HarfBuzz at 20!

looking back with 20/20 vision

behdad

WebEnginesHackfest2026

a sequel to *Ten Years of HarfBuzz*

behdad

WebEnginesHackfest2016

Agenda

- **HarfBuzz**
- **Rust**
- **Standardization**
- **Long shot**

HarfBuzz

C++ platform with a C library API
and accompanying cmdline tools

HarfBuzz

/hærfˈbɒːz/

From Persian حرف (***Harf***: letter) and باز (***Buzz***: open).

Transliteration of the Persian calque for *OpenType*.

noun

The Open Source *text shaping* engine.

adjective

Insincerely talkative: glib. A nod to the project's GNOME origins.

What is HarfBuzz?

- **hb-shape**: Text shaper: figures out what shapes to display at what positions, to show legible text in every **language**, **writing system**, and **font**.

and

- **Integrations**: **hb-ft** (FreeType), **hb-glib**, **hb-cairo**, **hb-icu**, …
- **hb-subset:** Font *subsetter* and variable-font *instancer*
- **hb-draw** / **hb-paint:** Callback-based monochrome / color drawing API
- **hb-raster:** Analytical grayscale rasterizer. No hinting.
- **hb-vector:** Vector output into SVG / PDF. Useful for printing color fonts.
- **hb-gpu:** Slug algorithm implementation with four shader language backends.
- …

History

- **Started** in 2005 as cross-Linux-desktop text shaper.
- **Rewrite** started in 2006.
- **Designed** using C++ operator-overloading.
- **Stable** C API/ABI.
- **Replaced** three Open Source shaping engines.
- **Expanded** non-shaping capabilities 2017 onwards.
- **Low bus-factor** 🙈. Khaled Hosny, myself, ~3 domain experts.

Wasn’t HarfBuzz just shaping?

HarfBuzz already had:

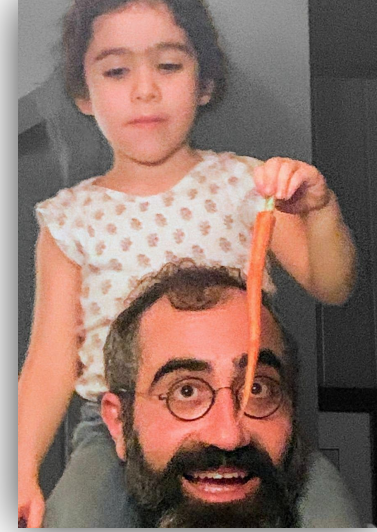
- Efficient binary data reading infrastructure.
- Description of pretty much all font tables.

Adding new functionality to HarfBuzz—as separate modules or libraries—made logical sense.

Extensive configuration system allows for trimming down the library to bare essentials. Heavily used by embedders and wasm solutions.

Full demo at harfbuzz-world.cc.

Vision: the `ffmpeg` of text shaping

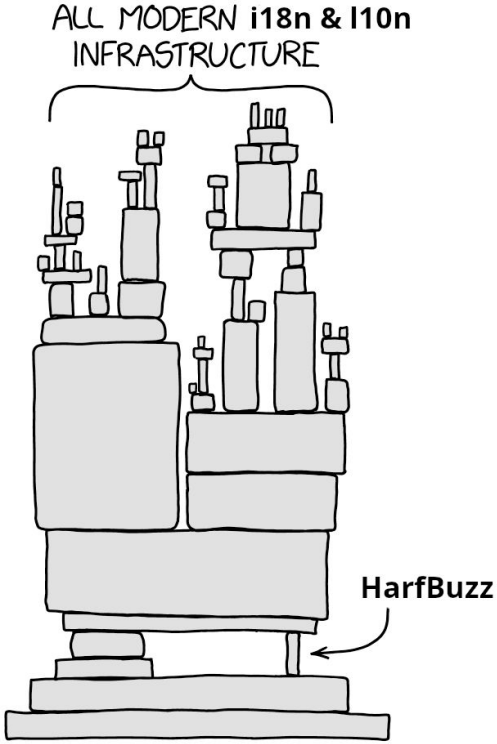


Ubiquitous, pragmatic, understands everything, survives bad input.

