BMP 文件结构的探索

WhatIf 2004-9-10

一、文件格式

Bmp 文件是非常常用的位图文件，无论是游戏还是其他都被广泛使用。针对 bmp 文件的处理也有一堆现成的 api 进行调用，然而文件内部究竟怎样，如何自己来解析这样的文件呢？为了消除无聊，我用了几天时间来研究了一下，同时作为学习笔记，进行记录。

首先，整个 bmp 文件的内容可以分为 3 到 4 块。之所以分为 3 到 4 块而不是固定的值，是因为，对于 bmp 来说可能存在调色板或者一些掩码。具体稍候讨论。

第一块是 bmp 的文件头用于描述整个 bmp 文件的情况。结构如下：

```
typedef struct tagBITMAPFILEHEADER {
  WORD      bfType;
  DWORD     bfSize;
  WORD      bfReserved1;
  WORD      bfReserved2;
  DWORD     bfOffBits;
    } BITMAPFILEHEADER, *PBITMAPFILEHEADER;
```

这些信息相当有用，如果你想直接来解析 bmp 文件。第一个 bfType 用于表示文件类型，如果它是 bmp 文件，那么它这个位置的值一定是"BM" 也就是 0x4D42。第二个 bfSize 表示整个文件的字节数。第三第四个 则保留，目前无意义，最后一个相当重要，表示，位图的数据信息离文件头的偏移量，以字节为单位。

第二块是位图信息头，即 BITMAPINFOHEADER，用于描述整个位图文件的情况。以下挑重要的数据进行解释

```
typedef struct tagBITMAPINFOHEADER{
  DWORD  biSize; //表示本结构的大小
  LONG   biWidth; //位图的宽度
  LONG   biHeight; //位图的高度
  WORD   biPlanes; //永远为 1 ,由于没有用过所以 没做研究 附 msdn 解释
   //Specifies the number of planes for the target device. This value must be set to 1.
  WORD   biBitCount;//位图的位数  分为 1 4 8 16 24 32 本文没对 1 4 进行研究
  DWORD  biCompression; //本以为压缩类型，但是却另外有作用，稍候解释
  DWORD  biSizeImage; //表示位图数据区域的大小以字节为单位
  LONG   biXPelsPerMeter;
  LONG   biYPelsPerMeter;
  DWORD  biClrUsed;
  DWORD  biClrImportant;
} BITMAPINFOHEADER, *PBITMAPINFOHEADER;
```

第三块就是调色板信息或者掩码部分，如果是 8 位位图 则存放调色板 ；16 与 32 位 位图则存放 RGB 颜色的掩码，这些掩码以 DWORD 大小来存放。

最后一块就是位图的数据实体。

以上文件信息可以在任意一篇 bmp 文件结构的文章中找到描述，所以本文只是稍微带过。

二、4 字节对其问题

关于数据读取。Bmp 文件有个重要特性，那就是对于数据区域而言，每行的数据它必须凑满 4 字节，如果没有满，则用冗余的数据来补齐。这个特性直接影响到我们读取位图数据的方法，因为在我们看来 （x,y）的数据应该在 y*width+x 这样的位置上 但是因为会有冗余信息 那么必须将 width 用 width＋该行的冗余量来处理，而由于位图文件有不同的位数，所以这样的计算也不尽相同。

下面列出计算偏移量的一般公式。

首先将位图信息读入一个 UCHAR 的 buffer 中 ：

8 位：

```
int pitch;
if(width%4==0){
    pitch=width;
}else{
    pitch=width+4-width%4;
}
```

index=buffer[y*pitch+x]; 因为 8 位位图的数据区域存放的是调色板索引值，所以只需读取这个 index

16 位

```
int pitch=width+width%2;
buffer[(y*pitch+x)*2]
buffer[(i*pitch+j)*2＋1]
```

两个 UCHAR 内，存放的是(x,y)处的颜色信息

24 位

```
int pitch=width%4;
buffer[(y*width+x)*3+y*pitch];
buffer[(y*width+x)*3+y*pitch+1];
buffer[(y*width+x)*3+y*pitch+2];
```

32位

由于一个象素就是4字节 所以无需补齐

虽然计算比较繁琐，但是这些计算是必须的，否则当你的位图每行的象素数不是4的倍数，那么y*width+x带给你的是一个扭曲的图片，当然如果你想做这样的旋转，也不错啊，至少我因为一开始没有考虑（不知道这个特性） 让一个每行象素少1字节的16位图片变成了扭曲的菱形。

