Font character oversampling for rendering from atlas textures

TL,DR: Run oversample.exe on a windows machine to see the benefits of oversampling. It will try to use arial.ttf from the Windows font directory unless you type the name of a .ttf file as a command-line argument.

Benefits of oversampling

Oversampling is a mechanism for improving subpixel rendering of characters.

Improving subpixel has a few benefits:

- With horizontal-oversampling, text can remain sharper while still being sub-pixel positioned for better kerning
- Horizontally-oversampled text significantly reduces aliasing when text animates horizontally
- Vertically-oversampled text significantly reduces aliasing when text animates vertically
- Text oversampled in both directions significantly reduces aliasing when text rotates

What text oversampling is

A common strategy for rendering text is to cache character bitmaps and reuse them. For hinted characters, every instance of a given character is always identical, so this works fine. However, stb_truetype doesn't do hinting.

For anti-aliased characters, you can actually position the characters with subpixel precision, and get different bitmaps based on that positioning if you re-render the vector data.

However, if you simply cache a single version of the bitmap and draw it at different subpixel positions with a GPU, you will get either the exact same result (if you use point-sampling on the texture) or linear filtering. Linear filtering will cause a sub-pixel positioned bitmap to blur further, causing a visible de-sharpening of the character. (And, since the character wasn't hinted, it was already blurrier than a hinted one would be, and now it gets even more blurry.)

You can avoid this by caching multiple variants of a character which were rendered independently from the vector data. For example, you might cache 3 versions of a char, at 0, 1/3, and 2/3rds of a pixel horizontal offset, and always require characters to fall on integer positions vertically.

When creating a texture atlas for use on GPUs, which support bilinear filtering, there is a better approach than caching several independent positions, which is to allow lerping between the versions to allow finer subpixel positioning. You can achieve these by interleaving each of the cached bitmaps, but this turns out to be mathematically equivalent to a simpler operation: oversampling and prefiltering the characters.

So, setting oversampling of 2x2 in stb_truetype is equivalent to caching each character in 4 different variations, 1 for each subpixel position in a 2x2 set.

An advantage of this formulation is that no changes are required to the rendering code; the exact same quad-rendering code works, it just uses different texture coordinates. (Note this does potentially increase texture bandwidth for text rendering since we end up minifying the texture without using mipmapping, but you probably are not going to be fill-bound by your text rendering.)

What about gamma?

Gamma-correction for fonts just doesn't work. This doesn't seem to make much sense -- it's physically correct, it simulates what we'd see if you shrunk a font down really far, right?

But you can play with it in the oversample.exe app. If you turn it on, white-on-black fonts become too thick (i.e. they become too bright), and black-on-white fonts become too thin (i.e. they are insufficiently dark). There is no way to adjust the font's inherent thickness (i.e. by switching to bold) to fix this for both; making the font thicker will make white text worse, and making the font thinner will make black text worse. Obviously you could use different fonts for light and dark cases, but this doesn't seem like a very good way for fonts to work.

Multiple people who have experimented with this independently (me, Fabian Giesen,and Maxim Shemanarev of Anti-Grain Geometry) have all concluded that correct gamma-correction does not produce the best results for fonts. Font rendering just generally looks better without gamma correction (or possibly with some arbitrary power stuck in there, but it's not really correcting for gamma at that point). Maybe this is in part a product of how we're used to fonts being on screens which has changed how we expect them to look (e.g. perhaps hinting oversharpens them and prevents the real-world thinning you'd see in a black-on-white text).

(AGG link on text rendering, including mention of gamma: http://www.antigrain.com/research/font_rasterization/)

Nevertheless, even if you turn on gamma-correction, you will find that oversampling still helps in many cases for small fonts.