Adoption strategy (in hindsight):

- **Market-share** is king
- Embrace **Open Source**
- **Permissive license** (old-MIT)
- **Industry standard** for text shaping
- If we do our job **right**, nobody will notice
- Best-of-class **performance**
- Unmatched **agility**. Zero-day Unicode updates
- **Robustness** trumps **correctness**, which trumps **performance**. Achieve **all**
- Leverage adoption to extend **font-format**

HarfBuzz shapes majority of text on modern screens.



Adoption: Open Source

- **Operating systems:** Android, ChromeOS, Linux distros (Fedora, Ubuntu, …)
- **Browsers:** Chromium, Firefox, WebKitGTK, Chromium derivatives…
- **Desktops:** GNOME, KDE, EFL, Flutter, GTK+ / Qt derivatives…
- **Apps:** LibreOffice, XeTeX, Inkscape, GIMP, VLC, VS Code & other Electron apps, Emacs, many others…
- **Programming languages:** Official OpenJDK Java implementation, .NET API (`HarfBuzzSharp` namespace).

Exceptions:

- ***Wine*** remains the only major Open Source platform to use its own text shaper. Porting to HarfBuzz is theoretically possible, but no one has tried.

Adoption: Proprietary

Many major proprietary platforms use HarfBuzz:

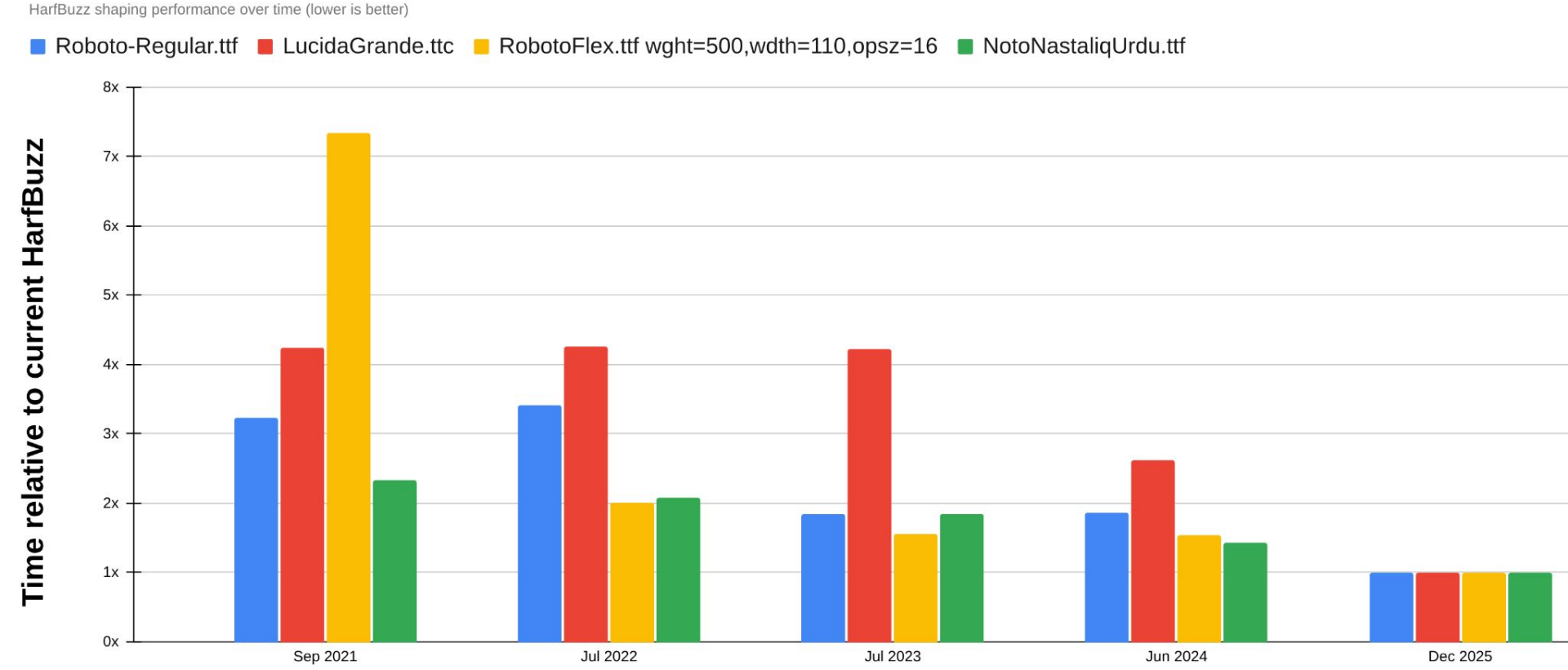
- **Google**’s Chrome, text-ads backend in the data-centers.
- **Adobe** Creative Suite: Photoshop, Illustrator, InDesign.
- **Microsoft** Edge, .NET, VS Code.
- **Amazon** Kindle & FireTV.
- Gaming consoles & engines.
- Smart TVs, car displays.
- Smart watches, VR headsets, other consumer gadgets .
- **Figma**, **Canva**, **Instagram**, etc, etc.
- Almost anywhere that cross-platform text layout is needed.

