BRAC UNIVERSITY

Complex Text Rendering

Submitted By:
Saad Bin Mahbub(07201002)
Md. Ashraf-ul-Hauque(08101019)
Imran Kader (08101008)

Submission Date:12th April, 2012

Complex Text Rendering

Submitted By:

Saad Bin Mahbub(07201002)

Md. Ashraf-ul-Hauque(08101019)

Imran Kader (08101008)

Supervisor:

Prof. Dr. Mumit Khan

Department of Computer Science

And Engineering (CSE),

BRAC University, Dhaka

_____

(signature)

Complex Text Rendering on Android

Table of content:

1. Introduction---------- 4
2. Background Information---------- 4
2.1 Complex TextRendering----------4
2.2 BriefHistory of Fonts---------- 4
2.3 Basic shaping forms---------- 5
3. Libraries----------8
3.1 ICU----------8
3.2 SKIA---------- ----------9
3.3 Harfbuzz---------- -10
4. Shaping with ICU LayoutEngine ---------- 10
5 Methodology----------11
5.1 Setting up a Linux build environment----------11
5.2 Download the Android source----------

-----11
5.3 Compile the code----------11
5.4 Bengali on Android----------12
5.5 Software required on your computer----------12
5.6 Modifying the font----------12
5.7 Generating the substitution code for the font----------14
5.8 Recompile android source code----------14
6. Conclusion----------15
7. References----------15

1. Introduction:

Android is a fairly new operating system for smart-phones. The Availability and demand for android phones in the market is increasing by the day. This availability and demand for android mobile phones has also reached Bangladesh. But unfortunately Android does not support Bengali since it is a complex script. So far android supports mostly the Latin languages which are rendered only by linear progression, on the other hand complex scripts need special processing.

2. Background information:

2.1 Complex Text Rendering:

The Latin script, which is the most commonly used script among software developers, is also the least complex script to display especially when it is used to write English. Using the Latin script, characters can be displayed from left to right in the order that they are stored in memory. Some scripts require rendering behavior that is more complicated than the Latin script. We refer to these scripts as "complex scripts" and to text written in these scripts as "complex text." Examples of complex scripts are the Indic scripts (for example, Devanagari, Tamil, Bengali, and Gujarati), Thai, and Arabic.

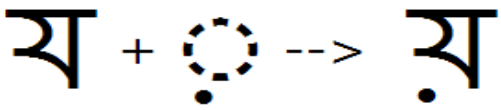
2.2 Brief History of Fonts:

Fonts were a collection of glyphs and a simple one-to-one mapping between characters and glyphs. Preliminary support for simple ligatures was available in some font formats. When Unicode was introduced there was a need for formats allowing complex transformations of glyphs(substitution and positioning). Two technologies were established to solve the problem OpenType Layout from Microsoft and Adobe and AAT from Apple. These to technologies and the Truetype and Type1 font formats were put together to form OpenType.

2.3 Basic shaping forms:

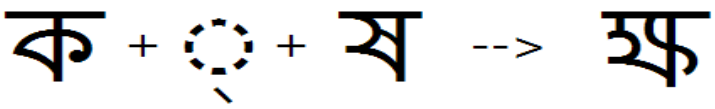
Nukta:

Nukta feature used to substitute pre-composed glyph (Yya) for Ya + Nukta:



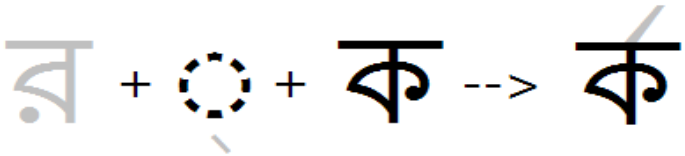
Akhand:

Ka + halant + Ssa is substituted with the KaSsa ligature:



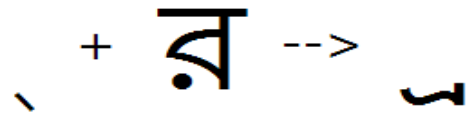
Reph

Reph feature substitutes the mark glyph form of Ra. After final reordering, positioning is adjusted in the 'abvm' GPOS feature:

