Let GDI do your RLE compression for you

devblogs.microsoft.com/oldnewthing/20090408-00

April 8, 2009

Raymond Chen

This is another trick along the lines of using DIB sections to perform bulk color mapping. GDI will do it for you; you just have to know how to ask. Today's mission is to take a 4bpp bitmap and compress it in `BI_RLE4` format. Now, sure, there are programs out there which already do this conversion, but the lesson is in the journey, not in the destination.

The secret is the `GetDIBits` function. You give this function a bitmap and a bitmap format, and out come the bits in the format you requested; GDI will convert as necessary.

Note: I'm going to take a risk and write "sloppy" code. This is code that is not production quality but is enough to get the point across, so put your nitpicking notepads away.

```
void ConvertToRLE4(LPCTSTR pszSrc, LPCTSTR pszDst)
{
    // error checking elided for expository purposes
    HBITMAP hbm = (HBITMAP)LoadImage(NULL, pszSrc, IMAGE_BITMAP,
                                     0, 0,
                                     LR_LOADFROMFILE |
                                     LR_CREATEDIBSECTION);


    DIBSECTION ds;


    // error checking elided for expository purposes
    GetObject(hbm, sizeof(ds), &ds);


    if (ds.dsBmih.biBitCount != 4) {
        // error – source bitmap is not 4bpp
    }


    struct BITMAPINFO16COLOR {
        BITMAPINFOHEADER bmih;
        RGBQUAD rgrgb[16];
    } bmi16;
    bmi16.bmih = ds.dsBmih;


    bmi16.bmih.biCompression = BI_RLE4;


    BYTE *rgbPixels = new BYTE[bmi16.bmih.biSizeImage];
    HDC hdc = GetDC(NULL);
    if (GetDIBits(hdc, hbm, 0, bmi16.bmih.biHeight, rgbPixels,
                  (LPBITMAPINFO)&bmi16, DIB_RGB_COLORS)
        != bmi16.bmih.biHeight) {
        // error – bitmap not compressible
    }
    ReleaseDC(NULL, hdc);


    BITMAPFILEHEADER bfh = { 0 };
    bfh.bfType = MAKEWORD('B', 'M');
    bfh.bfOffBits = sizeof(BITMAPFILEHEADER) + sizeof(bmi16);
    bfh.bfSize = bfh.bfOffBits + bmi16.bmih.biSizeImage;


    // error checking elided for expository purposes
    HANDLE h = CreateFile(pszDst, GENERIC_WRITE, 0, NULL,
                          CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL);
    DWORD dwWritten;
    WriteFile(h, &bfh, sizeof(bfh), &dwWritten, NULL);
    WriteFile(h, &bmi16, sizeof(bmi16), &dwWritten, NULL);
    WriteFile(h, rgbPixels, bmi16.bmih.biSizeImage, &dwWritten, NULL);
    CloseHandle(h);
    delete[] rgbPixels;
}
```

Let’s start from the top. After loading the bitmap and verifying that it is a 4bpp bitmap, we declare a `BITMAPINFO16COLOR` structure that is just a `BITMAPINFO` structure that holds 16 colors instead of just one. We copy the `BITMAPINFOHEADER` from the `DIBSECTION` to our structure for two reasons:

1. We want to make some changes, and
2. GDI expects the color table to come immediately after the `BITMAPINFOHEADER`.

The second reason is the more important one. We can’t use the `BITMAPINFOHEADER` that is part of the `DIBSECTION` structure because the `DIBSECTION` structure puts `dsBitfields` after the `BITMAPINFOHEADER` instead of a color table.

After copying the `BITMAPINFOHEADER`, we make the key change: Changing the compression type to `BI_RLE4`. We allocate a pixel buffer of a size equal to the uncompressed size of the original bitmap and use `GetDIBits` to fill it with compressed data. Key points:

- Before calling the `GetDIBits` function, we must set the `biSizeImage` member of the `BITMAPINFO` structure to the size of the buffer we passed as `rgbPixels`. In our case, this happened implicitly since we allocated `rgbPixels` based on the value of `bmi16.bmih.biSizeImage`.
- On successful exit from the `GetDIBits` function, the `GetDIBits` function sets the `biSizeImage` member of the `BITMAPINFO` structure to the number of bytes actually written to the buffer.
- On successful exit from the `GetDIBits` function, the `GetDIBits` function fills the color table if you’re using a bitmap format that requires a color table. It’s important that you allocate enough memory to hold the color table; if you forget, then you have a buffer overflow bug.

Since the `GetDIBits` function returns the number of scan lines successfully read, if the value is different from the value we requested, then something went wrong. In our case, the most likely reason is that the bitmap is not compressible according to the `BI_RLE4` algorithm.

Now that we have the compressed bits, it’s just grunt work to turn it into a `BMP` file. The `BMP` file format specifies that the file begins with a `BITMAPFILEHEADER`, followed by the `BITMAPINFOHEADER`, the color table, and the pixels. So we write them out in that order.

Easy peasy.

Raymond Chen

Follow