Pitfalls of transparent rendering of anti-aliased fonts

devblogs.microsoft.com/oldnewthing/20060614-00

June 14, 2006

Raymond Chen

Windows provides a variety of technologies for rendering monochrome text on color displays, taking advantage of display characteristics to provide smoother results. These include grayscale anti-aliasing as well as the more advanced ClearType technique. Both of these methods read from the background pixels to decide what pixels to draw in the foreground. This means that rendering text requires extra attention.

If you draw text with an opaque background, there is no problem because you are explicitly drawing the background pixels as part of the text-drawing call, so the results are consistent regardless of what the previous background pixels were. But if you draw text with a transparent background, then you must make sure the background pixels that you draw against are the ones you really want.

The most common way people mess this up is by drawing text multiple times. I've seen programs which draw text darker and darker the longer you use it. We'll see here how this can happen and what you need to do to avoid it. Start with the scratch program and make these changes:

```
HFONT g_hfAntialias;
HFONT g_hfClearType;
BOOL
OnCreate(HWND hwnd, LPCREATESTRUCT lpcs)
{
 g_hfAntialias = CreateFont(-20, 0, 0, 0, FW_NORMAL, 0, 0, 0,
    DEFAULT_CHARSET, OUT_DEFAULT_PRECIS, CLIP_DEFAULT_PRECIS,
    ANTIALIASED_QUALITY, DEFAULT_PITCH, TEXT("Tahoma"));
 g_hfClearType = CreateFont(-20, 0, 0, 0, FW_NORMAL, 0, 0, 0,
    DEFAULT_CHARSET, OUT_DEFAULT_PRECIS, CLIP_DEFAULT_PRECIS,
    CLEARTYPE_QUALITY, DEFAULT_PITCH, TEXT("Tahoma"));
 return g_hfAntialias && g_hfClearType;
}
void
OnDestroy(HWND hwnd)
{
 if (g_hfAntialias) DeleteObject(g_hfAntialias);
 if (g_hfClearType) DeleteObject(g_hfClearType);
 PostQuitMessage(0);
}

void MultiPaint(HDC hdc, int x, int y, int n)
{
 LPCTSTR psz = TEXT("The quick brown fox jumps over the lazy dog.");
 int cch = lstrlen(psz);
 for (int i = 0; i < n; i++) {
   TextOut(hdc, x, y, psz, cch);
 }
}
void
PaintContent(HWND hwnd, PAINTSTRUCT *pps)
{
 int iModePrev = SetBkMode(pps->hdc, TRANSPARENT);
 HFONT hfPrev = SelectFont(pps->hdc, g_hfAntialias);
 MultiPaint(pps->hdc, 10,  0, 1);
 MultiPaint(pps->hdc, 10, 20, 2);
 MultiPaint(pps->hdc, 10, 40, 3);
 SelectFont(pps->hdc, g_hfClearType);
 MultiPaint(pps->hdc, 10, 80, 1);
 MultiPaint(pps->hdc, 10,100, 2);
 MultiPaint(pps->hdc, 10,120, 3);
 SelectFont(pps->hdc, hfPrev);
 SetBkMode(pps->hdc, iModePrev);
}
```

This program creates two fonts, one with anti-aliased (grayscale) quality and another with ClearType quality. (I have no idea why people claim that there is no thread-safe way to enable ClearType on an individual basis. We're doing it just fine here.)

Run this program and take a close look at the results. Observe that in each set of three rows of text, the more times we overprint, the darker the text. In particular, notice that overprinting the anti-aliased font makes the result significantly uglier and uglier!

What went wrong?

The first time we drew the text, the background was a solid fill of the window background color. But when the text is drawn over itself, the background it sees is the previous text output. When the algorithm decides that "This pixel should be drawn by making the existing pixel 50% darker," it actually comes out 75% darker since the pixel is darkened twice. And if you draw it three times, the pixel comes out 88% darker.

When you draw text, draw it exactly once, and draw it over the background you ultimately want. This allows the anti-aliasing and ClearType engines to perform their work with accurate information.

The programs that darken the text are falling afoul of the overprinting problem. When the programs decide that some screen content needs to be redrawn (for example, if the focus rectangle needs to be added or removed), they "save time" by refraining from erasing the background and merely drawing the text again (but with/without the focus rectangle). Unfortunately, if you don't erase the background, then the text ends up drawn over a previous copy of itself, resulting in darkening.

The solution is to draw text over the correct background. If you don't know what background is on the screen right now, then you need to erase it in order to set it to a known state. Otherwise, you will be blending text against an unknown quantity, which leads to inconsistent (and ugly) results.

If you keep your eagle eyes open, you can often spot another case where people make the overprinting mistake: When text in a control (say, a check box) becomes darker and darker the more times you tab through it. This happens when programs don't pay close attention to the flags passed in the `DRAWITEMSTRUCT` that is passed to the `WM_DRAWITEM` message. For example, some people simply draw the entire item in response to the `WM_DRAWITEM` message, even though the window manager passed the `ODA_FOCUS` flag, indicating that you should only draw or erase the focus rectangle. This is not a problem if drawing the entire item includes erasing the background, but if you assume that the `WM_ERASEBKGND` message had erased the background, then you will end up overprinting your text in the case where you were asked only to draw the focus rectangle. In that case, the control is not erased; all you have to do is draw the focus rectangle. If you also draw the text, you are doing what the `MultiPaint` function did: Drawing text over text, and the result is text that gets darker each time it repaints.

Raymond Chen

Follow