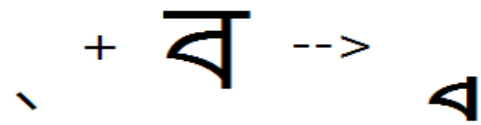


Below form of consonant

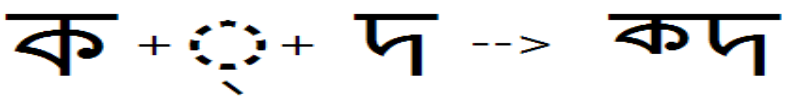
Halant + Ra (preceded by a consonant which does not form a ligature) substitutes a below-base Ra:



Halant + Ba (preceded by a consonant which does not form a ligature) substitutes a below-base Ba:



Half form of consonant



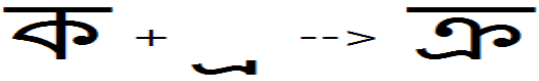
Half feature substitutes half form of Ka:

Post-base form of consonant

The post-base form of the Ya is substituted, when it is the last consonant in a syllable:

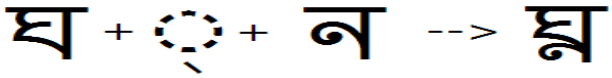
Vattu variants

The ‘vatu’ feature used to substitute a ligature of Ka + below-base Ra:

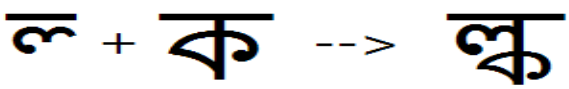


Conjunct forms

The ‘cjct’ feature used to substitute a ligature for Gha + Na:



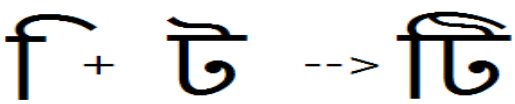
Pre-base substitutions



Example 3 - half La + full Ka is substituted by the LaKa conjunct:

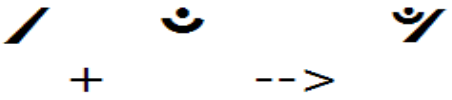
Example 4 - half Ka + half Ssa + full Nna is substituted by the KaSsaNna ligature:

Example 5 - The 'pres' feature is also used to substitute ligatures with the I-Matra:

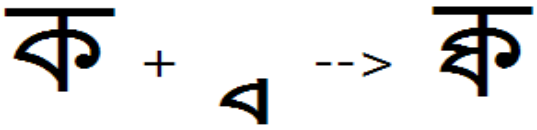


Above-base substitutions:

The 'abvs' feature used to substitute a ligature for the reph + candrabindu glyphs:



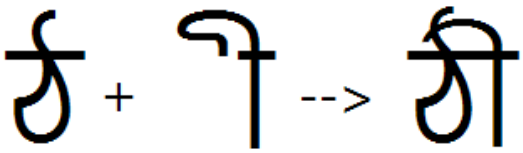
Below-base substitutions:



Example 1- 'blws' substitution for base + below-base conjuncts:

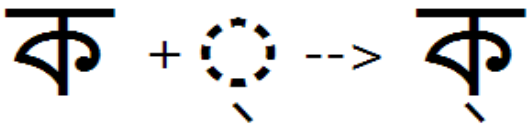
Post-base substitutions

Example 1- 'psts' used to substitute conjunct of base and post-base matra:



Halant form of consonants

Example 1 - 'haln' feature used to substitute halant form of base glyph:



3. Libraries:

Libraries we worked with in our thesis are listed below:

3.1 ICU: ICU is a mature, widely used set of C/C++ and Java libraries providing Unicode and Globalization support for software applications. ICU is widely portable and gives applications the same results on all platforms and between C/C++ and Java software.

Some of the services that it provides are the following.

