Prettybird: A DSL for Programmatic Font Compilation

Anonymous Author(s)

SPHINX OF BLACK QUARTZ
JUDGE MY VOW

Figure 1: A pangram of the Latin alphabet using the font "Prettybird Roman," compiled with Prettybird.

ABSTRACT

Prettybird is a new domain-specific font description language that simplifies the early steps of font design. Its simple grammar and reusable functions make font design more approachable to beginners while reducing the amount of time required for experts to craft a new font. Additionally, Prettybird avoids certain pitfalls of METAFONT, including reduced syntactic complexity, better integration with graphical font design software, and improved recursive features for automating the design of high-complexity glyphs.

1 PROBLEM AND MOTIVATION

Improvements in display technology over time have led to an increase in public interest in the field of digital typography. As a result, interest in non-standard font usage has grown over time. Figure 2 shows the approximately 80% increase in webfont requests from 2010–2022 [5].

This rising demand for custom or otherwise non-standard fonts has been driven primarily by rapid growth in the popularity of web application development, which has made diverse fonts more accessible to artists and designers. As such software has become more common, font design has attracted more beginners.

However, the creation of a new font requires a large amount of time. Many of the existing tools required to design high-quality custom fonts are either archaic or expensive. These hurdles hinder the ability of beginners to participate in this form of expression.

Designing tools that best utilize the strengths of both graphical font design software and font description languages provides existing font designers with more agency in designing unique fonts, and new font designers with a modern and versatile approach to the practice.

2 BACKGROUND AND RELATED WORK

METAFONT is an existing font description language developed by Knuth [3] as a companion to TEX. Although a comprehensive and versatile system, METAFONT was designed early enough in the field of computing that it was abandoned as graphical font design tools overcame hardware limitations [1].

METAFONT also suffers from a complex syntax, in part due to the technological limitations placed on it by its implementation languages. Its reliance on symbols and short keywords can make it seem archaic to programmers accustomed to more verbose description languages such as SVG and LATEX.

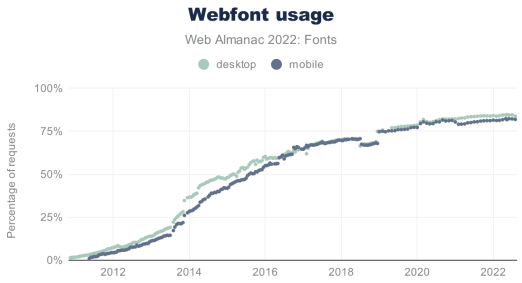

Figure 2: Webfont usage from Nov 15 2010 to Aug 1 2022 [5]

3 APPROACH AND UNIQUENESS

Prettybird aims to replicate the broad capabilities of METAFONT while resolving its syntax issues, and integrating with graphical font design software to provide a modern and versatile font design experience for the maximum amount of users. It provides higher-level features like recursion to allow more intricacy and complexity in glyphs. In summary, we make the following contributions:

- We design and implement a compiler that generates BDF, SVG, and TTF font binaries from Prettybird source code.
- The compiler back-end additionally outputs curve data viewable within the FontForge GUI, for quick and seamless visual feedback to users.
- We design and implement a novel type system consisting of 2 types (numbers and pairs) that is functionally equivalent to METAFONT's 8-type system.
- The compiler back-end also implements a type-checker that enforces type-correctness of atom arguments
- We present an anonymous user survey displaying user preference of Prettybird over METAFONT

3.1 Language Design

Prettybird operates on the concepts of a *glyph space* stack and *atoms* to provide a foundation for outline font description. A glyph space is a two-dimensional plane containing curve information, and an atom is a fundamental curve function. Possible atoms include vectors, ellipses, rectangles, and quadratic Bézier curves.

```
// attach_point is the top-centermost point where the serif
// and symbol meet
define horizontal_serif(attach_point, face_direction, width) {
    draw vector(attach_point - (width, 0),
                attach_point + (width, 0))
    draw bezier(attach_point - (width, 0),
                attach_point - (width, 0) + (width * 0.7,
                    face_direction * 4),
                attach_point - (width, 0) + (width * 0.7, 0))
    draw bezier(attach_point + (width, 0),
                attach_point + (width, face_direction * 4) -
                    (width * 0.7, 0),
                attach_point + (width, 0) - (width * 0.7, 0))
}

define bubble(top, bottom, width, direction) {
    // inner bubble
    draw bezier(top, top + (direction * width, width),
                top + (direction * width, 0))
    draw bezier(top, top + (direction * width * 1.25, width),
                top + (direction * width * 1.25, 0))
    // outer bubble
    draw bezier(top + (direction * width, width), bottom,
                bottom + (direction * width, 0))
    draw bezier(top + (direction * width * 1.25, width),
                bottom,
                bottom + (direction * width * 1.25, 0))
}

char B {
    base { blank(48, 72) }

    steps {
        draw filled rectangle((21, 12), (27, 52))

        horizontal_serif((24, 12), 1, 10)
        horizontal_serif((24, 52), -1, 10)

        bubble((32, 12), (32, 28), 8, 1)
        bubble((27, 28), (27, 52), 12, 1)
    }
}
```

