비트맵 파일의 구조

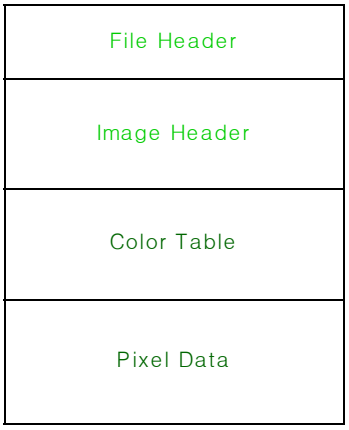
Posted 2008.06.05 21:48

BMP화일은 이미지 파일중에 가장 간단한 파일중 하나입니다. 기본적으로 bmp화일은 1,4,8,16,24 및 32비트를 지원합니다. 하지만 16과 32비트는 거의 사용하는 예가 드뭅니다. bmp화일은 사용이 쉬운만큼 용량이 큰 파일입니다. 그렇게 때문에 이 파일역시 기본적인 압축형식을 지원합니다. 그 방식은 간단한 압축방식인 Run-Length Compresstion방식이며, 4비트와 8비트를 지원합니다.(RLE4, RLE8). 어쨌든 이 압축방식의 특성은 간단한 색상의 그림블록의 경우에는 비교적 효율이 좋으나, 복잡한 색상 혹은 완벽히 랜덤한 색상의 경우는 전혀 압축이 되지 않을수 있습니다. 그래서, 대부분의 BMP파일은 압축을 사용하는 예가 드물게 됩니다.

몇해동안 bmp화일역시 여러방식이 존재했습니다. 하지만, 대부분은 사장이 되고, 현재 남은 형식은 Windows version3를 사용하고 있습니다. 하지만, 불행히도 가끔 이 방식을 따르지 않는 형식이 존재합니다. 이는 주로 OS/2에서 사용하는 방식입니다. 더 자세한 것은 다음에 설명하도록 하겠습니다.

Bitmap데이터 형식

기본적인 Bitmap화일의 구조는 다음과 같습니다.



보는것과 같이 BMP파일은 상당히 단순한 구조를갖고 있습니다. 또한, 만약 Window환경에서 프로그래밍을 하는 경우에는 windows.h 헤더화일에 정의가 되어 있습니다. 다른 환경에서 프로그래밍을 하는 경우에는 아래와 같이 설명하는 예를 직접 정의한 헤더화일을 가지고 계셔야 합니다.

File Header

파일 헤더는 BITMAPFILEHEAER라는 구조체에 정의되어 있습니다. 이 헤더의 주된 임무(?)는 현재의 파일 포맷이 정말 BMP인지에 대한 정보를 주는것입니다. 대부분의 파일은 자신의 성격에 맞게 확장자 이름이 정해져있습니다. BMP화일 역시 확장자는 *.BMP로 정의 되어 있겠죠. 그러나 불행히도, 우연히 다른 파일이 XX.BMP라고 썼을수 있으므로, BMP디코더가 이를 분석해 내야 합니다. 그 분석방법에는 아래와 같이 세가지방법이 있습니다. 그 방법을 알기에 앞서 BITMAPFILEHEADER의 레이어별 구조를 보면,

Filed Name	Size in Bytes	Description
bfType	2	BM이라는 캐릭터형이 저장되어있습니다.
bfSize	4	파일의 전체 크기를 표시합니다.
bfReserved1	2	사용하지 않습니다.
bfReserved2	2	사용하지 않습니다.
bfOffBits	4	실질 데이터(pixel)의 시작좌표를 OFFSET주소를 나타냅니다.

위 구조를 이해하셨다면, 현재의 파일포맷이 정말 BMP파일인지를 구분하는 방법을 아래와 같이 추려낼수 있습니다.

1. 파일의 첫 2바이트가 BM으로 시작하는가.

2. 바이트 단위의 정확한 파일크기와 헤더내의 bfSize가 있어야할 값과 일치하는가.
3. bfReserved1과 bfReserved2가 있어야할 위치에 0값이 있는가.

