Parametric Type Design in the Era of Variable and Color Fonts

Santhosh Thottingal

Abstract. Parametric fonts are programmatically defined fonts with variable parameters, pioneered by Donald Knuth with his METAFONT technology in the 1980s. While Donald Knuth's ideas in METAFONT and subsequently in METAPOST are often seen as legacy techniques from the pre-graphical user interface (GUI) era of type design, recent trends such as variable fonts suggest a resurgence of certain principles. This paper explores a modern type design process built on parametric design principles, specifically using METAPOST. The author created two variable fonts with this method and released them under a free, open-source license. The paper details the methodology, workflow, and insights gained from this process.

1. Introduction

Parametric type design uses algorithms to create customized and complex typography. It involves defining parameters such as stem width, serifs, and other letter-form components, which can be adjusted to generate a wide range of variations within a single font. The parametric type design approach was pioneered by Donald Knuth with his METAFONT (Knuth, 1979) technology in the early 1980s. However, this approach has not gained widespread acceptance in mainstream type design due to the perceived need for close collaboration between designers and programmers to mathematically define the design. Knuth attributes this to the fact that "asking an artist to become enough of a mathematician to understand how to write a font with 60 parameters is too much." (Knuth, 1985).

The contemporary type design process uses advanced graphical user interface (GUI) editors. These editors not only facilitate the design aspect but also streamline the entire workflow, encompassing the creation of a fully functional typeface and its subsequent proofing. It is worth

Santhosh Thottingal 0009-0004-7350-5234
Swathanthra Malayalam Computing
E-mail: santhosh.thottingal@gmail.com

Y. Haralambous (Ed.), *Grapholinguistics in the 21st Century 2024. Proceedings*
Grapholinguistics and Its Applications (ISSN: 2681-8566, e-ISSN: 2534-5192), Vol. 12.
Fluxus Editions, Brest, 2026, pp. 757–787. https://doi.org/10.36824/2024-graf-thot
ISBN: 978-2-487055-12-4, e-ISBN: 978-2-487055-13-1

noting that these GUI-based tools often come at a substantial cost, are proprietary, and, in some cases, depend on specific operating systems.

Nevertheless, certain foundational concepts introduced by Knuth have persisted and evolved over time. One such concept was the notion of generating multiple instances of a font by manipulating key parameters, a concept now recognized as variable fonts, which have garnered considerable popularity in contemporary typography (Constable, n.d.[b]; Hudson, n.d.).

In light of the fact that typefaces have evolved into increasingly complex software entities rather than mere digital renditions of calligraphic designs, the discipline of type design has transformed into a branch of software engineering. This transformation has necessitated that type designers either possess a working knowledge of software engineering and mathematical principles or collaborate closely with individuals skilled in the realm of 'type engineering'.

As a professional with a background in software engineering who later ventured into type design, I set out to revisit METAFONT and integrate it into the modern type design workflow. The author created two variable fonts with this method and released them under a free, open-source license. The paper details the methodology, workflow, and insights gained from this process.

2. Parametric Design

The fundamental premise of parametric design is that design outcomes are governed by explicitly defined parameters. Altering these parameters independently yields distinct outputs without necessitating a complete redesign. METAFONT is a programming language that helps to define such designs.

2.1. METAFONT and METAPOST

Donald Knuth started work on font creation software in 1977 and produced the first version of METAFONT in 1979. Due to shortcomings in the original METAFONT language, Knuth developed an entirely new METAFONT system in 1984, and it is this revised system that is used today. The AMS Euler typeface was created by Hermann Zapf with the assistance of Donald Knuth using METAFONT. Computer Modern is another typeface created using METAFONT. METAFONT produces bitmap fonts that can be embedded in PostScript documents.

METAPOST, a successor to METAFONT, is a graphics programming language developed by John Hobby that allows users to produce high-quality graphics (Hobby, 1994). METAPOST produces vector graphics

from a geometric/algebraic description. The language shares METAFONT's declarative syntax for manipulating lines, curves, points, and geometric transformations. Unlike METAFONT, METAPOST can produce SVG output.

```
beginfig(1);
side:=10;
draw (0,0) -- (side,0) -- (side,side) --(0,side) -- (0,0);
endfig;
```

LISTING 1. METAPOST code to draw a square

Listing 1 is an example METAPOST program. It produces a square like this. □ Changing the value of *side* to say, 20, draws the same square with a bigger size like this: □. Here, the *side* is a parameter to the design of the square. By changing the value of *side* and without any other modifications to the design, we produce distinct renditions. This is the simplest explanation of parametric design. Let us try to draw the Malayalam letter Ra.

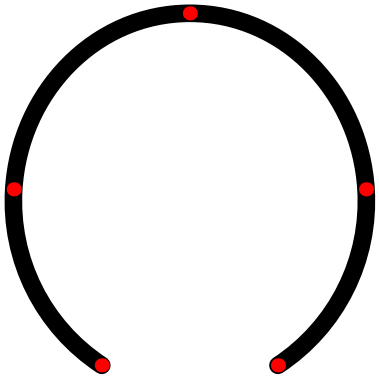
```
beginfig(0);
width:=100;  height:=100;

z0 = (x1 + width/4, 0);
z1 = (0, y0 + height/2);
z2 = (x1 + width/2, y1 + height
 /2);
z3 = (x2 + width/2, y1);
z4 = (x3 - width/4, y0);

pickup pencircle scaled 10;
draw z0..z1..z2..z3..z4;

%Add dots to indicate the points
pickup pencircle scaled 4;
for i=0 upto 4:
    draw z[i] withcolor red;
endfor;
endfig;

```



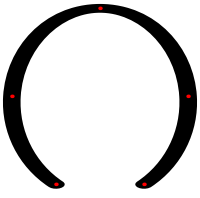
LISTING 2. METAPOST code to draw the Malayalam letter Ra

From the program Listing 2, we can see that we are drawing along the 5 points z_0 to z_4 and using curves to connect them. We are using a 'pen' called pencircle scaled 10 times. Pencircle is just a circle, and it will be moved along the curve and filled. There are *width* and *height* that control

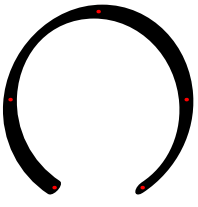
the sweep of the arc too. Similar to 'pencircle', there are other predefined pens like 'pensquare'.

2.2. The Pens

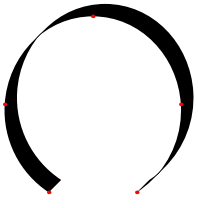
The concept of pens, also known as nibs, is the powerful component of METAPOST. A pen could be of any shape[1]. Let us use a pencircle, but deform it first by scaling 10 in x dimension and 5 in y dimension.`pickup pencircle xscaled 10 yscaled 5` makes it an elliptical pen: ⬬ Then, if we move that pen along the same path, we get:



We can rotate this elliptical nib to an angle and use it as a nib using `pickup pencircle xscaled 10 yscaled 5 rotated 45`. We get this nib: ⬮ Now, if we draw the same shape using this slanted nib, we get:



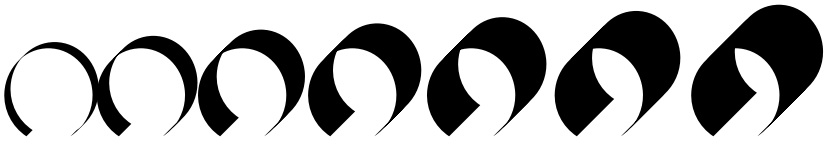
Let us attempt to define a broad nib calligraphy pen. They are straight lines with little thickness (we can assume 0 thickness, or razor-thin), held at an angle like 40 or 45. It will look like this: ／ If we draw with this calligraphy pen, we get:



1. Any convex polygon, to be precise.

As we saw from the examples above, we were able to produce a varying number of strokes from the simple pens defined. However, more complex character shapes, such as those found in serif fonts, cannot be adequately represented using pen strokes alone. One approach is to use a combination of different pens. Another approach is to draw the outlines directly instead of using the outline created using pens.

Not only the type of pen but also its aspects can be parameterized. For example, if we change the length of the calligraphic nib in the previous example, we can generate the same drawing with varying stroke width:



This is particularly useful in type design for generating weight variants such as thin, light, regular, bold, black, etc. Since each of the drawings will have an exact number of nodes, they are easily interpolatable to generate a variable font with a 'weight' axis.

