TERKA: Enabling Tamil script rendering in .NET Micro Framework

Jan Kučera[1,2], Vladimír Mach[1], Dominik Škoda[1], Matěj Zábský[1]

[1] Faculty of Mathematics and Physics
Charles University, Czech Republic

[2] Department of South and Central Asia
Charles University, Czech Republic

jan.kucera@matfyz.cz, vladimir@mach.im, dominik.skoda@gmail.com, matej@ihvar.cz

Abstract

In this paper, we present our contribution to the .NET Micro Framework that enables complex script rendering on small, embedded devices, and we provide a reference implementation for rendering the Tamil script. We also discuss the design of Tamil bitmap fonts and suggest the smallest proportional font possible.

Keywords: embedded devices, text rendering, bitmap fonts, Tamil engine

Introduction and Overview

Microsoft .NET Micro Framework is an open source platform for developing small, resource constrained devices, that emerged in Microsoft Research over 10 years ago. It allows developers to leverage their existing knowledge of managed development and powerful toolset to develop and debug embedded products rapidly.

On one side, hardware platforms like .NET Gadgeteer that build on top of the .NET Micro Framework are used for rapid prototyping in commercial research [1] or at secondary schools and universities for teaching students the basic principles of software development, hardware engineering and industrial design [2]. On the other side, .NET Micro Framework is becoming one of the technology backing up the Internet of Things development [3].

The framework comes with simple text rendering engine that uses highly optimized bitmap font format with no support for complex scripts, glyph shaping, text directionality or extended Unicode planes, effectively limiting the number of scripts and languages the .NET Micro Framework can render.

Our Rendering Engine

We present a new, state-machine based rendering engine, which is an enhanced Turing machine working on a tape of glyphs, as an in-place replacement for the engine currently in use. Backward compatibility and minimum resources requirements were our main goals when creating the engine.

First, we extend the current *Tiny Font* file format to support arbitrary number of appendix blocks where all additional data such as language specific instructions can be stored. Each Unicode plane is stored in separate appendix with the same format of characters in the Basic Multilingual Plane. Glyphs that directly map 1:1 to characters are stored as the respective characters themselves; additional glyphs are stored in the Supplemental Private Use Area planes. This has the advantage of avoiding large character-to-glyph mapping tables.

The virtual tape of glyphs is then populated from the user's string and machine rules are executed on the tape. Rules can be grouped into *features*, usually corresponding to OpenType features. Developers can turn on and off individual features during runtime, enabling fine typography as a by-product of complex script rendering, to our knowledge the first time on devices targeting such small amount of resources.

Each rule can be expressed as a function

$$\rho : Q \times C \times P_c \times (t + o_c) \rightarrow Q \times A \times P_a \times (t + o_a) \times o_t$$

where

- Q is a finite set of states, state $q \in Q$;
- C is a finite set of conditions, condition $c \in C$;
- P_c is a finite set of parameters which condition $c \in C$ accepts, $c \times p_c \rightarrow \{0,1\}$;
- t is the current position on the tape;
- o_c is a relative offset to t at which the condition $c \in C$ is evaluated;
- A is a finite set of actions, $a \in A$;
- P_a is a finite set of parameters which action $a \in A$ accepts;
- o_a is a relative offset to t at which the action $a \in A$ is performed;
- o_t is a number of glyphs to advance the tape with, $t \xrightarrow{\rho} t + o_t$.

Our OpenType Compiler

Since .NET Micro Framework is designed to run on devices with very small amount of memory and computational power, we have designed the engine so that all information required to render given script must be present in the font file. To save developers and font designers considerable amount of work, we also present an OpenType compiler that analyses the OpenType tables of standard fonts and converts them to the rendering engine state machines.

First, the OpenType tables (GSUB and GPOS) are parsed and used to construct a state-machine representation of the instructions, which is then optimized by merging equivalent states into a single state and normalized to minimize space requirements of the final state-machine.

Enabling Tamil Script

Some rendering systems, like Graphite or Apple Advanced Typography rely on shaping instructions to be present in the font files themselves, while the rendering systems on Windows platforms, like Uniscribe or DirectWrite, traditionally supply the language knowledge needed to render particular script themselves.

To ensure maximum flexibility in the constrained .NET Micro Framework environment and to prevent useless waste of memory and computation power for applications that do not require rendering all languages at once, we have decided that the language knowledge must be present in the font files.