세가지를 모두 체크하면, bmp파일 여부를 판명할수 있습니다. 하지만, 디코더의 크기를 줄이기 위해, 첫 번째것만 체크하는 경우도 있습니다.

파일의 헤더의 역할은 포맷의 명확성을 표시하는 한편, 파일내에 있어서 실질 데이터(pixel data)의 오프셋 주소값을 나타내줍니다. 다시말해, 실질데이터의 파일내 위치를 정의해 주는 역할을 합니다. 이는 bfOffBits에서 설정해있습니다. 대부분의 어플리케이션에서는 실질데이터는 BITMAPINFOHEADER구조체나 팔렛트다음에 위치하도록 되어있습니다. 어쨌든, 가끔 그 사이에 다른 데이터가 존재할수 있으므로(이를 filler byte라함), 명확성을 위해, bfOffbits를 이용하여, BITMAPFILEHEADER구조체와 실질데이터를 구분합니다.

Image Header

이미지 헤더는 BITMAPFILEHEADER 구조체 다음에 바로 위치하도록 되어있습니다. 이미지 헤더는 2가지 종류가 있습니다. 앞에서 언급했듯이 BMP파일은 두가지 종류가 있습니다. windows version 3와 OS/2에서 사용하는 BMP가 있다고 말씀드렸습니다. 이 경우, 이를 구분하여 저장해야 합니다. 먼저 전자의 경우에 저장해야할 이미지헤더구조체는 BITMAPINFOHEADER에 정의 되어있고, 후자의 경우는 BITAMPCOREHEADER에 정의 되어 있습니다. 이 두가지를 구분하는 방법은 어떠한 식별 필드가 존재하는 것이 아니라, 그 크기로 식별을 해야합니다. 이를 설명하기 위해서는 일단 위 두가지의 구조체를 알아보도록 하겠습니다.

BITAMPINFOHEADER구조체

Field Name	Size	Description
biSize	4	헤더 크기(최소 40bytes)
biWidth	4	이미지 폭
biHeight	4	이미지 높이
biPlanes	2	현재 지원값은 1입니다.
biBitCount	2	비트수 1,4,8,16,24,32
biCompression	4	압축타입 : BI_RGB(0),BI_RLE8(1),BI_RLE4(2),BI_BITFIELDS(3)
biSizeImage	4	이미지 크기
biXPelsPerMeter	4	미터당 픽셀수 x축
biYPelsPerMeter	4	미터당 픽셀수 y축
biClrUsed	4	실질적으로 사용될 컬러맵의 엔트리수
biClrImportant	4	주로 사용되는 컬러수

BITMAPCOREHEADER구조체

Field Name	Size	Description
bcSize	4	헤더 크기(12bytes)
bcWidth	2	이미지 폭
bcHeight	2	이미지 높이
bcPlanes	2	현재 지원값은 1입니다.
bcBitCount	2	비트수 1,4,8,24

위의 구조체를 보면, 두가지를 구분할 방법을 찾으셨을껍니다. 바로 헤더의 크기로 구분하면 되겠죠 ?윈도우에서 지원하는 BMP의 경우 헤더의 크기가 최소 40바이트가 필요합니다. 그렇기 때문에 수치적으로 쉽게 구분을 할수 있습니다.

이제 구조체를 하나하나 보면, 이미지헤더의 역할은 이미지의 비트수와 크기, 차원, 그리고 압축여부를 알려주는 역할을 합니다. 즉, 이미지 자체의 정보를 갖고 있다고 보시면됩니다. 위 구조체는 한번 읽어보시면, 이해가 가실것이고, 좀더 설명이 필요한 것만, 체크해 보도록 하겠습니다.위의 항목중 biHeight를 보면, 이는 기본적으로 unsigned 값입니다. 즉, 양수값입니다. 만약 이 값이 음수였다면, 실질데이터(pixel data)가 bottom up방식이 아닌 top down 방식으로 저장되어있다는 의미입니다. 이경우는 압축을 지원할수 없습니다. 그렇기 때문에 대부분의 biHeight값은 양수의 값을 갖게 됩니다.

