Summary for variable fonts support for Typst

Jassiel Ovando

Abstract

At most, the important axis are:

- weight
- italic
- slant
- and optical size

So we should focus on them, rather full fledged support for all possible axes, we could make a tracking issue for it, with subissues for each possible axis as necessities arise.

Contents

Axis definitions

Italic (ital)

Adjust the style from roman to italic. This can be provided as a continuous range within a single font file, like most axes, or as a toggle between two roman and italic files that form a family as a pair. Although, there might be cases where the font designer uses it as slant (see Basenji Variable and others from the same foundry).

Property	Value
Default	0
Min	0
Max	1

Property	Value
Step	1

Testing fonts.

- Inter

Weight (`wght`)

Adjust the style from lighter to bolder in typographic color, by varying stroke weights, spacing and kerning, and other aspects of the type. This typically changes overall width, and so may be used in conjunction with Width and Grade axes.

Property	Value
Default	400
Min	1
Max	1000
Step	1

Testing fonts.

- Google Sans Flex

Optical size (`opsz`)

Adapt the style to specific text sizes. At smaller sizes, letters typically become optimized for more legibility. At larger sizes, optimized for headlines, with more extreme weights and widths. In CSS this axis is activated automatically when it is available.

Property	Value
Default	14
Min	5
Max	1200
Step	0.1

Testing fonts.

- Google Sans Flex

Slant (`slnt`)

Adjust the style from upright to slanted. Negative values produce right-leaning forms, also known to typographers as an ‘oblique’ style. Positive values produce left-leaning forms, also called a ‘backslanted’ or ‘reverse oblique’ style.

Property	Value
Default	0
Min	−90
Max	90
Step	1

Testing fonts.

- Google Sans Flex
    - Has fake italics via predefined slanting, so obliques

Width (`wdth`)

Adjust the style from narrower to wider, by varying the proportions of counters, strokes, spacing and kerning, and other aspects of the type. This typically changes the typographic color in a subtle way, and so may be used in conjunction with Weight and Grade axes.

Property	Value
Default	100
Min	25
Max	200
Step	0.1

Testing fonts.

- Google Sans Flex

AR Retinal Resolution (`ARRR`)

Resolution-specific enhancements in AR/VR typefaces to optimize rendering across the different resolutions of the headsets making designs accessible and easy to read.

Property	Value
Default	10
Min	10
Max	60
Step	1

Ascender Height (YTAS)

A parametric axis for varying the height of lowercase ascenders.

Property	Value
Default	750
Min	0
Max	1000
Step	1

Testing fonts.

- Roboto Flex

Bleed (BLED)

Bleed adjusts the overall darkness in the typographic color of strokes or other forms, without any changes in overall width, line breaks, or page layout. Negative values make the font appearance lighter, while positive values make it darker, similarly to ink bleed or dot gain on paper.

Property	Value
Default	0
Min	0
Max	100
Step	1

Bounce (BNCE)

Shift glyphs up and down in the Y dimension, resulting in an uneven, bouncy baseline.

Property	Value
Default	0

Property	Value
Min	−100
Max	100
Step	1

Casual (CASL)

Adjust stroke curvature, contrast, and terminals from a sturdy, rational Linear style to a friendly, energetic Casual style.

Property	Value
Default	0
Min	0
Max	1
Step	0.01

Testing fonts. Recursive

Contrast (CTRS)

Contrast describes the stroke width difference between the thick and thin parts of the font glyphs. A value of zero indicates no visible/apparent contrast. A positive number indicates an increase in contrast relative to the zero-contrast thickness, achieved by making the thin stroke thinner. A value of 100 indicates that the thin stroke has disappeared completely. A negative value indicates “reverse contrast”: the strokes which would conventionally be thick in the writing system are instead made thinner. In western-language fonts this might be perceived as a 19th-century, “circus” or “old West” effect. A value of −100 indicates that the strokes which would normally be thick have disappeared completely.

Property	Value
Default	0
Min	−100
Max	100
Step	1

Counter Width (XTRA)

A parametric axis for varying counter widths in the X dimension.

