bmp文件大体上分成四个部分

|  |
| --- |
| 位图文件头BITMAPFILEHEADER |
| 位图信息头BITMAPINFOHEADER |
| 调色板Palette |
| 实际的位图数据ImageDate |

第一部分为位图文件头**BITMAPFILEHEADER**，是一个结构，其定义如下：

　　typedef struct tagBITMAPFILEHEADER {

　　WORD bfType;

　　DWORD bfSize;

　　WORD bfReserved1;

　　WORD bfReserved2;

　　DWORD bfOffBits;

　　} BITMAPFILEHEADER;

　　这个结构的长度是固定的，为14个字节(WORD为无符号16位整数，DWORD为无符号32位整数)，各个域的说明如下：

**0** 　**bfType**

指定文件类型，必须是0x424D，即字符串“BM”

**2 bfSize**

　　指定文件大小，包括这14个字节。

**6** **bfReserved1，bfReserved2**

　　为保留字，不用考虑

**10**　**bfOffBits**

为从文件头到实际的位图数据的偏移字节数，即图中前三个部分的长度之和。

　　第二部分为位图信息头**BITMAPINFOHEADER**，也是一个结构，其定义如下：

　　typedef struct tagBITMAPINFOHEADER{

　　DWORD biSize;

　　LONG biWidth;

　　LONG biHeight;

　　WORD biPlanes;

　　WORD biBitCount

　　DWORD biCompression;

　　DWORD biSizeImage;

　　LONG biXPelsPerMeter;

　　LONG biYPelsPerMeter;

　　DWORD biClrUsed;

　　DWORD biClrImportant;

　　} BITMAPINFOHEADER;

　　这个结构的长度是固定的，为40个字节(LONG为32位整数)，各个域的说明如下：

**14**　**biSize**

　　指定这个结构的长度，为40。

**18**　**biWidth**

　　指定图象的宽度，单位是象素。

**22**　**biHeight**

　　指定图象的高度，单位是象素。

**26**　**biPlanes**

　　必须是1，不用考虑。

**28**　**biBitCount**

　　指定表示颜色时要用到的位数，常用的值为1(黑白二色图), 4(16色图), 8(256色),16(16bit高彩色)， 24(24bit真彩色)，32(32bit增强型真彩色)。

**30**　**biCompression**

　　指定位图是否压缩，有效的值为BI\_RGB，BI\_RLE8，BI\_RLE4，BI\_BITFIELDS(都是一些Windows定义好的常量)。要说明的是，Windows位图可以采用RLE4，和RLE8的压缩格式，但用的不多。我们今后所讨论的只有第一种不压缩的情况，即biCompression为BI\_RGB的情况。

**34**　**biSizeImage**

　　指定实际的位图数据占用的字节数，其实也可以从以下的公式中计算出来：

　　biSizeImage=biWidth’ × biHeight

　　要注意的是：上述公式中的biWidth’必须是4的整倍数(所以不是biWidth，而是biWidth’，表示大于或等于biWidth的，最接近4的整倍数。举个例子，如果biWidth=240，则biWidth’=240；如果biWidth=241，biWidth’=244)。

　　如果biCompression为BI\_RGB，则该项可能为零

**38**　**biXPelsPerMeter**

　　指定目标设备的水平分辨率，单位是每米的象素个数，关于分辨率的概念，我们将在第4章详细介绍。

**42**　**biYPelsPerMeter**

　　指定目标设备的垂直分辨率，单位同上。

**46**　**biClrUsed**

　　指定本图象实际用到的颜色数，如果该值为零，则用到的颜色数为2。

**50**　**biClrImportant**

指定本图象中重要的颜色数，如果该值为零，则认为所有的颜色都是重要的。

　　第三部分为调色板**Palette**，当然，这里是对那些需要调色板的位图文件而言的。有些位图，如真彩色图是不需要调色板的，BITMAPINFOHEADER后直接是位图数据。

　　调色板实际上是一个数组，共有biClrUsed个元素(如果该值为零，则有2个元素)。数组中每个元素的类型是一个RGBQUAD结构，占4个字节，其定义如下：

　　typedef struct tagRGBQUAD {

　　BYTE rgbBlue; //该颜色的蓝色分量

　　BYTE rgbGreen; //该颜色的绿色分量

　　BYTE rgbRed; //该颜色的红色分量

　　BYTE rgbReserved; //保留值

} RGBQUAD;

|  |  |  |  |
| --- | --- | --- | --- |
| **颜色** | R | G | B |
| 红 | 255 | 0 | 0 |
| 蓝 | 0 | 255 | 0 |
| 绿 | 0 | 0 | 255 |
| 黄 | 255 | 255 | 0 |
| 紫 | 255 | 0 | 255 |
| 青 | 0 | 255 | 255 |
| 白 | 255 | 255 | 255 |
| 黑 | 0 | 0 | 0 |
| 灰 | 128 | 128 | 128 |

　　第四部分就是实际的图象数据了。对于用到调色板的位图，图象数据就是该象素颜在调色板中的索引值。对于真彩色图，图象数据就是实际的R、G、B值。下面针对2色、16色、256色位图和真彩色位图分别介绍。

