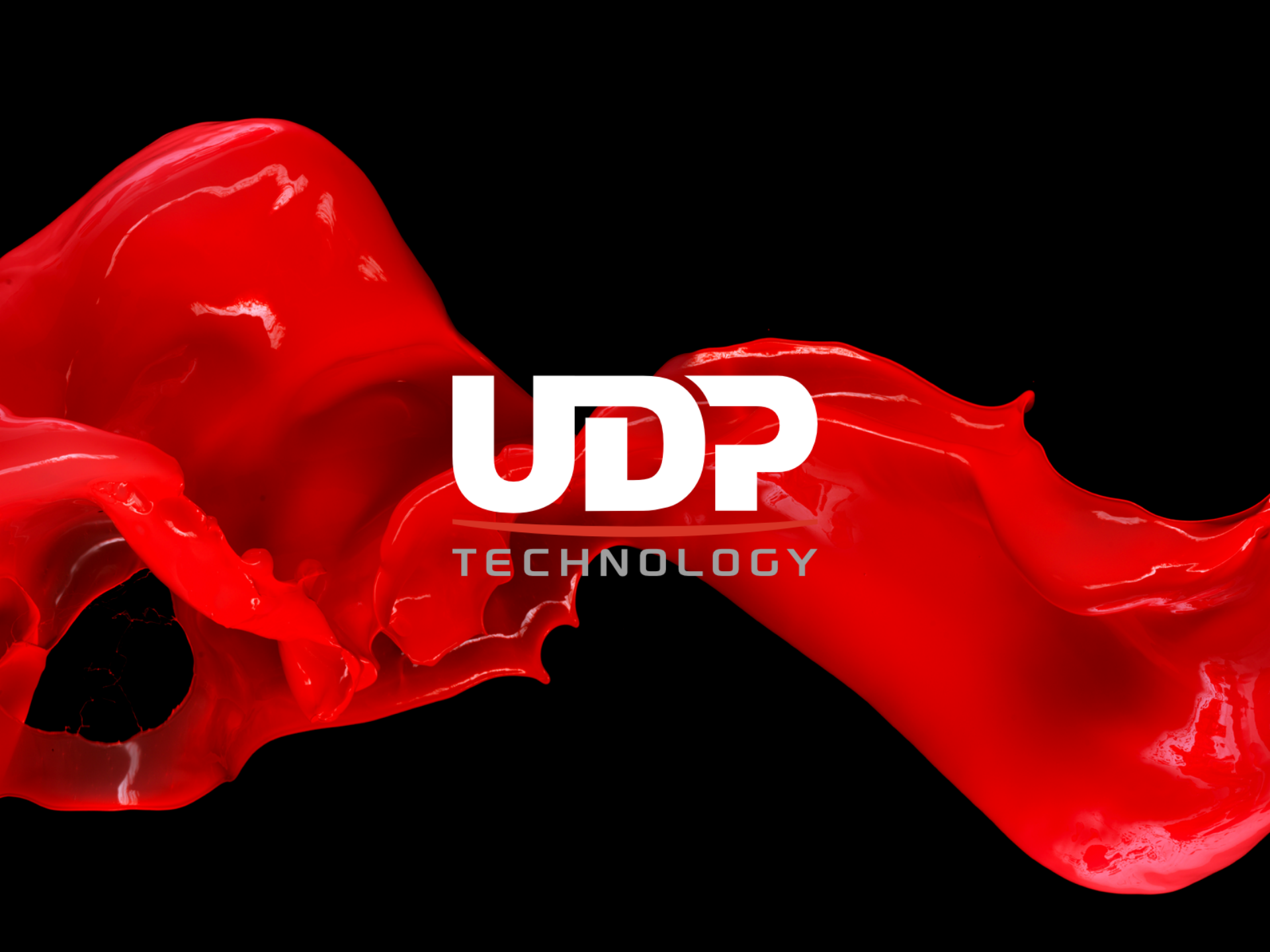

UDP
TECHNOLOGY


Introducing BIA Render
The New Rendering Framework for IPX

Introducing BIA Render

- BIA Render is the new rendering framework for IPX.
- It was originally conceived to add support for VCA Burnt in Annotation.
- But, BIA Render is versatile and can be used as a universal drawing framework for IPX.
- Burnt-in-Text (BIT), and Privacy mask have been ported to use BIA Render.

