December 21, 2018, updated on August 10, 2019

The sad state of font rendering on Linux

Preamble

As it turns out, font rendering is a highly controversial topic. If you don't see anything wrong with Linux font rendering, please disregard this as a shitpost. Thanks.

I spent the first 25 years of my life on Windows and therefore I am biased towards Windows font rendering with ClearType. I also agree with research which suggests that this rendering approach makes reading easier on the eyes. There are also some facts you can't argue with, such as non-linear rendering without anti-aliasing looking awful.

With that out of the way, let's proceed.

Basic concepts

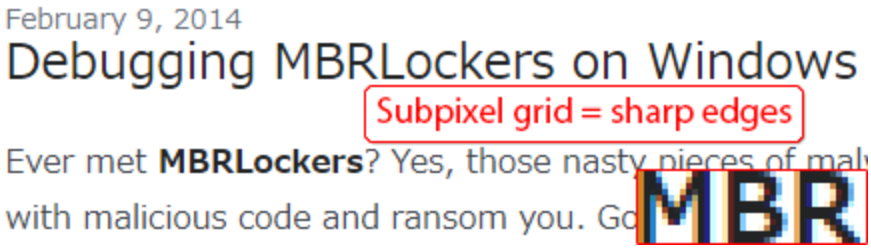
Fonts are vector data that gets rasterized when displayed to the user. Computer displays are low-DPI devices for complex reasons, and such DPI (96) is not enough to display fonts without a myriad of trade-offs. Most notable are sub-pixel rendering, sub-pixel positioning, font hinting and anti-aliasing. You can read up more here and here if you want the full background.

Each operating system approaches font rendering differently. To get Windows-like rendering that I prefer, anti-aliasing, sub-pixel rendering, sub-pixel positioning and font hinting based on byte code embedded into fonts - basically, every step of the

technological progress made in the last 30 years - need to be active.

Windows implementation

Simply put, Windows uses every font rendering improvement technology available *and* goes a step further to use fonts specifically designed to look great when combined with this technology. It is state of the art and then some.



One obvious thing you can see on the zoomed in part of the image is that black and white text is not actually black and white on Windows 7. Since sub-pixel rendering relies on toggling red, green and blue parts of a single pixel, it introduces *colour fringing*. A small percentage of people can see so well, that they actually notice this when reading. They can disable this feature at the cost of less sharp fonts or migrate to Windows 8 or newer.

Windows font rendering keeps changing with every new version of the OS. Windows 7 had the sharpest fonts, while from Windows 8 onwards Microsoft went back to grayscale font smoothing, resulting in blurrier fonts without colour fringing. This is how FreeType code in `/include/freetype/ftdriver.h` documents Windows font rendering evolution over time:

GETINFO	framework	version	feature
3	GDI (Win 3.1),	v1.0	16-bit, first version
TrueImage			
33	GDI (Win NT 3.1),	v1.5	32-bit

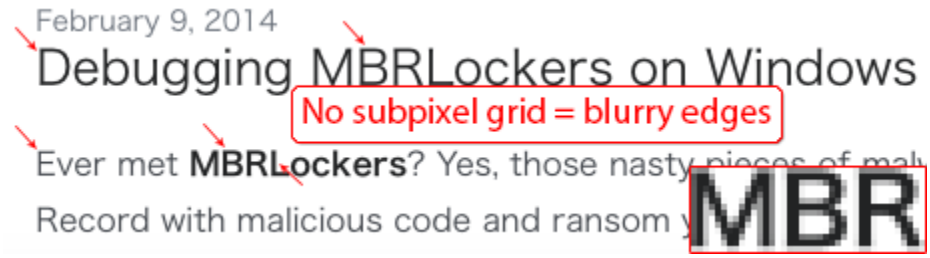
GETINFO	framework	version	feature
HP Laserjet			
34	GDI (Win 95)	v1.6	font smoothing,
		new SCANTYPE opcode	
35	GDI (Win 98, 2000)	v1.7	(UN)SCALED_COMPONENT_OFFSET
		bits in composite glyphs	
36	MGDI (Win CE 2)	v1.6+	classic ClearType
37	GDI (XP and later),	v1.8	ClearType
GDI+ old (before Vista)			
38	GDI+ old (Vista, Win 7),	v1.9	subpixel ClearType,
WPF		Y-direction ClearType,	
		additional error checking	
39	DWrite (before Win 8)	v2.0	subpixel ClearType flags

GETINFO	framework	version	feature
		in GETINFO opcode,	
		bug fixes	
40	GDI+ (after Win 7),	v2.1	Y-direction ClearType flag
DWrite (Win 8)		in GETINFO opcode,	
		Gray ClearType	

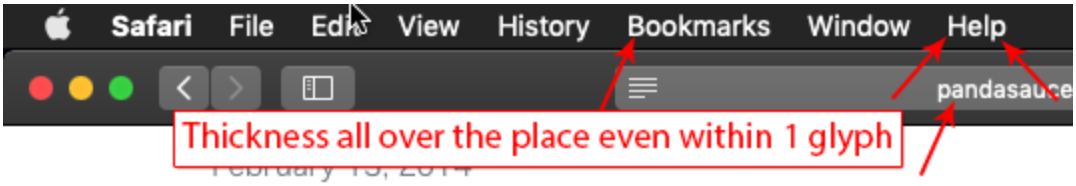
I don't see colour fringing, so my preference is 39/DWrite/v2.0 rendering of Windows 7.

OS X implementation

OS X has the absolute worst font rendering from the readability aspect. It does not use hinting at all and relies only on grayscale anti-aliasing. This combination makes everything appear very bold and blurry:

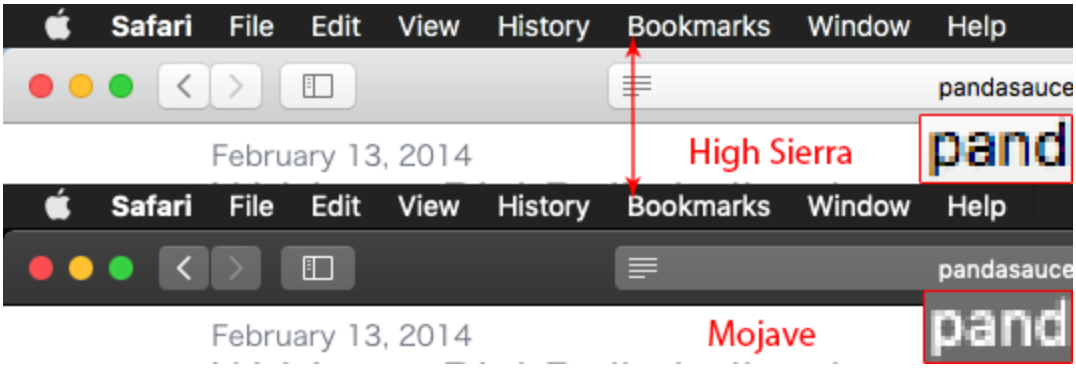


There is not even a hint of any consistency in the rendering either, thickness is all over the place even within a single glyph, with different strokes "sticking" together because of the lack of pixels:



This distracts your subconsciousness and wastes attention on just recognizing glyphs and their edges, resulting in massive eye strain. Don't believe me? Do a controlled reading speed experiment against Windows.

