据是以正文字串形式写出来,但是许多TIFF任选项都不遵守这一规则。

TIFF文件的缺点是:如果读者拿到一张装满TIFF文件的磁盘,则用自己的软件显示TIFF文件的可能性只有5%。尽管如此,TIFF格式还是受到了人们的普遍欢迎,它是一有用的格式,它很适用于中等大小的图象应用程序,大多数桌面印刷软件包输入TIFF文件来得到彩色或半色调灰度图象。TIFF图象文件格式有一个好处是不依赖于专用厂家应用系统或者专用的计算机。对Macintosh和PC系统而言,TIFF文件的可读性和不可读性都是一样的。

TIFF文件最多能支持24位彩色。可以在一文件中存储任意大小的图象。事实上,在TIFF文件中可以存储多个图象,还可存一些其它的信息。

这里介绍的程序代码不能读所有的TIFF文件,它只能处理TIFF文件的基本格式。

5.1 TIFF文件结构

TIFF文件就由一个头标和随后的一系列标记域组成,在TIFF文件中该域内可以是任何类型的值。TIFF中定义了很多域,关于每个具体域的含义读者可参阅相应TIFF使用手册或其它介绍TIFF文件格式的资料。

驱动PC兼容系统的Intel微处理器处理多字节数据的次序与Motorola微处理系统正好相反,这种处理在图形文件之外很少出现,因为很少有数据类型可以在系统之间传输,原始正文是与机器无关的。

设计TIFF标准的目的是允许PC机可以处理TIFF文件的Intel数据,使Macintosh机可以处理Motorola数据;当网络可用时,在某一点上允许两个系统都可以解码另一机上的TIFF文件。因此,在每一TIFF文件中都带有创建它的机器的属性,它是放在文件的开始部分的。

彩色TIFF文件的头很简单,如下所示:

```
typedef struct {
        unsigned int numbertype;
        unsigned int version;
        unsigned long offset;
        } TIFFHEAD;
```

(1) 从TIFF文件的头开始,将TIFF文件的头8个字节放入这些域中,如果文件使用Intel数据,则numbertype为4949H,如果使用Motorola数据,numbertype为4D4DH,这里的49H代表ASCII码的I,4DH代表ASCII码的M。TIFF解码程序在解码时需要有一函数从文件中取出字和长整数,这些函数设置字节次序,在文件头的numbertype域中正是以这个次序出现的。

(2) version元素总为42。但是,根据number-

type 域的不同值，42 也可以以其它的整数类型出现。长整数 offset 也可以是 Intel 或 Motorola 数。注意，numbertype 元素是与硬件无关的，不管解码的字节次序如何，这两个字节都是一样的。

(3) TIFF 文件头的 offset 元素表示文件的开始到第一个“图象文件目录”的偏移。正是图象文件目录使得 TIFF 文件具有较大的弹性和复杂性。每一目录可以有任何多个标记，每一标记都说明要打开的图象的参数，这些内容是非常重要的。例如，有一专用标记说明画图象的艺术家的姓名，有一个标记说明数字化图象的扫描仪的型号，有两个标记说明半色调光点大小，最后两个现在还没有用到。

图象文件目录的开始位置是定义目录中标记数的整数，每一标记有 12 字节长，它的结构如下所示：

```
typedef struct {
        unsigned int tag;
        unsigned int type;
        unsigned long length;
        unsigned long offset;
        }TIFFTAG;
```

这些元素的每一个都以 Intel 或 Motorola 数的形式存放，因此不能以正常的形式将它们读入结构中。

(1) tag 域说明了几十个已定义的 TIFF 标记中标记的标记号码。例如，如果标记号为 256，该标记称为 ImageWidth，定义了要打开的图象的像素宽。

