The curious pattern of pre-emptively rejecting the solution to your problem: Redrawing during resizing

devblogs.microsoft.com/oldnewthing/20191111-00

November 11, 2019

Raymond Chen

A customer had a program that drew a little bit of content against the right-hand edge of its client area. They found that when the user resized the window from small to large, they got bad rendering:

```
X
X
X
X
X
```

The customer added, "If we add the `CS_ HREDRAW` window style, then the problem goes away, but we don't want to use it."

This is another curious case of pre-emptively rejecting the solution to your problem. They found the answer, and then asked for a way to solve the problem without using the answer.

Upon pressing further, we learned that the reason they don't want to use the `CS_ HREDRAW` window style is that it introduces flicker.

You can solve that by using a flicker-free updating model, like double-buffering.

But suppose they don't want to do double-buffering, for whatever reason. Maybe the cost of a full repaint is too high, and they don't want to repaint the parts that didn't change.

What you can do is invalidate only the part that needs to be redrawn.

Let's demonstrate the problem with a simplified version that merely draws a thin border along the right edge. This is basically laziness on my part so I don't have to deal with fonts.

Start with the scratch program and make these changes:

```
void
PaintContent(HWND hwnd, PAINTSTRUCT *pps)
{
    RECT rc;
    GetClientRect(hwnd, &rc);
    Polyline(pps->hdc, (POINT*)&rc, 2);
    rc.left = rc.right - 2;
    FillRect(pps->hdc, &rc, GetSysColorBrush(COLOR_DESKTOP));
}
```

In addition to drawing a two-pixel border along the right edge, the program also draws a diagonal line inside the window. This lets you see whether any unrelated content was repainted.

Observe that as-is, the program exhibits the problem when you resize the window wider.

Observe also that this change fixes the problem:

```
    wc.style = CS_HREDRAW;
```

However, it comes at a cost of redrawing the entire window, as evidenced by the fact that the diagonal line is always updated to match the window size.

Okay, change that line back to `wc.style = 0;` because we are going to try to solve the problem without triggering a full repaint.

What we want to do is be informed when the window is about to be resized, so we can invalidate the last two pixels. Enter the `WM_ WINDOWPOSCHANGING` message. This message is sent as part of the resizing operation. The window size hasn't changed yet, but it's about to!

```
BOOL OnWindowPosChanging(HWND hwnd, WINDOWPOS* lpwpos)
{
    if (!(lpwpos->flags & SWP_NOSIZE)) {
        RECT rc;
        GetClientRect(hwnd, &rc);
        rc.left = rc.right - 2;
        InvalidateRect(hwnd, &rc, TRUE);
    }
    return FORWARD_WM_WINDOWPOSCHANGING(hwnd, lpwpos, DefWindowProc);
}

    HANDLE_MSG(hwnd, WM_WINDOWPOSCHANGING, OnWindowPosChanging);
```

When we are informed that the window position is about to change, we check whether it's due to a change in size. If so, then we get the client rectangle (which will be the old client rectangle) and invalidate the last two columns of pixels, which is exactly the size of the right-aligned content. We then allow the message to be processed normally.

When you run this program, you'll notice two things:

1. The two-pixel border on the right hand side draws correctly. In particular, the previous border erases when the window changes width.
2. The main content of the window does not repaint. You can see this because the diagonal line is drawn from corner to corner of the *old* window size.

Raymond Chen

Follow