三、有了数据分离RGB分量。

由于我的测试代码用了GDI，所以我必须讲得到的某一个点的值 分离成 24位模式下的RGB分离，这不是一件容易的工作。位图麻烦的地方之一就是他的格式太多，所以我们还是要分格式再讨论。

8位

通过第二部分提到的操作我们得到了一个index，这个值的范围是0～255 一共256个 正好是调色板的颜色数量。

在8位bmp图片中 数据信息前256个RGBQUAD的大小开始就是调色板的信息。不过如果要组织成调色板还要一定的转换 因为里面是RGBQUAD信息 r b 两个与调色板中的顺序是颠倒的。因为我不需要调色板设置所以我字节读取到RGBQUAD数组中，并且通过下面的表达式获取RGB值：

```
UCHAR r=quad[index].rgbRed;
UCHAR g=quad[index].rgbGreen;
```

UCHAR b=quad[index].rgbBlue;

16位

这是最麻烦的一个。因为在处理时有555 565 两种格式的区别，而且还有所谓压缩类型的区别。

之前的bitmapinfoheader里面提到一个biCompression

现在我们分两种情况讨论：BI_RGB和BI_BITFIELDS

当他等于BI_RGB时 只有555 这种格式，所以可以放心大胆的进行如下的数据分离：

```
UCHAR b=buffer[(i*pitch+j)*2]&0x1F;
UCHAR g=(((buffer[(i*pitch+j)*2+1]<<6)&0xFF)>>3)+(buffer[(i*pitch+j)*2]>>5);
UCHAR r=(buffer[(i*pitch+j)*2+1]<<1)>>3;
```

希望不要被这个表达式折磨的眼花缭乱，我想既然你在看这篇文章，你就有能力阅读这样的代码，否则只能说你还没有到阅读这方面的地步，需要去学习基础的语法了。

有一点值得提醒的是由于有较多的位操作 ，所以在处理的时候在前一次操作的上面加上一对括号，我就曾经因为没有加而导致出现误差，另外虽然buffer中一个元素代表的是一个UCHAR 但是右移操作会自动增长为两字节 所以需要在进行一次与操作截取低位的1字节数据。

现在讨论BI_BITFIELDS。

这个模式下 既可以有555 也可以有565 。

555 格式 xrrrrrgggggbbbbb

565 格式 rrrrrgggggbbbbb

显然不同的格式处理不同，所以我们要首先判断处到底属于那种格式。

Bitmapinfoheader的biCompression为BI_BITFIELDS时，在位图数据区域前存在一个RGB掩码的描述是3个DWORD值，我们只需要读取其中的R或者G的掩码，来判断是那种格式。

以红色掩码为例 0111110000000000的时候就是555格式 1111100000000000就是565格式。

以下是565格式时的数据分离：

```
UCHAR b=buffer[(i*pitch+j)*2]&0x1F;
UCHAR g=(((buffer[(i*pitch+j)*2+1]<<5)&0xFF)>>2)+(buffer[(i*pitch+j)*2]>>5);
UCHAR r=buffer[(i*pitch+j)*2+1]>>3;
```

现在我们得到了RGB各自的分量，但是还有一个新的问题，那就是由于两字节表示了3个颜色 555下每个颜色最多到0x1F 565格式下最大的绿色分量也就0x3F。所以我们需要一个转换 color=color*255/最大颜色数 即可

如565下RGB(r*0xFF/0x1F,g*0xFF/0x3F,b*0xFF/0x1F)

24位

```
UCHAR b=buffer[(i*width+j)*3+realPitch];
UCHAR g=buffer[(i*width+j)*3+1+realPitch];
UCHAR r=buffer[(i*width+j)*3+2+realPitch];
```

32位

```
UCHAR b=buffer[(i*width+j)*4];
UCHAR g=buffer[(i*width+j)*4+1];
UCHAR r=buffer[(i*width+j)*4+2];
```

四、剩余的问题

当数据取到了，颜色也分离出来了 ，但是可能你绘出的位图是倒转的，这是因为有些位图的确是翻转的。通过bitmapinfoheader的biHeight可以判断是正常还是翻转，当biHeight>0的时候颠倒，它小于0