2.3. The Curves

The `draw` command of METAPOST connects the points using straight lines or curves. The points are defined by coordinates in the form of (x,y). Points connected using dots are curves ... Points connected using -- are straight lines. --- connect a straight line and curve with a smooth joint. So examples:

```
draw (0, 0) -- (20,0) --(20, 10)
draw (0, 0) .. (10,0) .. (20,0)
draw (0, 0) -- (10,0) .. (20,0)
draw (0, 0) --- (10,0) .. (20,0)
```

When METAPOST draws a smooth curve through a sequence of points, each pair of consecutive points is connected by a cubic Bézier curve, which needs, in order to be determined, two intermediate control points in addition to the end points. The points on the curved segment from points p_0 to p_1 with post control point c_0 and pre control point c_1 are determined by the formula

$$p(t) = (1-t)^3 p_0 + 3(1-t)^2 t c_0 + 3(1-t)t^2 c_1 + t^3 p_1,$$

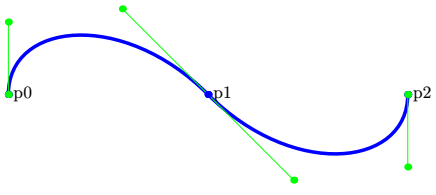
where $t \in [0,1]$. METAPOST automatically calculates the control points such that the segments have the same direction at the interior knots.

In the Figure below, the additional control points are drawn as green dots and connected to their parent point with green line segments. The curve moves from the starting point p_0 in the direction of the post control point of p_0, but bends after a while towards p_1. The further away the post control point is, the longer the curve keeps this direction. Similarly, the curve reaches a point from the direction of the pre-control point. The further away the pre-control point is, the earlier the curve gets this direction. It is as if the control points pull their parent point in a certain direction, and the further away a control point is, the stronger it pulls.

```
beginfig(1);
u := 2cm;
pair p[];
p0 = (0,0);  p1 = (2u,0);  p2
  = (4u,0);
path q; q := p0{dir 90}..p
 1..{dir 90}p2;
draw q withpen pencircle
 scaled 2 withcolor blue;

for i=0 upto length(q):
    dotlabel.rt("p" & decimal
(i), point i of q);
    draw point i of q
      withpen pencircle
scaled 4
      withcolor blue;
    p3 := precontrol i of q;
    p4 := postcontrol i of q;
    draw p3--p4 withcolor
green;
    draw p3 withpen pencircle
 scaled 4 withcolor green;
    draw p4 withpen pencircle
 scaled 4 withcolor green;
endfor;
endfig

```



By default in METAPOST, the incoming and outgoing direction at a point on the curve are the same so that the curve is smooth. The algorithm behind the curve smoothening in METAPOST is developed by John D. Hobby. (Hobby, 1986b)[2]. METAPOST allows the control points to be specified directly in the following format:

2. The hobby package with Tikz package of LaTeX can also create similar smooth curves. Before developing the METAPOST- based workflow, I attempted to create SVGs using Tikz; however, I abandoned it for various reasons, including inflexibility

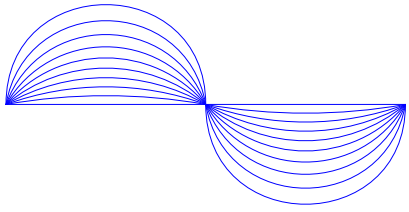
```
draw (0,0)..controls (26.8,-1.8) and (51.4,14.6)..(60,40)
```

It is easy to adjust these control points in a GUI editor, but in METAPOST, we want to focus only on the curves. For this, METAPOST provides a mechanism to declare the direction, tension, and curl of the curves along with the path expression.

```
draw (0, 0){dir 30}..{dir 45}(10, 10)..{right}(20, 0)..{dir 30}(30,
10) =
```

To illustrate how the direction specification changes the curves, let us plot curves with varying directions as below:

```
1 beginfig(0)
2 for a=0 upto 9:
3     draw (0,0){dir 10a}
4     ..{dir -10a}(4cm, 0)
5     ..{dir 10a}(8cm, 0) withcolor
      blue;
6 endfor
7 endfig;
```



METAPOST also allows specifying the tension and curl of the curves. Since they are not used in the typefaces I designed, we are skipping the explanation here for brevity.

2.4. SVGs from METAPOST

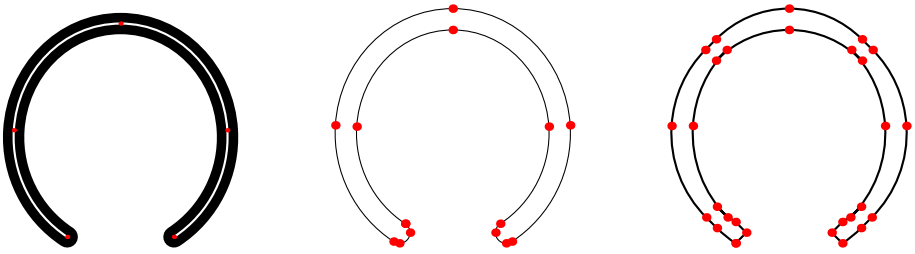
The latest version of METAPOST can output SVGs. The previous examples of rendering a glyph with different pens might lead us to believe that we can directly use those SVGs in a font. However, a close inspection of the SVGs produced will disappoint us. Let us revisit the 'Ra' example:

In Figure 1a, the white line is the path we defined and drew. The thick black line along that path is just the path's stroke width. So this is an SVG with just 5 nodes. This is not an SVG we can use for type design. In a glyph, we need a closed outline. Our expected drawing should be as shown in Figure 1b.

While designing fonts with METAFONT Knuth outlines (Knuth, 1989) a method for 'envelope' calculation. Internally, in METAFONT all the pen-circles are approximated by convex polygons with 24 sides. Then this polygon is used just like pensquare (that has 4 sides). Then the outlines of the polygon pen are calculated to get the 'envelope' or outline

---

and SVG-related issues. https://ctan.math.illinois.edu/graphics/pgf/contrib/hobby/hobby.pdf.



(a) SVG output from pencircle method

(b) Expected SVG for use in a typeface

(c) Envelope(outline stroke) calculated from a convex polygon pen

FIG. 1. Comparison of SVGs produced by METAPOST

stroke. This works well for the METAFONT since it outputs bitmaps. But for METAPOST, since the output is a vector image and we want the SVG to be a very clean vector with very few nodes, this solution has issues, as we see in Figure 1c. It produces several unwanted nodes, and they are unpredictable when parameters such as width/height are changed. We resolved these issues by using the macros provided by the MetaType1 package.

2.5. MetaType1

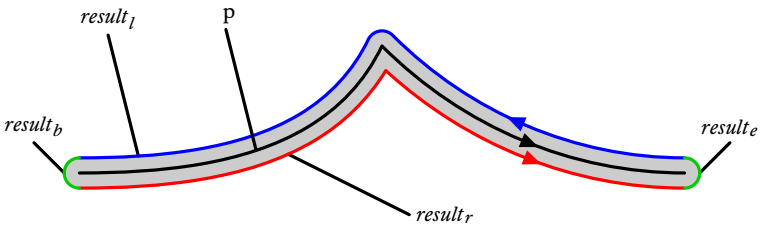
MetaType1 is a tool created by Bogusław Jackowski, Janusz Nowacki, and Piotr Strzelczyk for creating PostScript Type 1[3] fonts (Jackowski, Nowacki, and Strzelczyk, 2001). MetaType1 was used to create the Latin Modern fonts, which are derived from Computer Modern fonts and include many more accented characters (Jackowski and Nowacki, 2003). Most important fonts produced with MetaType1 are: Latin Modern, Latin Modern Math, TEX Gyre, Antykwa Toruńska, Antykwa Półtawskiego, Kurier, and Iwona.

Even though we are not generating any PostScript fonts here, MetaType1 was very crucial in the development of the workflow. The utility library, as part of MetaType1, includes many macros to produce accurate pen envelopes suitable for typefaces.

The Nupuram project extensively used the macro *pen_stroke* defined in MetaType. Macro *pen_stroke* performs an operation known as "expanding stroke"; we'll call the result of the operation a "pen envelope" (for a

3. Type 1 fonts are a legacy format created by Adobe in 1984. See Haralambous (2007, pp. 655–676).

given path). The macro has one optional parameter, *opts* (*text*), and two obligatory ones: input path *p* (*expr*) and a *result* (*suffix*). A user has an access to subpaths of the envelope, namely: $result_r$ is the right edge of the envelope, $result_l$—its left edge, $result_b$—is a fragment of the pen outline joining left and right edge of the envelope at the beginning node of the path, $result_e$—is a similar fragment at the ending node of the path (see the picture below). If the path *p* is cyclic, then $result_e$ and $result_b$ are undefined; the variable *result* also contains the fully expanded stroke.



