How do I set the alpha channel of a GDI bitmap to 255?

devblogs.microsoft.com/oldnewthing/20210915-00

September 15, 2021

Raymond Chen

Most GDI operations will destroy the alpha channel, because GDI was invented back in the days of monochrome and CGA, where you had one, maybe two bits per pixel, and the paper introducing the concept of an alpha channel wouldn't be published for another year.

Since the alpha channel hadn't been invented yet, the top eight bits of 32bpp pixel formats were unused. Whenever GDI needed to generate a 32bpp pixel, say as the result of text rendering, the results had zero in the top eight bits because, well, the bits had no defined meaning. And if you thought those bits were your alpha channel, well, you just lost your alpha channel.

Okay, so you accept that GDI operations are going to wipe out your alpha channel. How do you get it back, assuming you just want to make your bitmap opaque again?

Fortunately, there's still a GDI operation that doesn't destroy the alpha channel: `BitBlt`. Probably because `BltBlt` is defined in terms of bitwise operations, so the work is done without really thinking about what the bits *mean*. And we can take advantage of that.

```
// hdc is a memory DC with a 32bpp bitmap selected into it.
// This function sets the alpha channel to 255 without
// affecting any of the color channels.

void MakeBitmapOpaque(
    HDC hdc, int x, int y, int cx, int cy)
{
    BITMAPINFO bi = {};
    bi.bmiHeader.biSize        = sizeof(BITMAPINFOHEADER);
    bi.bmiHeader.biWidth       = 1;
    bi.bmiHeader.biHeight      = 1;
    bi.bmiHeader.biPlanes      = 1;
    bi.bmiHeader.biBitCount    = 32;
    bi.bmiHeader.biCompression = BI_RGB;

    RGBQUAD bitmapBits = { 0x00, 0x00, 0x00, 0xFF };

    StretchDIBits(hdc, x, y, cx, cy,
                  0, 0, 1, 1, &bitmapBits, &bi,
                  DIB_RGB_COLORS, SRCPAINT);
}
```

The first step is to create a `BITMAPINFO` structure that describes a 1×1 32bpp bitmap.

We then create that single pixel consisting of an alpha channel (which resides in the reserved bits) of 255, and zero for the color channels.

And then we ask to stretch that 1×1 bitmap over the destination bitmap, using the `SRCPAINT` raster operation.

The secret sauce is that the `SRCPAINT` raster operation means that the source and destination should be OR'd together to form the result. Our source pixel is `{ 0x00, 0x00, 0x00, 0xFF }`, so this means that for each destination pixel, the color channels are left unchanged (OR'd with zero) and the alpha channel is set to 255 (OR'd with 255).

Bingo, this sets the alpha channel of the entire bitmap back to 255.

This lets you use functions like `FillRect` or `DrawText`, and let them destroy the alpha channel of your opaque bitmap. Then come back and repair the alpha channel by setting the bitmap back to opaque without changing any of the color channels.

This trick makes the specified portion of the bitmap opaque. If you want it to have a different alpha channel, you could use two `StretchBlt` operations, one to zero out the alpha channel, and another to OR in the desired value.

```
// hdc is a memory DC with a 32bpp bitmap selected into it.
// This function sets the alpha channel without
// affecting any of the color channels.

void SetBitmapAlphaChannel(
    HDC hdc, int x, int y, int cx, int cy, BYTE alpha)
{
    BITMAPINFO bi = {};
    bi.bmiHeader.biSize        = sizeof(BITMAPINFOHEADER);
    bi.bmiHeader.biWidth       = 1;
    bi.bmiHeader.biHeight      = 1;
    bi.bmiHeader.biPlanes      = 1;
    bi.bmiHeader.biBitCount    = 32;
    bi.bmiHeader.biCompression = BI_RGB;

    if (alpha != 255) {
        RGBQUAD zeroAlpha = { 0xFF, 0xFF, 0xFF, 0x00 };
        StretchDIBits(hdc, x, y, cx, cy,
                      0, 0, 1, 1, &zeroAlpha, &bi,
                      DIB_RGB_COLORS, SRCAND);
    }

    RGBQUAD alphaOnly = { 0x00, 0x00, 0x00, alpha };
    StretchDIBits(hdc, x, y, cx, cy,
                  0, 0, 1, 1, &alphaOnly, &bi,
                  DIB_RGB_COLORS, SRCPAINT);
}
```

The `SRCAND` raster operation performs a logical AND of the source and destination. We set the entire alpha channel to zero by ANDing it with zero, and then we set the value to the desired value by ORing the new value in. (And we can skip the AND step if the desired alpha is 255.)

Raymond Chen

Follow