It’s easy to get data into the GPU, but harder to get it out

devblogs.microsoft.com/oldnewthing/20211223-00

December 23, 2021

Raymond Chen

Back in the old days, computer graphics were handled by the CPU by directly manipulating the frame buffer,[1] and the graphics card’s job was simply to put the pixels in the frame buffer onto the screen. As computer graphics technology has progressed, more and more work has been offloaded onto the GPU. Nowadays, the GPU draws triangles without CPU assistance, it has a z-buffer, it runs pixel shaders, it does texture mapping. The CPU shovels raw data into the GPU, and the GPU does the work of combining those pixels to form a final result, which is sent to the screen. All the work offloaded to the GPU means that the CPU is freed up for other things.

In order to accomplish this feat, graphics cards are designed so that the CPU can quickly pump data into the GPU, and the GPU has convenient, fast access to its memory. What was not optimized is getting data back out, so in practice, reading data out of the GPU is relative slow. This trade-off is typically a huge win, because the CPU rarely needs to look at the final result. Not caring about the final result also means that the CPU can use tricks like double-buffering and triple-buffering[2] to keep the graphics pipeline full.

What this means for you is that operations like `GetPixel` and `BitBlt` from the screen will be comparatively slow, because the data needs to be fetched out of the GPU’s frame buffer.

In practice, the frame buffer for what’s on the screen right now may not even exist. Once the frame is presented, the CPU reuses that frame buffer to compose the next frame. This means that reading from the frame buffer is even slower: The CPU first has to *regenerate* the frame buffer, so it can read the desired pixel or pixels from it. This means that a single `GetPixel` call on the screen DC is no longer just reading four bytes of memory from a frame buffer. Instead, it has to run a full render pass over the screen in order to figure out what’s there, and only then can it fetch four bytes from the frame buffer to read the pixel you’re interested in.

So try not to read from the screen DC. It’ll be really slow because the compositor first has to regenerate the screen contents before it can give you the pixel you want. Look for alternatives. For example, instead of reading the pixel in order to do alpha blending, use the `AlphaBlend` function to offload the work to the GPU. Or use a layered window with an alpha channel. Both of these change “reading data from the GPU” (which is slow) to “pushing

data into the GPU" (which is fast). If you absolutely must read from the screen DC, then do it in bulk with `BitBlt`. The cost of reading from the screen is in the render pass. If you read 100 pixels one at a time, that's 100 render passes, each of which produces a single pixel. Much better is to read 100 pixels with a single `BitBlt`. That way, you pay for only one render pass to get all your pixels.

[1] In the *really* old days, computer graphics were handled by the CPU feeding pixels to the output device in real time, known as "racing the beam".

[2] There was a time when double-buffering and triple-buffering had the pejorative names "trouble-buffering" and "cripple-buffering".

Raymond Chen

Follow