그밖에 biPlanes은 차원(gif의 움직이는 파일처럼)을 나타내는데, 현재는 1차원만을 지원합니다.(움직이는 bmp는 없습니다.) 또,biBitCount는 이미지의 비트수를 나타냅니다. biXPelsPerMeter와 biYPelsPerMeter는 모니터상의 화면과 프린트시의 화면크기가 다를수 있으므로, 이를 설정하기 위한 필드입니다.

BITAMPCOREHEADER구조체를 보면, 대부분은 BITMAPINFOHEADER와 비슷한 기능을 합니다.차이점은 이는 압축을 지원하지 않으며, 지원하는 비트수가

적다는 차이가 있습니다.(하지만, 사실상 비트수에서는 16과 32비트수는 거의 사용하지 않습니다.)

Color Palette

컬러팔레트는 세가지 포맷형식이 있습니다. 이는 bmp화일의 비트수에 의해 구분이 되는데, 1,4,8비트를 갖는 이미지인 경우 RGB값을 나타내기 위해 컬러맵을 사용합니다. 비트수는 이미지 헤더에서 biBitCount나 bcBitCount에서 판단할수 있습니다. 여기서 윈도우 bmp화일인 경우는 RGBQUAD 구조체를 사용하고, OS/2에서는 RGBTRIPLE구조체를 이용합니다.

RGBQUAD구조체

Field	Size	Description
rgbBlue	1	파란색 값
rgbGreen	1	빨간색 값
rgbRed	1	녹색 값
rgbReserved	1	항상 0 값

RGBTRIPLE구조체

Field	Size	Description
rgbtBue	1	파란색 값
rgbtGreen	1	빨간색 값
rgbtRed	1	녹색 값

세 번째 포맷형식은 컬러맵을 사용하지 않습니다. 일단 BI_BITFIELDS를 세팅하기 위한 biCompression이 없는 216,24,32비트 이미지인 경우는 컬러팔렛트를 사용하지 않습니다. 이는 팔레트 자체가 용량이 커버리기 때문입니다. 만약, 16,32비트이고, BI_BITFIELDS인 biCompression값을 갖는다면, RGBQUAD구조체에는 마스크를 사용합니다.(RGBTRIPLE은 압축을 지원하지 않으므로, 생각하지 않으셔도 됩니다.) 마스크에 대해 대강 설명하면, 미술시간에 종이에 무늬를 그리고, 칼로 짤라내고, 그 위에 스프레이 같은걸로 막 뿌리면, 종이 아래에 자른 무늬가 생기죠 ? 이때 짜른 종이가 바로 마스크입니다. 컴퓨터에서 마스크란 일련의 비트열입니다. 예를 들면 열 개의 비트열이 있다고 생각해보죠. 그중에 특정위치의 비트값을 알아내고 싶다면, 마스크와 AND연산 그리고 SHIFT연산을 이용하여 구해낼수 있습니다.

데이터 : 1100110101
마스트 : 0000111000 <- 4,5,6번째 비트를 알고싶을때.
———————— <- AND연산
결과값 : 0000110000
———————— <- 오른쪽 SHIFT연산
결과값 : 0000000110

마스크에 대해서는 대충 이해가 가셨으리라 봅니다. 계속 컬라팔레트에대해 알아보죠.
컬러팔레트에서 전부 세가지의 마스크를 이용합니다. 이는 RED,GREEN,BLUE에 해당하는 마스크입니다. 이는 영인 부분과 영이 아닌부분으로 구분되어 있으며, 데이터 비트에대한 마스킹을 합니다.
32비트 이미지인 경우 아래와 같은 값을 갖고 있습니다.

RED	00 0000000000 0000000000 1111111111
GREEN	00 0000000000 1111111111 0000000000
BLUE	00 1111111111 0000000000 0000000000
	상위비트 하위비트

여기까지, 파일의 헤더와 팔레트에 대해 알아보았습니다. 다음에는 실제데이터와 압축방식에 대해 알아보도록 하겠습니다.