HarfBuzz: Correctness

- Initially: match Microsoft *Uniscribe* / *DirectWrite*.
- Systematically tested against Uniscribe.
- Diverge only when warranted.
- ***Rule of 2 out of 3***: When Microsoft **DirectWrite** and Apple **CoreText** disagree, **side with one of them**. Or improve on both.
- Over **6,000** shaping tests.
- Heavily **fuzzed**.

Thanks to billions of users, HarfBuzz is now its own reference point.

HarfBuzz: Performance improvements over five years



An ecosystem of bindings and ports

Official supported on **`github.com/harfbuzz`**:

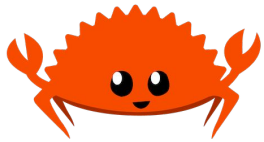
- **HarfBuzz:** C++ platform with a C API.
- **HarfRust:** High-fidelity Rust shaper port.
- **uharfbuzz:** Ergonomic Python bindings.
- **harfbuzzjs:** JavaScript & WebAssembly.

Numerous other bindings and ports too:

- **HarfBuzzSharp:** Part of .NET API.
- **ImageSharp:** Native .NET port.
- **go-text/typesetting:** Go port.
- **luaharfbuzz:** Lua bindings.
- **gobject-introspection:** Clunky auto-generated bindings.

Rust

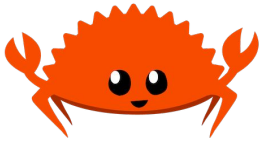
addressing *safe* shaping



Rust text shaping

- **2017** **`harfbuzz_rs`** *Servo*-originated high-level safe API to HarfBuzz
- **2019** **Allsorts** *YesLogic*'s clean-room implementation
- **2020** **RustyBuzz** *RazrFalcon*'s fully safe port of HarfBuzz to Rust
- **2021** **Swash** *Chad Brokaw*'s implementation from scratch
- **2024** **HarfRust** Fork of *RustyBuzz*: one crate to rule them all

Google Oxidize



“Wherein we contemplate moving shaping, rasterization, font compilation, and general font inspection and manipulation from Python & C++ to Rust.”

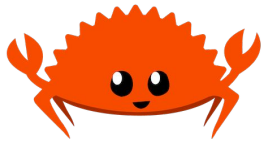
—*Rod Sheeters*

Tech-lead & manager, Google Fonts

January 9, 2022

HarfRust

one crate to rule them all



HarfRust

/hærfˈrɒːst/

From Persian حرف (***Harf***: letter) and راست (***Rust***: true / straight).

Transliteration of the Persian calque for *TrueType*, the precursor to OpenType.