Figure 3: Code example to generate “B” glyph

Glyph spaces are spawned upon function entry. When a function terminates, the positive curve data within the glyph space, drawn using atom calls or other function calls, is binary-unioned with the glyph space directly underneath the current one in the stack. This provides a simple programmatic foundation for glyph design that intuitively resembles layered drawing.

3.2 Compiler Implementation

Prettybird’s compiler automatically generates BDF-, SVG-, and TTF-formatted font files from Prettybird source code. The compiler backend additionally utilizes the FontForge API to convert the language’s IR (glyph spaces) into curve information compatible with FontForge’s graphical font editor UI.

As part of the continued work on Prettybird, we are currently extending the language to add support for an interactive preview window using FontForge that will allow users to combine the benefits of language-based and GUI-based font design. Additionally, we are adding support for user-defined brushes, a feature carried over from METAFONT.

3.3 User Study

Our anonymous user study presented respondents with code samples and glyphs of Prettybird and METAFONT. Prettybird code

Figure 4: The “B” glyph generated by the code example

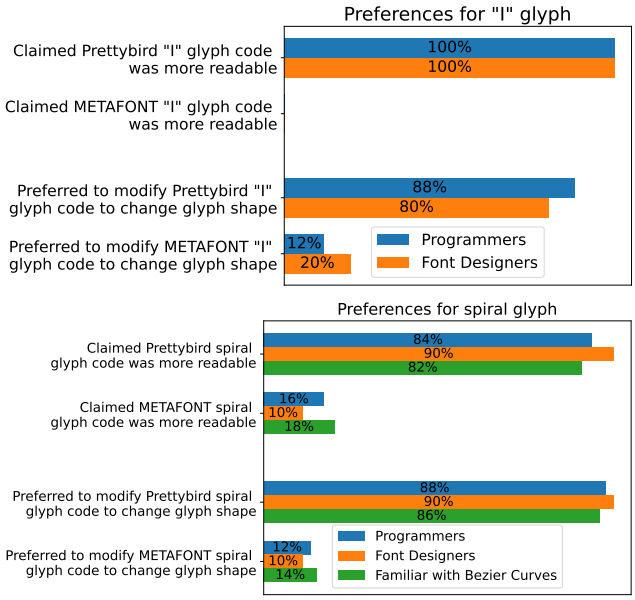

Figure 5: Preferences between Prettybird and METAFONT for the capital "I" and spiral glyphs.

samples were written to replicate the output of corresponding METAFONT samples taken from the METAFONTbook [4] and Dotted and Dashed Lines in METAFONT [2].

Figure 5 shows that respondents in all categories of familiarity with programming, font design, and Bézier curves overwhelmingly preferred the readability of Prettybird to METAFONT, with a sample size of 62. Respondents were selected from online font design communities and university computer science programs, self-identifying themselves into the aforementioned categories of familiarity.

4 RESULTS AND CONTRIBUTIONS

Prettybird is an accessible and powerful description language for programmatic font design. The language offers a simple grammar and common font design utilities to provide a modern mode of font design. By improving on multiple aspects of METAFONT, Prettybird provides the opportunity to revitalize the art of programmatic font design.

REFERENCES

[1] Nelson H. F. Beebe. 2005. The design of TEX and METAFONT: A retrospective.
[2] Jeremy Gibbons. 1970. Dotted and Dashed Lines in METAFONT. (02 1970).
[3] Donald Ervin Knuth. 1990. *Computers and typesetting.* Vol. D - METAFONT: The Program. Addison Wesley.
[4] Donald Ervin Knuth. 1990. *Computers and typesetting.* Vol. C - The METAFONTbook. Addison Wesley.
[5] Bram Stein. 2022. Webfont usage. *Fonts | 2022 | The Web Almanac by HTTP Archive* (Sep 2022). https://almanac.httparchive.org/en/2022/fonts#fig-1