The case of the image that came out horribly slanted: Taking the pitch into account

devblogs.microsoft.com/oldnewthing/20240905-00

September 5, 2024

Last time, we recognized from the specific corruption in a bitmap that the problem was a mismatched stride. Here's the code that they used to convert from an `ID2D1Bitmap1` to a GDI `HBITMAP`. Their strategy was to use the `ID2D1Bitmap1::Map` method to get access to the D2D bitmap's pixel memory, and then use `SetDIBits` to write those pixels into the GDI `HBITMAP`.

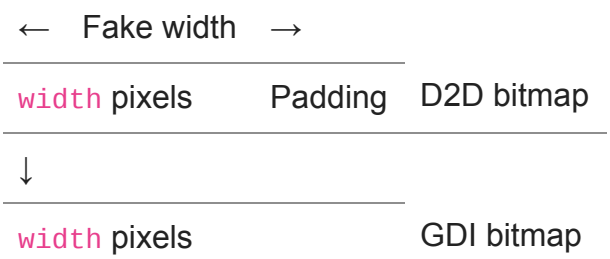
```
HRESULT D2D1BitmapToGdiBitmap(
    ID2D1Bitmap1* d2dBitmap,
    HDC hdc,
    HBITMAP* result)
{
    *result = nullptr;

    // Verify that it's in PARGB format
    assert(d2dBitmap->GetPixelFormat().format ==
            DXGI_FORMAT_B8G8R8A8_UNORM &&
           d2dBitmap->GetPixelFormat().alphaMode ==
            DXGI_ALPHA_MODE_PREMULTIPLIED);

    auto size = d2dBitmap->GetPixelSize();

    // Get a pointer to the pixel memory for reading.
    D2D1_MAPPED_RECT rect;
    RETURN_IF_FAILED(d2dBitmap->Map(D2D1_MAP_OPTIONS_READ, &rect);

    // Make sure we unmap even on early exit.
    auto unmap = wil::ScopeExit([&]() { d2dBitmap->Unmap(); });

    // Create a GDI HBITMAP of matching size.
    wil::unique_hbitmap gdiBitmap(
        CreateCompatibleBitmap(hdc, size.width, size.height));
    RETURN_IF_NULL_ALLOC(gdiBitmap);

    // D2D bitmaps are top-down, but GDI defaults to bottom-up.
    // A negative height indicates that the pixels are top-down.
    BITMAPINFO bmi{};
    bmi.bmiHeader.biSize = sizeof(bmi.bmiHeader);
    bmi.bmiHeader.biPlanes = 1;
    bmi.bmiHeader.biBitCount = 32; // B8G8R8A8_UNORM is 32bpp
    bmi.bmiHeader.biCompression = BI_RGB;
    bmi.bmiHeader.biHeight = -static_cast<LONG>(size.height);
    bmi.bmiHeader.biWidth = size.width;

    RETURN_IF_WIN32_BOOL_FALSE(SetDIBits(hdc, gdiBitmap.get(),
        0, size.height, rect.bits, &bmi; DIB_RGB_COLORS));

    *result = gdiBitmap.release();
    return S_OK;
}
```

Notice that this code never uses the `rect.pitch`. It assumes that the stride of the D2D bitmap is the same as the stride of the GDI bitmap.

GDI's requirement for bitmap stride is that the stride is always a multiple of four, specifically, the smallest multiple of four that can hold all the bytes of a single row of pixels. This code assumed that GDI and D2D agreed on the bitmap stride, and when it was run on a system

whose video driver wanted stricter alignment, you got the stretched-and-slanted distorted result that is a signature of a stride bitmap.

We can fix the problem here by telling GDI that we have an artificially wide source bitmap, so that the extra source pixels occupy the padding bytes in the D2D bitmap.



The output bitmap only has room for `width` pixels, so the pixels represented by the padding get clipped out. And now that we got the strides to match, each row of pixels in the D2D bitmap starts where GDI expects them to be.

```
HRESULT D2D1BitmapToGdiBitmap(
    ID2D1Bitmap1* d2dBitmap,
    HDC hdc,
    HBITMAP* result)
{
    *result = nullptr;

    // Verify that it's in PARGB format
    assert(d2dBitmap->GetPixelFormat().format ==
            DXGI_FORMAT_B8G8R8A8_UNORM &&
           d2dBitmap->GetPixelFormat().alphaMode ==
            DXGI_ALPHA_MODE_PREMULTIPLIED);

    auto size = d2dBitmap->GetPixelSize();

    // Get a pointer to the pixel memory for reading.
    D2D1_MAPPED_RECT rect;
    RETURN_IF_FAILED(d2dBitmap->Map(D2D1_MAP_OPTIONS_READ, &rect);

    // Make sure we unmap even on early exit.
    auto unmap = wil::ScopeExit([&]() { d2dBitmap->Unmap(); });

    // Create a GDI HBITMAP of matching size.
    wil::unique_hbitmap gdiBitmap(
        CreateCompatibleBitmap(hdc, size.width, size.height));
    RETURN_IF_NULL_ALLOC(gdiBitmap);

    // D2D bitmaps are top-down, but GDI defaults to bottom-up.
    // A negative height indicates that the pixels are top-down.
    BITMAPINFO bmi{};
    bmi.bmiHeader.biSize = sizeof(bmi.bmiHeader);
    bmi.bmiHeader.biPlanes = 1;
    bmi.bmiHeader.biBitCount = 32; // B8G8R8A8_UNORM is 32bpp
    bmi.bmiHeader.biCompression = BI_RGB;
    bmi.bmiHeader.biHeight = -static_cast<LONG>(size.height);
    bmi.bmiHeader.biWidth = rect.pitch / 4; // 4 bytes per pixel

    RETURN_IF_WIN32_BOOL_FALSE(SetDIBits(hdc, gdiBitmap.get(),
        0, size.height, rect.bits, &bmi; DIB_RGB_COLORS));

    *result = gdiBitmap.release();
    return S_OK;
}
```

The math turned out easy because a 32bpp bitmap has 4 bytes per pixel, and GDI's stride is a multiple of 4, so reversing the formula for GDI stride comes out easy:

stride = ROUNDUP(*biWidth* × *biBitCount*, 32) / 8
stride = ROUNDUP(*biWidth* × 32, 32) / 8
stride = *biWidth* × 32 / 8

stride = *biWidth* × 4
biWidth = *stride* / 4

More generally, then the math tells us that the GDI bitmap width to produce a specific stride is ⌊*stride* × 8 / *biBitCount*⌋ (This works only if the stride is a multiple of four. If it's not, then you'll never get it to match GDI, since GDI stride is always a multiple of four.)

But wait, what if the stride is negative?

Whaaaaaat?

We'll look at negative stride next time.

Bonus chatter: A member of the Direct2D team suggested that instead of transferring the bits manually, create a GDI-compatible render target, say via `CreateDCRenderTarget` (with `D2D1_RENDER_TARGET_USAGE_GDI_COMPATIBLE` in addition to the existing `DXGI_FORMAT_B8G8R8A8_UNORM` and `DXGI_ALPHA_MODE_PREMULTIPLIED`), render to that render target, and then use `ID2D1GdiInteropRenderTarget::GetDC` to get an `HDC` to `BitBlt` from. Now you don't have to worry about stride. Let D2D deal with it for you.