For finding an envelope, a default path (*default_nib*, returned by *fix_nib*) is used except for nodes for which the parameter *opts* sets another pen. Mastering the usage of the parameter *opts* allows a user to achieve nontrivial effects. The parameter *opts* is a list (space-separated or semicolon-separated) of the following operators: (1) *nib*, (2) *cut*, (3) *tip*, and (4) *ignore_directions*.

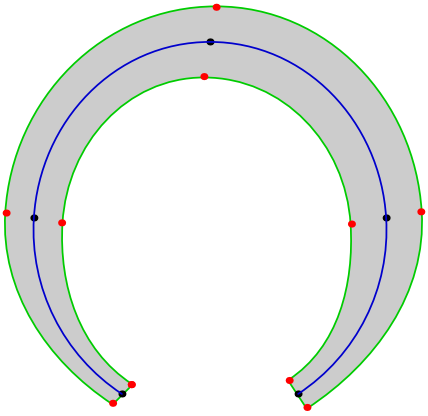
The macro *nib* has two parameters: *nib*(pen)(list_of_nodes), where 'pen' is a path returned by macro *fix_nib*, and 'list_of_nodes' contains comma-separated numbers (times) of the nodes of the path *p* at which a given pen is to be used.

To illustrate this, let us revisit the same arc examples. As shown below, by using the *pen_stroke* macro and using different pens at different nodes in the path, we get a perfect vector output. We can see the arc is drawn using 5 points. And corresponding to each node, 2 nodes are present in the outline stroke, on the outer and inner sides of the outline stroke. Depending on the pen thickness at each node, we get smooth curves that vary in thickness.

The resulting stroke modulation and its extrapolatable nature with respect to parameters such as nib thickness are key tools for building a typeface, potentially a variable typeface. The terminals can be cut at any angle, and the nibs at any angle or width. Each node in the path can be any nib at any width, shape, and angle.

```
input plain_ex;
beginfig(0);
width:=200;      height:=200;
thick:=width/5;  thin:=thick*2/3;
% .. path definition goes here ..
path p, s;  p := z0..z1..z2..z3..z4;

vardef thicknib = fix_nib(thick, 0, 0) enddef;
vardef thinnib = fix_nib(thin, 0, 0) enddef;
pen_stroke(
    cut(thinnib, 45)(0)
    nib(thinnib scaled 1.2 rotated -10)(1)
    nib(thicknib rotated 80)(2)
    nib(thicknib rotated 10)(3)
    cut(thinnib scaled 1.25, rel 90)(4)
)(p)(s);
draw s withpen pencircle scaled 1;
fill s withcolor .8white;
endfig;
```



LISTING 3. Usage of *pen_stroke* macro

3. Earlier Attempts

Although METAFONT never became mainstream, hundreds of fonts have been created with it: an incredible list has been compiled by Luc Devroye[4].

Jeroen Hellingman created Malayalam metafonts in 1994 (Hellingman, 1998). These were created as uniform strokes. Karel Piska later converted all these fonts to Type 1 fonts. He did this by an accurate analytic conversion to outlines using METAPOST output. After theoretical conversion, FontForge is used for removing overlap, simplification, rounding to an integer, auto hinting, generating outline fonts, and necessary manual modifications.[5]

4. List of fonts created using METAFONT. Compiled by Luc Devroye. http://luc.devroye.org/metafont.html.

5. CTAN: indic-type1—Indic Type 1 fonts converted from public METAFONT sources https://ctan.org/pkg/indic-type1.

4. Modern Typeface Design Workflow With METAPOST

As we explained earlier, the first step is to define the glyphs using METAPOST. Then we compile them to SVGs. These SVGs are then converted to the GLIF format of Unified Font Object format[6]. The Unified Font Object (UFO) is a cross-platform, cross-application, human-readable, future-proof format for storing font data. This is the modern format used by type designers. Type design tools offer a feature to export their internal format to UFO too[7].

Once we have the UFO-formatted font source, we need to write the Opentype features and save them as part of the same UFO file. Then type compiling tools like fontmake[8] can compile this UFO to a font binary such as .ttf, .otf, or webfonts. The opentype features, glyph to unicode mapping, kerning, and font meta information—these are all part of the prepared UFO. A set of programs prepares all of these based on a simple configuration file.

Preparing the OpenType features used to be a major task in Malayalam typeface design. We used to manually write them. In the Nupuram typeface, they are automatically generated based on the script grammar encoded in the application logic.[9]. For a script, Malayalam, OpenType rules are an integral part of the typeface. Rapid experimentation and variation generation with Nupuram were possible due to the automatic generation of OpenType rules.

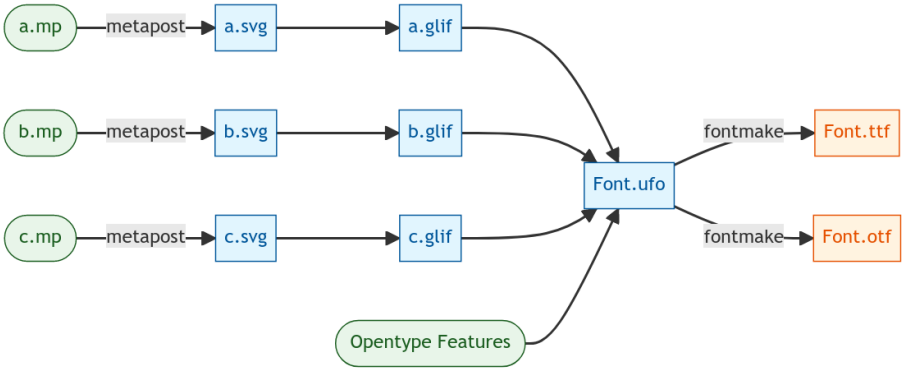


FIG. 2. High level workflow of type design with METAPOST

6. https://unifiedfontobject.org/versions/ufo3/.
7. For example, Glyphs application's .glyphs format can be converted to ufo format.
8. fontmake : https://github.com/googlefonts/fontmake.
9. Source code repository of Nupuram https://gitlab.com/smc/fonts/Nupuram.

FIG. 3. Vilakkum Velichavum title design by SA Nair

5. Nupuram Type Family

5.1. Design

Nupuram[10] draws inspiration from the title posters of early Malayalam movies around 1960-1970, particularly those crafted by the renowned title designer, S Appukkuttan Nair(Popularly known as SA Nair)[11]. These title designs feature letterforms with wide, flat, sharp terminals, thin vertical strokes, and thick horizontal strokes.[12]. Even though there are hundreds of posters done by SA Nair with same design concept, there is no strict uniformity in these designs due to their entirely handmade nature. Adapting these distinctive traits to a typeface required many efforts. In this endeavor, while I departed from the pronounced sharpness of the terminals, I preserved the broad strokes while slightly reducing their thickness.

The vertical stems in glyphs are thin, and horizontal ones are thick, akin to the characteristics of reverse contrast typefaces. The letters are closer to a handwritten style than a regular print style. One can easily notice the playful, casual, personal aesthetic. Usable at display and text sizes too.

10. The word Nupuram means 'anklet' https://en.wikipedia.org/wiki/Anklet.

11. SA Nair, Malayalam Movie and Music Database https://m3db.com/sa-nair.

12. Examples: Thakilu Kottampuram (https://tinyurl.com/52enjcaa), Vilakkum Velichavum (https://tinyurl.com/49s97eup), Angadi(https://tinyurl.com/7cpnzy4s).

FIG. 4. Nupuram design

Based on the macros we explained in Section 2.5, the nibs of Nupuram glyphs are defined. There are three 3 nibs—*thicknib*, *thinnib* and *terminalnib*. Here, *thicknib*, *thinnib* are razor nibs or nibs with 0 width. Their lengths are parameterized. *terminalnib* is a razor nib or an elliptical nib based on whether the terminals are rounded (soft) or flat (sharp).