Inspired by the work of Nedumaran [4], we have designed a state-machine representing the language knowledge needed to render the Tamil script (see Figure 1). Each state transition can be represented by a single rule, which gives us the total number of 7 rules needed for the Tamil language, operating on a group of consonants (including the conjunct consonant க்ஷ which participates in glyph reordering) and vowel marks.

Note that the ஸ்ரீ ligature does not need to be included in the rules, as it does not form syllables with other vowels. Both க்ஷ and ஸ்ரீ ligatures as well as all other ligatures (puḷḷi, i, i□, u, u□) are already part of the OpenType tables and will be automatically included by the compiler discussed above.

The 7 language rules for Tamil are:

௧. (Entry, *consonant*) →
(SawConsonant, −, +1)

௨. (SawConsonant, ொ) →
(InsertingAA, *rewrite* ொ *to* ெ, +1)

௩. (SawConsonant, ோ) →
(InsertingAA, *rewrite* ோ *to* ே, +1)

௪. (SawConsonant, ௌ) →
(InsertingAU, *rewrite* ௌ *to* ெ, +1)

௫. (SawConsonant, *vowel*) →
(Entry, *reorder* Axto xA, +2)

௬. (InsertingAA, −) →
(SawConsonant, *insert* ா, −1)

௭. (InsertingAU, −) →
(SawConsonant, *insert* ௗ, −1)

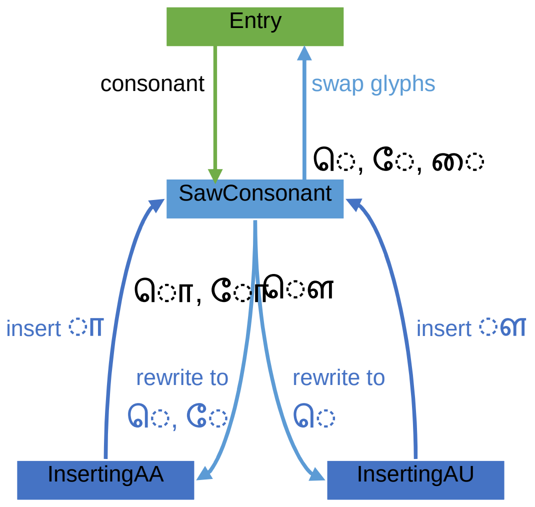


Figure 1. Tamil script state machine

Let's demonstrate how the rendering engine works on the example of கொ. The glyph tape is initialized as follows:

க	ொ		
▲			

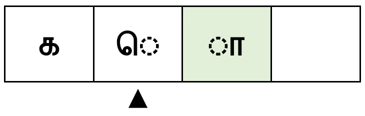
The state-machine begins operation in the *Entry* state, and reaches the consonant க. The rule ௧ applies, machine switches to the *SawConsonant* state and advances the tape by one.

க	ொ		
	▲		

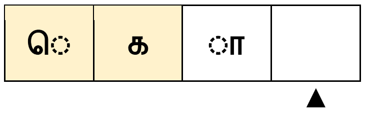
At this position, the ொ mark is seen and rule ௨ applies. The ொ mark is rewritten to the ெ mark, state switched to *InsertingAA* and tape advanced by one.

க	ெ		
		▲	

This time the unconditional rule ௭ applies, the mark ா is inserted, the machine is switched back to *SawConsonant* state and tape advanced backwards.



Here the only rule that matches is the rule ௫, which reorders the two glyphs and moves the head again at the end of the tape.



After all glyphs are processed, they are ready to be rendered on the screen. Other consonant and vowel combinations work in a similar fashion.

These language rules are supposed to be packed into a feature, applied after all other OpenType shaping features. For Tamil, the standard order of features is documented on Microsoft Typography website[1].

Bitmap Fonts

Less constrained devices and computers deal with vector fonts easily. On the other side, small microprocessors that work with bitmaps rarely deal with Unicode strings, not mentioning supporting scripts like Tamil, and that resulted in lower interest in Tamil bitmap fonts.

The Latin alphabet and Tamil letters have different layout statistics and therefore cannot fulfil the same expectations when designing text rendering systems and displays. For example, the most common displays used to render Latin text have 5×7 dots per character, but even 3×5 is readable.[2] LED displays with various number of segments can be used to display not only numbers but also text.

Can we do the same for Tamil? Can we use the common dot matrix displays to render Tamil script? What are actually the minimum size requirements of Tamil script? Can we design reasonable fixed-width Tamil font? We would like to bring attention of engineers and font designers to these questions and provide some initial research in this area.