(2) type 域说明了用于定义当前的标记中信息的数据类型，有 5 种数据类型可用于定义 TIFF 标记：

1——该标记把数据定义为一字节。

2——该标记把数据定义为一 ASCII 字符的字串。

3——该标记把数据定义为无符号整数。

4——该标记把数据定义为无符号长整数。

5——该标记把数据定义为由 2 个长整数组成的任意数字，一个表示分数的分子，一个表示分母。

(3) length 域定义当前标记中数据的长度，但是要注意它事实上不是以字节定义长度的，它是以标记所指定的存储单位来定义长度。因此，对于说明为 1 个字节长的串的标记来说，它的长度值为 1，对说明为 500 个字节长的串的标记来说，它的长度值也为 1(因为以串为单位)。

(4) 标记的 offset 值定义了标记表示的真实信息：如果 length 元素说明了放在 4 字节中的数据，offset 元素的真正大小就是存在 offset 域中的数据；反之，它就存在 TIFF 文件的某个地方，它的地址就由 offset 域说明(相对于文件的开始)。

在许多已定义的 TIFF 标记中，几乎所有都是任选的，为了提供足够的信息来理解 TIFF 文件的结构，在 TIFF 文件中必须有如下标记：

- NewSubfileType　　定义图象的特性。
- ImageWidth　　定义图象的像素宽。
- ImageDepth　　定义图象的像素深。
- BrtsPerSample　　定义像素位数。
- StripOffsets　　说明图象在文件中的开始位置。
- Compression　　定义 TIFF 的压缩类型。
- PhotometricInterpretation　　定义图象数据是以一般形式存储是以交错形式存储，根据它对彩色像素作出正确解释。

Compression 标记是 TIFF 格式的任选项之一，理想的 TIFF 解码程序能处理任何一种 TIFF 文件，可以解释所有 TIFF 压缩类型。Compression 标记的值可以是下面这些值之一：

1——非压缩

2——CCITT 组 3 Huffman 编码

3——CCITT 组 3 FAX 压缩

4——CCITT 组 4 FAX 压缩

5——LZW 压缩

32 773——Macintosh PacBits 行程编码

在压缩任选项中，这里讨论类 1 和类 32 773。2、3、4 三种 CCITT 压缩类型需要有非常庞大的 makeup 代码表，在任何时候都不适用于彩色 TIFF 文件，LZW 压缩类型也需要有庞大的代码，现在使用不太广泛。

读完图象文件目录中的所有标记之后，就会遇到一个长整数值，该值很可能为 0。如果它为 0，那么说明文件中所有标记已读完；如果该值不为 0，那么表明还有其它图象文件目录，该长整数表示了偏移量。一般不会处理到由多个图象目录中组成 TIFF 文件，建议使用该文件来支持预示图象。

5.2 TIFF 文件解码程序

TIFF 的早期版本没有给出定义彩色调色板的方法，而是让用户用每个像素的红、绿、蓝的百分比来创建图象，可以使用 BitsPerSample 标记来定义对彩色百分比的分辨率。