Property	Value
Default	400
Min	−1000
Max	2000
Step	1

Testing fonts.

- Roboto Flex

Cursive (CRSV)

Control the substitution of cursive forms along the Slant axis. ‘Off’ (0) maintains Roman letterforms such as a double-storey a and g, ‘Auto’ (0.5) allows for Cursive substitution, and ‘On’ (1) asserts cursive forms even in upright text with a Slant of 0.

Property	Value
Default	0.5
Min	0
Max	1
Step	0.1

Testing fonts. Recursive

Descender Depth (YTDE)

A parametric axis for varying the depth of lowercase descenders.

Property	Value
Default	−250
Min	−1000
Max	0
Step	1

Testing fonts.

- Roboto Flex

Edge Highlight (EHLT)

Controls thickness of edge highlight details.

Property	Value
Default	12
Min	0
Max	1000
Step	1

Element Expansion (ELXP)

As the Element Expansion axis progresses, the elements move apart.

Property	Value
Default	0
Min	0
Max	100
Step	1

Element Grid (ELGR)

In modular fonts, where glyphs are composed using multiple copies of the same element, this axis controls how many elements are used per one grid unit.

Property	Value
Default	1
Min	1
Max	2
Step	0.1

Element Shape (ELSH)

In modular fonts, where glyphs are composed using multiple copies of the same element, this axis controls the shape of the element.

Property	Value
Default	0
Min	0

Property	Value
Max	100
Step	0.1

Extrusion Depth (EDPT)

Controls the 3D depth on contours.

Property	Value
Default	100
Min	0
Max	1000
Step	1

Figure Height (YTFI)

A parametric axis for varying the height of figures.

Property	Value
Default	600
Min	−1000
Max	2000
Step	1

Testing fonts.

- Roboto Flex

Fill (FILL)

Fill in transparent forms with opaque ones. Sometimes interior opaque forms become transparent, to maintain contrasting shapes. This can be useful in animation or interaction to convey a state transition. Ranges from 0 (no treatment) to 1 (completely filled).

Property	Value
Default	0
Min	0
Max	1
Step	0.01

Flare (FLAR)

As the flare axis grows, the stem terminals go from straight (0%) to develop a swelling (100%).

Property	Value
Default	0
Min	0
Max	100
Step	1

Grade (GRAD)

Finesse the style from lighter to bolder in typographic color, without any changes overall width, line breaks or page layout. Negative grade makes the style lighter, while positive grade makes it bolder. The units are the same as in the Weight axis.

Property	Value
Default	0
Min	−1000
Max	1000
Step	1

Testing fonts.

- Google Sans Flex

Horizontal Element Alignment (XELA)

Align glyph elements from their default position (0%), usually the baseline, to a rightmost (100%) or leftmost (-100%) position.

Property	Value
Default	0
Min	−100
Max	100
Step	1

Horizontal Position of Paint 1 (XPN1)

The position of the paint moves left and right. Negative values move to the left and positive values move to the right, in the X dimension. Paint 1 is behind Paint 2.

Property	Value
Default	0
Min	−100
Max	100
Step	1

Horizontal Position of Paint 2 (XPN2)

The position of the paint moves left and right. Negative values move to the left and positive values move to the right, in the X dimension. Paint 2 is in front of Paint 1.

Property	Value
Default	0
Min	−100
Max	100
Step	1

Hyper Expansion (HEXP)

Expansion of inner and outer space of glyphs.

Property	Value
Default	0
Min	0
Max	100
Step	0.1

Informality (INFM)

Adjusts overall design from formal and traditional (0%) to informal and unconventional (up to 100%).

Property	Value
Default	0

Property	Value
Min	0
Max	100
Step	1

Lowercase Height (YTLC)

A parametric axis for varying the height of the lowercase.

Property	Value
Default	500
Min	0
Max	1000
Step	1

Testing fonts.

- Roboto Flex

Monospace (MONO)

Adjust the style from Proportional (natural widths, default) to Monospace (fixed width). With proportional spacing, each glyph takes up a unique amount of space on a line, while monospace is when all glyphs have the same total character width.