When this blog post got featured on Reddit and HackerNews, a number of people pointed out that prior to Mojave, which was the latest release at the time the post was written, OS X used to have sub-pixel anti-aliasing, text rendering was great, I was wrong and Apple did not deserve the hate. To verify this, I temporarily got my hands on a box running High Sierra and did some comparisons.



As you can see here, indeed, OS X had sub-pixel anti-aliasing in High Sierra, which provided less boldness and bluriness with somewhat better consistency in glyph thickness. However, colour fringing on High Sierra is rather apparent. Rendering is still rather blurry, closer to the FreeType auto-hinter than to what *I* would consider an optimal result.

I have not reverse-engineered Apple's font rendering code, but this would explain what I am seeing:

- Microsoft patented ClearType rendering and Apple took care not to step on their toes
- Sub-pixel rendering techniques patented in ClearType may have a better trade-off

between colour fringing and sharpness than those used in Apple's Quartz

- Apple uses sub-pixel positioning and anti-aliasing, but does not use hinting or uses something akin to FreeType's autohinter
- In case of grayscale anti-aliasing, everything is extra bold because full pixels instead of sub-pixels have to be used for font smoothing, and this distorts the glyph massively as horizontal resolution now is roughly 3 times smaller
- In case of sub-pixel positioning without respecting the sub-pixel grid, as is the case with grayscale anti-aliasing and no hinting on Mojave, stroke thickness consistency goes out of the window, because you have just increased the number of stroke thickness variations that you need to render while also reducing your ability to render them accurately

For decades OS X remained a very ugly baby, until in 2015 Apple just gave us ~~courage~~ HiDPI in form of Retina. This was a bid to make all hinting technology obsolete and put everyone else to shame with their old fashioned sub-pixel hacks.

Here is what Apple didn't account for, or accepted the risk for:

- An overwhelming majority of legacy software has 96 DPI hardcoded into it
- Software maintainers didn't buy into the idea of massive code rewrites and UI tinkering to support DPIs other than 96, because the costs are high and incentives are nil
- In the case of mixed use of HiDPI and non-HiDPI displays, you need to somehow scale the entire UI back-and-forth when moved between displays, which introduces more hacks and corner cases where end result is more suboptimal than before
- Windows never had the font rendering problems that Apple did, so:
    - Majority of screens currently in use are non-HiDPI, meaning you are almost guaranteed to run into the mixed DPI use case
    - Majority of screens currently in use will not be upgraded to HiDPI until they break or corporate hardware upgrade cycles kick in, because majority of the market share is covered by Microsoft's approach, which doesn't need HiDPI to produce easily readable text

    - Majority of screens currently in use will not be upgraded to HiDPI even when they get replaced, because HiDPI costs extra for no obvious benefit to your average consumer or corporation

As a result, all major operating systems have introduced (OS X, Windows) or are working on introducing (Linux) a workaround for cases where HiDPI either is not supported by the application or mixed DPI screens are used.

It boils down to having a primary DPI, used to rasterize fonts, and a per-display DPI, which the operating system uses to resize applications when they enter that display. For applications that don't support HiDPI, their entire UI just gets upscaled.

Resizing happens using the image already rasterized at a different DPI. This results in the entire UI of that application being blurry, both for scaling down and for scaling up.

Sometimes the operating system also needs to make a guess whether the application in question supports HiDPI or not, which can result in further cases of poor rendering where it shouldn't be poor.

Finally, we have the use case of applications spanning multiple displays, either because the user needs it, or just because at that point in time the window is being dragged between screens.

If you embraced the Apple ecosystem lock-in and Retina-only setups, you are probably loving it. Everyone else, likely to include Apple developers supporting these corner cases and users experiencing those cases, not so much.

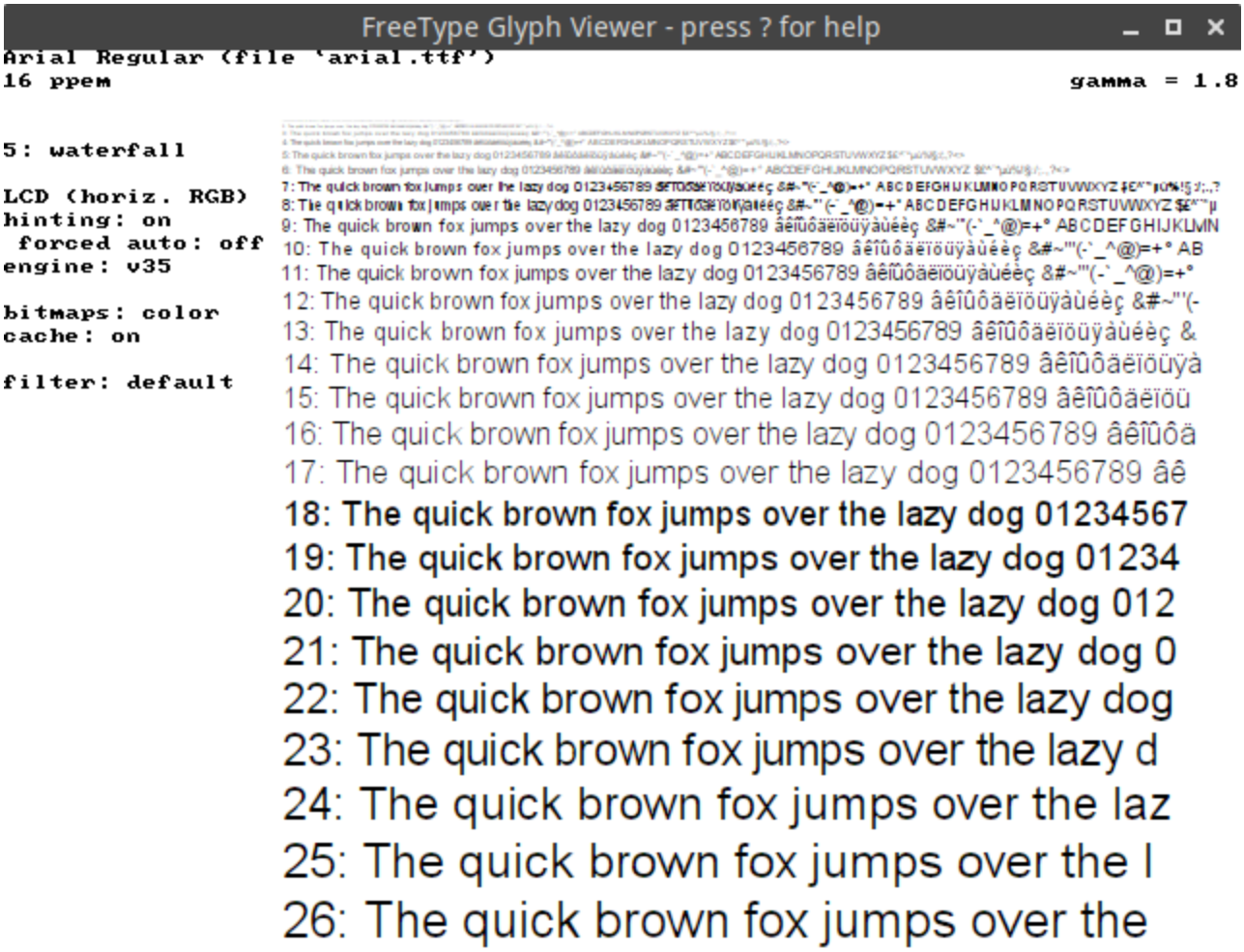
As of 2019, HiDPI experience is still a mixed bag that solves one simple problem at the cost of introducing multiple complex problems. It made rendering code more complex than ever before while providing a user experience that is worse than ever before.