正如前面讨论的那样，理想的 TIFF 文件解码程序能解码所有类型的标记，下面的 READTIF 程

序比较简明,而且功能也有限,并不能解码所有的 TIFF 文件。

```
/* * * * * * * * * * * * * * * * * * * * * * * * *
        TIFF 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[80];
    int r;
    if(argc > 1) {
        strmfe(path,argv[1],"TIF");
        strupr(path);
        if((fp = fopen(path,"rb")) != NULL) {
            fi.setup=dosetup;
            fi.closedown=doclosedown;
            r=unpacktiff(fp,&fi);
            printf("\%s",results[r]);
            fclose(fp);
        } else printf("Error opening %s",path);
    } else puts("Argument:    path to a TIFF file");
}
/* unpack a TIFF file */
unpacktiff(fp,fi)
    FILE *fp;
    FILEINFO *fi;
{char *p, *pr;
    unsigned long l;
    int i,j,n;
    numbertype=fgetword(fp);
    if(numbertype=='II' || numbertype=='MM') {
    fgetword(fp);
    while((l=fgetlong(fp)) != 0L) {
      fseek(fp,l,SEEK_SET);
      setdefaults(fi);
      n=fgetword(fp);
      for(i =0;i<n;++i) decodetag(fi,fp);
      l=ftell(fp);
      fi->bits=fi->samples * fi->bitspersample;
      if(fi->bits==1) continue;
      if(fi->bits <= 8)
          fi->bytes = pixels2bytes(fi->width) *
              fi->bits;
      else fi->bytes=fi->width * RGB_SIZE;
      if(fi->width != 0 &&
        fi->depth != 0 &&
        fi->offset != 0L) {
        if((p=malloc(fi->bytes)) == NULL)
            return(MEMORY_ERROR);
        if((pr=malloc(fi->width)) == NULL) {
            free(p);
            return(MEMORY_ERROR);
        }
        (fi->setup)(fi);
        if(fi->count==1L) {
        fseek(fp,fi->offset,SEEK_SET);
        for(i=0;i<fi->depth;++i) {
            if(readline(p,fp,fi->bytes,fi) !=
              fi->bytes) {
                free(pr);
                free(p);
                freebuffer();
                return(BAD_READ);
            }
            convertline(pr,p,fi);
            putline(pr,i);
        }
    }
    else {
        for(l=0L;l<fi->count;++l) {
        fseek(fp,fi->offset+(l * sizeof(long)),
            SEEK_SET);
        fseek(fp,fgetlong(fp),SEEK_SET);
        for(j=0;j<(int)fi->rowsperstrip;++j) {
          if((int)((l * fi->rowsperstrip)+j) >=
            fi->depth) break;
          if(readline(p,fp,fi->bytes,fi) !=
            fi->bytes) {
              free(pr);
              free(p);
              freebuffer();
              return(BAD_READ);
          }
          convertline(pr,p,fi);
          putline(pr,(int)(l * fi->rowsperstrip)+j);
        }
      }
    }
    free(pr);
    free(p);
    (fi->closedown)(fi);
```

```
    fseek(fp,1,SEEK_SET);
  }
  }
  return(GOOD_READ);
  } else return(BAD_FILE);
}
convertline(dest,source,fi)
    char *dest, *source;
    FILEINFO *fi;
{int i;
    switch(fi->bits) {
      case 24:
          for(i=0;i<fi->width;++i) {
              dest[i]=greyvalue(source[RGB_RED],
                  source[RGB_GREEN],
              source[RGB_BLUE]);
              source+=RGB_SIZE;
          }
          break;
  case 8:
          memcpy(dest,source,fi->width);
          break;
  case 4:
          for(i=0;i<fi->width;) {
              dest[i++]=(*source>>4) & 0x0f;
              dest[i++]=*source & 0x0f;
              ++source;
          }
          break;
  }
}
setdefaults(fi)
    FILEINFO *fi;
{int i;
    fi->width=0;
    fi->depth=0;
    fi->bits=0;
    fi->planarconfig=1;
    fi->samples=1;
    fi->compression=1;
    fi->offset=0L;
    for(i=0;i<256;++i)
        memset(fi->palette+(i*RGB_SIZE),i,
            RGB_SIZE);
}
decodetag(fi,fp)
    FILEINFO *fi;
    FILE *fp;
{long length,offset,pos;
    int tag,type,i;
    tag=fgetword(fp);
    type=fgetword(fp);
    if(type == TIFFlong) {
        length=fgetlong(fp);
        offset=fgetlong(fp);
    }
    else {length=(unsigned long)fgetword(fp);
        fgetword(fp);
        offset=(unsigned long)fgetword(fp);
