Separating the metadata from the DIB pixels: Changing the raster operation

devblogs.microsoft.com/oldnewthing/20090716-00

July 16, 2009

Raymond Chen

For a few days now, we've used the `SetDIBitsToDevice` function in conjunction with a precomputed `BITMAPINFO` to draw a DIB with an alternate color table without modifying the `HBITMAP`.

The `SetDIBitsToDevice` function operates like a `BitBlt` with raster operation `SRCCOPY`. If you want another raster operation, you can use `StretchDIBits`, which has a final raster operation parameter. Despite its name, you don't have to stretch with `StretchDIBits`. Just pass a source and destination of equal size, and you've performed a NOP stretch, but you get the benefit of the raster operation.

```
void
PaintContent(HWND hwnd, PAINTSTRUCT *pps)
{
 if (g_pvBits) {
    StretchDIBits(pps->hdc, 0, 0,
                      g_bmiGray.bmiHeader.biWidth,
                      g_bmiGray.bmiHeader.biHeight, 0, 0,
                      g_bmiGray.bmiHeader.biWidth,
                      g_bmiGray.bmiHeader.biHeight,
                      g_pvBits,
                     (BITMAPINFO*)&g_bmiGray, DIB_RGB_COLORS,
                      NOTSRCCOPY);
 }
}
```

I changed the call from `SetDIBitsToDevice` to `StretchDIBits`, setting the source and destination rectangles to the same size (so no actual stretching occurs), and specifying a raster operation of `NOTSRCCOPY` so the result on screen is a negative grayscale.

Some people may object to performing a stretch operation and requesting no stretching, but that's perfectly fine. At least in this case, GDI is not stupid. If you ask it to perform a stretch operation but pass parameters that don't do any stretching, it will optimize this to a non-stretching operation. You don't need to hand-optimize it. Instead of writing

```
if (cxSrc == cxDst && cySrc == cyDst) {
 BitBlt(hdc, xDst, yDst, cxDst, cyDst,
        hdcSrc, xSrc, ySrc, dwRop);
} else {
 StretchBlt(hdc, xDst, yDst, cxDst, cyDst,
            hdcSrc, xSrc, ySrc, cxSrc, cySrc, dwRop);
}
```

… just go straight to the `StretchBlt`:

```
StretchBlt(hdc, xDst, yDst, cxDst, cyDst,
           hdcSrc, xSrc, ySrc, cxSrc, cySrc, dwRop);
```

The `StretchBlt` function will convert the operation to a `BitBlt` if `cxSrc == cxDst` and `cySrc == cyDst`. You don't have to hand-optimize it. The GDI folks hand-optimized it for you.

(In fact, for a long time, the `SetDIBitsToDevice` function simply called `StretchDIBits`, saying that the input and output rectangles were the same size, and `StretchDIBits` detected the absence of stretching and used an optimized code path. Consequently, "optimizating" the code by calling `SetDIBitsToDevice` was actually a pessimization.)

Back to `StretchDIBits`. So far, we've been drawing the entire bitmap at the upper left corner of the destination device context. The last remaining feature of `BitBlt` is the ability to draw a rectangular chunk of a source bitmap at a destination location, so let's do that. We'll draw the bottom right corner of the bitmap in the bottom right corner of the window, with negative colors, and *just to show we can*, we'll also stretch it.

```
void
PaintContent(HWND hwnd, PAINTSTRUCT *pps)
{
 if (g_pvBits) {
  RECT rc;
  GetClientRect(hwnd, &rc);
  int cxChunk = g_bmiGray.bmiHeader.biWidth / 2;
  int cyChunk = g_bmiGray.bmiHeader.biHeight / 2;
  StretchDIBits(pps->hdc, rc.right - cxChunk * 2,
                rc.bottom - cxChunk * 2,
                cxChunk * 2, cyChunk * 2,
                g_bmiGray.bmiHeader.biWidth - cxChunk,
                g_bmiGray.bmiHeader.biHeight - cyChunk,
                cxChunk, cyChunk,
                g_pvBits, (BITMAPINFO*)&g_bmiGray,
                DIB_RGB_COLORS, NOTSRCCOPY);
 }
}
```

So far, we’ve been operating on DIB pixels that are held inside a DIB section. But there’s no requirement that the bits passed to `StretchDIBits` come from an actual DIB section. We’ll look at the total disembodiment of DIBs next time, as well as looking at some unexpected consequences of all our game-playing.