The fun and profit of manipulating the DIB color table can be done without having to modify it

devblogs.microsoft.com/oldnewthing/20090714-00

July 14, 2009

Raymond Chen

If I were Michael Kaplan, I'd have a more clever title like *I'm not touching you!* or *Look but don't touch* or maybe *Looking at a DIB through* `BITMAPINFO` *-colored glasses*.

We saw some time ago that you can manipulate the DIB color table to perform wholesale color remapping. But in fact you can do this even without modifying the DIB color table, which is a handy trick if you want to do color remapping but you don't want to change the bitmap itself. For example, the bitmap is not one that is under your control (so you shouldn't be modifying it), or the bitmap might be in use on multiple threads (so modifying it will result in race conditions).

Let's demonstrate this technique by converting the "Gone Fishing" bitmap to grayscale, but doing so without actually modifying the bitmap. As always, we start with our scratch program and make the following changes:

```
HBITMAP g_hbm;

BOOL
OnCreate(HWND hwnd, LPCREATESTRUCT lpcs)
{
 // change path as appropriate
 g_hbm = (HBITMAP)LoadImage(g_hinst,
                      TEXT("C:\\Windows\\Gone Fishing.bmp"),
                      IMAGE_BITMAP, 0, 0,
                      LR_CREATEDIBSECTION | LR_LOADFROMFILE);
 return TRUE;
}
void
OnDestroy(HWND hwnd)
{
 if (g_hbm) DeleteObject(g_hbm);
 PostQuitMessage(0);
}
void
PaintContent(HWND hwnd, PAINTSTRUCT *pps)
{
 if (g_hbm) {
  BITMAP bm;
  if (GetObject(g_hbm, sizeof(bm), &bm) == sizeof(bm) &&
                bm.bmBits != NULL &&
                bm.bmPlanes * bm.bmBitsPixel <= 8) {
   struct BITMAPINFO256 {
    BITMAPINFOHEADER bmiHeader;
    RGBQUAD bmiColors[256];
   } bmiGray;
   ZeroMemory(&bmiGray, sizeof(bmiGray));
   HDC hdc = CreateCompatibleDC(NULL);
   if (hdc) {
    HBITMAP hbmPrev = SelectBitmap(hdc, g_hbm);
    UINT cColors = GetDIBColorTable(hdc, 0, 256, bmiGray.bmiColors);
    for (UINT iColor = 0; iColor < cColors; iColor++) {
     BYTE b = (BYTE)((30 * bmiGray.bmiColors[iColor].rgbRed +
                      59 * bmiGray.bmiColors[iColor].rgbGreen +
                      11 * bmiGray.bmiColors[iColor].rgbBlue) / 100);
     bmiGray.bmiColors[iColor].rgbRed   = b;
     bmiGray.bmiColors[iColor].rgbGreen = b;
     bmiGray.bmiColors[iColor].rgbBlue  = b;
    }
    bmiGray.bmiHeader.biSize        = sizeof(bmiGray.bmiHeader);
    bmiGray.bmiHeader.biWidth       = bm.bmWidth;
    bmiGray.bmiHeader.biHeight      = bm.bmHeight;
    bmiGray.bmiHeader.biPlanes      = bm.bmPlanes;
    bmiGray.bmiHeader.biBitCount    = bm.bmBitsPixel;
    bmiGray.bmiHeader.biCompression = BI_RGB;
    bmiGray.bmiHeader.biClrUsed     = cColors;
    SetDIBitsToDevice(pps->hdc, 0, 0,
                      bmiGray.bmiHeader.biWidth,
                      bmiGray.bmiHeader.biHeight, 0, 0,
                      0, bmiGray.bmiHeader.biHeight,
                      bm.bmBits,
                     (BITMAPINFO*)&bmiGray, DIB_RGB_COLORS);
    BitBlt(pps->hdc, bm.bmWidth, 0, bm.bmWidth, bm.bmHeight,
           hdc, 0, 0, SRCCOPY);
    SelectBitmap(hdc, hbmPrev);
    DeleteDC(hdc);
   }
  }
 }
}
```

Things start off innocently enough, loading the bitmap into a DIB section for use during painting.

We do our work at paint time. First, we confirm that we indeed have a DIB section and that it is 8bpp or lower, because bitmaps at higher than 8bpp do not use color tables.

We then select the bitmap into a DC so we can call `GetDIBColorTable` to get its current color table. (This is the only step that requires the bitmap to be selected into a device context.) We then edit the color table to convert each color to its grayscale equivalent.

Finally, we fill in the `BITMAPINFO` structure with the description of the bitmap bits, and then we call `SetDIBitsToDevice` to send the pixels to the destination DC.

Just for good measure, we also `BitBlt` the original unmodified bitmap, to prove that the original bitmap is intact and unchanged.

This mini-program is really just a stepping stone to other things you can do with this technique of separating the metadata (the `BITMAPINFO`) from the pixels. We'll continue our investigations tomorrow.

(Before you all run out and use this technique everywhere you can imagine, wait for the remarks in Friday's installment.)