的时候正常，不过测试写到现在看到的文件都是颠倒过来的。

五、相关测试代码：

采用MFC 目的只是实现自行解析位图文件

```
void CBmpTestView::OnDraw(CDC* pDC)
{
	CBmpTestDoc* pDoc = GetDocument();
	ASSERT_VALID(pDoc);

	// TODO: 在此处为本机数据添加绘制代码

	if(filename==""){
		return;
	}
	FILE *fp=fopen(filename,"r");
	if(fp==NULL){
		pDC->TextOut(100,200,"no file found");
		return;
	}
	BITMAPFILEHEADER fileheader;
	BITMAPINFO info;

	fread(&fileheader,sizeof(fileheader),1,fp);
	if(fileheader.bfType!=0x4D42){
		pDC->TextOut(100,200,"无位图文件 请选择位图文件");
		fclose(fp);
		return ;
	}
	fread(&info.bmiHeader,sizeof(BITMAPINFOHEADER),1,fp);
	long width=info.bmiHeader.biWidth;
	long height=info.bmiHeader.biHeight;
	UCHAR *buffer=new UCHAR[info.bmiHeader.biSizeImage];
	fseek(fp,fileheader.bfOffBits,0);
	fread(buffer,info.bmiHeader.biSizeImage,1,fp);

	if(info.bmiHeader.biBitCount==8){
		int pitch;
		if(width%4==0){
			pitch=width;
		}else{
			pitch=width+4-width%4;
		}
		RGBQUAD quad[256];
		fseek(fp,fileheader.bfOffBits-sizeof(RGBQUAD)*256,0);
```

```
        fread(quad,sizeof(RGBQUAD)*256,1,fp);
        if(height>0){
            //height>0 表示图片颠倒
            for(int i=0;i<height;i++){
                for(int j=0;j<width;j++){
                    int index=buffer[i*pitch+j];
                    UCHAR r=quad[index].rgbRed;
                    UCHAR g=quad[index].rgbGreen;
                    UCHAR b=quad[index].rgbBlue;
                    pDC->SetPixel(j,height-i,RGB(r,g,b));
                }
            }
        }else{
            for(int i=0;i<0-height;i++){
                for(int j=0;j<width;j++){
                    int index=buffer[i*pitch+j];
                    UCHAR r=quad[index].rgbRed;
                    UCHAR g=quad[index].rgbGreen;
                    UCHAR b=quad[index].rgbBlue;
                    pDC->SetPixel(j,i,RGB(r,g,b));
                }
            }
        }
    }else if(info.bmiHeader.biBitCount==16){
        int pitch=width+width%2;
        if(height>0){
            //height>0 表示图片颠倒
            if(info.bmiHeader.biCompression==BI_RGB){
                //该模式只有555
                for(int i=0;i<height;i++){
                    for(int j=0;j<width;j++){
                        //5 5 5 格式
                        UCHAR b=buffer[(i*pitch+j)*2]&0x1F;
                        UCHAR
g=(((buffer[(i*pitch+j)*2+1]<<6)&0xFF)>>3)+(buffer[(i*pitch+j)*2]>>5);
                        UCHAR r=(buffer[(i*pitch+j)*2+1]<<1)>>3;
                        pDC->SetPixel(j,height-
i,RGB((r*0xFF)/0x1F,(g*0xFF)/0x1F,(b*0xFF)/0x1F));
                    }
                }
            }else if(info.bmiHeader.biCompression==BI_BITFIELDS){
                //该模式在bitmapinfoheader之后存在RGB掩码 每个掩码1 DWORD
                fseek(fp,fileheader.bfOffBits-sizeof(DWORD )*3,0);
                DWORD  rMask;
```

```
                    fread(&rMask,sizeof(DWORD ),1,fp);
                    if(rMask==0x7C00){
                        // 5 5 5 格式
                        MessageBeep(0);
                        for(int i=0;i<height;i++){
                            for(int j=0;j<width;j++){
                                UCHAR b=buffer[(i*pitch+j)*2]&0x1F;
                                UCHAR
g=(((buffer[(i*pitch+j)*2+1]<<6)&0xFF)>>3)+(buffer[(i*pitch+j)*2]>>5);
                                UCHAR r=(buffer[(i*pitch+j)*2+1]<<1)>>3;
                                pDC->SetPixel(j,height-
i,RGB((r*0xFF)/0x1F,(g*0xFF)/0x1F,(b*0xFF)/0x1F));
                            }
                        }
                    }else if(rMask==0xF800){
                        //5 6 5 格式
                        for(int i=0;i<height;i++){
                            for(int j=0;j<width;j++){
                                UCHAR b=buffer[(i*pitch+j)*2]&0x1F;
                                UCHAR
g=(((buffer[(i*pitch+j)*2+1]<<5)&0xFF)>>2)+(buffer[(i*pitch+j)*2]>>5);
                                UCHAR r=buffer[(i*pitch+j)*2+1]>>3;
                                pDC->SetPixel(j,height-
i,RGB(r*0xFF/0x1F,g*0xFF/0x3F,b*0xFF/0x1F));
                            }
                        }
                    }
                }
            }else{
                if(info.bmiHeader.biCompression==BI_RGB){
