BMP文件格式详解（BMP file format）

BMP文件格式，又称为Bitmap（位图）或是DIB(Device-Independent Device，设备无关位图)，是Windows系统中广泛使用的图像文件格式。由于它可以不作任何变换地保存图像像素域的数据，因此成为我们取得RAW数据的重要来源。Windows的图形用户界面（graphical user interfaces）也在它的内建图像子系统GDI中对BMP格式提供了支持。

下面以Notepad++为分析工具，结合Windows的位图数据结构对BMP文件格式进行一个深度的剖析。

BMP文件的数据按照从文件头开始的先后顺序分为四个部分：

* bmp文件头(bmp file header)：提供文件的格式、大小等信息
* 位图信息头(bitmap information)：提供图像数据的尺寸、位平面数、压缩方式、颜色索引等信息
* 调色板(color palette)：可选，如使用索引来表示图像，调色板就是索引与其对应的颜色的映射表
* 位图数据(bitmap data)：就是图像数据啦^\_^

下面结合Windows结构体的定义，通过一个表来分析这四个部分。

|  |  |  |
| --- | --- | --- |
| 数据段名称 | 对应的Windows结构体定义 | 大小（Byte） |
| bmp文件头 | BITMAPFILEHEADER | 14 |
| 位图信息头 | BITMAPINFOHEADER | 40 |
| 调色板 |  | 由颜色索引数决定 |
| 位图数据 |  | 由图像尺寸决定 |

我们一般见到的图像以24位图像为主，即R、G、B三种颜色各用8个bit来表示，这样的图像我们称为真彩色，这种情况下是不需要调色板的，也就是说位图信息头后面紧跟的就是位图数据了。因此，我们常常见到有这样一种说法：位图文件从文件头开始偏移54个字节就是位图数据了，这其实说的是24或32位图的情况。这也就解释了我们按照这种程序写出来的程序为什么对某些位图文件没用了。

下面针对一幅特定的图像进行分析，来看看在位图文件中这四个数据段的排布以及组成。

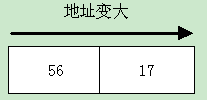
我们使用的图像显示如下：

D:\Works_YOKOGAWA\git_repository\test_github_repository\my_test\doc\ImageFormat\KeyTop_UNIT_kSPm.bmp

这是一幅16位的位图文件，因此它是含有调色板的。

在拉出图像数据进行分析之前，我们首先进行几个约定：

1. 在BMP文件中，如果一个数据需要用几个字节来表示的话，那么该数据的存放字节顺序为“低地址村存放低位数据，高地址存放高位数据”。如数据0x1756在内存中的存储顺序为：



这种存储方式称为小端方式(little endian) , 与之相反的是大端方式（big endian）。对两者的使用情况有兴趣的可以深究一下，其中还是有学问的。

2. 以下所有分析均以字节为序号单位进行。

下面我们对从文件中拉出来的数据进行剖析：

（1）bmp文件头

Windows为bmp文件头定义了如下结构体：

typedef struct tagBITMAPFILEHEADER

{

UINT16 bfType;

DWORD bfSize;

UINT16 bfReserved1;

UINT16 bfReserved2;

DWORD bfOffBits;

} BITMAPFILEHEADER;

其中，

|  |  |  |  |
| --- | --- | --- | --- |
| 变量名 | 地址偏移 | 大小（字节） | 作用 |
| bfType | 0000h | 2 | 说明文件的类型，可取值为： \* BM – Windows 3.1x, 95, NT,….. \* BA – OS/2 Bitmap Array \* CI – OS/2 Color Icon \* CP – OS/2 Color Pointer \* IC – OS/2 Icon \* PT – OS/2 Pointer |
| bfSize | 0002h | 4 | 说明该位图文件的大小，用字节为单位 |
| bfReserved1 | 0006h | 2 | 保留，必须设置为0 |
| bfReserved2 | 0008h | 2 | 保留，必须设置为0 |
| bfOffBits | 000Ah | 4 | 说明从文件头开始到实际的图像数据之间的字节的偏移量。这个参数是非常有用的，因为位图信息头和调色板的长度会根据不同情况而变化，所以我们可以用这个偏移值迅速地从文件中读取到位图数据。 |

