Text Layout Engine

-Susmit

28 Feb 2017

OVERVIEW & PURPOSE

To build a text layout engines for kivy (pango,harfbuzz) to enable her for reshaping for alphabets such as Arabic,Persian,Thai and Devanagri .

What is Text Layout Engine?

- TLE identifies the script that user is interested in and display the text accordingly.
- https://en.wikipedia.org/wiki/Complex_text_layout

Complex text layout complexities:-

1. **Bidirectionality : characters can either appear right to left or left to right in a line.**
    - Visual order differs from storage order.
    - Ex Arabic,Herbrew read read left to right
    - Bidirectional Text
2. **Shaping :where a character may change its shape ,dependent on its location and/or surrounding characters.**
    - Arabic character shapes change to connect adjacent characters.
    - A character in Arabic script can have as many as four different shape-forms,depending on context
3. **Ligatures:mandatory special forms, and no unicode equivalent**
    - Arabic and devanagari represent some character sequence with ligatures.
4. **Positioning: adjustment vertical ,horizontal**

- Thai and other script require characters to reposition.

5. **Reordering:character positions depend on context.**
    - Some hindi characters reorder based on context

6. **Split Character: Some character appear in more than one position.**
    - Thai and many indic languages display a single character in multiple positions.

Layout Engine Working Overview

- The font for a particular script contains rules.
- Two main categories called GPOS(glyph positioning) and GSUB(glyph substitution)
- There are features like “ccmp”(composition and decomposition), “blws”(below base substitution) etc falling under GSUB rule.Other features like “blwn” (below base mark positioning),”abvm” (above base mark positioning) “kern” etc .fall under GPOS rule.
- The fonts may contain language tags for the languages they support.
- All combinations of characters used by particular languages are accessed by rules or lookups defined in the fonts.
- The rendering engine has to identify the script, select the fonts, apply correct rules from the fonts and display it.

Layout Engine Working

- User input is stored in a buffer/memory.
- Identify a script by looking at the Unicode values in the buffer.

- Determine the bidirectional levels for the text.
- Update the language tag using information.
- Determine a language engine from the updated language tag and script.
- Determine a set of possible fonts from the updated language tag and the font properties for the character. These fonts are sorted according to how well they match the language tag and font properties.
- Apply the rules defined in the font to the Unicode values stored in the buffer.
- Do character, word, line boundary analysis.
- The output of this process is usually per line. These are then fed into the renderer.

Pango

- The basic job of Pango is to take Unicode text, possibly annotated with extra attributes and convert that into appropriately positioned glyphs selected from the fonts on the system.
- In some cases, the user uses routines from Pango directly to render the positioned glyphs; in other cases Pango is used as part of a larger system which uses the positioned glyphs
- From the point of view of the application programmer, Pango looks quite simple. There is a layout object, PangoLayout that holds one or more paragraphs of Unicode text.Once a PangoLayout is created, the application can determine the size of the text, or render it to an output device.

Pango Rendering Pipeline

Beneath the high-level API shown above that deals with paragraphs of text, there is a low level API that exposes the details of Pango's layout pipeline. The steps in this pipeline are:

• Itemization: Input text is broken into a series of segments with unique font, shape engine, language engine, and set of extra attributes.

• Text boundary determination: The text is analyzed to determine logical attributes such as possible line break positions, and word and sentence boundaries.

• Shaping: each item of text is passed to the shape engine for conversion into glyphs.

• Line breaking: the shaped runs are grouped into lines; if the individual runs are longer than particular lines, then they are broken at line break positions,

Harfbuzz

- HarfBuzz evolved from code that was originally part of the FreeType project. It was then developed separately in Qt and Pango. Then it was merged back into a common repository
- HarfBuzz only does shaping. That is, hb_shape() is functionally equivalent to pango_shape()
- Currently used in latest versions of Firefox, GNOME, ChromeOS, Chrome, LibreOffice, XeTeX, Android, and KDE, among other places.
- sample_example

Harfbuzz Limitation

- It does not provide an itemizer
- It doesn't provide Unicode Bidirectional Algorithm Implementation
- It neither provides Line Breaking implementation.
- No Glyph rasterization
- Glyph metrics information

नमस्ते (Hi! in Hindi :P)