Manipulating the DIB color table for fun and profit

devblogs.microsoft.com/oldnewthing/20061115-01

November 15, 2006

Raymond Chen

If you create a DIB section at 8bpp or lower, then it will come with a color table. Pixels in the bitmap are represented not by their red/blue/green component values, but are instead indices into the color table. For example, a 4bpp DIB section can have up to sixteen colors in its color table.

Although displays that use 8bpp or lower are considered woefully outdated nowadays, bitmaps in that format are actually quite useful precisely due to the fact that you can manipulate colors in the bitmap, not by manipulating the bits themselves, but instead by manipulating the color table.

Let's demonstrate this by taking the "Gone Fishing" bitmap and converting it to grayscale. Start with our scratch program and make these 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);
 if (g_hbm) {
  HDC hdc = CreateCompatibleDC(NULL);
  if (hdc) {
   HBITMAP hbmPrev = SelectBitmap(hdc, g_hbm);
   RGBQUAD rgbColors[256];
   UINT cColors = GetDIBColorTable(hdc, 0, 256, rgbColors);
   for (UINT iColor = 0; iColor < cColors; iColor++) {
    BYTE b = (BYTE)((30 * rgbColors[iColor].rgbRed +
                     59 * rgbColors[iColor].rgbGreen +
                     11 * rgbColors[iColor].rgbBlue) / 100);
    rgbColors[iColor].rgbRed = b;
    rgbColors[iColor].rgbGreen = b;
    rgbColors[iColor].rgbBlue = b;
   }
   SetDIBColorTable(hdc, 0, cColors, rgbColors);
   SelectBitmap(hdc, hbmPrev);
   DeleteDC(hdc);
  }
 }
 return TRUE;
}
void
OnDestroy(HWND hwnd)
{
 if (g_hbm) DeleteObject(g_hbm);
 PostQuitMessage(0);
}
void
PaintContent(HWND hwnd, PAINTSTRUCT *pps)
{
 if (g_hbm) {
  HDC hdc = CreateCompatibleDC(NULL);
  if (hdc) {
   HBITMAP hbmPrev = SelectBitmap(hdc, g_hbm);
   BITMAP bm;
   if (GetObject(g_hbm, sizeof(bm), &bm) == sizeof(bm)) {
    BitBlt(pps->hdc, 0, 0, bm.bmWidth, bm.bmHeight, hdc, 0, 0, SRCCOPY);
   }
   SelectBitmap(hdc, hbmPrev);
   DeleteDC(hdc);
  }
 }
}
```

The `OnDestroy` function merely cleans up, and the `PaintContent` function simply draws the bitmap to the window's client area. All the work really happens in the `OnCreate` function.

First, we load the bitmap as a DIB section by passing the `LR_CREATEDIBSECTION` flag. This opens up the exciting world of DIB sections, but all we care about is the color table. That happens when we call `GetDIBColorTable`. Since color tables are supported only up to 8bpp, a color table of size 256 is big enough to handle the worst case. Once we get the color table, we go through each color in it and convert it to grayscale, then set the new color table into the DIB section. That's all.

Notice that we were able to change the color of every single pixel in the bitmap by modifying just 1KB of data. (Four bytes per `RGBQUAD` times a worst-case of 256 colors.) Even if the bitmap were 1024 × 768, modifying just the color table is enough to change all the colors in the bitmap.

Manipulating the DIB color table is how flags like `LR_LOADMAP3DCOLORS` and `LR_LOADTRANSPARENT` do their work. They don't walk the bitmap updating every single pixel; instead, they just load the color table, look for the colors they are interested in, and change that entry in the color table. This technique of editing the color table is what I was referring to when I suggested you could use DIB sections to avoid the pesky DSna raster operation. And it's faster, too. But it only works on bitmaps that are 8bpp or lower.

You may also have noticed that `LR_LOADTRANSPARENT` doesn't actually load a transparent bitmap. Rather, it loads a bitmap that **appears to be** transparent provided that you draw it against a window whose color is `COLOR_WINDOW`. Why this misleading name? Because at the time this flag was invented, GDI didn't support transparent bitmaps. (And even today, it still doesn't really support then, with the notable exception of functions like `AlphaBlend`.) The best you could do was fake it.

Raymond Chen

Follow