5.2. Typographic Units

The nib length (which results in the thickness of strokes) is defined by *thick* and *thin* configurations. The value *thick* is expressed in relation to EM Size. the emsize is defined as $em := 1000$. Then this Em value is divided by basic typography units, defined as $u := em/10$. The u is the unit we will use to define all typographic parameters such as ascent, descent, bearing, thick, thin, etc.

```
1   em :=          1000;        % Height of characters - Em square
2   u :=           em/10;       % Unit width. Em square divisions.
3   ascent :=      8u;          % Ascender Height
4   descent :=     2u;          % Descender Height
5   xheight :=     2/3*ascent;  % Height of English small letters
6   mheight :=     3/4*ascent;  % Height of Malayalam letters
7   Xheight :=     8u;          % Height of English capital letters
8   thick :=       1u;          % Thickness of thickest lines
9   thin :=        0.7;         % Thickness of thinnest lines-ratio of
      thick.
10  subthick :=    0.666u;      % Thickness of thickest lines in
      subscribed characters
11  xthick :=      1;           % Extra thickness for terminals
12  slant :=       0;           % Slant of characters.Give angle values
13  condense :=    1;           % Condense factor. < 1 for condense, > 1
      for expand
14  lbearing :=    0.4u;        % Default left bearing
15  rbearing :=    0.4u;        % Default right bearing
16
```

LISTING 4. Basic typographic parameters defined in Nupuram typeface

By changing any of these base parameters, we can build a new design. But not all changes produce a good or useful design. To make a typeface

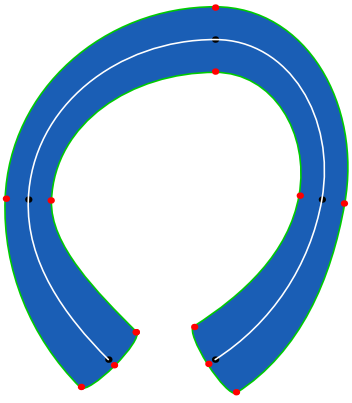
with weight variants, we change the *thick* parameter alone. To change the extra thickness of terminals, we change the *xthick* parameter. To get a slanted glyph with 15, we give *slant* = 15 and so on. We will discuss this in detail in the next section about variable fonts.

Using all these parameters, let us try to draw the letter 'Ra', but this time as per the Nupuram design. See Listing 5.

```
input plain_ex;
beginfig(0);
width:=200; height:=200; thick:=width/5;
thin:=0.7; m:=width;   xthick:=1.1;
terminalround:=0.5;
z0=(x1 + m/4 , 0);   z1=(0, m/2);
z2=(x0 + m/3, y1 + m/2);    z3=(x2 + m/3, y2-
    m/2);
z4=(x2, 0);
path p,s; p:=z0{dir 135}..z1.. z2{right}..z
    3{dir 260}..z4;

vardef thicknib = fix_nib(thick, thick, 0)
    enddef;
vardef thinnib = fix_nib(thick*thin, thick*
    thin, 0) enddef;
vardef terminalnib = fix_nib(thick*xthick,
    thick*terminalround, 0) xyscaled(xthick,
     terminalround) enddef;
vardef terminalangle expr t of p = angle(
    direction t of p)+90 enddef;
pen_stroke(
    nib(thinnib)(1,3) nib(thicknib)(2)
    nib(terminalnib rotated terminalangle 0
    of p)(0)
    nib(terminalnib rotated terminalangle 4
    of p)(4)
)(p)(s);
draw s;fill s;
endfig;

```



Listing 5. The 'Ra' letter in Nupuram

5.3. Variable Fonts

The glyphs and their variations are fully controllable by simple configuration files. On top of these common configurations, we define font-specific variations in simple configuration files. For example, the configuration for creating Nupuram-Bold font will look like this

```
input ./config/Regular;
thick:= 1.25u;
```

It imports (includes) the Regular configuration and sets *thick* = 1.25*u*. You will quickly see that it is an increase on the *thick* value set in the configuration for the 'Regular' variant

```
thick := 0.90u;
soften := 0;
```

Let us look at the configuration for the Nupuram Thin variant by setting *thick* = 0.5*u*.

```
input ./config/Regular;
thick := 0.5u;
```

Similarly, the configuration for Nupuram Condensed is defined by setting *condense* := 0.8

```
input ./config/Regular;
condense := 0.8;
```

An oblique variant will have the following configuration *slant* := 15

```
input ./config/Regular;
slant :=15; % Degree of slanting
```

Whether the terminals are flat (sharp edge) or rounded (smooth) is controlled by the *terminalnib* that uses *terminalround* parameters. By setting it *terminalround* := 0.15 we can get a sharp terminal.[13]

13. Note that we are not setting it to 0 because, at zero, the curve segment at the terminal becomes a straight line. This affects the interpolation. We can only do interpolation of a curve with another curve with more or less curvature.

TABLE 1. Nupuram variation axis tags, their ranges, default, and descriptions

Axis	Tag	Range	Default	Description
Weight	`wght`	100 to 900	400	Thin to Black. Can be defined with `font-weight` CSS property.
Slant	`slnt`	-15 to 0	0	Upright (0) to Slanted (about 15).
Width	`wdth`	75 to 125	100	Condensed to Expanded. Can be defined with usual `font-stretch` CSS property.
Soft	`SOFT`	0 to 100	50	Sharp to normal to Super-Soft terminals.

```
input ./config/Regular;
terminalround:=0.15;

```

Since all of the above variants are the same design, with the exact number of SVG nodes and the same curves, they are interpolatable. We can use them as master designs in a variable fonts design space and get a variable font with 'weight', 'width', 'slant,' and 'soft' axes.

All 4 axes can be considered a 4-dimensional space. By carefully choosing the values of any of these 4 axes, we can practically produce an infinite number of styles. Modern typesetting software and web pages (CSS) allow selecting this style with quick previews. Even though a user can choose any of these styles, the typeface also comes with a predefined 'named instances,' which are predefined combinations of these values. For example, 'Nupuram Condensed Thin' is a named instance that sets the 'width' and 'weight' axis values to 'Condensed' and 'Thin'. Nupuram has 32 such named instances.

Nupuram is the first variable font in Malayalam with 4 design axes.

5.4. Debugging and Proofing

GUI-based font editors usually have advanced preview and proofing systems integrated. This is also an essential requirement for any type design system. While designing Nupuram, I used the Visual Studio IDE along with a live SVG preview. Figure 8 shows an example setup. In development mode, in addition to glyphs, guidelines, and coordinates, points are also visualized to support the design process.

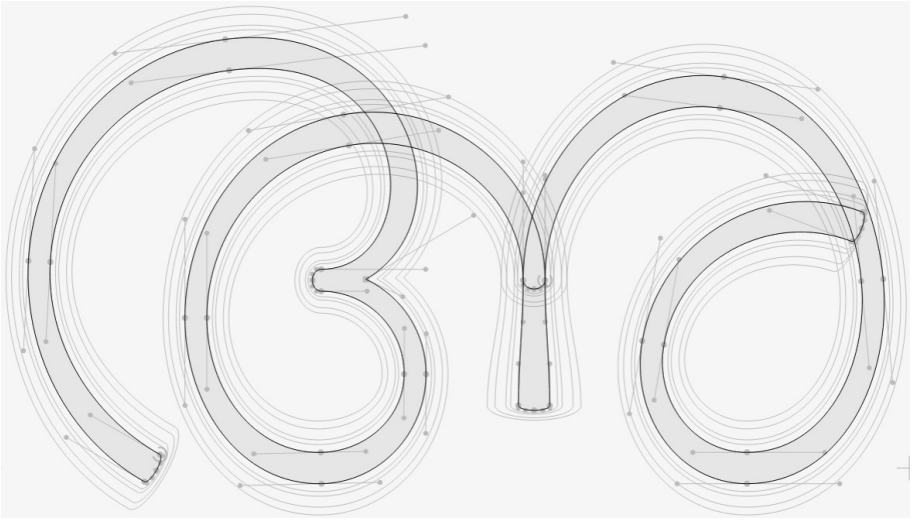


FIG. 5. Visualization of Nupuram `wght` axis. The thickness of strokes varies here.

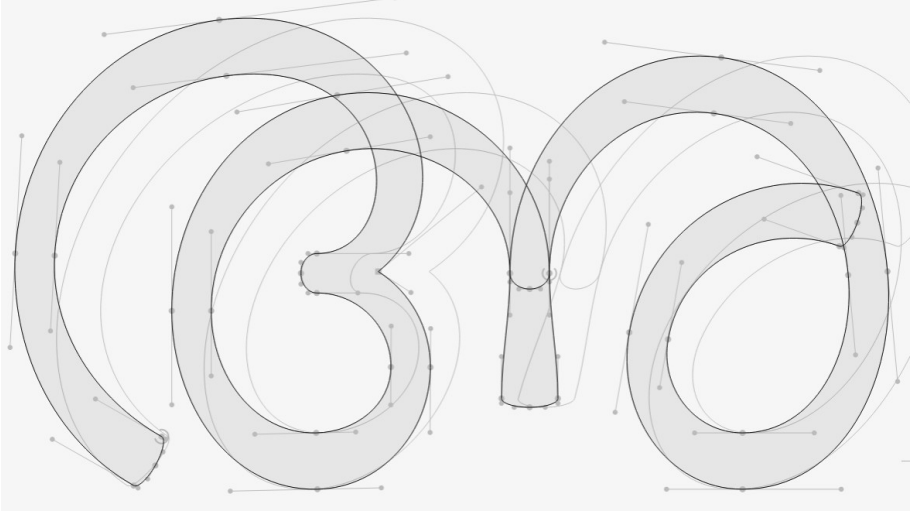


FIG. 6. Visualization of Nupuram `slnt` axis. The slant angle changes from 0 to -15.