Property	Value
Default	0
Min	0
Max	1
Step	0.01

Testing fonts. Recursive

Morph (MORF)

Letterforms morph: Changing in unconventional ways, that don't alter other attributes, like width or weight. The range from 0 to 60 can be understood as seconds.

Property	Value
Default	0
Min	0
Max	60
Step	1

Rotation in X (XROT)

Glyphs rotate left and right, negative values to the left and positive values to the right, in the X dimension.

Property	Value
Default	0
Min	−180
Max	180
Step	1

Rotation in Y (YROT)

Glyphs rotate up and down, negative values tilt down and positive values tilt up, in the Y dimension.

Property	Value
Default	0
Min	−180
Max	180
Step	1

Rotation in Z (ZROT)

Glyphs rotate left and right, negative values to the left and positive values to the right, in the Z dimension.

Property	Value
Default	0
Min	−180
Max	180
Step	1

Roundness (ROND)

Adjust shapes from angular defaults (0%) to become increasingly rounded (up to 100%).

Property	Value
Default	0
Min	0
Max	100
Step	1

Testing fonts.

- Google Sans Flex

Scanlines (SCAN)

Break up shapes into horizontal segments without any changes in overall width, letter spacing, or kerning, so there are no line breaks or page layout changes. Negative values make the scanlines thinner, and positive values make them thicker.

Property	Value
Default	0
Min	−100
Max	100
Step	1

Shadow Length (SHLN)

Adjusts the font's shadow length from no shadow visible (0%) to a maximum shadow applied (100%) relative to each family design.

Property	Value
Default	50
Min	0
Max	100
Step	0.1

Sharpness (SHRP)

Adjust shapes from angular or blunt default shapes (0%) to become increasingly sharped forms (up to 100%).

Property	Value
Default	0
Min	0
Max	100
Step	1

Size of Paint 1 (SZP1)

Modifies the size of a paint element going from an initial size (0) to positive values that increase the size (100%) or negative values that shrink it down (-100%). Reducing the size can create transparency.

Property	Value
Default	0
Min	−100
Max	100
Step	1

Size of Paint 2 (SZP2)

Modifies the size of a paint element going from an initial size (0) to positive values that increase the size (100%) or negative values that shrink it down (-100%). Reducing the size can create transparency. Paint 2 is in front of Paint 1.

Property	Value
Default	0
Min	−100
Max	100
Step	1

Softness (SOFT)

Adjust letterforms to become more and more soft and rounded.

Property	Value
Default	0
Min	0
Max	100
Step	0.1

Spacing (SPAC)

Adjusts the overall letter spacing of a font. The range is a relative percentage change from the family's default spacing, so the default value is 0.

Property	Value
Default	0
Min	−100
Max	100
Step	0.1

Thick Stroke (XOPQ)

A parametric axis for varying thick stroke weights, such as stems.

Property	Value
Default	88
Min	−1000
Max	2000
Step	1

Testing fonts.

- Roboto Flex

Thin Stroke (YOPQ)

A parametric axis for varying thin stroke weights, such as bars and hairlines.

Property	Value
Default	116
Min	−1000
Max	2000
Step	1

Testing fonts.

- Roboto Flex

Uppercase Height (YTUC)

A parametric axis for varying the heights of uppercase letterforms.

Property	Value
Default	725
Min	0
Max	1000
Step	1

Testing fonts.

- Roboto Flex

Vertical Element Alignment (YELA)

Align glyphs elements from their default position (0%), usually the baseline, to an upper (100%) or lower (-100%) position.

Property	Value
Default	0
Min	−100
Max	100
Step	1

Vertical Extension (YEXT)

The axis extends glyphs in the Y dimension, such as the Cap Height, Ascender and Descender lengths. This is a relative axis, starting at 0% and going to the typeface's individual maximum extent at 100%.

Property	Value
Default	0
Min	0
Max	100
Step	1

Vertical Position of Paint 1 (YPN1)

The position of the paint moves up and down. Negative values move down and positive values move up. Paint 1 is behind Paint 2.