        fgetword(fp);
    }
    switch(tag) {
      case SubfileType: break;
      case ImageWidth:fi->width=
                (unsigned int)offset;
          break;
      case ImageLength:
          fi->depth=(unsigned int)offset;
          break;
      case RowsPerStrip:
          if(type==TIFFlong)
              fi->rowsperstrip=offset;
          else fi->rowsperstrip=offset & 0xffffL;
          break;
      case StripOffsets:
          if(type==TIFFlong) fi->offset=
              offset;
          else fi->offset=offset & 0xffffL;
          fi->count=(int)length;
          break;
      case StripByteCounts:
          if(type==TIFFlong) fi->bytecount=
              offset;
          else fi->bytecount= offset & 0xffffL;
          break;
      case SamplesPerPixel:
          fi->samples=(int)offset;
          break;
      case BitsPerSample:
          if(length > 1L) {
              pos=ftell(fp);
              fseek(fp,offset,SEEK_SET);
              fi->bitspersample=fgetword(fp);
              fseek(fp,pos,SEEK_SET);
          } else fi->bitspersample=(int)offset;
          break;
```

```
case PlanarConfiguration:
        fi->planarconfig=(int)offset;
        break;
case Compression:
        fi->compression=(int)offset;
        break;
case Group3Options: break;
case Group4Options: break;
case FillOrder: break;
case Threshholding: break;
case CellWidth: break;
case CellLength: break;
case MinSampleValue: break;
case MaxSampleValue: break;
case PhotometricInterp: break;
case GrayResponseUnit: break;
case GrayResponseCurve: break;
case ColorResponseUnit: break;
case ColorResponseCurves: break;
case XResolution: break;
case YResolution: break;
case ResolutionUnit: break;
case Orientation:break;
case DocumentName: break;
case PageName: break;
case XPosition: break;
case YPosition: break;
case PageNumber: break;
case ImageDescription: break;
case Make: break;
case Model: break;
case FreeOffsets: break;
case FreeByteCounts: break;
case ColorMap: pos=ftell(fp);
        fseek(fp,offset,SEEK_SET);
        for(i=0;i<(1<<fi->bitspersample);
            ++i) {
            if(i >= 256) break;
                fi->palette[i * RGB_SIZE +
                    RGB_RED]=
                fgetword(fp) >> 8;
        }
        for(i=0;i<(1<<fi->bitspersample);
            ++i) {
            if(i >= 256) break;
                fi->palette[i * RGB_SIZE +
                    RGB_GREEN]=
                fgetword(fp) >> 8;
        }
        for(i=0;i<(1<<fi->bitspersample);
            ++i) {
            if(i >= 256) break;
                fi->palette[i * RGB_SIZE +
                    RGB_BLUE]=
                fgetword(fp) >> 8;
        }
        fseek(fp,pos,SEEK_SET);
        break;
    }
}
/* read a compressed PackBits line */
readline(p,fp,bytes,fi)
    char *p;
    FILE *fp;
    int bytes;
    FILEINFO *fi;
{int c,i,n=0;
    if(fi->compression==1)
        return(fread(p,1,bytes,fp));
    else if(fi->compression==0x8005) {
        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);
    }
    else return(0);
}
/* get a word from a TIFF file */
unsigned int fgetword(fp)
    FILE *fp;
{if(numbertype == 'II') return((fgetc(fp) & 0xff) +
    ((fgetc(fp) & 0xff) << 8));
    else return(((fgetc(fp) & 0xff) << 8) +
        (fgetc(fp) & 0xff));
}
/* get a long integer from a TIFF file */
unsigned long fgetlong(fp)
```

```
    FILE *fp;
{if(numbertype == 'II')
      return((unsigned long)(fgetc(fp) & 0xff) +
        ((unsigned long)(fgetc(fp) & 0xff) << 8) +
        ((unsigned long)(fgetc(fp) & 0xff) << 16) +
        ((unsigned long)(fgetc(fp) & 0xff) << 24));
    else return(((unsigned long)(fgetc(fp) & 0xff) <<
            24) +
        ((unsigned long)(fgetc(fp) & 0xff) << 16) +
        ((unsigned long)(fgetc(fp) & 0xff) << 8) +
        (unsigned long)(fgetc(fp) & 0xff));
}
```