Linux implementation

Linux distros use the **FreeType** library for rendering fonts. FreeType comes with 3

different engines for interpreting TrueType hinting instructions: v35, v38, v40. It also includes an autohinter (a.k.a. autofitter), which ignores the byte code instructions embedded into the fonts and tries to hint fonts automatically.

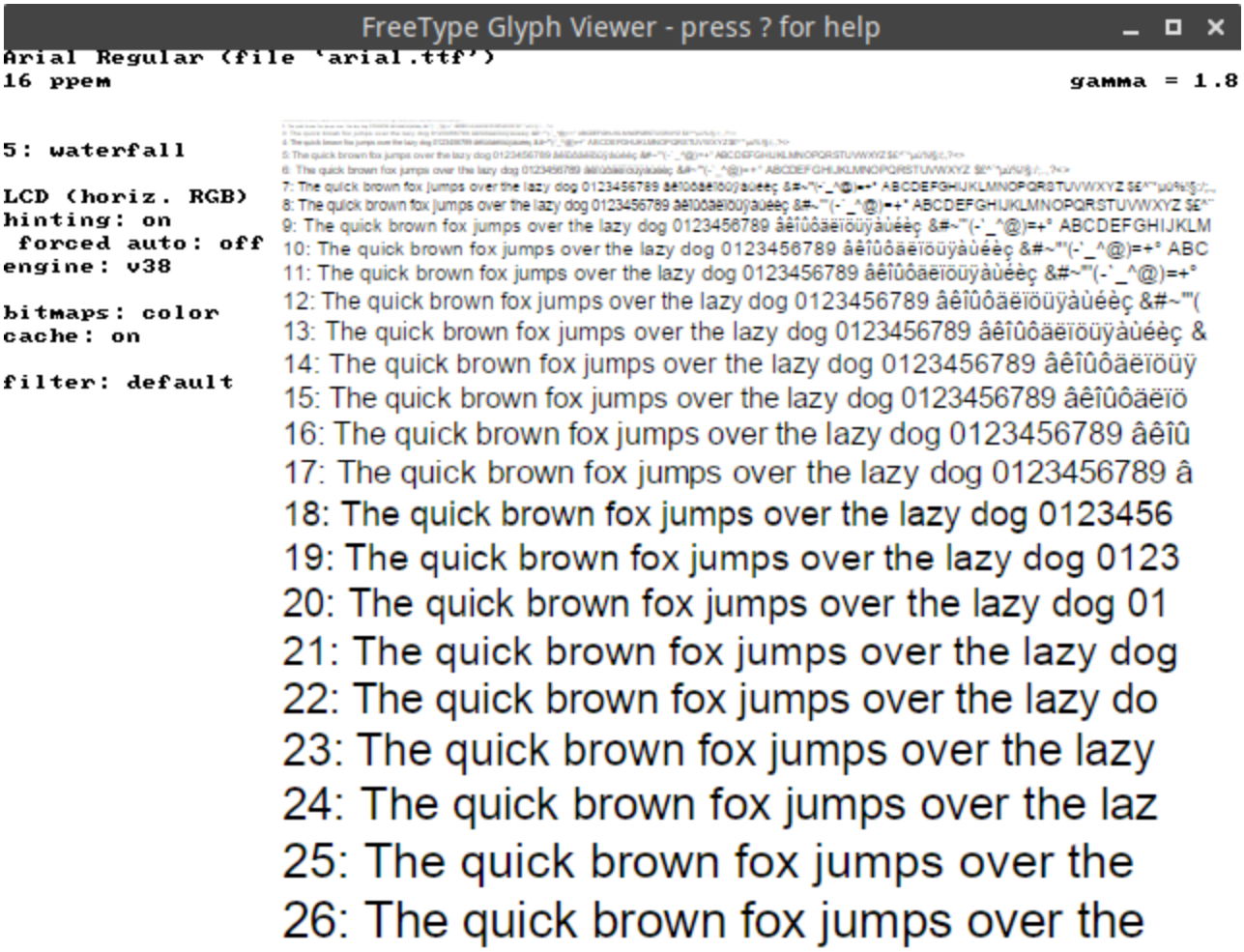
v35 is the legacy engine that is used by default on more risk-averse distros, notably RHEL derivatives. Usually it is compiled without the use of any patent-infringing technologies. It used to be the default in the mainline code base before v40 came around. This is what it looks like:



While it looks fantastic on the first image, once you put different font sizes next to it on the second image, you can immediately see the problem with this engine.

v38 is the “Infinality” engine. Infinality was a set of patches for earlier versions of FreeType that focused on ignoring patent issues and just getting the best possible font rendering in. Some of those were merged into mainline FreeType, but not enabled by default. The author of the patches eventually vanished and I can’t vouch for how much of it is in and how actively it is maintained by the man behind FreeType. This is what it looks like:


FreeType Glyph Viewer - press ? for help
Arial Regular (file 'arial.ttf')
16 ppem
gamma = 1.8
glyph order idx: 0, name: .notdef
1: all glyphs
LCD (horiz. RGB)
hinting: on
forced auto: off
engine: v38
bitmaps: color
cache: on
filter: default

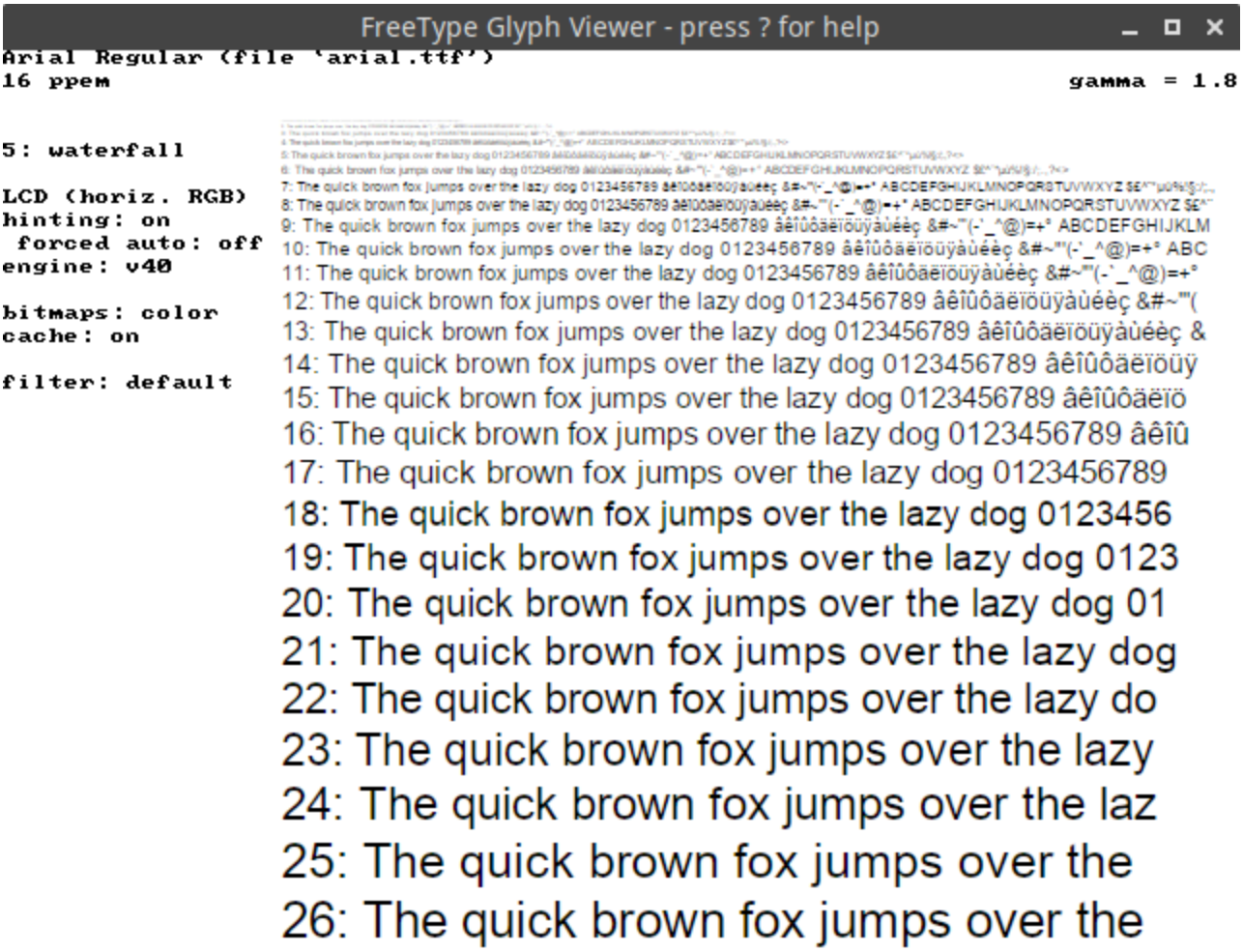


It is somewhat less sharp, but more smooth. The issues on the second image are less apparent, but still very obviously there.

There is widespread criticism stating that font rendering is *visibly* slower on v38 because of all those patches, supported by this post on FreeType website. I couldn't find any profiling with comparisons and for the life of me I can't see the *visibly* slower rendering.

v40 is the new engine, based on stripped down Infinality code, as described in the same blog post. This engine uses Microsoft patented technology where the patents have expired and is usually compiled with patent infringing code activated. This is what it looks like:


FreeType Glyph Viewer - press ? for help
Arial Regular (file 'arial.ttf')
16 ppem
gamma = 1.8
glyph order idx: 0, name: .notdef
1: all glyphs
LCD (horiz. RGB)
hinting: on
forced auto: off
engine: v40
bitmaps: color
cache: on
filter: default