Property	Value
Default	0
Min	−100
Max	100
Step	1

Vertical Position of Paint 2 (YPN2)

The position of the paint moves up and down. Negative values move down and positive values move up. Paint 2 is in front of Paint 1.

Property	Value
Default	0
Min	−100
Max	100
Step	1

Volume (VOLM)

Expands and exaggerates details of a typeface to emphasize the personality. Understood in a percentage amount, it goes from a neutral state (0%) to a maximum level (100%).

Property	Value
Default	0
Min	0
Max	100
Step	1

Wonky (WONK)

Toggle the substitution of wonky forms. ‘Off’ (0) maintains more conventional letterforms, while ‘On’ (1) maintains wonky letterforms, such as leaning stems in roman, or flagged ascenders in italic. These forms are also controlled by Optical Size.

Property	Value
Default	0
Min	0
Max	1
Step	1

Year (YEAR)

Axis that shows in a metaphoric way the effect of time on a chosen topic.

Property	Value
Default	2000
Min	−4000
Max	4000
Step	1

Roboto Flex

Most axis-feature rich variable font I could found.

- Weight
- Grade
- Optical size
- Slant
- Width
- Thick stroke
- Thin stroke
- Counter width
- Uppercase height
- Lowercase height
- Ascender height
- Descender height
- Figure height

Cheee Variable

- Yeast
- Gravity

Implementation notes

Architecture Overview

The variable fonts implementation follows a layered architecture, inspired by the allsorts crate and building upon LaurenzV's closed PR approach:

```
┌──────────────────────────────────────────────────────────────────────┐
│                        User-Facing API Layer                         │
│                                                                      │
│  text(weight: 450, style: "italic", size: 12pt, ...)                 │
└──────────────────────────────────────────────────────────────────────┘
                                    │
                                    ▼
┌──────────────────────────────────────────────────────────────────────┐
│                     Semantic Axis Mapping Layer                      │
│                                                                      │
│  - weight → wght axis                                                │
│                                                                      │
│  - style (italic/oblique) → ital/slnt axis                           │
│  - size → opsz axis (automatic)                                      │
│  - stretch → wdth axis                                               │
│                                                                      │
└──────────────────────────────────────────────────────────────────────┘
                                    │
                                    ▼
┌──────────────────────────────────────────────────────────────────────┐
│                    Font Selection & Instantiation                    │
│                                                                      │
│  FontBook::select() / select_fallback()                              │
│  → Returns FontKey with InstanceParameters                           │
│                                                                      │
└──────────────────────────────────────────────────────────────────────┘
```

Type	Purpose
`Field<T>`	Enum for static or variable values (with range + default)
`StaticField<T>`	Fixed value for non-variable fonts
`VariableField<T>`	Range + default for variable axes
`SlantAxis`	Slant/italic axis info (`None`, `Slnt{...}`, `Ital{...}`)
`OpticalSizeAxis`	Optical size axis info (`None`, `Opsz{min, max, default}`)
`FontVariantCoverage`	Complete coverage info including all axes

```
                                    │
                                    ▼
┌──────────────────────────────────────────────────────────────────────┐
│                    Variable Font Infrastructure                      │
│                                                                      │
│  InstanceParameters::set_axis(&[u8; 4], f32)                         │
│  → Generic axis value application via ttf-parser/rustybuzz           │
└──────────────────────────────────────────────────────────────────────┘
```

Key Components

1. Axis Metadata Storage (`crates/typst-library/src/text/font/variant.rs`).

Defines data structures to store variable font axis information:

2. Axis Detection (`crates/typst-library/src/text/font/book.rs`). In `FontInfo::from_ttf()`, variable axes are detected from the `fvar` table:

```rust
if ttf.is_variable() {
    for axis in ttf.variation_axes() {
        match axis.tag {
            b"wght" => /* weight axis */,
            b"wdth" => /* width/stretch axis */,
            b"slnt" => /* slant axis */,
            b"ital" => /* italic axis */,
            b"opsz" => /* optical size axis */,
        }
```

```
    }
}
```