READTIF 指出解码 TIFF 文件的基本方法，它可以处理许多常遇到的彩色 TIFF 文件。因为彩色数据压缩效果不佳，所以绝大多数彩色 TIFF 文件不压缩图象数据，特别是对真彩色 RGBTIFF 图象而言。

下面是 RGBTIFF 解码八位彩色 TIFF 文件得到的输出：TIFF tag listing for :ZOE. TIF

Intel number format

SubfileType	— 1
ImageWidth	— 320
ImageLength	— 200
BitsPerSample	— 8
Compression	— 32 773
PhotomtricInterp	— 3
StripOffsets	— 1 568
PlanarConfiguration	— 1
Software	— GraphicWorkshop 6. 1
ColorMap	— Offset=32 Length=1 536

在把 TIFF 图象文件交给 READTIF 处理之前，要先检查一下要表示的是哪些标记。

(1) SubFileType 标记应该是图象文件目录中的第一个标记。事实上，在版 TIFF 标准中，它已被一称为 NewSubFile 的标记所替代，这些标记说明由目录定义的图象的用途。SubFileType 的参数可以是下面这些值之一：

1—单个全规格图象。

2——当前 TIFF 文件在另一目录中定义的缩小图象。

3——多页图象的一页。

NewSubFile 标记定义的信息许多都是相同的，但是在它的参数中可以进行位设置，如下所示：

位 0——当前 TIFF 文件在另一目录定义的缩小图象。

位 1——透明屏蔽。

如果以上的位都为 0，则默认的图象为整屏图象，即相当于 SubFileType 标记的参数为 1。

(2) PhotometricInterpretation 标记 TIFF 解码程序如何处理在 TIFF 文件中发现的图象数据，它的参数如下所示：

0——图象为反视频的黑白图象

1——图象为正常黑白图象

2——图象使用 RGB 彩色

3——图象使用调色板彩色

4——图象为黑白透明屏蔽

(3) ColorMap 标记说明彩色映射的偏移量或者调色板表，可以假设调色板数据是按 3 字节 RGB 格式存放的。颜色映射表的每个入口用 48 位颜色存储，其中 24 位通常使用不到。每个 RGB 值以 3 个整数的形式存储，但是，它们不是顺序存储的。所有的红色值先存储，然后存储绿色值，最后存储兰色值，将 16 位颜色强度值转换为 8 位值很容易完成。所有要做的操作就是向右移 8 位，舍弃颜色分辨率的低 8 位。

在 TIFF 中，总是根据标记号的不同，按数字次序来存储 TIFF 标记。使用这种方法，在读完一目标图象文件之后，就可以首先遇到包含有重要信息的标记。

TIFF 文件的行格式被认为是 TIFF 说明最难理解的一部分。压缩行信息有许多方法，同样根据设置的标记的不同也有一些方法将像素压到一行中，这里的代码不能处理所有的情况。24 位彩色的 TIFF 文件是以 3 字节 RGB 值业存储行的，8 位彩色文件的行是以字节数组形式存放的，该字节是彩色映射表的指针。4 位图象的存储格式相同。5 到 7 位彩色的图象是以 8 位文件存储的，2 到 3 位彩色的图象是以 4 位文件的形式存储的。TIFF 说明也支持位平面彩色行。有些商用应用程序能正确读位平面 TIFF 文件。

(4) TIFF 文件的彩色位数由 BitsPerSample 乘以 SamplesPerPixel 决定。在 24 位文件中，这两个值应分别以 8 和 3，很多情况下会发现 24 位文件的 BitsPerSample 标记长度值大于 1，这表明它的 offset 值才真正表示偏移量，如果要检查该 offset 指向何处，就会发现它指向一个包含 3 个数值(通常为 8、8、8)的表。每一个数表示每个象素在 RGB 彩色的 3 个字节中占据的位数。理论上说，RGB 图象

象素的红、绿、蓝值可以占不同的位数。如果在这种情况下遇到一24位图象，就可以用BitsPerSample标记指出的表的3个值相加的方法求出每个象素的位数。但是，因为这里的3个值常常是相同的，就可以首先找到表的入口，然后乘以samplesPerPixel标记的参数，这样也可以获得同样的结果。传统的TIFF处理器常常使用这种方法，并且一般都有效，毫无疑问可以简化解码TIFF标记的代码。