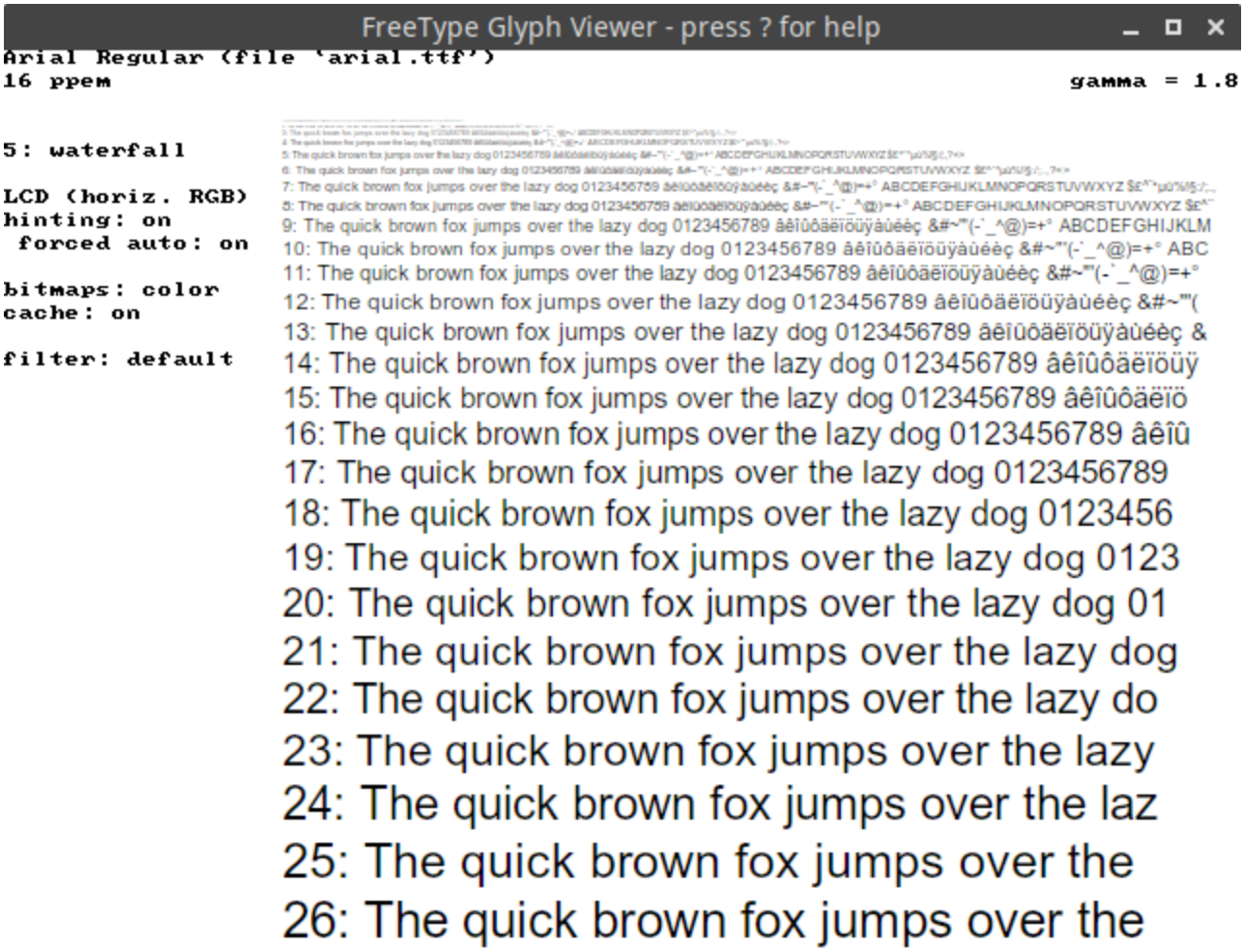


There are some differences if you compare pixel by pixel and obviously the kerning is different.

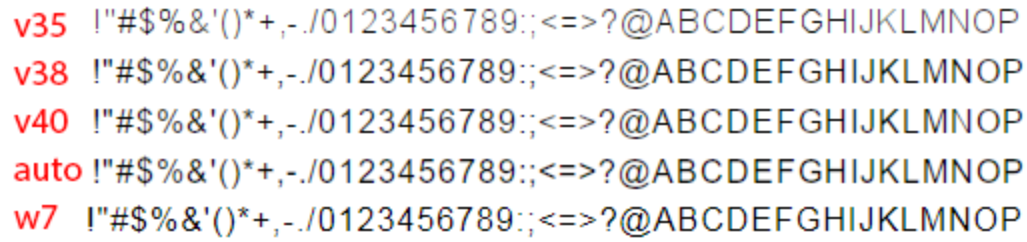
Ubuntu also enables other patented technologies disabled by default in FreeType: ClearType sub-pixel rendering and “GX/AAT table validation”. You can read more about the patents in David Turner’s blog post, but I haven’t noticed any impact of table validation code on the fonts that I see every day.

Finally, the **autohinter**:


FreeType Glyph Viewer - press ? for help
Arial Regular (file 'arial.ttf')
16 ppem
gamma = 1.8
glyph order idx: 0, name: .notdef
1: all glyphs
LCD (horiz. RGB)
hinting: on
forced auto: on
bitmaps: color
cache: on
filter: default



Thickness linearity between font sizes on the second image is fantastic. But compare the overall thickness at standard web font size (16px) to any other option and you will see that this comes at the cost of making everything bold by default:



Note: I am working on an updated version of this comparison that includes FreeType Harmony sub-pixel rendering, Windows 10 and OS X samples.

It is also very easy to tell which option is the most blurry and difficult to read on this image.

With all these side-by-side, my preferred Linux option is v38 engine. In less perfect conditions (e.g. non black-and-white text), v40 seems easier on the eyes.

If anyone wants more screenshots to compare, here are some:

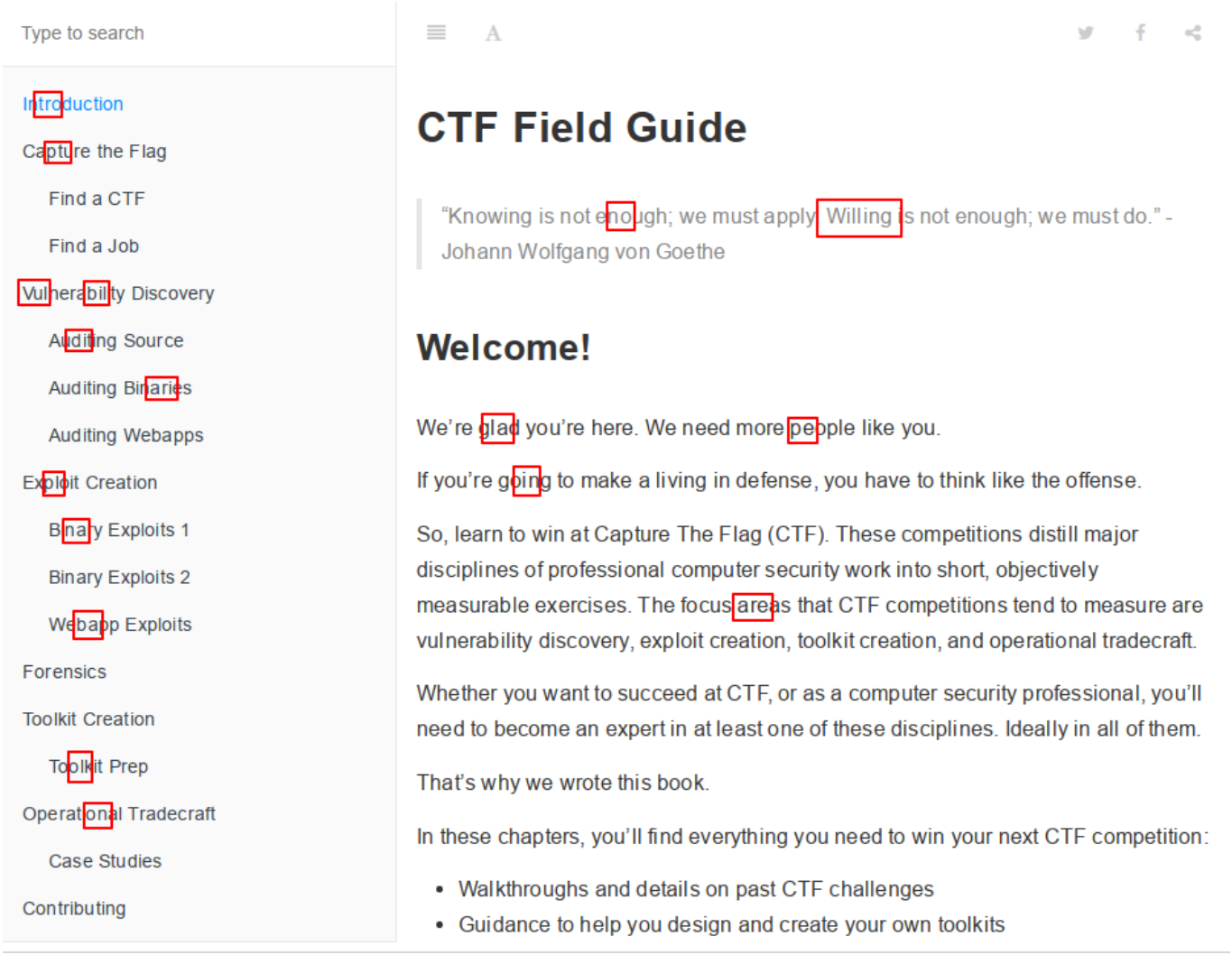
- High Sierra, LCD font smoothing enabled
- High Sierra, LCD font smoothing disabled
- Mojave
- Windows 7
- Linux (v38, sub-pixel, ClearType branch, sub-pixel positioning, Skia)