However, we also need a more advanced proofing system to render an arbitrary text content with the given font to evaluate and fine-tune

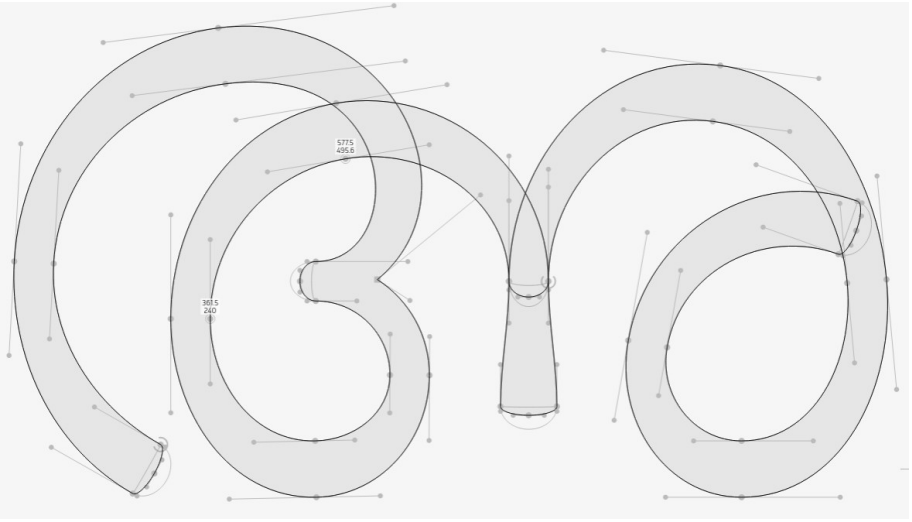


FIG. 7. Visualization of Nupuram SOFT axis. The sharpness or softness varies from 0 to 100.

various design aspects. For this, I developed a web-based type test and preview system[14]. See Figure 9.

To ensure the quality of glyphs and various technical details to be taken care of for a production-ready typeface, automatic testing is incorporated using the fontbakery tool[15].

5.5. Latin Glyphs

All the Malayalam characters defined in Unicode version 15 are present in the font. Nupuram also has Latin script support. Nupuram supports 294 languages covering approximately 2.8B speakers[16]

The Latin glyphs follow the same design as Malayalam. See Figure 10.

5.6. Nupuram Calligraphy

Nupuram Calligraphy simulates a wide-nib calligraphy pen, with a 40 ° nib rotation. This is a variable font with a 'weight' axis. The width of the

14. This online tool is available at `https://smc.gitlab.io/fonts/Nupuram/tests/`.

15. fontbakery: A font quality assurance tool `https://github.com/fonttools/fontbakery`.

16. Calculated using hyperglot tool `https://hyperglot.rosettatype.com/`.

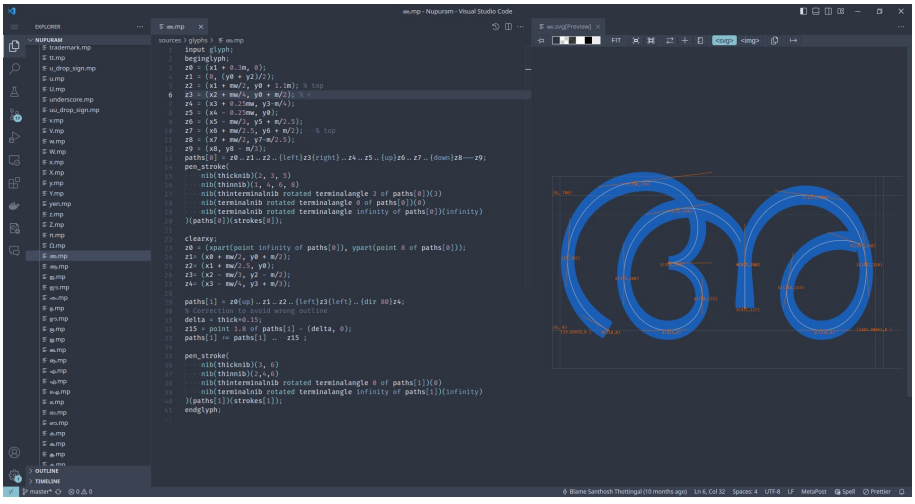


FIG. 8. An example typedesign setup with VS Code IDE with METAPOST and SVG generated, shown side by side. Changing the code automatically refreshes the image.

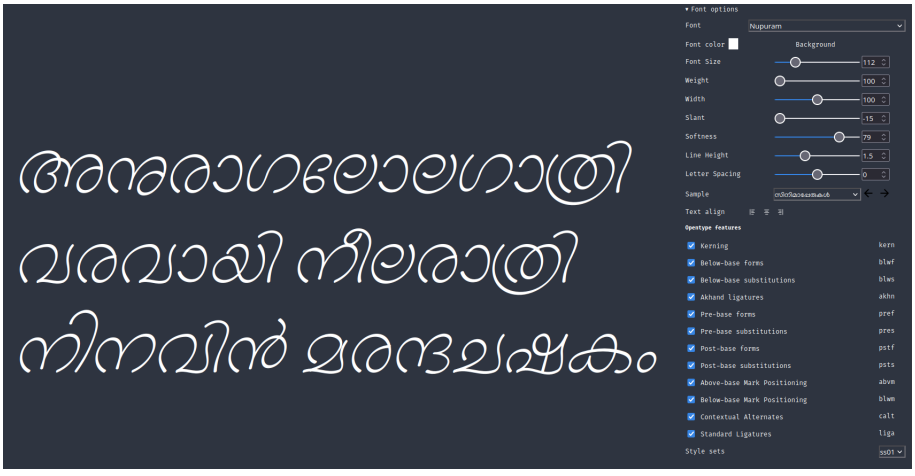


FIG. 9. Web-based type testing, proofing, playground used in the Nupuram design process

calligraphy pen can be varied to get different weights. The width of the nib can be varied as required. So by defining 3 widths, narrow, medium, and wide, we can get 3 variants of this glyph. Using these master glyphs, we can interpolate to any nib size using the variable font technology.

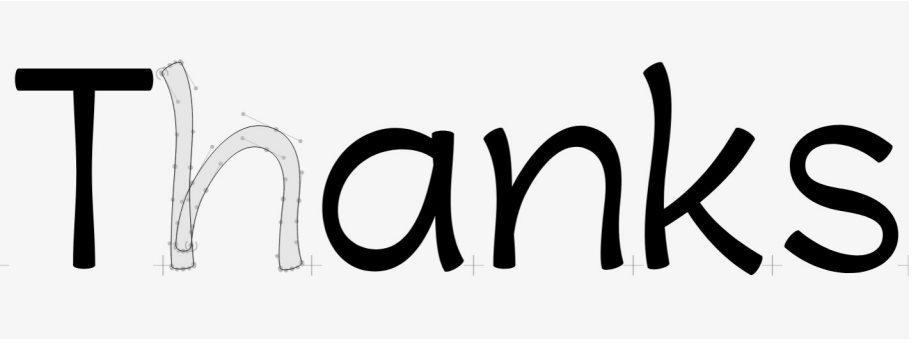


Fig. 10. Nupuram English sample

The Earth is a very small stage in a vast cosmic arena. Think of the rivers of blood spilled by all those generals and emperors so that, in glory and triumph, they could become momentary masters of a fraction of a dot. Think of the endless visited by the inhabitants of one corner of this pixel the scarcely distinguishable inhabitants of some other corner, how frequent their misunderstandings, how eager they are to kill one another, how fervent their hatreds. The Earth is the only world known so far to harbor life. There is nowhere else, at least in the near future, to which our species could migrate.

Fig. 11. Nupuram sample English paragraph

That is how we made the Nupurum Calligraphy variable font. An illustration of weight axis variations is given in Figure 12.