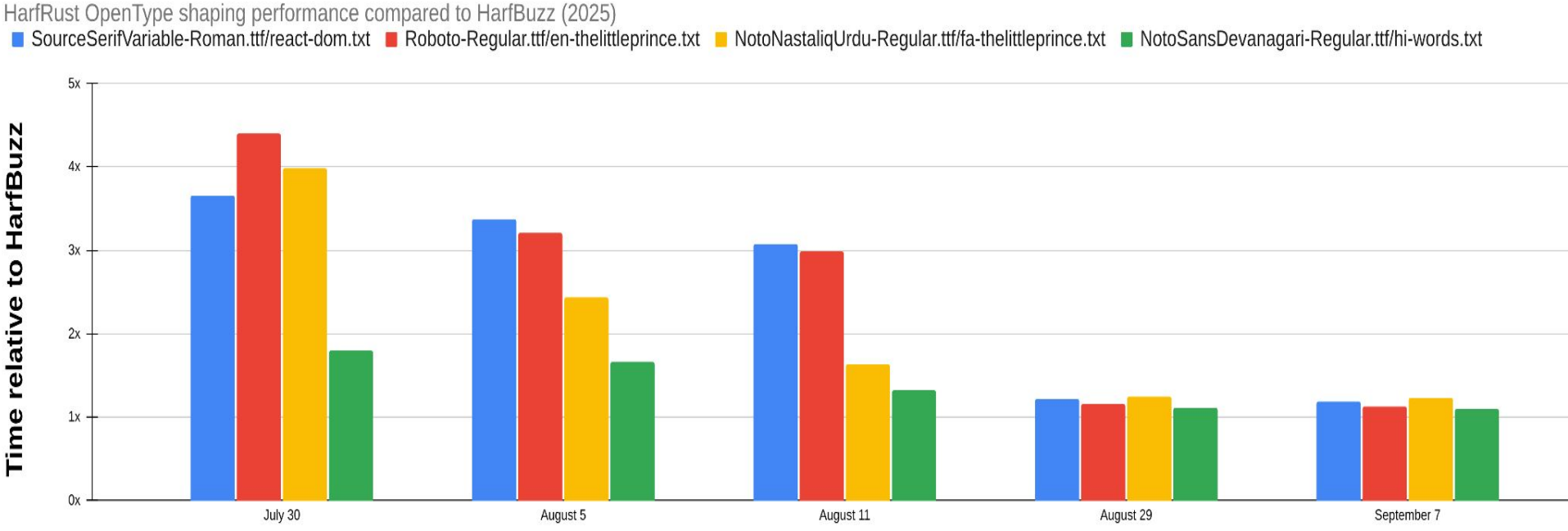
noun

The same trusty Open Source *text shaping* engine, in fully safe Rust.