Text: Unicode text handling, full character properties and character set conversions

Analysis: Unicode regular expressions; full Unicode sets; character, word and line boundaries

Comparison: Language sensitive collation and searching

Transformations: normalization, upper/lowercase, script transliterations

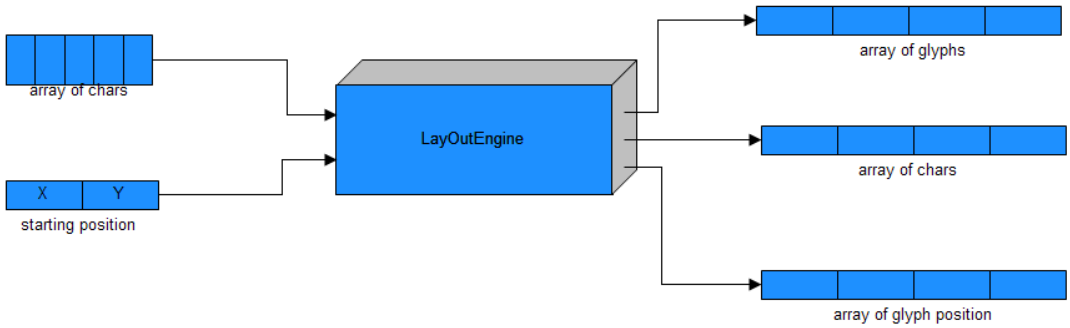
Locales: Comprehensive locale data and resource bundle architecture, via the Common Locale Data Repository

Complex Text Layout: Arabic, Hebrew, Indic and Thai

Time: Multi-calendar and time zone

Formatting and Parsing: dates, times, numbers, currencies, messages and rule based.

Android OS uses ICU4c version 4.2.1. It supports complex text layout for Arabic, Hebrew, Indic and Thai scripts. Bengali is member of Indic Script Family. ▪ LayoutEngine.cpp ▪ is the virtual base class used to do complex text layout. "IndicLayoutEngine", "IndicClassTables", "IndicRearrangement", "IndicReordering" - these files set relues for Indic Script.



LayOut Processing:

- text must be in a single font, script, and language
- instance of a LayOutEngine is created by calling "layoutEngineFactory(fonts,script,language)"
- Fonts are identified by instances of the "LEFontInstance" class.
- Script and language codes are identified from "ScriptAndLanuageTags"

3.2 SKIA:

Skia is a graphics engine which is used in android for various graphical purposes. It is one of many external libraries used by the Andriod OS. What interests us about SKIA is that it provides back-end support for fonts. Skia has its own portable cache for drawing text, but the actual creation of individual glyph images and metrics are left to the porting layer. There are two classes that must be supplied by the port:

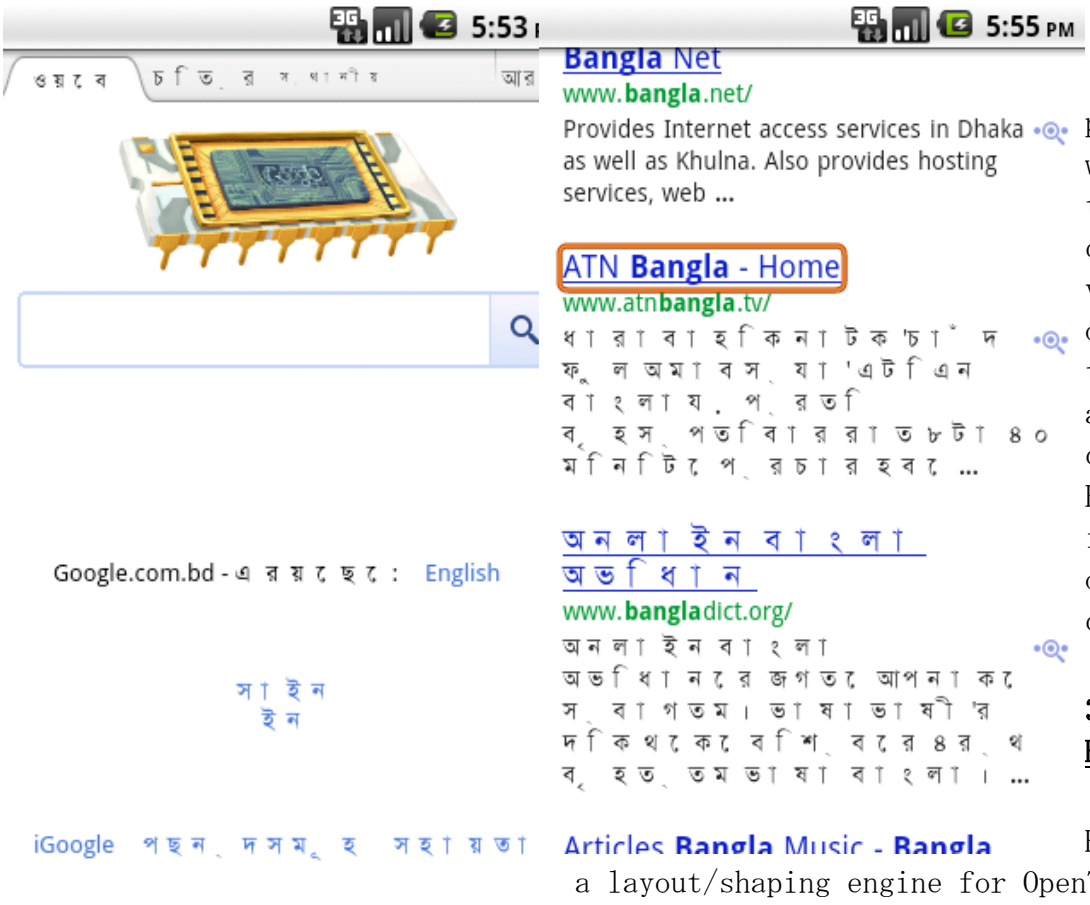
- **SkFontHost** - this is a collection of static methods that are responsible for finding font files by name, and for providing read-access to them, most for the scaler context
- **SkScalerContext** - this is a class that is instantiated for each font *strike*, that is each combination of a typeface+pointsize+matrix+flags.