解码程序READTIF首先打开一TIFF文件，确定它的数据类型，找出第一个图象文件目录，然后跳着检查每个标记，指出那些对打开图象有用的值。注意在decode标记的switch句子中有许多空白条件语句，这表示那些在其它上下文中有用的标记，但它们对READTIF无用。解码一个图象文件目录中的所有标记之后，就必须知道正在解码的图象的范围、颜色种数以及图象数据文件中的偏移量，假设这些值都不为0，那么图象就可以被打开了。

TIFF文件的这些参数没必要有默认值，默认值就是标记不存在显式定义时使用的值，例如，Compression标记的默认值是1(非压缩)，如果用户要创建一合法的非压缩TIFF文件，那么就可以用Compression标记的默认值。READTIF的Setdefaults函数在将范围、彩色种数和偏移量都置为0时，创建相应的默认值。

(5) TIFF文件放图象数据的偏移量由Stripoffset标记定义。TIFF允许将图象数据分成"条"，这样处理大图象就容易多了，因为没有一个条会需要大于64K的内存来存储。在这种结构下TIFF文件由一些压缩条和表示每一条在文件中的偏移量的表组成，Stripoffsets标记提供偏移量表的偏移量。大多数TIFF文件都是以一大图象(单个条)的方式存储的，Stripoffsets标记直接提供图象的偏移量。但是，为了保证TIFF解码程序能解码最多数量的TIFF文件，应该提供代码来处理两种类型的图象。因此在READTIF中真正用于读图象行的for循环就比较复杂。在多个条的文件中必须多次寻找，从表到图象数据来回移动，对文件的每个条进行适当处理。

如果设计一较复杂的TIFF解码程序，可能要考虑在开始读图象行之前就将条的偏移量表读入长整数数组中，这毫无疑问会使解码程序加快速度。虽然这需要分配一个内存缓冲，但如果在一个时刻只处理一个条就不成问题了。

正如到现在为止讨论过的所有图形图象文件那样，READTIF程序使用VGA卡的模式13H来显示TIFF文件。

HP承建世界杯赛现场报道计算机网络

惠普公司将承建1998年世界杯足球赛的现场报道计算机网络。该网络将覆盖分布于法国各地的10个比赛场地和位于巴黎的CFO(整套设备操作人员)总部以及国际新闻中心，使用光缆和3类、5类UTP(非屏蔽双绞线)电缆作为联网的介质并通过帧中继网将它们联成一体。

世界杯赛现场报道计算机网络基础设施包括31台AdvanceStack Switch 2000交换器，约200台集线器(包括100Mbit和10Mbit)，约30台支持帧中继用的Internet路由器。同时，WAN主干网采用双备份帧中继网。此外所有的PC都要配备10/100兆的网卡。考虑到网络的物理拓扑以及成本，100Mbit设备只在必须使用的场合才使用，如：为新闻摄像准备的视频流服务器与PC机之间的通路；服务器到交换器的联接；交换器到交换器的联接；在CFO总部的图象处理准备阶段等。

大多数100Mbit集线器采用的是100VG，因为其主要的高速应用就是为新闻界提供从视频服务器到PC之间的MPEG-2视频数据流(速率约为4Mbit/秒到PC机)。这个视频Intranet需要提供球队和球员的资料以及统计数字。

快速以太网链路将用于交换器到交换器的联接或交换器到服务器的联接。共享的10Mbit以太网链路(通过AdvanceStack交换型集线器)用于联接主要提供电子邮件和完成打印任务的PC机。

联网的PC机数将超过1 800台，其中约500台供新闻界使用，约400台供CFO总部使用。另外还有约600台HP打印机以及约100台HP服务器将联入网络。

网络上使用的应用软件包括新闻界视频应用的来自视频文件服务器的MPEG-2视频流。这是1个基于Web的应用程序的一部分，它来自每赛场的本地服务器，叫做"World Cup OnLine"(或者叫"Info France98"或"Info98")。办公室通用应用软件包括字处理软件、电子表格软件、数据库、商业、图象、演示图象以及Email等。而关于使用的应用软件的详细说明可查询HP Internet World Cup网址：的技术说明：http://www.grenoble.hp.com/wc98。