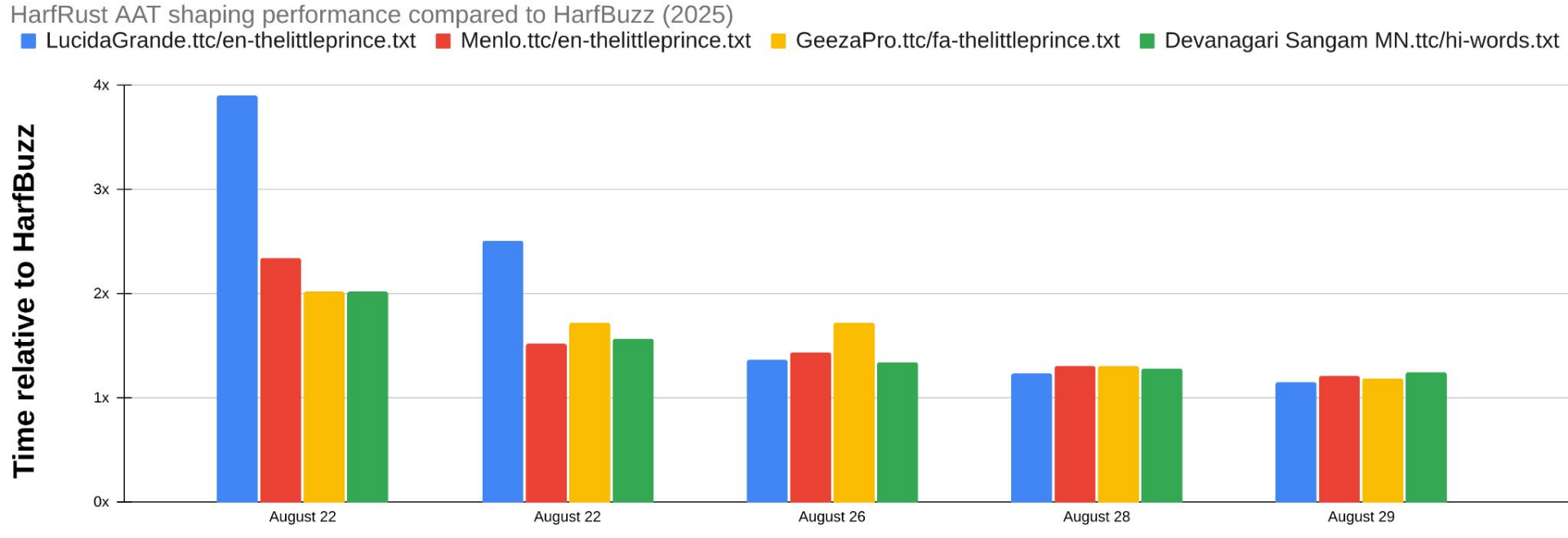
not to be confused with

RustType, a font rasterizer in Rust. Small fraction of *FreeType* functionality.

HarfRust: Performance: OpenType



HarfRust: Performance: AAT

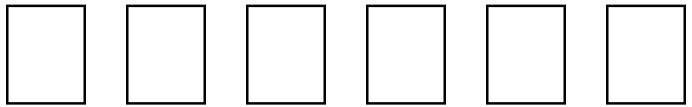


HarfRust: Rollout

- Variety of Rust crates (Parley, Blitz, etc)
- Chrome experiment using `hb-harfrust` (by Dominik Röttsches)
  - Performance promising. Working on parity with HarfBuzz.
  - Expect to port to HarfRust natively, skipping the HarfBuzz bridge for shipping.
- Servo experiment (by Nico Burns)
  - Dual backend (HB & HR) comparing results, discovering bugs.

Standardization

extending the font format



Font format as bottleneck

By mid-2010's we had:

- Open Source font sources,
- Open Source font compiler,
- Open Source operating systems & browsers,
- Open Source font shaping & rendering libraries.

I had an aha moment:

If we could extend the font format,
we could unlock new kinds of innovation.

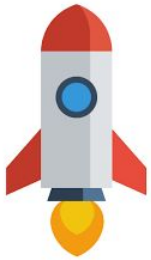
Font format innovation path

- OpenType is property of Microsoft.
- New work in OpenType *cannot* happen without Microsoft's commitment.
- OpenType is mirrored as ISO OpenFontFormat, with its own slow process.
- We took the ISO path in 2022. *Still* not released as a standard.

Font format innovation path

- Google Chrome reluctant to ship its own format extensions.
- Unless **standardized** or **shipped by another major player** first.
- In October 2022, Apple silently shipped our proposed *avar2* technology, just two months after I wrote down the proposal!
- Chrome, just recently in March 2026!
- That’s a 3.5 year *standardization tax* we have to pay.
- While Microsoft and Apple ship their own extension and document later.

Looong shot



WebAssembly *in* the font

Beyond-Emulation: WebAssembly for President!

- Current code-*vs*-data boundary is arbitrary.
- *Securely* inject programmability into the font file.
- *Generic* solution to many hard shaping problems.
- No *specific* shaping engine changes needed.
- Put another way: a plugin system for the shaping engine.
- Bonus: work around the standardization problem.