                    //该模式只有555
                    for(int i=0;i<0-height;i++){
                        for(int j=0;j<width;j++){
                            //5 5 5 格式
                            UCHAR b=buffer[(i*pitch+j)*2]&0x1F;
                            UCHAR
g=(((buffer[(i*pitch+j)*2+1]<<6)&0xFF)>>3)+(buffer[(i*pitch+j)*2]>>5);
                            UCHAR r=(buffer[(i*pitch+j)*2+1]<<1)>>3;
                            pDC-
>SetPixel(j,i,RGB((r*0xFF)/0x1F,(g*0xFF)/0x1F,(b*0xFF)/0x1F));
                        }
                    }
                }else if(info.bmiHeader.biCompression==BI_BITFIELDS){
                    //该模式在bitmapinfoheader之后存在RGB掩码 每个掩码1 DWORD
```

```
                    fseek(fp,fileheader.bfOffBits-sizeof(DWORD )*3,0);
                    DWORD  rMask;
                    fread(&rMask,sizeof(DWORD ),1,fp);
                    if(rMask==0x7C00){
                         // 5 5 5 格式
                         MessageBeep(0);
                         for(int i=0;i<0-height;i++){
                              for(int j=0;j<width;j++){
                                   UCHAR b=buffer[(i*pitch+j)*2]&0x1F;
                                   UCHAR
g=(((buffer[(i*pitch+j)*2+1]<<6)&0xFF)>>3)+(buffer[(i*pitch+j)*2]>>5);
                                   UCHAR r=(buffer[(i*pitch+j)*2+1]<<1)>>3;
                                   pDC-
>SetPixel(j,i,RGB((r*0xFF)/0x1F,(g*0xFF)/0x1F,(b*0xFF)/0x1F));
                              }
                         }
                    }else if(rMask==0xF800){
                         //5 6 5 格式
                         for(int i=0;i<0-height;i++){
                              for(int j=0;j<width;j++){
                                   UCHAR b=buffer[(i*pitch+j)*2]&0x1F;
                                   UCHAR
g=(((buffer[(i*pitch+j)*2+1]<<5)&0xFF)>>2)+(buffer[(i*pitch+j)*2]>>5);
                                   UCHAR r=buffer[(i*pitch+j)*2+1]>>3;
                                   pDC->SetPixel(j,i,RGB(r*0xFF/0x1F,g*0xFF/0x3F,b*0xFF/0x1F));
                              }
                         }
                    }
               }
          }
          //pDC->TextOut(100,200,"16位图");
     }else if(info.bmiHeader.biBitCount==24){
          int pitch=width%4;
          //b g r
          if(height>0){
               //height>0 表示图片颠倒
               for(int i=0;i<height;i++){
                    int realPitch=i*pitch;
                    for(int j=0;j<width;j++){
                         UCHAR b=buffer[(i*width+j)*3+realPitch];
                         UCHAR g=buffer[(i*width+j)*3+1+realPitch];
                         UCHAR r=buffer[(i*width+j)*3+2+realPitch];
                         pDC->SetPixel(j,height-i,RGB(r,g,b));
                    }
```

```
            }
        }else{
            for(int i=0;i<0-height;i++){
                int realPitch=i*pitch;
                for(int j=0;j<width;j++){
                    UCHAR b=buffer[(i*width+j)*3+realPitch];
                    UCHAR g=buffer[(i*width+j)*3+1+realPitch];
                    UCHAR r=buffer[(i*width+j)*3+2+realPitch];
                    pDC->SetPixel(j,i,RGB(r,g,b));
                }
            }
        }

        //pDC->TextOut(100,200,"24位图");

    }else if(info.bmiHeader.biBitCount==32){
        // b g r a
        if(height>0){
            //height>0 表示图片颠倒
            for(int i=0;i<0-height;i++){
                for(int j=0;j<width;j++){
                    UCHAR b=buffer[(i*width+j)*4];
                    UCHAR g=buffer[(i*width+j)*4+1];
                    UCHAR r=buffer[(i*width+j)*4+2];
                    pDC->SetPixel(j,height-i,RGB(r,g,b));
                }
            }
        }else{
            for(int i=0;i<height;i++){
                for(int j=0;j<width;j++){
                    UCHAR b=buffer[(i*width+j)*4];
                    UCHAR g=buffer[(i*width+j)*4+1];
                    UCHAR r=buffer[(i*width+j)*4+2];
                    pDC->SetPixel(j,i,RGB(r,g,b));
                }
            }
        }
        //pDC->TextOut(100,200,"32位图");
    }
    delete buffer;
    fclose(fp);
}
```