Burnt In Annotations

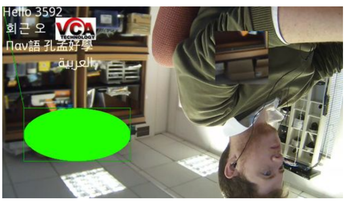
Introduction

The Burnt In Annotations project aims to create a simple graphics framework that can become the standard burnt in annotations framework for IPX. This graphics framework will be designed to fulfil the following functions:

- Burnt In Text (BIT)
- Data Time
- Camera Focus/Zoom overlay
- VCA Tracking Overlays
- Logo Overlay
- Debug Information

Contents

Screenshots

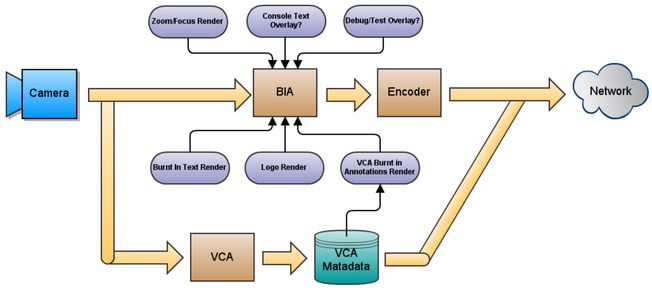


Requirements

- Must be able to render a selection of graphical objects:
  - Lines
  - Rectangle Outlines
  - Filled Rectangle
  - Fixed Width Text
- May optionally render:
  - Ellipse Outlines
  - Filled Ellipses
  - Filled Polygons
- Must be able to blt graphics into frame
- May optionally render transparently
- Must be able to render in real time
- Must have a low CPU load

Design

Overview



The BIA module runs in the video encode thread before encoding, and renders graphics from a series of layers.

Each layer is actually a callback that has been registered with the BIA module. When the callback is invoked, it calls functions in the BIA rendering API which are immediately drawn into the image frame.

In the same frame, the VCA engine computes the analytics and stores the output metadata in a buffer. In the next frame, the VCA BIA layer will parse the metadata buffer, and render these results into a frame.

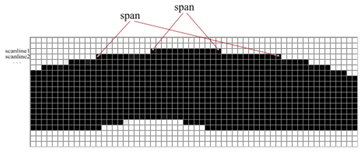
Features

Drawing Shapes without Heavyweight Functions

Avoiding dividing, square root, cosine etc. Is always preferable on any platform. This is particularly important on IPX because the DM368 has no divide unit.

- Bresenham's Line Algorithm can be used to draw diagonal lines without the need to use divide.
- A similar algorithm – the Midpoint Circle Algorithm can be adapted to render ellipses.

Scanline Renderer



A Scanline Renderer can be used to render solid shapes including: Rectangles, Ellipses, Polygons, Fat Lines and Text.

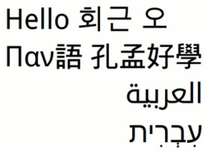
It works as follows:

1. Iterate around the outline of the shape to be rendered, computing a "guard pixel" that marks the edge of the shape for each line. These guard pixels are added to a list.
2. If necessary quick-sort the list, so that the list of pixels is sorted from left to right, and top to bottom – as in text flow. Pairs of guard pixels now form spans. (Guard pixels for simple shapes such as rectangles and ellipses can be generated in order so that this sorting step is not needed.)
3. Fill in the pixels in the spans between pairs of guard pixels.

This scheme has a few key advantages:

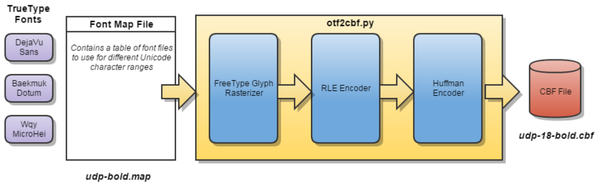
- Flexible: Any shape can be rendered so long as it's outline can be iterated.
- Modular: Low-level pixel operations are kept separate from high level geometry calculations. Different pixel formats can be supported by simply writing alternative pixel painting back-end functions.
- Good Performance In Complex Shapes: The outline of a polygon shape can be calculated as an O(N) operation (where N is the perimeter of the polygon). The list of guard pixels can then be sorted as an O(log N) operation, and the pixel painting is an O(N) operation.

TrueType Font Rendering



Good quality text rendering with good coverage of world languages is a desirable feature for UDP camera. TrueType fonts provide this.

Initial work has been done to explore the possibility of using the open source font rendering stack based on Pango and FreeType: TrueType Font Rendering with FreeType and Pango. This proved impractical because of the rendering performance, and large size of the needed shared libraries and font files. Instead I plan to implement a bitmap based font rendering engine.



Code To Be Deleted

BIA will make the following code redundant. We can therefore remove it from IPX.

- alg_swosd.c
- av_server/av_capture/application/ipnc/src/fonts/* – ~700kB compiled code

How to use "otf2cbf.py"

set the python environment.

file path : rootfs/system/fonts/otf2cbf.py

otf2cbf.py -o OUTFILE -m FONTMAP -s SIZE Ex) python otf2cbf.py -o udp-18-kor.cbf -m udp-regular-extended.map -s 18

Map file is structured as shown below. We can modify map file and we get cbf file that we want to.

```
[chan@fonts(limit/1.11.2)]$ cat udp-bold-extended.map
# Font File                     Start       End

# Cyrillic
ttf/DejaVuSans-Bold.ttf,        0x00000400, 0x000004FF      # Cyrillic
ttf/DejaVuSans-Bold.ttf,        0x00000500, 0x0000052F      # Cyrillic Supplement

# European
ttf/DejaVuSans-Bold.ttf,        0x00000000, 0x0000007F      # Basic Latin
ttf/DejaVuSans-Bold.ttf,        0x00000080, 0x000000FF      # Latin-1
ttf/DejaVuSans-Bold.ttf,        0x00000100, 0x0000017F      # Latin Extended-A
ttf/DejaVuSans-Bold.ttf,        0x00000180, 0x0000024F      # Latin Extended-B

# Greek
ttf/DejaVuSans-Bold.ttf,        0x00001F00, 0x00001FFF      # Greek Extended

# Japanese and Chinese
ttf/wqy-microhei.ttf,           0x00003000, 0x000030ff      # Punctuation Hiragana and Katakana
ttf/wqy-microhei.ttf,           0x00004e00, 0x00009faf      # CJK Unified Ideographs

# Korean
ttf/baekmuk-dotum.ttf,          0x0000AC00, 0x0000D7A3      # Hangul Precomposed Symbols

[chan@fonts(limit/1.11.2)]$
```