The case of the image that came out horribly slanted: Diagnosis

devblogs.microsoft.com/oldnewthing/20240904-00

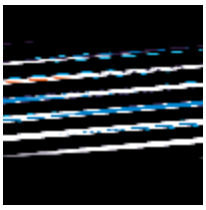
September 4, 2024

A customer wanted to to transfer pixels between a `ID2D1Bitmap1` and a GDI `HBITMAP`. The original D2D bitmap looked like this:

This usually got converted to a GDI bitmap correctly, but once in a while, they'd run into a system where the result looked like this:

Just from the screen shot, I knew what the problem was. Do you see what happened?

Here are some clues.



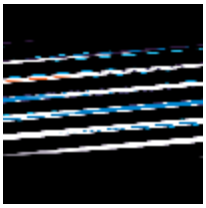
Notice that the result is not total garbage. There is a striped pattern to the result.

Notice also that the result has roughly the same color distribution of pixels as the intended result. The pixels are just being put in the wrong place.

And if you kind of squint, you can see the correct image. It just got stretched really thin and twisted.

Let's unstretch and untwist it:

This twisting and stretching is what happens when you mismatch the bitmap stride.



Imagine two metronomes ticking at slightly different rates.

```
X   X   X   X   X   X   X   X   X   X   X
Y     Y     Y     Y     Y     Y     Y     Y     Y
```

Let's insert a line break in the timeline each time the fast X metronome ticks, so this lets us see what the Y metronome looks like from X's point of view:

```
X
↓
Y     |
|Y    |
|  Y  |
|    Y|
|     |
Y     |
|Y    |
|  Y  |
|    Y|
|     |
```

The ticks of the slower metronome get gradually more and more late until they get so late that they "wrap around" and are back in sync (but a full cycle behind). And then they get more and more late, and wrap around again, and the whole thing just repeats.

A bitmap's *stride* (sometimes known as *pitch*) is the value to add (in bytes) to get from the start of one row of pixels to the start of the next row of pixels. If the bitmap was produced at one stride but consumed at a different stride, then you get the mismatched metronome effect, where the producer is starting each row every *X* bytes, but the consumer expects each row to begin every *Y* bytes. If the producer's stride is larger, then from the consumer's standpoint, the bytes of each row arrive later and later. (And if the producer's stride is smaller, then the consumer sees each row starting more and more prematurely.) The result is that each row of pixels is offset slightly, and as more and more rows are processed, the offset accumulates, leading to the slanting effect.

The choice of stride can be hardware-dependent. Some video drivers prefer that each row of pixels begin on an exact multiple of, say, 64 bytes. If your bitmap uses a pixel format that is 4 bytes per pixel, then a bitmap of width *n* will have 4*n* bytes of pixel data for each row, and if that's not a multiple of 64 (if *n* is not a multiple of 16), then there will be padding bytes at the end of each row.

This explains why the problem occurred only on some systems: You need a graphics card which imposes stronger alignment than that required by GDI.

Next time, we'll look at the code for doing the conversion and see where the stride needs to be taken into account.