3. Instance Parameters (`crates/typst-library/src/text/font/mod.rs`).

`InstanceParameters` stores axis values to apply when instantiating a variable font:

```rust
pub struct InstanceParameters(SmallVec<[AxisValue; 2]>);

impl InstanceParameters {
    pub fn set_weight(&mut self, weight: FontWeight);
    pub fn set_stretch(&mut self, stretch: FontStretch);
    pub fn set_slant(&mut self, degrees: f32);
    pub fn set_italic(&mut self, italic: bool);
    pub fn set_optical_size(&mut self, size_pt: f32);
    pub fn set_axis(&mut self, tag: &[u8; 4], value: f32);  // Generic
}
```

4. Font Selection (`crates/typst-library/src/text/font/book.rs`).

`FontBook::select()` and `select_fallback()` now:

1. Find the best matching font based on variant coverage
2. Build `InstanceParameters` with appropriate axis values
3. Return a `FontKey` containing both font index and instance parameters

```rust
pub fn select(
    &self,
    family: &str,
    variant: FontVariant,
    optical_size: Option<f32>,  // Text size in points
) -> Option<FontKey>;
```

5. Font Instantiation (`crates/typst-library/src/lib.rs`).

`WorldExt::font_by_key()` handles variable font instantiation:

```
fn font_by_key(&self, key: &FontKey) -> Option<Font> {
    if key.instance_params.is_empty() {
        return self.font(key.index);  // Static font
    }
    // Create variable font instance with axis values (memoized)
    create_variable_font_instance(data, index,
key.instance_params.clone())
}
```

6. Shaping Integration (`crates/typst-layout/src/inline/shaping.rs`). The shaping context passes font size for optical sizing:

```
pub trait SharedShapingContext<'a> {
    fn size(&self) -> Abs;  // For optical size axis
    fn variant(&self) -> FontVariant;
    // ...
}
```

Axis Implementation Status

Key Design Decisions

1. **Automatic optical sizing**: Like CSS `font-optical-sizing: auto`, the `opsz` axis is automatically set based on text size in points.
2. **Semantic mapping over raw values**: Users control weight/style/stretch through existing Typst APIs, not raw axis values. This matches 95%+ of use cases.

Axis	Tag	Detected	Auto-Applied	User-Exposed
Weight	`wght`	✅	✅	✅ `text(weight: ...)`
Width/Stretch	`wdth`	✅	✅	✅ `text(stretch: ...)`
Italic	`ital`	✅	✅	✅ `text(style: "italic")`
Slant	`slnt`	✅	✅	⚠️ Via style only, no direct control
Optical Size	`opsz`	✅	✅	✅ Automatic from text size
Custom axes	`****`	❌	❌	❌ Future: `text(axes: (...))`

3. **Memoized instantiation**: Variable font instances are cached via `comemo::memoize` to avoid re-parsing for repeated requests.
4. **Graceful fallback**: If a variable font doesn't have an axis, it's simply not set—no errors.
5. **Range clamping**: Axis values are clamped to the font's supported range to prevent invalid instances.

Files Modified

Future Work

1. **User-exposed slant control**: Allow `text(slant: -12deg)` for direct slant axis control
2. **Generic axis API**: `text(axes: (ROND: 100, CASL: 0.5))` for arbitrary axes

File	Changes
`crates/typst-library/src/text/font/variant.rs`	Added `OpticalSizeAxis`, updated `FontVariantCoverage`
`crates/typst-library/src/text/font/mod.rs`	Added `set_optical_size()` to `InstanceParameters`, exported `OpticalSizeAxis`
`crates/typst-library/src/text/font/book.rs`	Added `opsz` detection, updated `select()`/ `select_fallback()` with optical size parameter
`crates/typst-library/src/lib.rs`	Variable font instantiation in `font_by_key()`
`crates/typst-layout/src/inline/shaping.rs`	Added `size()` to `SharedShapingContext`, pass optical size to font selection
`crates/typst-layout/src/math/shaping.rs`	Added `size` field, implemented `size()` for math context
`crates/typst-layout/src/math/fragment.rs`	Pass size to math shaping
`crates/typst-layout/src/math/mod.rs`	Pass optical size in `get_font()`
`crates/typst-layout/src/inline/line.rs`	Pass optical size in `apply_shift()`
`crates/typst-library/src/visualize/image/svg.rs`	Updated for new `select()` signature (passes `None` for optical size)