Funnily enough, in my adventures I learned that some people prefer the fuzzy and blurry rendering of OS X defaults on Windows. They found a way to replace the Windows rendering engine with FreeType for this purpose and the tool is called MacType. Since FreeType is highly configurable, it allows you to disable hinting and mimic OS X approach to rendering fonts. You will see below why this actually makes for a worse user experience without even talking about stability issues that this is bound to introduce.

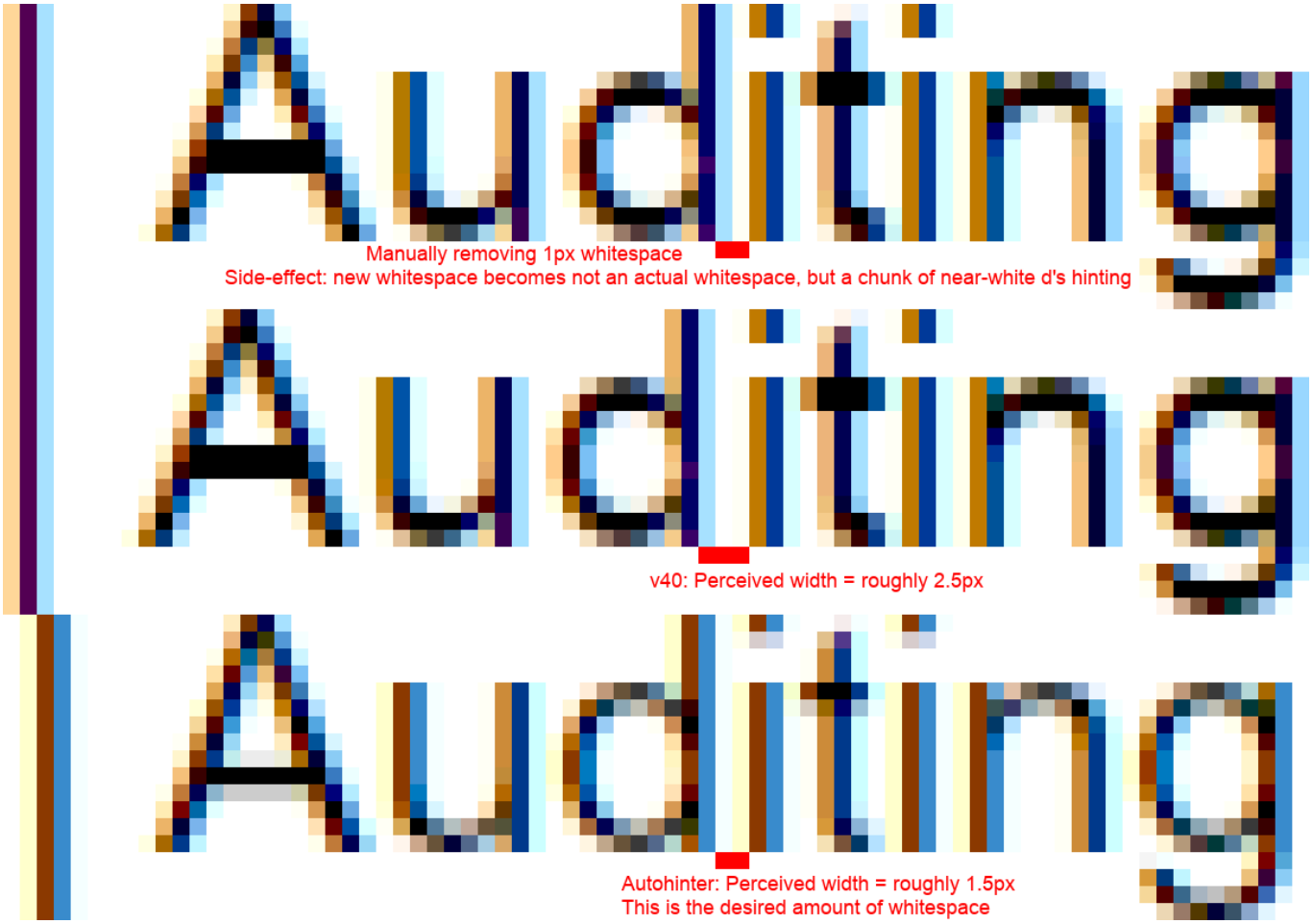
Some users even disable all sub-pixel rendering and anti-aliasing, claiming that it reduces eyestrain and that anti-aliasing damages eyesight. It's kind of like anti-vaxxing (hello from 2019 if you are reading this in the future).

