How do I find the bounding box for a character in a font?

devblogs.microsoft.com/oldnewthing/20100526-00

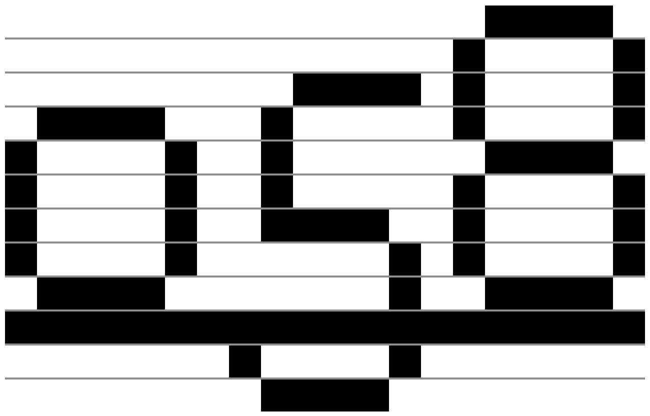
May 26, 2010

Raymond Chen

A customer had the following question:

> I'm looking for a way to get the height of text which consists entirely of digits, for example `"123"`, as these characters do not have any descent or internal leading. I expected functions like `GetTextExtent` to return the character's ascent minus the internal leading, but in fact it returns the ascent plus the descent plus the internal leading. I considered getting the font metrics and taking the `TEXTMETRICS.tmAscent`, but I'm worried that numbers in other languages might have a nonzero descent and internal leading. Is there a function I can call to return the "real" height of the text?

Well, first of all, this question makes an assumption about digits that isn't even true in English. Fonts developed in recent years tend to keep all digits the same height (and often the same width), but fonts designed before the advent of computers (or computer fonts which were inspired by old-timey fonts) will often vary the height, and sometimes even have digits with descenders. Here's an example from the font Georgia:



058

Observe that the number zero is six pixels tall, whereas the number eight is nine pixels tall, and the number five has a two-pixel descender!

Okay, so you're going to have to take the descent into account for all languages, including English. Internal leading is the space above a character to separate it from elements above it. For example, you need some space above a capital T so that the horizontal bar remains readable. Again, the assumption that English doesn't need internal leading is false.

Okay, but what about the original question? Well, when I heard this question, my first thoughts went back to the early days of Win32 when the coolest new GDI feature was paths, and everybody was showing off the fancy text effects you could pull off with the aid of paths. My initial instinct was therefore to use the same technique as those cool demos by combining `BeginPath`, `TextOut`, and `EndPath`. Once I had a path, I could get its dimensions by using `PathToRegion` and `GetRgnBox`.

Fortunately, it turns out that there's an easier way. The `GetGlyphOutline` function returns the glyph metrics, which describe the bounding box of the pixels of a character.

```
// Create an identity matrix
static const MAT2 c_mat2Identity = {
    { 0, 1 }, /* eM11 = 1.0 */
    { 0, 0 }, /* eM12 = 0.0 */
    { 0, 0 }, /* eM21 = 0.0 */
    { 0, 1 }, /* eM22 = 1.0 */
 };
GetGlyphOutline(hdc, L'0', GGO_METRICS, &gm, 0, NULL, &c_mat2Identity);
```

The dimensions of the character are returned in the `GLYPHMETRICS` structure, and in particular, you can derive the bounding box from the `gmptGlyphOrigin`, `gmBlackBoxX`, and `gmBlackBoxY` members.

Raymond Chen

Follow