5.7. Nupuram Color

Color fonts (also known as chromatic fonts) can use multiple colors, including gradients, in a single glyph, rather than the flat, single color used by typical, non-color (monochromatic) fonts. This relatively new technology allows designers to set the font's color palette to express themselves with color in ways that previously would not have been possible outside advanced graphics applications.[17] Nupuram has a Color font version with COLRv1 specification. The colors can be customized by users, for example, using CSS. Nupuram Color is also a variable font with a customizable 'wght' axis.

17. Introducing color fonts, Rod Sheeter `https://fonts.google.com/knowledge/introducing_type/introducing_color_fonts`.

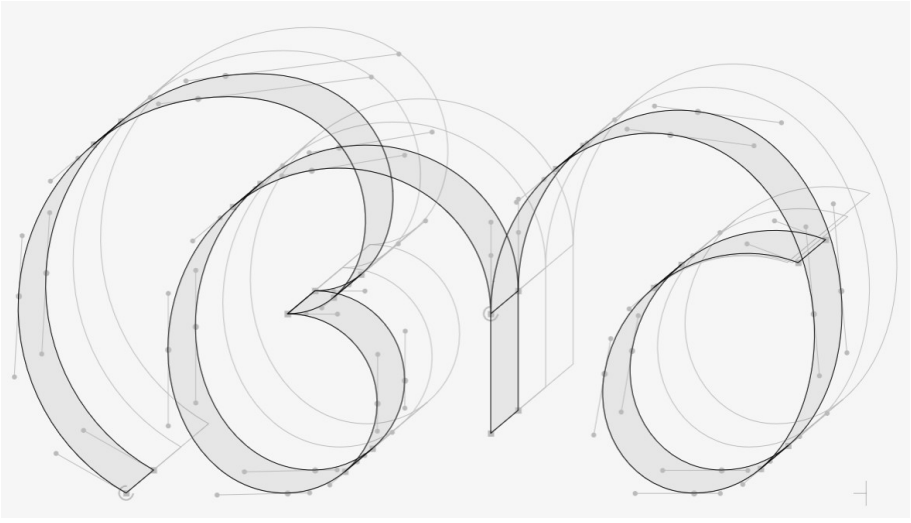


FIG. 12. Weight axis variations of Nupuram Calligraphy subfamily

FIG. 13. Nupuram Calligraphy sample

Nupuram Color creates a solid 3D object illusion. The anatomy of the color font is based on an observation in Nupuram Calligraphy. In the calligraphy variant, we moved a calligraphic pen through the glyph path. If we use pen strokes with thick and thin strokes as explained earlier, we get a modulated glyph outline as Figure 15a. If the calligraphic pen we used for Nupuram Calligraphy moves through the outline, we will get Figure 15b.

It is slightly confusing, but we start to see a 3D shape in it. Let us fill this with blue to get a clearer picture. See Figure 15c. Now we can relate this to the calligraphy glyph we constructed earlier. Same stroke modulation, but two times—for outer and inner lines.

If we look at 15c carefully, this is a 3D structure, but a hollow one. The facing size and backside is void, creating a hollow structure. To get a solid 3d shape, let us take the envelope of the whole drawing. What I mean by that is to take the outline of this structure. See Figure 15d.

Now it is not hollow. If you look carefully, you will see it is a 3D letter of 'va'. But to help your eyes with the 3D perspective, it needs colors or lighting to create depth. To start with, let us place our original 'va'

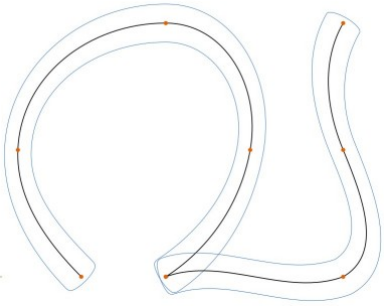
FIG. 14. Sample rendering of Nupuram color font with default color palette

outline on top of it in a lighter color. And let us make the lighting and coloring a bit more realistic by using color gradients. We get the final rendering as Figure 15f.

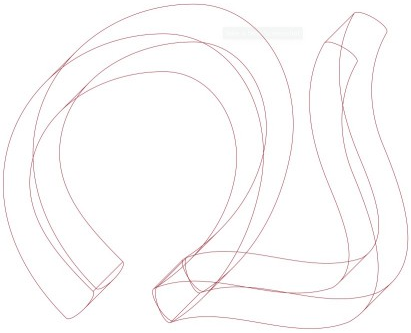
Nupuram Color font offers 18 predefined palettes that users can select. Or a user can specify the colors using CSS, for example. This color font uses 3 colors for its shadow-ish look. They are Dark, Light, and Base colors. Base is the facing color, Light is the central glowing area color. Dark is the color for the shadow part. The colors are used to create an internal gradient.

Nupuram is the first color font in Malayalam.

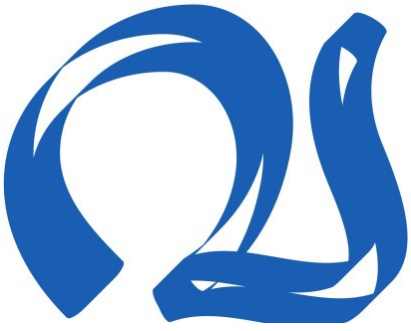
Each glyph in Nupuram color is constructed of two layers. The facing layer is Nupuram Regular. The background layer is constructed using the techniques explained above. Then each of these layered glyphs is compiled into a UFO. For each layer, we define a linear gradient from the top to the middle, then reflect it vertically. By choosing complementary colors for the gradients on the front and background layers, we create the illusion of a 3D object illuminated. This kind of sophisticated glyph manipulation and layer generation was possible thanks to METAPOST's ease of experimentation and variant generation. Drawing each of these glyphs by hand in a GUI environment would be quite tiresome and would take months of effort.



(a) Letter 'va' with pen stroke and outlines



(b) Moving calligraphy nib along the outline



(c) Figure 15c filled with color

(d) Envelope of Figure 15c

(e) Nupuram regular placed on top of Figure 15d

(f) Color gradients applied on Figure 15e. Final rendering

FIG. 15. Construction of Nupuram Color glyph

5.8. Nupuram Dots and Arrows

Since a pen can be any shape that covers a predefined path, one experiment I did in Nupuram was to produce a variant in which the paths are filled with equally spaced dots. We get a dotted font, often used in an educational context, to learn how to write a letter by writing on top of it. We call this variant Nupuram Dots. See Figure 16.

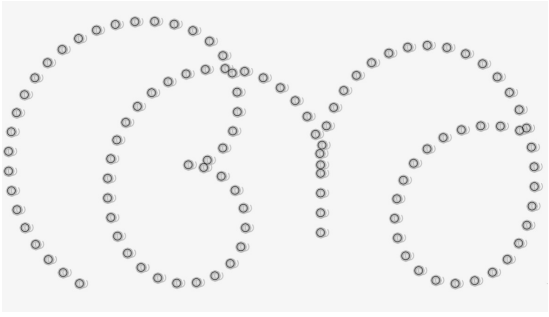


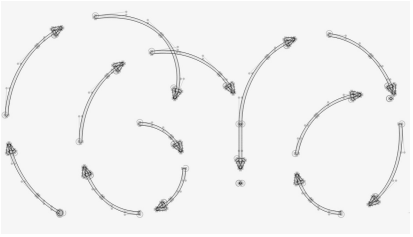
FIG. 16. Nupuram Dots—a font for learning to write

I tried to replace the dots with arrows to indicate the direction of writing. These arrows are equally placed on the pen path. See Figure 17a. To make it more useful, I placed the arrows font on top of the regular font. And gave two colors for each layer, making it a color font. This color can be customized on the user side, as we explained in the section about Nupuram color. See Figure 17b.

Special fonts, such as dot fonts, play a valuable role in facilitating the learning to write letters, especially for young learners and those with fine motor skill challenges. By tracing arrows or connecting the dots, students develop the muscle memory and hand-eye coordination necessary for proper letter formation. Dot fonts are particularly popular in early childhood education, where they help children transition from drawing shapes to writing letters.

Developing a dots or arrow font like this is a full-blown typeface project in the common typeface design approach. Here, we see a quick derivation of an existing typeface.

Optimizing these curves with the curve harmonization techniques was required for aesthetic reasons. Linus Romer's implementation of Curvature combs and harmonization (Romer, 2023) of METAPOST was used for the purpose.