3. **Font object integration**: As mentioned in LaurenzV's PR, a `font` object could expose axis information and control

References

- Axis definitions from Google Fonts
- https://github.com/yeslogic/allsorts-tools/blob/master/src/subset.rs
- https://github.com/yeslogic/allsorts/releases/tag/v0.15.0
- https://helpx.adobe.com/after-effects/using/variable-font-axes-support.html
- https://fonts.adobe.com/

Related issues and comments

Support variable fonts. No relevan information, just user request to support it. It's the main issue to support variable fonts.

variable fonts in Flutter.

> Oh there's 2 more things, which might not be relevant to flutter, but maybe are more relevant to flutter than a typical design app...
>
> Flutter developers and end users of flutter apps will need a easy way to know what's actually available in each VF family; that's because they will often be constructing UIs for end users to set axes values, as much as they will set axes values within the apps own code itself.
> eg How to obtain a dict of axes in each variable font family (which can be more than 1 TTF file, despite a lot of oversimplification in docs that "a VF is lots of font files turned into 1 file", especially 2, one roman and one Italic) and their tags, labels, min/default/max values, and flags, which are all in the font files fvar tables.
>
> A nuance here is that axes have a bit mask of flags, and the only one defined so far is "HIDDEN", which isn't meant to mean "inaccessible to users" but rather "shown on request", like from some progressive disclosure or preference UI.

> At a high level, there's 3 benefits to VFs: to compress, to express, and to finesse. Compress is as stated nicely by @srix55 above, to use a bunch of weight styles in a smaller file-size disk representation by run time instantiation of those weights. Express is where the maker user slides the slider or dials the dial or whatever to find just the right value of the axis that expresses their intent, which could be just a very specific weight like 427, or some FUNK or WONK axis to whatever value they feel like, and typically it's clearly visible to anyone what the charge was. Finesse is like opsz or GRAD where the value is set indirectly, based on the context of the text (like font size or dark mode) and are often very subtle adjustments.
>
> So, with this understanding, it may be that some axes are intended for indirect use, or direct but infrequent use, and are best left out of the axes controls shown to every user, always, but rather only shown to users who somehow indicated they do want to see every axis in the font.
>
> It would be nice if flutter made that information also easily available to developers so they can do the right things when exposing axes to their end users.
>
> I guess there may already be such a getter method for font features, and as above a separate and similar method for variations would be great. If there isn't already that for features, that's also necessary, since axes can trigger features and control of both is necessary in that case. The Frances and Recursive fonts demo this.
>
> There is at least 3 "axis registry" data sets, one from ISO MPEG Open Font Format, which is fed mainly by the next one, the Microsoft OpenType Specification which actually has a website version of the spec and is almost always in sync with the first; and the Google Fonts one at GitHub.com/GoogleFonts/axisregistry , which is a superset of, although not directly derivative from, the others.
> These provide additional contextual information about axes, like longer descriptions that explain an axis, typical interesting instance locations, etc.

This is similar to the above, being about providing convenience methods for app developers to expose the full depth of facilities offered by VFs.

For context, fonts.google.com has updated 100s of font families to be variable fonts, including the # 2 most popular family Open Sans - and the previous "Open Sans Condensed" family is no longer listed (although the CSS API serves it so existing users aren't effected) because those styles are now available in the Width (wdth) axis.

I also kindly note that in comments above and eg #67688 (comment) there's a conflation between OpenType features (aalt, cwsh, ss01, etc) and OpenType variations (wght, wdth, WONK, GRAD, etc); features are binary toggles, introduced in the late 90s, and variations are continuous ranges introduced in 2019 - and reaching mass adoption in 2022. The underlying libraries (freetype, harfbuzz) that processes them, that Dart/Flutter depend on, treat them rather separately.