Adding Bangla Font:

To add Bangla font in our Android source code we located the folder *android-os/frameworks/base/data/fonts* that contains the font and copied the font there. The font used in this case was the **SolaimanLipi** which is an Open Type font. We then added the name of the font in the android MK file. This was done so that the system recognizes the while building and makes a copy of it in the system's fonts folder. Now the **gSystemFonts[]** array in the **SKFontHost_android** was edited to include the name of our font:

```
static const FontInitRec gSystemFonts[] = {
    { "DroidSans.ttf",              gSansNames  },
    { "DroidSans-Bold.ttf",         NULL        },
    { "DroidSerif-Regular.ttf",     gSerifNames },
    { "DroidSerif-Bold.ttf",        NULL        },
    { "DroidSerif-Italic.ttf",      NULL        },
    { "DroidSerif-BoldItalic.ttf",  NULL        },
    { "DroidSansMono.ttf",          gMonoNames  },
    /*  These are optional, and can be ignored if not found in the file system.
        These are appended to gFallbackFonts[] as they are seen, so we list
        them in the order we want them to be accessed by NextLogicalFont().
     */
    { "DroidSansArabic.ttf",        gFBNames    },
    { "DroidSansHebrew.ttf",        gFBNames    },
    { "DroidSansThai.ttf",          gFBNames    },
    { "DroidSansJapanese.ttf",      gFBNames    },
    { "DroidSansFallback.ttf",      gFBNames    },
    { "DroidSansBengali.ttf",       gFBNames    }
};
```

And the results were as follows:



The problem with the text is clearly visible the order of the letters are not correct. Fixing this issue is our primary concern.

3.3 Harfbuzz:

Harfbuzz is a layout/shaping engine for OpenType fonts. Its purpose is to standardize text layout in FOSS(Free and Open Source Software) hence it is the main component of modern GNU/Linux text rendering.

4. Shaping with ICU LayoutEngine:

The core idea behind shaping with ICU's layout engine is to create a Font Instance. The layout engine of ICU is platform idependent. However LEFontInstance is fully platform dependent, meaning the font instance should be implemented by the software engineer uses the ICU.

The font instance will select, read and extract data from font. However implementing the Font instance is very complicated.