The ugliest part

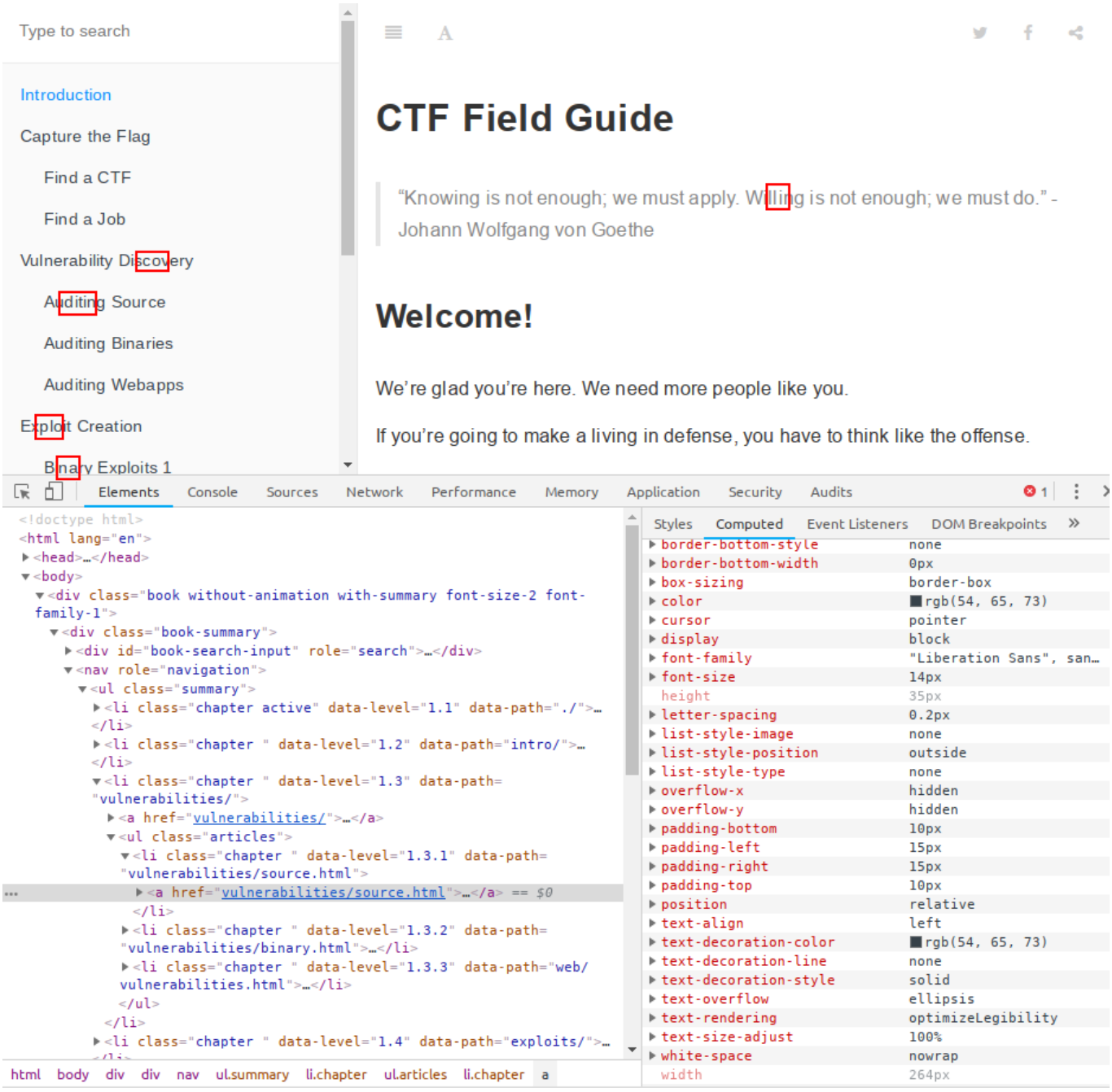
With current state of Linux, it does not matter which engine you pick. They all are broken in the same way:



There is a pixel or less of extra space between kerning pairs and the problem grows exponentially when letter-spacing values are fiddled with:



Before someone jumps in and says “stop using Microsoft fonts”, here is the exact same behaviour exhibited when rendering Liberation Sans, the font most commonly recommended to replace standard web ones:



This is because for a very long time Linux did not support the sub-pixel grid at any level and a lot of assumptions were made. One of such assumptions was that glyph positions have to be whole integers, and those integers were mapped to whole pixels. Of course, with sub-pixel rendering that is no longer the case, so poor rounding decisions result in what you see above.

The solution to this is to position glyphs with sub-pixel precision. For a ridiculously long time, Google's Skia was the only rendering library that supported it.

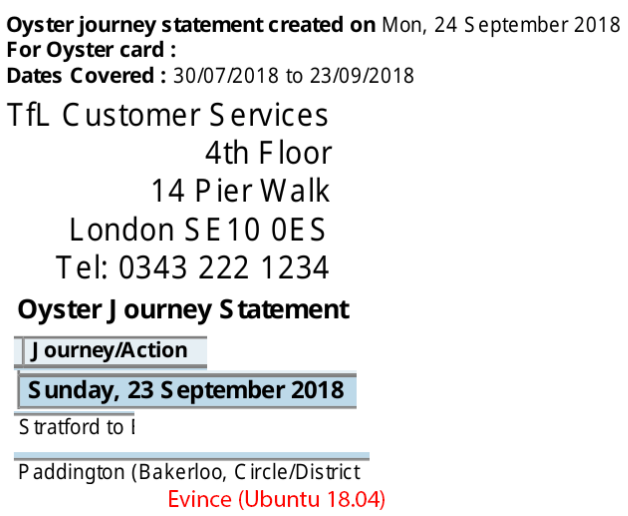
Even now, Chromium still has it disabled by default, but this awesome ~~bug~~ feature lets you enable it by starting Chromium-based applications with `--disable-font-subpixel-positioning` flag. Yes, I know, it says "disable", but it does the opposite as of v76. I've submitted patches to make this the default behaviour, but I have no idea how to get them merged and how long it will take for these patches to trickle down into all apps that use Chromium: Electron, Spotify and Slack come to mind.