　　对于2色位图，用1位就可以表示该象素的颜色(一般0表示黑，1表示白)，所以一个字节可以表示8个象素。

　　对于16色位图，用4位可以表示一个象素的颜色，所以一个字节可以表示2个象素。

　　对于256色位图，一个字节刚好可以表示1个象素。

对于真彩色图，三个字节才能表示1个象素。

　　要注意两点：

　　(1) 每一行的字节数必须是4的整倍数，如果不是，则需要补齐。这在前面介绍biSizeImage时已经提到了。

(2) 一般来说，.BMP文件的数据从下到上，从左到右的。也就是说，从文件中最先读到的是图象最下面一行的左边第一个象素，然后是左边第二个象素……接下来是倒数第二行左边第一个象素，左边第二个象素……依次类推 ，最后得到的是最上面一行的最右一个象素。

|  |  |  |
| --- | --- | --- |
| **单色位图（320\*240）** | | |
| 文件头 | 0 位图类型 | ‘BM’或 0x42、0x4D |
| 2 文件大小 | 9664 或 0xc0、0x25、0x00、0x00 |
| 6 | 0 |
| 10 文件开始到数据开始之间的偏移量 | 62 或 0x3e、0x00、0x00、0x00 |
| 信息头 | 14 信息头的长度 | 40 或 0x28、0x00、0x00、0x00 |
| 18 位图的宽度 | 320 或 0x40、0x01、0x00、0x00 |
| 22 位图的高度 | 240 或 0xf0、0x00、0x00、0x00 |
| 26 | 1 或 0x01、0x00 |
| 28 像素的位数 | 1 或 0x01、0x00 |
| 30 压缩 | 0 |
| 34 位图数据的大小 | 9602 或 0x82、0x25、0x00、0x00 |
| 38 水平分辨率 |  |
| 42 垂直分辨率 |  |
| 46 使用的颜色数 | 0 |
| 50 重要的颜色数 | 0 |
| 调色板 | 54 白（1） | 0XFF、0XFF、0XFF、0X00 |
| 58 黑（0） | 0X00、0X00、0X00、0X00 |
| 实际图像数据 | 62 | 9602个数据 |

**单色BMP格式**

黑白图，每个像素只占一位，而颜色表总共有两项，占8个字节。故图像数据区的开始是 0x36 + 8 = 0x3E字节。

颜色表总是从0x36=54字节开始。

00000000代表黑色，00FFFFFF代表白色。比如F0这样一个字节，实际表示八个像素，前面四个胃白色，后面四个像素为黑色

**16色BMP格式**

16个颜色，每个像素占据4位，颜色表总共16项，占据64个字节，故数据区是从 0x36 + 0x40 = 0x76字节开始。

**256色BMP格式**

256个颜色，每个颜色占据8位，颜色表总共256项，占据 256 x 4 = 1024字节。故图像数据区开始是 第0x36 + 0x400 = 0x436字节。

|  |  |  |
| --- | --- | --- |
| **16bit高彩色位图** | | |
| 文件头 | 0 位图类型 | ‘BM’或 0x42、0x4D |
| 2 文件大小 |  |
| 6 | 0 |
| 10 文件开始到数据开始之间的偏移量 | 54 或 0x36、0x00、0x00、0x00 |
| 信息头 | 14 信息头的长度 | 40 或 0x28、0x00、0x00、0x00 |
| 18 位图的宽度 |  |
| 22 位图的高度 |  |
| 26 | 1 或 0x01、0x00 |
| 28 像素的位数 | 16 或 0x10、0x00 |
| 30 压缩 |  |
| 34 位图数据的大小 |  |
| 38 水平分辨率 |  |
| 42 垂直分辨率 |  |
| 46 使用的颜色数 |  |
| 50 重要的颜色数 |  |
| 实际图像数据 | 54 | 三个字节表示一个像素 |

**BMP图片的重要特征**

数据区里的数据是线性的，行主序，依次是 点一的B值，点一的G值，点一的R值，点二的B值，点二的G值，点二的R值，等等，需要注意的是，Windows中普遍采用了行倒向扫描的约定，即，BMP文件中原点在左下角，图像矩阵数组第一行实际是图片的最后一行。

还有一点也需要注意：Windows要求每一行的数据的长度必须是4Bytes的整数倍，如果不是，要以值为0的字节补充，如果读取的时候不处理，会得到一个倾斜的图像。

unsigned char data[9604];

FATFS fs;

FRESULT res;

UINT br;

**/\***

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**\* Function name : Pic\_Viewer();**

**\* Description : 解码bmp文件**

**\* Input : nop**

**\* Output : nop**

**\* Return : nop**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**\*/**

void Pic\_Viewer(void)

{

FIL file;

u16 height;

u16 width;

u8 bit\_count;

res = f\_mount(0, &fs);

res = f\_open(&file, "stm32.bmp", FA\_OPEN\_EXISTING | FA\_READ);  **//打开图片**

if(res!=FR\_OK) printf("bmp图片错误... ");

**//读取文件信息【文件头（14字节），信息表（40字节），调色板（不定）】**

**//真彩图无调色板（所以是0字节）**

**//其她图按颜色数来规定调色板的字节数【单色（8），16色（64），256色（1024）】**

res = f\_read(&file, data, 62, &br);

if (data[0]==0x42 && data[1]==0x4D) **//文件格式标记为BM**

{

width = data[18] + (data[19] << 8) + (data[20] << 16) + (data[21] << 24);

height = data[22] + (data[23] << 8) + (data[24] << 16) + (data[25] << 24);

if ((width == 320) && (height == 240))**//文件像素为Width320\*Height240**

{

bit\_count = data[28] +(data[29] << 8);

switch (bit\_count)

{

case 1: **//单色**

{

res = f\_read(&file, data, 9604, &br); //读取9604个数据

LCM\_Show\_bmp(data);

break;

}

case 4: **//16色**

{

break;

}

case 8: **//256色**

{

break;

}

case 16: **//16bit高彩色**

{

break;

}

case 24: **//24bit真彩色**

{

break;

}

case 32: **//32bit增强型真彩色**

{

break;

}

default:

break;

}

}

}

f\_close(&file); **// Close open files**

f\_mount(0, NULL); **// Unregister work area prior to discard it**

}