Q?

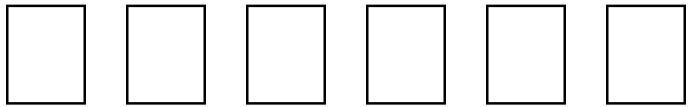
ask away your burning questions
or i'll see you in ten years

or not (╯°□°)╯ ︵ ┻━┻

Backup

Standardization

extending the font format



Font format innovation path: OpenType

- OpenType is the dominant font format specification.
- OpenType is property of Microsoft.
- Industry-wide collaborations with Microsoft have produced impressive results in the past: *variable fonts* and *COLRv1* for example.
- However, Microsoft's willingness to incorporate new changes in OpenType is subject to their engineering and editorial resource availability, which has *allegedly* been going down over the years.
- High chance of impasse. A different strategy is needed.
- After the *color fonts* snafu, Google reluctant to ship its own format extensions.
- Unless **standardized** or **shipped by another major player** first.
- In 2023, Apple silently shipped our proposed *avar2* technology. Chrome still *doesn't*.

Font format innovation path: ISO Open Font Format

- In mid 2000s, to curb competition, Microsoft donated a copy of OpenType to ISO for standardization.
- By 2007, ISO *Open Font Format* (*OFF*), formally known as ISO/IEC 14496-22 was born.
- Microsoft maintains OpenType as a separate specification to this day.
- ISO contribution process is slow, painful, and based on an “ad-hoc” working group’s consensus.
- The ad-hoc group’s recommendations are then voted on by the national delegations.
- The whole process takes between three to five years, from proposal to released standard.

Font format innovation path: ISO Open Font Format

- Chrome is reluctant to ship proposals before late stages of standardization.
- Android is reluctant to ship font format extensions not supported by Chrome.
- Chrome can be open to supporting what other platforms already support.
- This puts us in a double-bind: if we innovate in the file format, we either have to wait 4 years before Chrome will ship it, or hope that Apple would go ahead and ship it (and with no announcement for that matter).
- While in the meantime, Microsoft and Apple feel free to ship font format extensions in their products, and then *maybe* submit to ISO.
- This is a structural problem, not a failure of any single party.

Font format innovations: a timeline

- **1997** OpenType 1.0 was released by Microsoft and Adobe, bringing to an end the *Font Wars* of the 1990s. Documentation leaves a lot to be desired.
- **2000s** Microsoft support Indic scripts in *Uniscribe*, their shaping engine.
- **2007** *ISO Open Font Format* is released, as a derivative of OpenType.
- **2009** Introduction of *DirectWrite* as successor to Uniscribe, as *de facto* shaping standard.
- **~2012** Apple, Google, Microsoft, and Adobe+Mozilla each invented a different way to support color fonts,. All four find themselves standardized at the ISO Open Font Format as competing techs.
- **2014** I organized an informal OpenType face2face meeting with Microsoft, Apple, Adobe, other experts to advance Indic-like text shaping.

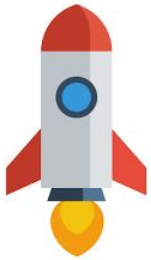
Font format innovations: a timeline

- **2015** Microsoft announced *Universal Shaping Engine* (USE), a more systematic and advanced version of HarfBuzz's homegrown *South-East Asian* shaper. HarfBuzz implements USE quickly and HarfBuzz 1.0.0 is released on July 26, 2015.
- **2016** Based on industry-wide interest, Microsoft invited Adobe, Apple, and Google to collaborate together on what became *OpenType 1.8 Variable Fonts* on September 14th, 2016. The Open Source stack & Chrome are first to ship.
- **2017** Microsoft reduced resources to OpenType and font software drastically, assigning *Peter Constable*, the OpenType editor, to other projects.
- **2019** With *Dominik Röttsches* and others at Google, we proposed *COLRv1*. With Microsoft buy-in, the work was added to OpenType.

