Dither me this

devblogs.microsoft.com/oldnewthing/20160922-00

September 22, 2016

Raymond Chen

For some reason, the Internet got all excited about dithering a few months ago, linking primarily to this article about eleven dithering algorithms. (Here's another article about dithering.)

Which reminded me of a dithering story.

The folks at Microsoft Research, when not working on their time machine, occasionally produce code for product teams. Some time ago, the Internet Explorer team received some code from Research that implemented an improved dithering algorithm.

One of my colleagues reviewed the code, and noticed that, among other things, the algorithm rejected the convention wisdom mentioned in Tanner Helland's article:

> Typically, when we finish processing a line of the image, we discard the error value we've been tracking and start over again at an error of "0" with the next line of the image.

Rather than throwing away the accumulated error, the algorithm distributed the error in the final pixel into the next scan row, so that the error from the last pixel in the row wouldn't be lost.[1] It also had a separate algorithm for the last row of the bitmap, pushing all the error to the right since there is no row below.

I have no idea what it did with the error from the final pixel. Maybe it uploaded it to a server where all the lost errors are archived.

My colleague asked why the algorithm was so meticulous about not throwing away accumulated error, at a cost of significant algorithmic complexity. The developer explained, "Because it's correct. We want to minimize the error, and if you throw it away at the end of the row, then you never get to compensate for it in the next row. The total error in the image is higher as a result."

This is a case of getting so engrossed by the mathematical perfection of your algorithm that you lose sight of the original problem.

The goal of dithering is to improve the appearance of an image when it is quantized, that is, when it is displayed with a lower color resolution than the original image. The principle behind Floyd-Steinberg dithering is to keep track of how far your quantized color is from the ideal color and to distribute that error into future nearby pixels, in order to give those pixels a chance to compensate for the quantization error.

But minimizing error is not the goal of dithering. It is merely a technique. The goal of dithering is to make an image that looks good.

My colleague asked, "What if the image is clipped due to layout, or because the user scrolls the page so that the image is only half-visible. Do you go back and re-dither the visible portion of the image?"

The developer admitted that the image is dithered only once. If the image is clipped, the clipping is applied to the dithered version.

"Well, then, when the image gets clipped, what happens to all the error in the clipped-out portion? Oh, wait, I'll tell you: It's thrown away."

As an extreme case of this, consider dragging another window so it partially covers the dithered image. Does the image re-dither to account for the fact that some of its error is now covered by another window? No, the dithered image is just the dithered image, and it gets clipped and occluded just like any other image.

So there's really no point in being so anal-retentive about the error in the last column and last row of the image. First of all, it creates a discontinuity in the algorithm, so that the last column and last row of the image get dithered differently from the other pixels, which may result in a "band" at the edge if there is a large area of uniform color.

Besides, that error can get lost to occlusion or clipping. And don't try to "repair" the image when that happens. Not only is it computationally expensive, but it would result in the somewhat disturbing effect that moving a window over the image results in the image developing a constantly changing "fringe" as the last visible row and column of the image constantly recalculates.

Getting rid of the special cases for the last row and column simplified the algorithm. What's more, in the era where CPUs had limited on-chip cache (measured in single-digit kilobytes), it meant that the code and look-ahead row could fit into cache, leaving all the memory bandwidth available for accessing the actual bitmap.

This was a soft real-time algorithm, because the user is waiting for the image to appear on the screen, so improvements in speed were far more valuable than barely perceptible improvements in visual quality.

[1] I forget how large the error diffusion matrix was exactly. Didn't realize there was going to be a quiz over 20 years later. Let's assume it was the classic Floyd-Steinberg dither, which pushes error only one pixel out. (Although I seem to recall it had a larger diffusion matrix, because there was some odd/even weirdness.)

Raymond Chen

Follow