(a) Nupuram Arrows—Showing writing direction

(b) Nupuram arrows color font with default color palette

FIG. 17. Nupuram Arrows

6. Type Design Workflow

A design and development workflow fine-tuned for this kind of type design was also required. It is important to see the results of an METAPOST instantly for fast feedback loops. I integrated METAPOST- based type design into the popular Visual Studio Code so that the designer can work on METAPOST code and live-preview the results with all design utilities, such as a design grid, point coordinates, anchors, and so on. I also prepared an online sandbox for METAPOST so that quick, cheap experiments are possible and sharing and collaboration on design concepts are feasible[18].

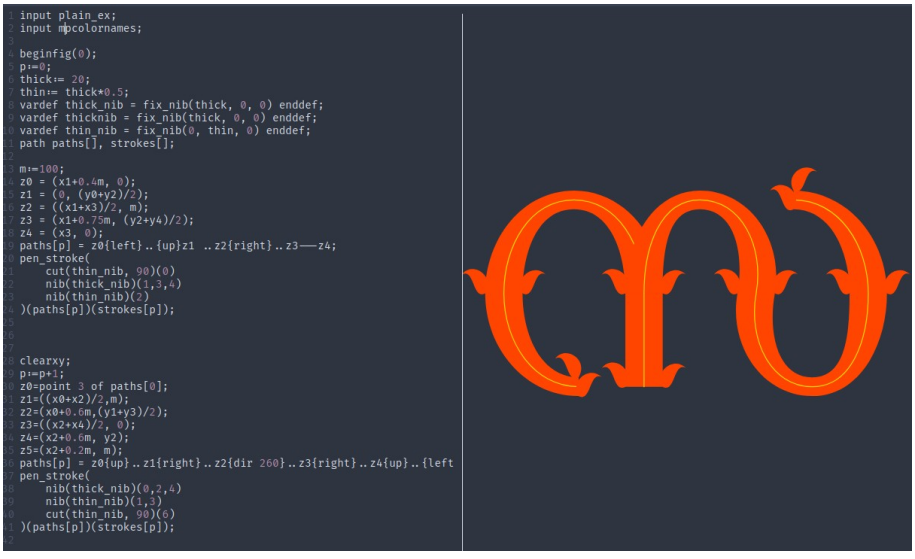
7. Derivatives

The educational typefaces featuring arrows and dots were the first of their kind in Malayalam. I used the SVGs generated by METAPOST code to create a website to learn to write Malayalam[19]. The SVG stroke paths are animatable to simulate a drawing animation. Along with Arrows and Dots variants, and by incorporating example words and pronunciation audio samples, this website has serious users now.

8. Malini Typeface

After Nupuram, I also designed another typeface with the same workflow, a typeface optimized for body text, which has stricter stroke modulation and optical fine-tuning requirements. Malini has 4 variable axes:

18. The metapost sandbox `https://mpost.thottingal.in`.
19. `https://learn.smc.org.in`.



```
input plain_ex;
input mpcolornames;

beginfig(0);
p:=0;
thick:= 20;
thin:= thick*0.5;
vardef thick_nib = fix_nib(thick, 0, 0) enddef;
vardef thicknib = fix_nib(thick, 0, 0) enddef;
vardef thin_nib = fix_nib(0, thin, 0) enddef;
path paths[], strokes[];

m:=100;
z0 = (x1+0.4m, 0);
z1 = (0, (y0+y2)/2);
z2 = ((x1+x3)/2, m);
z3 = (x1+0.75m, (y2+y4)/2);
z4 = (x3, 0);
paths[p] = z0{left}..{up}z1 ..z2{right}..z3--z4;
pen_stroke(
    cut(thin_nib, 90)(0)
    nib(thick_nib)(1,3,4)
    nib(thin_nib)(2)
)(paths[p])(strokes[p]);


clearxy;
p:=p+1;
z0=point 3 of paths[0];
z1=((x0+x2)/2,m);
z2=(x0+0.6m,(y1+y3)/2);
z3=((x2+x4)/2, 0);
z4=(x2+0.6m, y2);
z5=(x2+0.2m, m);
paths[p] = z0{up}..z1{right}..z2{dir 260}..z3{right}..z4{up}..{left
pen_stroke(
    nib(thick_nib)(0,2,4)
    nib(thin_nib)(1,3)
    cut(thin_nib, 90)(6)
)(paths[p])(strokes[p]);
```

FIG. 18. METAPOST code and live preview setup. Glyph designed by the author

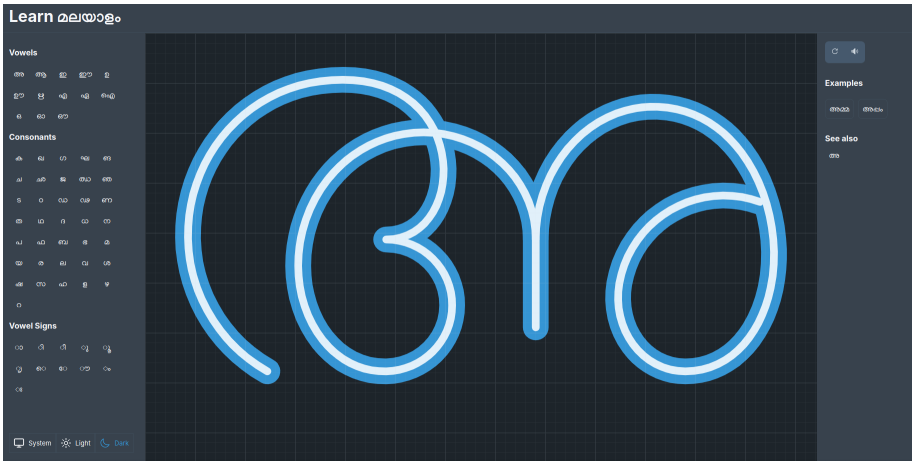


FIG. 19. Educational website to learn how to write letters of Malayalam, created using the same SVGs used for the typeface

Weight, Width, Slant, and Optical Size. Malini is the first typeface in Malayalam script with optical sizing. While Nupuram allowed more freedom in drawing, as it is designed as a display typeface with handwriting-like characteristics, Malini demanded more precise drawing. This chal-

lenge helped fine-tune the workflow again and to retrospectively apply some of the improvements to the Nupuram typeface.

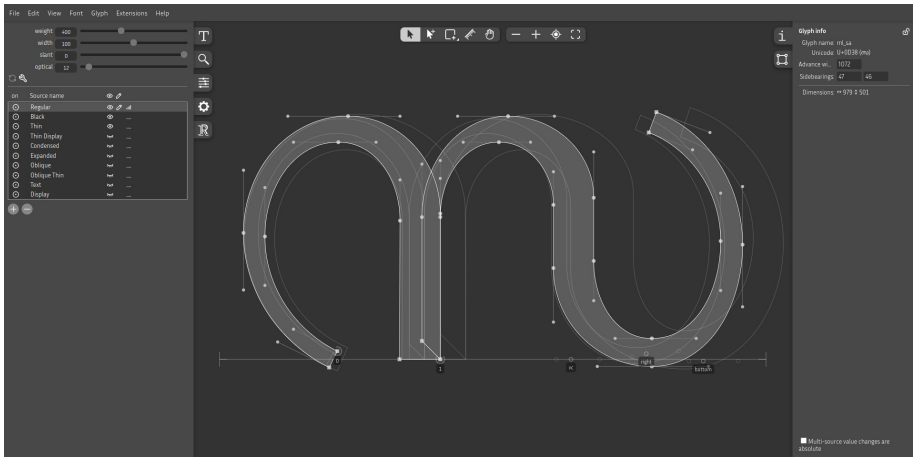


FIG. 20. Malini typeface—Letter Sa of Malayalam showing stroke modulation and various axes

അയാൾ കിണറ്റിലേക്ക് കൂപ്പു കുത്തി. കിണറു കടന്ന് ഉൾ ക്കിണറ്റിലേക്ക്. വെള്ളത്തിന്റെ വില്ലീസു പടുതകളിലൂടെ അയാൾ നീങ്ങി. ചില്ലുവാതിലുകൾ കടന്ന്, സ്വപ്നത്തിലൂടെ സാന്ധ്യപ്രജ്ഞയിലൂടെ, തന്നെ കൈനീട്ടി വിളിച്ച പൊരുളി ന്റെ നേർക്ക് അയാൾ യാത്രയായി. അയാൾക്കു പിന്നിൽ ചില്ലുവാതിലുകൾ ഒന്നൊന്നായടഞ്ഞു.

FIG. 21. Malini sample text rendering

9. Lessons

The parametric approach helped me to do easy experimentation. The traditional type design approach is inflexible once the key typographic metrics are set. In the METAPOST based approach, fine-tuning these parameters is possible at any stage of the project. It is very easy to experiment with different parameters and pick one that is satisfactory to the designer. Sometimes these experiments also reveal surprising results. The approach for a 3D-like color font was born from such an experiment.