Font format innovations: our roadmap

- **2022** I proposed a set of changes collectively called *Boring-Expansion* to the ISO Open Font Format, with Google's financial support and backing. This is an incremental extension of the font format to address decades-long limitations. Other parties participated in technical discussions. The final result has been stuck in the ISO bureaucracy currently.
- **2026** Started next phase of *Boring-Expansion*: new proposals to significantly speed up the slowest text shaping needs, code-named *layout-ng*.

Looong shot



WebAssembly *in* the font

Beyond-Emulation: WebAssembly for President!

- Time to *really* put the *smarts* into the *fonts*.
- Current code-*vs*-data boundary is quite arbitrary:
  - **Font file** provides *data* for how to convert from Unicode text into positioned glyphs.
  - **Layout engine** controls the layout based on font shaping result & Unicode properties. Sometimes with full programmability, as in *LuaTeX* or *SILE* typesetting systems.
- *Securely* inject programmability into the font file.
- *WebAssembly* already portable assembly format with tooling.
- *Generic* solution to many hard shaping problems.
- No *specific* shaping engine changes needed!
- Another use-case: *Graphite*, *DecoType ACE*, alt. shapers.

WebAssembly in smart-fonts! Parallel precedents

- OpenGL 1 fixed-pipeline	→	OpenGL 2+ / WebGPU *programmable* shaders
- Static web-servers	→	CGI, Perl, PHP, Python, JS… *scripting*
- CGI, etc, web-server scripting	→	WebAssembly in edge-computing
- Static web content	→	JavaScript / WebAssembly programming
- Linux physical servers	→	Linux as container of *many* virtual servers
- Pre-digital typesetting	→	METAFONT, TeX, LuaTeX

Beyond-Emulation: BIG question

Beyond-Emulation: the elusive Nastaliq style of Arabic

Requires collision-avoidance or approximation of:

- Easy to code in eg. C / Rust.
- Hard to express as data rules.

OpenType
(No mitigations)

OpenType
(Best effort)

WASM

Beyond-Emulation: Arabic typesetting

Many alternative models developed over the decades:

- *Graphite*
- *DecoType*
- *DigitalKhaatt*

Other custom-built solutions, some only available in Iran or Pakistan.

- We don’t *want* to standardize one solution. It would be way too *specific* for a *generic* font format.

Beyond-Emulation: SIL Graphite

SIL Graphite is an alternative shaping model to OpenType Layout.

Only Firefox, LibreOffice, XeTeX, and possibly minor clients support it, by configuring HarfBuzz with the extra `libgraphite2` backend.

Instead: compile `libgraphite2` to WebAssembly to embed in the font.

- 90kb `Wasm` table payload (40kb as WOFF2).
- 3x slower (with JIT) than natively compiled library.

Beyond-Emulation: how it works

- Proof-of-concept experimental implementation in HarfBuzz.
- Using the nimble `bytecodealliance/wasm-micro-runtime` C API.
- The runtime lacks basic security features like *timeboxing*.
- WebAssembly specification still *very* heavy-handed on instances:
  - Memory allocations happen in multiples of 64kb.
  - No virtual address-space or memory mappings between the host and the guest.
  - Multiple address-spaces envisioned but not fleshed out in the spec.
  - Acceptable performance *unachievable* without runtime instance sharing.
- Rust might have more battle-tested WebAssembly host runtimes.
- Fonts have been built both from C / C++ and Rust shaping logic code.
- `llama.ttf`, `translate.ttf`, etc.

Beyond-Emulation: where to go from here

- Research-level project for the foreseeable future.
- Might never happen in browsers & operating-systems.
- Maybe just enabled by trusted desktop-publishing systems.
- Rewrite the WebAssembly shaping API to be more independent of HarfBuzz internal data representations.
- Assess implementing in HarfRust using a Rust-based WebAssembly host.
- Viewed another way: a HarfBuzz-originated text shaping *plugin* system.
- Ultimately: a solution to the *font format extension* bottleneck.