이제 실제 픽셀 데이터의 형태와 이용방법, 그리고 압축에 대한 방식에 대해 알아보도록하겠습니다.

Pixel Data

픽셀데이터는 팔레트나 비트마스크 다음에 위치해 있습니다. 즉, BITMAPINFOHEADER나 BITMAPCOREHEADER구조체 다음에 위치해 있습니다. 그러나, 그 중간에 다른 용도의 바이트가 존재할수 있으므로, 단순한 다음포인터 주소값을 읽기보다는 bfOffBits의 값으로 픽셀 데이터의 위치를

정해야만 합니다.

픽셀은 줄단위로 보통 bottom up방식으로 저장되어 있습니다. 줄단위는 biHeight나 bcHeight필드의 수만큼 있으며, 줄자체의 크기는 biWidth와 biBitCount 혹은 bcWidth와 bcBitCount필드에 의해 정해집니다. 줄에대한 바이트 수는 아래와 같은 식에 의해 정의될수 있습니다.(단, 계산식은 integer수준에서 함)

줄당 바이트수=(width x bitcount +7 / 8) +3 /4

픽셀데이터의 형식은 픽셀당 비트수에 의존됩니다.

1,4 비트 이미지 ; 각 데이터바이트는 두 개나 여덟 개의 필드로 나누어집니다. 그리고 그것은, 각각 컬러팔레트값을 나타내게됩니다.

8비트 이미지 ; 줄단위의 각 픽셀은 1바이트로 나타내며, 그것이 컬러팔레트값을 나타냅니다.

16비트 이미지 ; 각 픽셀은 2바이트의 정수값을 나타내며, 만약 biCompression필드의 값이 BI_RGB인 경우, 각 컬러값은 5비트씩으로 나타내고, 최상위비트는 사용하지 않습니다. 그리고 biCompression필드가 BI_BITMAP인 경우 3개의 4바이트 비트마스크가 각 컬러요소에 사용되게 됩니다.(여기서, 컬러요소의 순서는 red,blue,green임을 주의하세요)

24비트 이미지 ; 각 픽셀은 blue, green, red요소의 값을 추려낼수 있는 세 개의 연속된 바이트열로 나타냅니다. 여기서 대부분의 이미지 파일형식은 픽셀의 순서가 반대가 됨을 주의해야 합니다.

32비트 이미지 ; 각 픽셀은 4바이트의 정수형으로 표현됩니다. 여기서 biCompression필드가 BI_RGB로 되어있으면, 세 개의 낮은순서순으로 된 바이트열이 즉, blue, green,red순서로 8비트값을 나타내게 됩니다. 결론적으로, 이 형식은 24비트 이미지형식과 유사하며, 나머지 바이트는 낭비하게 되는것입니다. 또, biCompression필드가 BI_BITFIELD로 되어있다면, 세 개의 4바이트 비트마스크가 각각의 색깔요소 사용에 필요한 비트를 추려내게됩니다. 이 비트 마스크순서는 red,blue,green이 됩니다.

Compression type

압축에는 여러 가지 방식이 있습니다. bmp파일은 그중에 가장 간단한 방식을 사용합니다. 그 방식이 Run-Length 압축방식이라하며, 기본적인 내용은 다음과 같습니다.

만약, 111111111122222222333333 이만큼의 데이터를 저장하기 위해서는 전부 24의 공간이 필요합니다.

하지만, 위에같은 경우는 두 개의 쌍으로 다음과 같이 나타내면 어떻게 될까요.

A1 82 63 (16진수로 A는 10을 의미합니다.)

위처럼 쓴다면, 단지 6개의 공간만이 필요하게되어 거의 25%정도의 공간이면 충분하게 됩니다. 이와같이 길이에 대한 데이터를 표시하는 방식이 Run-length압축방식입니다.

bmp파일에서 사용하는 Run-Length Encodging방식에는 두가지가 있습니다.