对照文件数据我们看到：



1-2： 424Dh = 'BM',表示这是Windows支持的位图格式。

3-5： 0000960Dh = 38413B = 3.478 kB，通过查询文件属性发现一致。

6-9： 这是两个保留段，为0。

A-D：00000436h = 1078。即从文件头到位图数据需偏移1078字节。我们稍后将验证这个数据。

共有14个字节。

（2）位图信息头

同样地，Windows为位图信息头定义了如下结构体：

|  |  |  |  |
| --- | --- | --- | --- |
| 变量名 | 地址偏移 | 大小 | 作用 |
| biSize | 0x000E | 4字节 | BITMAPINFOHEADER结构所需要的字节数。 |
| biWidth | 0x0012 | 4字节 | 说明图像的宽度，用像素为单位 |
| biHeight | 0x0016 | 4字节 | 说明图像的高度，用像素为单位 注： 这个值除了用于描述图像的高度之外，他还有另一个用处，就是指明该图像是倒向的位图，还是正向的位图。  如果该值是一个正数，说明图像是倒向的，如果该值是一个负数，则说明图像是正向的。大多数的BMP文件都是倒向的位图，也就是高度值是一个正数。 |
| biPlanes | 0x001A | 2字节 | 为目标设备说明颜色平面数，其值将总是设为1 |
| biBitCount | 0x001C | 2字节 | 说明比特数/像素，其值为1、4、8、16、24或32 |
| biCompression | 0x001E | 4字节 | 说明图像数据压缩的类型。取值范围为： 0 BI\_RGB 不压缩 1 BI\_RLE8 8比特游程编码（RLE），只用于8位位图；  2 BI\_RLE4 4比特游程编码（RLE），只用于4位位图； 3 BI\_BITFIELDS比特域，用于16、32位位图； 4 BI\_JPEG JPEG 位图含JPEG图像（仅用于打印机） 5 BI\_PNG PNG位图含PNG图像（仅用于打印机） |
| biSizeImage | 0x0022 | 4字节 | 说明图像的大小 以字节为单位。当用BI\_RGB格式时，可设置为0； |
| biXPelsPerMeter | 0x0026 | 4字节 | 说明水平分辨率，用像素/米表示，有符号整数 |
| biYPelsPerMeter | 0x002A | 4字节 | 说明垂直分辨率，用像素/米表示，有符号整数 |
| biClrUsed | 0x002E | 4字节 | 说明位图实际使用的彩色表中的颜色索引数（设为0的话，则说明使用所有调色板项） |
| biClrImportant | 0x0032 | 4字节 | 说明对图像显示有重要影响的颜色索引的数目；如果是0，表示都重要。 |

对照上面的图像数据：



12-15：0000003Ah = 58，图像宽为58像素，与文件属性一致。

16-19：00000028h = 40，图像高为40像素，与文件属性一致。这是一个正数，说明图像数据是从图像左下角到右上角排列的。

1A-1B：0001h, 该值总为1。

1C-1D：0008h = 8, 表示每个像素占8个比特，即该图像共有256种颜色。

1E-21：00000000h，BI\_RGB， 说明本图像不压缩。

22-25：00000960h，图像的大小，因为使用BI\_RGB，所以设置为0。

26-29：00000000h，水平分辨率，缺省。

2A-2D：00000000h，垂直分辨率，缺省。

2E-31：00000000h = 0, 说明使用所有调色板项。

32-35：00000000h = 0,表示都重要。

（3）调色板

下面的数据就是调色板了。前面也已经提过，调色板其实是一张映射表，标识颜色索引号与其代表的颜色的对应关系。它在文件中的布局就像一个二维数组palette[N][4],其中N表示总的颜色索引数，每行的四个元素分别表示该索引对应的B、G、R和Alpha的值，每个分量占一个字节。如不设透明通道时，Alpha为0。因为前面知道，本图有256个颜色索引，因此N = 256。索引号就是所在行的行号，对应的颜色就是所在行的四个元素。这里截取一些数据来说明：



索引：(蓝，绿，红，Alpha)

0号：(00，00，00，00)

1号：(FF，FF，FF，00)

2号：(F4，FF，FF，00)

3号：(11，11，11，00)

4号：(22，22，22，00)

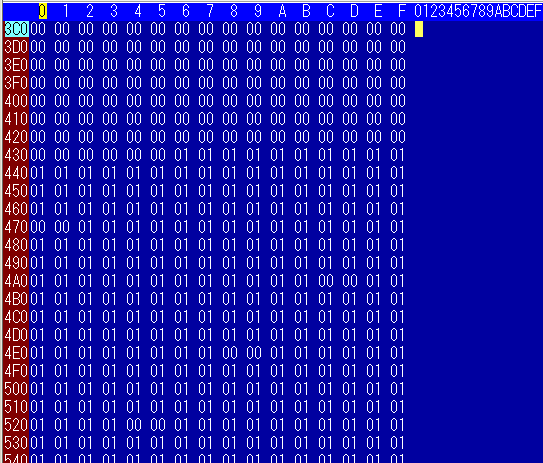
5号：(33，33，33，00) 等等。

一共有256种颜色，每个颜色占用4个字节，就是一共1024个字节，再加上前面的文件信息头和位图信息头的54个字节加起来一共是1078个字节。也就是说在位图数据出现之前一共有1078个字节，与我们在文件信息头得到的信息：文件头到文图数据区的偏移为1078个字节一致！