5. Methodology:

5.1 Setting up a Linux build environment:

apt-get install bison g++-multilib libc6-dev-i386 lib32ncurses5-dev zlib1g-dev flex ia32-libs x11proto-core-dev libx11-dev

lib32readline5-dev lib32z1-dev gperf gnupg valgrind zip sun-java6-jdk sun-java6-jre sun-java6-bin make git

5.2 Download the Android source code

Google provides a tool called repo to download the source code. Download repo from

https://code.google.com/p/git-repo/downloads/list

chmod +x repo

./repo init -u git://android.git.kernel.org/platform/manifest.git

-b <branch>

./repo-1.13 sync-j4

Where <branch> is the android version you want, eg froyo, etc. To check the list of branches

available, use git as follows

git clone git://android.git.kernel.org/platform/manifest.git

cd manifest

git branch -r

5.3 Compile the code

Change to the directory you downloaded the code to, and run

. build/envsetup.sh

lunch full-eng

make -j4

5.4 Bengali on Android

Android does not support complex scripts, including Bengali and other Indic languages. Here the process of recompiling the skia graphics library to support the following is described:

1. Reordering of the Bengali characters

2. Support for the substitution table in the Bengali font

5.5 Software required on your computer:

To get required softwares, run this command in terminal,

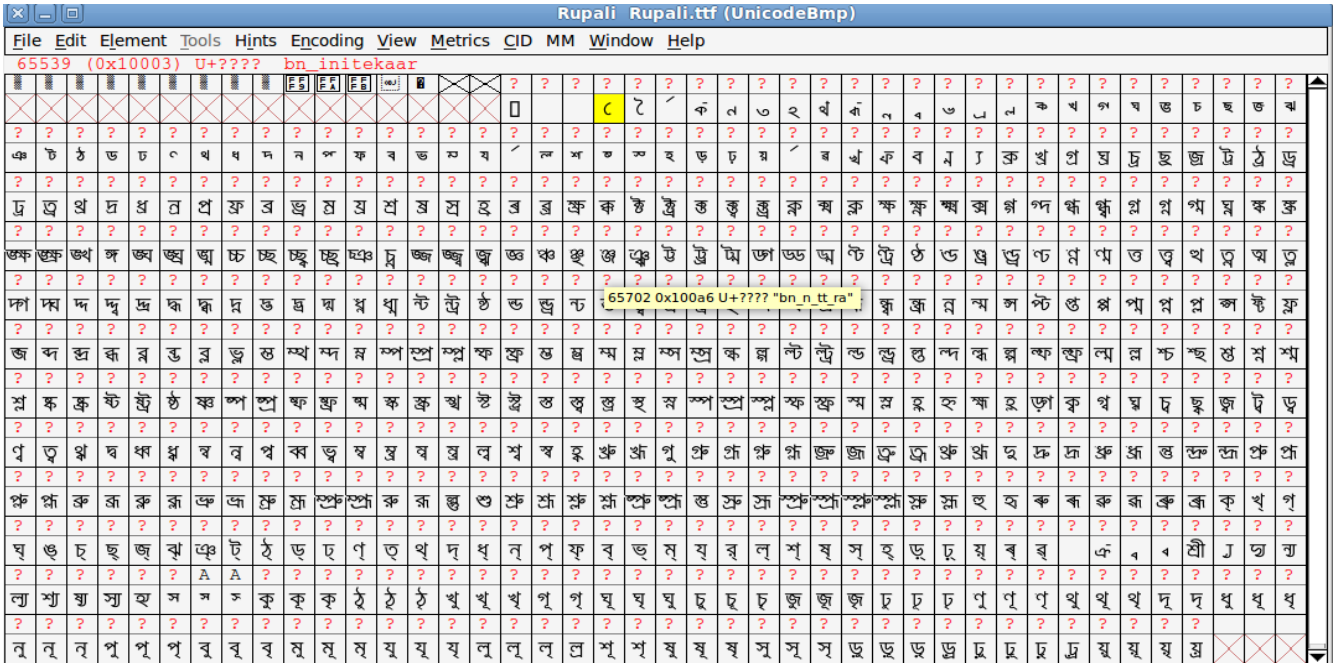
sudo apt-get install ruby font-forge fonttools

5.6 Modifying the font

Bengali fonts contain characters used by the substitution tables that are outside the normal range. As
we are working with 2 byte codes for unicode characters, we need to copy these characters to a
different range.