Type	Compare	Description
RLE8	8비트 이미지	8비트 이미지중 biCompression필드에 BI_RLE8(1)로 되어있는경우
RLE4	4비트 이미지	4비트 이미지중 biCompression필드에 BI_RLE4(2)로 되어있는경우

RLE8(Run-Length Encoding for 8bit per Pixel)

이 압축방식에서는 크게 2개의 바이트쌍으로 나눕니다. 첫 번째 바이트는 데이터의 길이를 나타내고, 두 번째 바이트는 픽셀값을 나타냅니다. 예를 들면,

0916 0016을 확장시키면

0016 0016 0016 0016 0016 0016 0016 0016 0016

이 됩니다.

여기서 두 개의 바이트쌍의 조합으로 몇가지의 특수바이트로 사용할수 있습니다. 일단 생각해 볼수 있듯이 첫 번째 바이트에는 0 값이 올수 없습니다. 그래서 이 값을 escape코드로 사용합니다.

Escape-code Followed by..	Description
00	개행문자를 의미합니다.이미지상에 다음 라인으로 넘어갑니다.
01	이미지의 끝을 의미합니다.
02	현재의 포지션에서 다른 포지션으로 이동합니다. 그리고 그 다음 두 개의 바이트쌍은 다른 포지션으로의 위치를 나타냅니다. 이것은 0인 픽셀이 많은 경우, 효과적으로 나타내기 위한 방식입니다.

예를 들어보면,

0516 1116 0016 0016 0316 0716 0216 0116 0016 0116

이것을 해석해 보면

1116 1116 1116 1116 1116

0716 0716 0716 0116 0116 End of Image

이되게 됩니다. 분석이 어렵지는 않죠 ? bmp파일의 압축에대해서는 여기서 일단락하겠습니다. 사실상 bmp에서 압축은 별 효과가 없습니다. 만약 원본의 이미지 파일이 같은 일련의 데이터가 없는 경우(대부분의 이미지가 이렇습니다.) 압축된 효과가 오히려 더 커져버리는 경우가 생길수 있습니다. 또, 압축이 된다 하더라도, 요즘 나오는 jpg나 png 혹은 gif만큼도 효과가 없기 때문입니다.

RLE4 (Run-Length Encoding for 4 bit per pixel)

이 포맷역시 RLE8과 거의 유사합니다. 차이점이라고 하면, 길이바이트 다음에 오는 바이트에 두 개의 픽셀값이 들어있게됩니다. 여기서, 두 개의 픽셀값은 각각 하나의 길이로 취급이 되며, 두 값이 같던, 틀리던간에, 전체의 길이만큼만 나타내게됩니다. 일단 이해가 안가시면 아래 보기를 보시죠.

0516 4316을 확장시키면

4 3 4 3 4

가 되게 됩니다.

이 경우 길이바이트가 홀수이기 때문에 픽셀값이 남더라도 표시하지 않게 됩니다.압축에 대해서는 여기서 마치겠습니다. 나머지 자세한 내용은 Bitmap관련자료를 참고하시면서, 스스로 분석해 보시기 바랍니다.

[출처] 비트맵 파일의 구조 | 작성자 엘키

'Progamming > API / MFC / C#' 카테고리의 다른 글

ActiveX 만들기 (1) 2008.10.29
ActiveX 컨트롤을 사용하기 (0) 2008.07.17
vs2005, vs2008 에서 typedef void * POINTER_64 PVOID64; 에러.. (1) 2008.07.08
[Embedded CE] Embedded Visual C++ 4 Setting (0) 2008.07.02
Thread 함수들...(WaitForSingleObject..) (0) 2008.06.06
비트맵 파일의 구조 (0) 2008.06.05
WaitforSingleObject 사용법 - 데브피아 개똥이 님 글 (0) 2008.06.05
ActiveX (0) 2008.06.04
닷넷 axMediaPlayer, axiwp, activeMovieControl 라이브러리 추가하기 (0) 2008.05.27
WSACreateEvent (0) 2008.05.19
소켓 입출력 모델 : WSAEventSelect (0) 2008.05.19

Filed under : Progamming/API / MFC / C#
Tag : bitmap 구조, 비트맵 구조