The FontFeature() method would perhaps be only appropriate for features, while a new FontVariation() method may be more appropriate. This would mirror the CSS properties font-feature-settings and font-variation-settings. Although, I also note that those properties are poorly designed for actually cascading (see eg https://web.dev/variable-fonts/#font-variation-settings-inheritance).

An equivalent/superior to the CSS property font-optical-sizing: auto is also strongly desirable for Flutter; I say superior because again that CSS property was poorly designed and a improvement proposal is at w3c/csswg-drafts#4430 (and Chrome has had bugs with this, see eg crbug.com/1305237)

Finally, CSS underspecifies how "outline stroking" and semi-opaque overlapping glyphs should work, and variable fonts exacerbate this - since they make overlapping contours, intra-glyph, very common; inter-glyph overlaps are also common and not consistently handled. While glyph overlaps were always allowed in the TrueType specification (since late 80s...) the PostScript font specs disallowed them, and many TTFs were made

> as derived from PS fonts, such that much software does not correctly handle them. https://jsbin.com/xewuzew demos some issues in current browsers with this.
>
> So, to implement full support for variable fonts, I recommend Flutter have those 4 things:
>
> - a way to set variable font axes values with inheritance
> - by default, when the opsz axis exists in a font, set the value to the font size, and allow a ratio override, plus override via the above axis value sette. (Ideally the auto value is resolved to physical Points, 1/72nd of an inch, although I am recommending that ideal as a partisan and recognize there are other positions on what value to use; but the key idea is that it is set based on font size)
> - correctly render semi-opaque text
> - correctly render outline-stroked text

Related projects

LaurenzV closed PR. Main initial attempt for supporting variable fonts in typst, but closed due to:

> md > This PR is still in-progress and aims to add some initial support for variable fonts. Initial because the aim is not to give the user full control over setting custom variation coordinates (this could be good future work, but probably makes more sense to implement in conjunction with the planned `font` object). The aim is to automatically select a correct instance based on the `wght` (font weight), `wdth` (font stretch), and `ital`/`slnt` (italic) axes, which I think should over 95%+ of the use cases. > > It also adds support for embedding CFF2 fonts in PDFs by converting them to a TTF font, which is not the best approach (better would be to convert to CFF), but it should do the job. CFF2 fonts are pretty rare ([less than 1% of variable fonts](https://almanac.httparchive.org/en/2024/fonts#variable-fonts)), but based on past issues it seems like some systems do use some CFF2-based Noto fonts, so this is definitely

> worth fixing. > > Fixes #185 (tracking support for specifying variable font axes is probably worth opening a new issue for). > > Example: > >

> #for (font, d_text) in (> (“Cantarell”, “I love using variable fonts!”), > (“Noto Sans CJK SC”, “我太喜欢使用可变字体了!”), > (“Roboto”, “I love using variable fonts!”), >) { > [= #font] > for i in range(100, 900, step: 100) { > text(weight: i, font: font)[#d_text] > } > } > ``> > Result: [test.pdf](https://github.com/user-attachments/files/21813827/test.pdf) > > TODOs: > > * Properly hook up thewdthandslant/italicaxes. > * Bulk test with maany fonts to ensure it works as expected. > * Fix or find a workaround for thepixglyph` bug that makes CFF2 fonts render as black rectangles in PNG export. > * Investigate whether the code leads to regressions in SVG rendering, where variable fonts are currently not supported. > * Add test cases > * Clean up code + documentation

```md

It’s located as a closed PR and submodule in `./modules/typst-laurenzv-variable-fonts`.

**All sort crate.** Located in `./modules/allsorts`, as a git submodule for reference, since they have a working variable font implementation.

***All sorts tools repository.*** Located in `./modules/allsorts-tools` also as a git submodule for reference, since it has usage examples and extra tools.

**Typst own subsetter (my fork for testing and development purposes, synced with upstream).** Forked and also as submodule for reference and possible editing/testing since it’s in charge of font subsetting in typst.

Located in `./modules/typst-subsetter`.
```