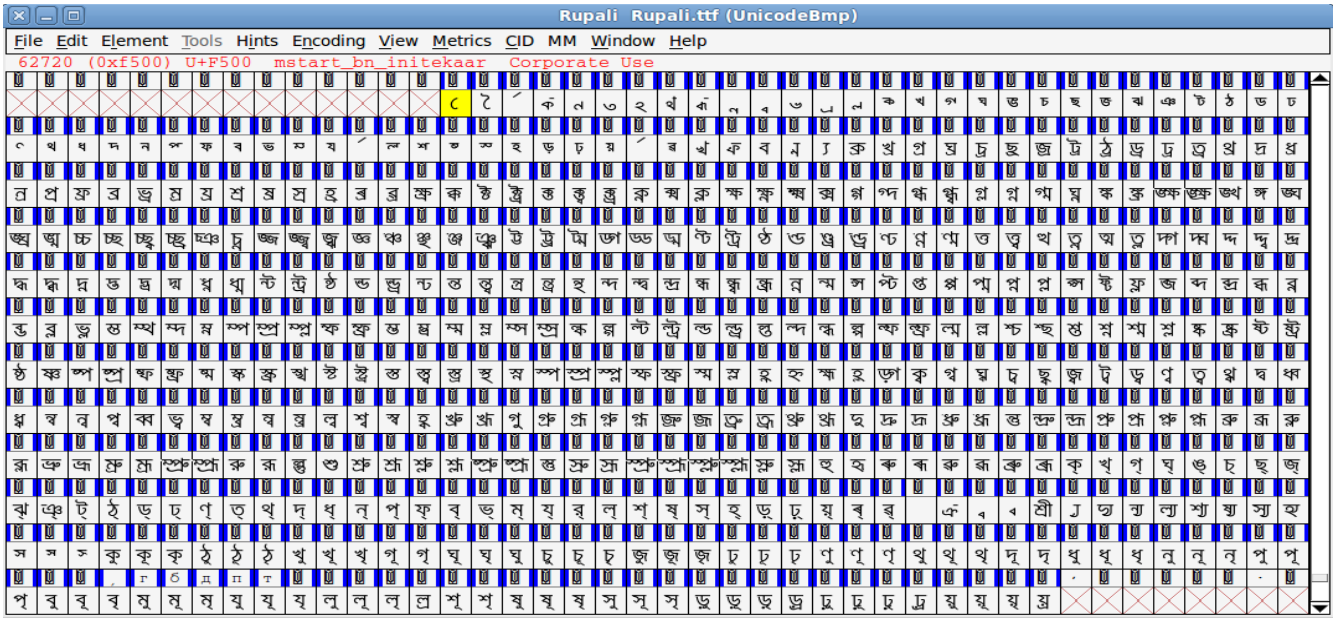
Open the font in font-forge. Go to the end of the range, and copy all the characters after 65535.

Note the name of the first character. In the example below, the first character is at 65539, and has
the name bn_initekaar.



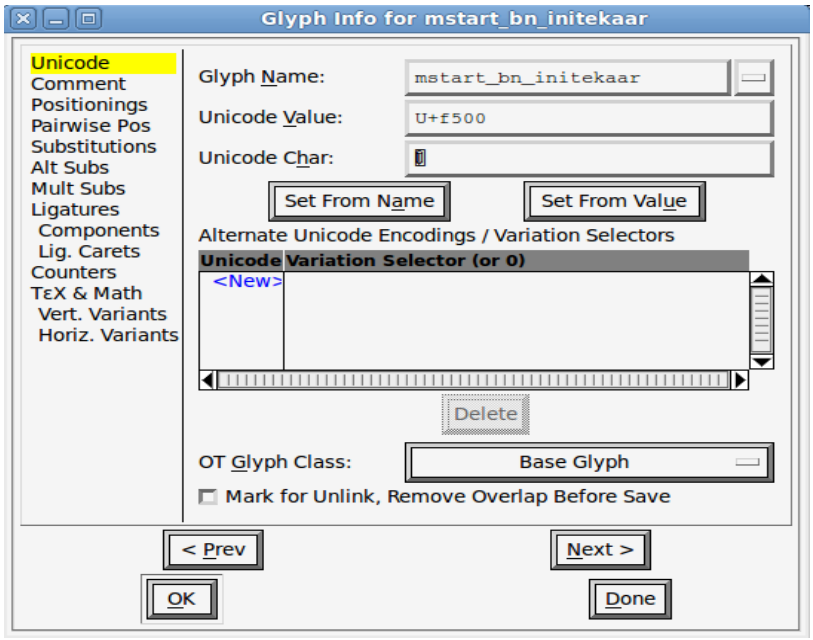
Paste the characters in a unused area of the range. In the example below the first character is at

62720 (corporate use) position .