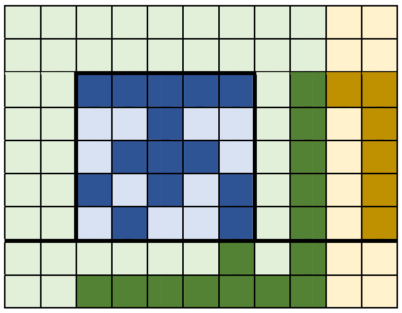


Figure 2. Minimum resolution for Tamil glyphs: 11×9 pixels

We are presenting a proportional Tamil font with the kernel of 5×5 pixels on the baseline, surrounded by standard zone of 2 pixels for vowel modifiers and extended zone of additional 2 pixels to the right as illustrated in Figure 2, needed only for the depicted ū variation. We tried to follow the recommendations of M.M. Manivannan [5] as much as possible.

The glyph with the finest structure horizontally — ணூ — requires 6 vertical lines and similarly த் requires 5 horizontal lines, which defines the minimum required resolution to avoid ink bleeding. Moreover, all consonants except ண fit into the 5×5 kernel, which improves readability. The full letter chart is shown in Figure 3.

Our rendering engine is the first platform that supports glyph substitutions and reordering while using bitmap fonts, so new skills might need to be developed for designing fonts for the platform. For smallest font sizes, supplying combined glyph bitmaps might save more space than manipulating glyphs using rules. The larger the font is the more beneficial is to reuse

[1] http://www.microsoft.com/typography/otfntdev/tamilot/features.htm
[2] See Windows Console font settings or http://robey.lag.net/2010/01/23/tiny-monospace-font.html

glyphs (and even their position) as much as possible, which is fortunately in the nature of the Tamil script. For example, Figure 4 shows reused ī modifier glyphs in the font.

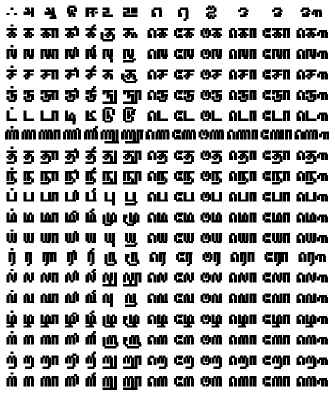


Figure 3. Letter chart 11×9 pixels

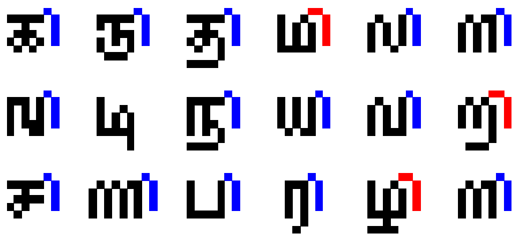


Figure 4. Glyph reuse in ī syllables

Conclusion and resources

The Tamil script is the first complex script implemented for our open-sourced .NET Micro Framework rendering engine, allowing engineers to create new Tamil-enabled devices, or to power existing off-the-shelf displays to render Tamil text directly from Unicode encoding. This can return Tamil to people on public places and further support the adoption of Unicode.

Handcrafted fonts always bring superior experience to those automatically rasterized, so we started designing our own bitmap fonts and welcome both comments and other designers to help us achieve pixel-perfect Tamil typography.

For full technical documentation and downloads, visit http://terka.microframework.cz/.

References

[1] N. Villar, J. Scott, S. Hodges, K. Hammil and C. Miller, ".NET Gadgeteer: A Platform for Custom Devices", in *Proceedings of Pervasive 2012*, Heidelberg, 2012.

[2] S. Hodges, J. Scott, S. Sentance, C. Miller, N. Villar, S. Schwiderski-Grosche, K. Hammil and S. Johnston, ".NET Gadgeteer: A New Platform for K-12 Computer Science Education", in *SIGCSE '13 Proceedings of the 44th ACM Technical Symposium on Computer Science Education*, New York, 2013.

[3] R. Gangwar, "Internet of Things (IoT) on Microsoft", Microsoft Corporation, 25 May 2014. [Online]. Available: http://blogs.msdn.com/b/rahul/archive/2014/05/25/internet-of-things-iot-on-microsoft.aspx. [Accessed 1 July 2014].

[4] M. Nedumaran, "Building Tamil Unicode Fonts for Mac OS X", in *Tamil Internet 2009 Conference Papers*, Cologne, 2009.

[5] M.M. Manivannan, "Notes on Tamil Orthography – puLLi, kAl, ai, ja, etc.", in *Tamil Internet 2013 Conference Papers*, Kuala Lumpur, 2013.