Leveraging my background as a software engineer and my adeptness with source code, I found that applying this familiar paradigm to type design significantly enhanced productivity. Shapes were manipulated as modularized methods within the source code, and this modular approach proved instrumental in generating design outputs that were not only consistent but also coherent. This marked a notable departure from the challenges I had encountered in traditional typeface design projects, where maintaining consistency and cohesion had often been a struggle.

Producing a superfamily of Malayalam typefaces with multiple variable and color fonts can easily take multiple years of effort and the collaboration of multiple individuals. However, the majority of Nupuram's work was finished in my free time in 6 months.

Capturing the details of all required glyphs and their OpenType features in the program was not easy, but it is an investment for future fonts. I could reuse most of the METAPOST code and the framework in the Malini typeface project[20], even though that is a typeface optimized for body text.

The ability to define the design precisely in mathematical language will guide the designer to stringent mathematical proportions for letter forms. Nevertheless, it is essential to recognize that such precision does not inherently guarantee the production of aesthetically pleasing glyphs. This tension between mathematical precision and aesthetic appeal emerged as a recurring theme throughout my type design process. For example, the programmer's inclination to align an element exactly at the 0.5 mark of the entire width versus the more subjective choice of proportions that align with visual harmony and appeal.

The ability to swiftly generate variations and, consequently, produce variable fonts proved to be a significant advantage in my work. As an advocate of free and open-source principles, my objective was to construct the entire type design workflow and develop typefaces without relying on proprietary software. Notably, at the time, there was a lack of reliable free software tools for typeface design tailored to variable font creation.

20. Malini is a new typeface project that the author is working on with METAPOST based workflow `https://github.com/smc/malini`.

I overcame that limitation with this new workflow. Nupuram is one of the early color fonts following the publication of the OpenType Color 1 specification. It is worth noting that support for variable and color fonts has yet to catch up in common software applications.

It is tempting to proliferate numerous subfamilies from a single typeface because it is easy to do so. In this regard, it is pertinent to recall Knuth's cautionary advice regarding this issue. In the Nupuram project, driven by a desire to explore the myriad possibilities, I created numerous subfamilies. Nevertheless, it remains a verifiable fact that an abundance of variations does not necessarily equate to an abundance of genuinely useful fonts.

I must acknowledge that METAPOST is not a programming language one can easily learn. It diverges significantly from the conventions of popular programming languages, presenting a distinctive learning curve. While valuable documentation resources do exist, there is a noticeable dearth of practical examples that designers can readily reference and learn from. Given this challenge, prior to advocating for the adoption of this workflow among fellow type designers, I recognized the need to address this limitation. So I documented Nupuram extensively. The project is published under a free and open source license for anybody to refer to and learn[21]. I created a simple METAPOST playground website `https://mpost.thottingal.in` where people can quickly write METAPOST code and preview the result. I started a repository of various type design concepts illustrated using METAPOST[22].

Upon embarking on this exploration with METAPOST, I encountered several articles that explain why METAPOST or METAFONT did not catch up. A recurring theme in these discussions was the debate over whether an arbitrary type design can be expressed using the pen-based approach, in which outlines are essentially the outputs of pen strokes. It is undeniable that outline-based design has established itself as a tried-and-true, successful design methodology, and this criticism is indeed valid. However, I contend that there is no inherent necessity for the exclusive use of pen-based design in every aspect of glyph creation. Whether it manifests as a stroke path or an outline, it essentially boils down to an array of coordinates from my perspective. The manipulation of arrays is a skill that I have honed as a programmer. For example, the serif shape for a Latin letter cannot be created using the outlines of a pen. But nothing prevents an METAPOST programmer from quickly drawing that outline without a pen definition.

---

21. Nupuram source code repository: `https://github.com/smc/nupuram`. Licensed under SIL Open Font License, Version 1.1.

22. Typeface design concepts illustrated using METAPOST `https://github.com/santhoshtr/type-concepts`.

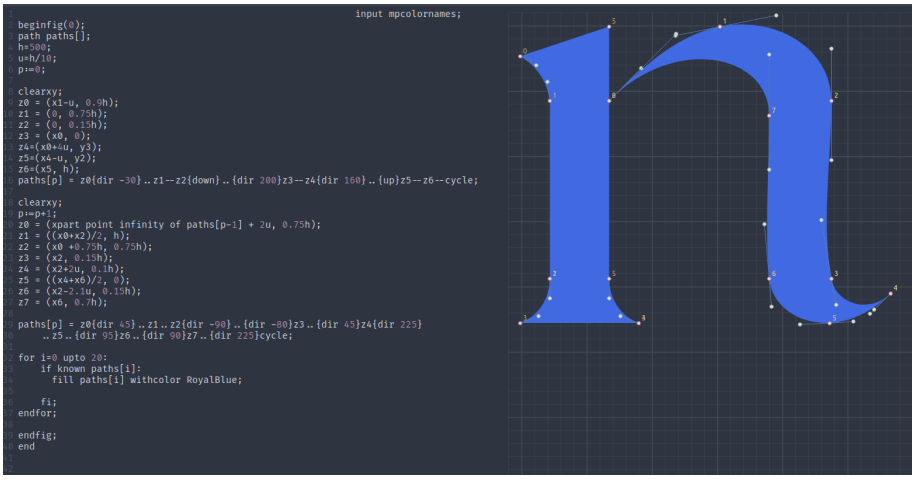


FIG. 22. Letter 'n' drawn using METAPOST using outline technique, without using any pens

10. Conclusion

In this paper, I have provided an outline of the parametric type design methodology, using the Nupuram typeface as an illustrative example. I also elaborated on the lessons learned. While this new approach enabled me to create a typeface with many features hitherto deemed unattainable, it is important to underscore that I am not proposing it as a replacement for the existing type design workflow. The approach outlined here is suitable for designers proficient in software engineering. As typefaces are becoming sophisticated software these days, I am hopeful that this approach will find utility among a broader spectrum of designers. I aspire to see the Nupuram and Malini typefaces, born from this exploration, become valuable assets for a diverse community of users.

References

Constable, Peter (n.d.[a]). *COLR—Color Table (OpenType 1.9.1)—Typography — learn.microsoft.com*. https://learn.microsoft.com/en-us/typography/opentype/spec/colr. [Accessed 08-06-2024].

——— (n.d.[b]). *OpenType Font Variations overview (OpenType 1.9.1)—Typography — learn.microsoft.com*. https://learn.microsoft.com/en-us/typography/opentype/spec/otvaroverview. [Accessed 08-06-2024].

Haralambous, Yannis (2007). *Fonts & Encodings. From Advanced Typography to Unicode and Everything in Between*. Sebastopol, CA: O'Reilly, pp. xx+1018.

Hellingman, Jeroen (1998). “Malayalam fonts”. In: *CTAN: language/malayalam*.

Hobby, John D. (1986a). “Smooth, easy to compute interpolating splines”. In: *Discrete and Computational Geometry* 1, pp. 123–140. DOI: 10.1007/BF02187690.

———— (1986b). “Smooth, easy to compute interpolating splines”. In: *Discrete & computational geometry* 1.2, pp. 123–140.

———— (1994). “A User’s Manual for METAPOST. AT&T Bell Laboratories”. In: *Computing Science Technical Report* 162.

Hudson, John (n.d.). *Introducing OpenType Variable Fonts — medium.com*. https://tinyurl.com/323xypke. [Accessed 08-06-2024].

Jackowski, Bogusław and Janusz M. Nowacki (2003). “Latin Modern: Enhancing Computer Modern with accents, accents, accents”. In: *TUGBoat: Proceedings of the 2003 Annual Meeting* 24.

Jackowski, Bogusław, Janusz M. Nowacki, and Piotr Strzelczyk (2001). “MetaType1: A MetaPost-based engine for generating Type 1 fonts”. In: *Proc. of EuroTEX*, pp. 111–119.

Knuth, Donald E. (1979). *TEX and METAFONT: New Directions in Typesetting*. USA: American Mathematical Society. ISBN: 0932376029.

———— (1985). “Lessons learned from METAFONT”. In: *Visible Language* 19.1, p. 35.

———— (1989). *The METAFONTbook*. Addison-Wesley Longman Publishing Co., Inc. Chap. 16, p. 150.

Romer, Linus (2023). “Curvature combs and harmonized paths in MetaPost”. In: *TUGboat* 44, pp. 236–239. DOI: 10.47397/tb/44-2/tb137romer-curvetools.