Change the name of the first character, by right clicking on the character, and selecting “Glyph info”. Enter the original name, prepended by “mstart_”, for example, mstart_bn_initekaar. The gencode.rb script looks for a character starting with “mstart_”

and maps the original substitution rules to the new character range.



Go to the menu option File → Generate Fonts to generate the font. Keep the type as true type.

5.7 Generating the substitution code for the font:

The substitution rules for the font are hard coded into the skia library. This means the library will only support the Khmer font that it is compiled for.

Go to the directory containing the font, and run

ttx -s Rupali.ttf - (or other font name).

This will produce several .xml files, including one for the GSUB table. Copy gencode.rb to the same directory. From the directory containing the xml and the gencode run this command,

chmod +x gencode.rb

Run gencode.rb with the following command in terminal,

./gencode.rb > gsub.cpp

This generates the necessary c code to do the font substitutions. Copy gsub.cpp to the android

directory external/skia/core/gsub.cpp

cp gsub.cpp ~/android/external/skia/core/gsub.cpp

5.8 Recompile android source code:

Copy the shape.cpp file to external/skia/core/shape.cpp in the android directory.

Modify the external/skia/Android.mk file, adding the new files to be compiled

LOCAL_SRC_FILES:= ¥

src/core/shape.cpp ¥
src/core/gsub.cpp ¥

Modify external/skia/src/ports/SkFontHost_android.cpp

Adding

    { "Rupali.ttf",   gFBNames   },

to

static const FontInitRec gSystemFonts[] = {

}

Modify external/skia/src/core/SkCanvas.cpp

Add the definitions

typedef unsigned short mychar;
size_t reorder(const void *vtext, size_t len, mychar * newtext);

Change the function

```
void SkCanvas::drawText(const void* text, size_t byteLength,

             SkScalar x, SkScalar y, const SkPaint& paint) {
to

    mychar mytext[1024];
    size_t mybyteLength;

    if(byteLength > 1024)
        mybyteLength = 0;
    else
        mybyteLength = reorder(text,byteLength,mytext);

    ITER_BEGIN(paint, SkDrawFilter::kText_Type)

    while (iter.next()) {

        if (mybyteLength > 0)  {

           iter.fDevice->drawText(iter, (const void *) mytext, mybyteLength, x, y,
paint);

        } else  {

                iter.fDevice->drawText(iter, text, byteLength, x, y, paint);
        }
    }

    ITER_END

}
```

Change the function

```
void SkCanvas::drawPosText(const void* text, size_t byteLength, const SkPoint
pos[], const SkPaint& paint) {
```

to

```
    mychar mytext[1024];
    size_t mybyteLength;

    if(byteLength > 1024)
        mybyteLength = 0;

    else
        mybyteLength = reorder(text,byteLength,mytext);

    ITER_BEGIN(paint, SkDrawFilter::kText_Type)

    while (iter.next()) {

        if (mybyteLength > 0)
                iter.fDevice->drawPosText(iter, (const void *)mytext, mybyteLength,
&pos->fX, 0, 2, paint);

        else
                iter.fDevice->drawPosText(iter, text, byteLength, &pos->fX, 0, 2,
paint);

    }

    ITER_END

}
```

To recompile android source code

```
cd ~/android
. build/envsetup.sh
lunch full-eng
make -j4
```

The compiled files libskia.so and libskiagl.so are in
~/android/out/target/product/generic/system/lib/

Android should now show Bengali properly.

6. Conclusion:

Although the shape we made have some bugs, it will help android phones to render most of Bengali language properly. Our next step will be to improve the shaper and removing the remaining bugs. We also want to provide this service to the people of Bangladesh.

References:

http://site.icu-project.org/

http://code.google.com/p/skia/

http://behdad.org/text/

www.wikipedia.org/

http://finder.com.kh/?page_id=55

http://www.readytext.co.uk/?p=1375

http://www.microsoft.com/typography/otfntdev/bengalot/shaping.aspx