（4）位图数据

下面就是位图数据了，每个像素占一个字节，取得这个字节后，以该字节为索引查询相应的颜色，并显示到相应的显示设备上就可以了。

注意：由于位图信息头中的图像高度是正数，所以位图数据在文件中的排列顺序是从左下角到右上角，以行为主序排列的。



也即我们见到的第一个像素01是图像最左下角的数据，第二个像素01为图像最后一行第二列的数据，…一直到最后一行的最后一列数据，后面紧接的是倒数第二行的第一列的数据，依此类推。

如果图像是24位或是32位数据的位图的话，位图数据区就不是索引而是实际的像素值了。下面说明一下，此时位图数据区的每个像素的RGB颜色阵列排布：

24位RGB按照BGR的顺序来存储每个像素的各颜色通道的值，一个像素的所有颜色分量值都存完后才存下一个下一个像素，不进行交织存储。

32位数据按照BGRA的顺序存储，其余与24位位图的方式一样。

像素的排布规则与前述一致。

对齐规则

讲完了像素的排列规则以及各像素的颜色分量的排列规则，最后我们谈谈数据的对齐规则。我们知道Windows默认的扫描的最小单位是4字节，如果数据对齐满足这个值的话对于数据的获取速度等都是有很大的增益的。因此，BMP图像顺应了这个要求，要求每行的数据的长度必须是4的倍数，如果不够需要进行比特填充（以0填充），这样可以达到按行的快速存取。这时，位图数据区的大小就未必是 图片宽×每像素字节数×图片高 能表示的了，因为每行可能还需要进行比特填充。

填充后的每行的字节数为：

http://images.cnblogs.com/cnblogs_com/jason_yao/bmp_9.png

，其中BPP（Bits Per Pixel）为每像素的比特数。

在程序中，我们可以表示为：

int iLineByteCnt = (((m\_iImageWidth \* m\_iBitsPerPixel) + 31) >> 5) << 2;

这样，位图数据区的大小为：

m\_iImageDataSize = iLineByteCnt \* m\_iImageHeight;

我们在扫描完一行数据后，也可能接下来的数据并不是下一行的数据，可能需要跳过一段填充数据：

skip = 4 - ((m\_iImageWidth \* m\_iBitsPerPixel)>>3) & 3;

五、拾遗

至此，我们通过分析一个具体的位图文件例子详细地剖析了位图文件的组成。需要注意的是：我们讲的主要是PC机上的位图文件的构成，对于嵌入式平台，可能在调色板数据段与PC机的不同。如在嵌入式平台上常见的16位r5g6b5位图实际上采用的掩模的方式而不是索引的方式来表示图像。此时，在调色板数据段共有四个部分，每个部分为四个字节，实际表示的是彩色版规范。即：

第一个部分是红色分量的掩模

第二个部分是绿色分量的掩模

第三个部分是蓝色分量的掩模

第四个部分是Alpha分量的掩模（缺省为0）

典型的调色板规范在文件中的顺序为为：

00F8 0000 E007 0000 1F00 0000 0000 0000

其中

00F8 0000为FB00h=1111100000000000（二进制），是蓝红分量的掩码。

　　E007 0000为 07E0h=0000011111100000（二进制），是绿色分量的掩码。

　 1F00 0000为001Fh=0000000000011111（二进制），是蓝色分量的掩码。

　 0000 0000设置为0。

将掩码跟像素值进行“与”运算再进行移位操作就可以得到各色分量值。看看掩码，就可以明白事实上在每个像素值的两个字节16位中，按从高到低取5、6、5位分别就是r、g、b分量值。取出分量值后把r、g、b值分别乘以8、4、8就可以补齐每个分量为一个字节，再把这三个字节按BGR组合，放入存储器，就可以转换为24位标准BMP格式了。

这样我们假设在位图数据区有一个像素的数据在文件中表示为02 F1。这个数据实际上应为F102：

r = (F102 AND F800) >> 8 = F0h = 240

g= (F102 AND 07E0）>> 3 = 20h = 32

　 b=(F102 AND 001F) << 3 = 10h =16

至此我们就可以显示了。（本文结束）