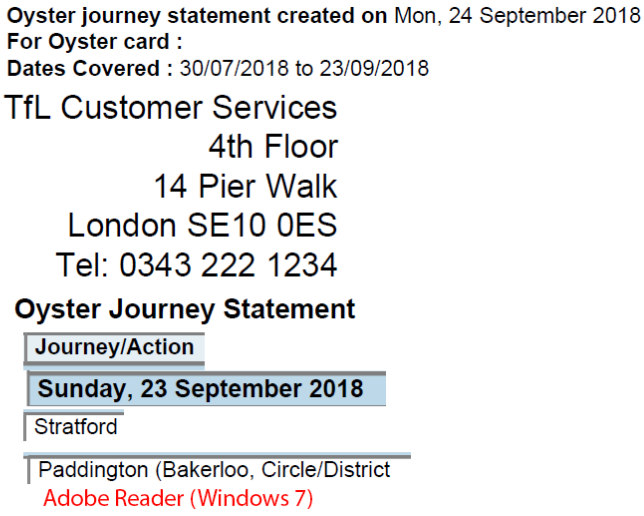
~~The problem with this is that Google's Skia is the only rendering library that supports it, and even there it is disabled by default. I am maintaining a gist with binary patches for Chrome until my proposal to enable it gets accepted upstream, but it doesn't help with the rest of the system.~~

~~More about it can be read in Behdad Esfahbod's whitepaper on linear rendering. He also documented how to fix it back in 2008, and the amount of effort required pretty much meant that we may be stuck with it forever. GTK maintainers attempted to pick this up recently, but it broke some tests that they could not quite figure out and thus Cairo team ignored the patch.~~

By the time I was updating this post in August 2019, Cairo received support for sub-pixel positioning in both xlib and image compositors. This means that GTK will soon have it too, as well as Pango and basically anything that relies on Cairo to render text. I am looking forward to the next Ubuntu LTS and might make a separate post about compiling this into the current Ubuntu LTS.

On top of sub-pixel positioning, we have this kind of ~~bullshit~~ weird behaviour in some programs, and that has nothing to do with sub-pixels:





This has to do with how those particular programs treat system font rendering settings and reconfigure the underlying libraries. It is on my TODO list to look into this too, but not with a particularly high priority ranking.

~~At this point, I give up. Font rendering on Linux is broken, very few people care, and those that do (like myself) are too busy with our day jobs to figure out all the prerequisite knowledge needed to fix it, let alone actually fix it.~~

Extras

Whichever engine you go with on Linux, there will be a few more things to address.

Fontconfig

On top of FreeType there is the `fontconfig` configuration layer that allows to toggle the technologies mentioned above on or off for different fonts, their size ranges and so on. Technically, you could have two completely opposite settings for two different font sizes or weights of the same font. You can read the official docs for the full details.

Microsoft Fonts

When you are browsing the internet, a lot of websites make implicit assumptions about your default fonts. If you are coming from the Windows world, a lot of websites will look

weird to you and page rendering may be downright broken if the fonts are missing from your system, e.g. plain text files opened in Chrome will have overlapping lines. To fix it, you need to get Windows fonts installed.

The traditional way of achieving this is through installing `ttf-mscorefonts-installer` or `msttcorefonts`. The `msttcorefonts` package looks like some Shenzhen basement knockoff that renders poorly and doesn't support Unicode. I suspect that these fonts have gone through multiple iterations in every Windows release and that the versions available through the repositories must be from around Windows 95 days.

Try generating some Zalgo and see the empty boxes for yourself. These are *not* the same fonts included in your Windows installation. If you have these packages installed, ditch them prior to continuing.

The workaround is to grab the fonts from your Windows installation. I Am Not A Lawyer and if you are worried about this going against Microsoft licensing, get legal advice. I just know that it would be illegal for me to provide you with the files, therefore: the fonts are available in `C:\Windows\Fonts`. You need the following files:

arialbd.ttf	**ARIALNB**.TTF	**ariblk**.ttf	**consolai**.ttf	**courbi**.ttf	**georgiai**
arialbi.ttf	**ARIALNI**.TTF	**comicbd**.ttf	**consola**.ttf	**couri**.ttf	**georgia**.
ariali.ttf	**ARIALN**.TTF	**comic**.ttf	**consolaz**.ttf	**cour**.ttf	**georgiaz**
ARIALNBI.TTF	**arial**.ttf	**consolab**.ttf	**courbd**.ttf	**georgiab**.ttf	**impact**.t

Depending on how you access that directory, you may have to copy the files based on font family names:

- Arial
- Comic Sans MS
- Consolas
- Courier New
- Georgia
- Segoe UI

- Times New Roman
- Trebuchet MS
- Impact
- Verdana
- Webdings

The files go in `/usr/share/fonts/MAKE_UP_A_FOLDER_NAME`.

GIMP

If you use GIMP, you may notice weird artifacts in the Text Tool. Create `/etc/gimp/2.0/fonts.conf`:

```
<fontconfig>
  <match target="font">
    <edit name="rgba" mode="assign">
      <const>none</const>
    </edit>
  </match>
</fontconfig>
```

Noto Font

Noto font family is an excellent font that supports all the different languages and symbols. It generally provides a better experience on Linux than the standard Microsoft fonts, without looking too different. It comes in sans, serif and monospace editions, which covers all the OS needs. Install it like this:

```
$ sudo apt install fonts-noto
```

Tweak Tool

- Hinting: `Slight`, which translates to “autohint”. I suggest it because it exhibits the advance widths rounding issue in kerning pairs the least. Personally, I use full hinting with v38.

- Anti-aliasing: `Subpixel`
- Window Titles: `Noto Sans UI Regular 11` or `Noto Sans Display Regular 11` (renamed in newer versions)
- Interface: `Noto Sans UI Regular 10` or `Noto Sans Display Regular 10` (renamed in newer versions)
- Documents: `Noto Serif Regular 11`
- Monospace: `Noto Mono Regular 13`

Application Settings

I find that different applications render best with certain font sizes set. Most likely, this is because it forces the least broken glyph form in absence of subpixel positioning which would give me a non-broken glyph.

Here they are:

- Terminator: `Ubuntu Mono 13.5`
- Sublime Text: `Ubuntu Mono 13.4`, `padding-top 4`, `padding-bottom 4`
- IntelliJ: `Ubuntu Mono 18`, `line height 1.4`
- Chrome, Spotify, Slack, Electron apps: add `--disable-font-subpixel-positioning` to the shortcut. I used to manually patch every binary release of Chrome to enable subpixel positioning, but thanks to this bug in Chromium that turned out to be not necessary.

Conclusion

Whether it was this post gaining traction or just the natural flow of life, but things have improved since the first edition of this post. I am grateful to everyone who contributed code, to everyone who reached out to point out inaccuracies in earlier editions of this post and to everyone who reached out suggesting workarounds for me to try out.

whoami

I'm Georgi (Russian: Георгий). Although I do various software security things for work, I particularly enjoy reverse engineering and breaking native code on Android and embedded systems. Check out more about me.