What does BIA Render do?

What does BIA Render do?

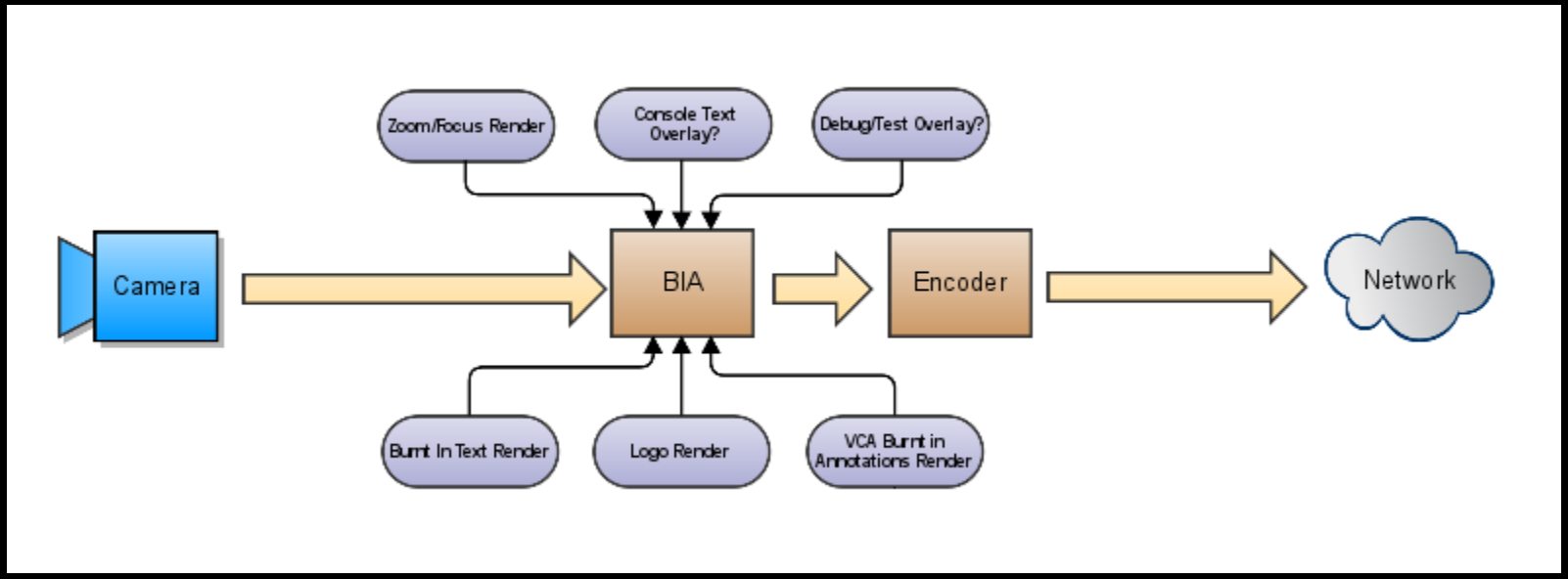
- BIA Render is a 2D Graphics framework like GDI, GDI+, Cairo, etc.
- It supports many drawing operations....

Here BIA Render is drawing an Ellipse, a Rectangle Frame, a Diagonal Line, Blitting an Image (the VCA Logo), and Blitting one part of the frame to another.

BIA Render supports high quality text rendering in many languages. Here we are rendering Burnt-In-Text in English, Korean, Chinese and Russian.

How does it work?

How does it work?



- BIA Render has a list of layers to render.
- Each layer is a function that executes drawing commands.

How does it work?

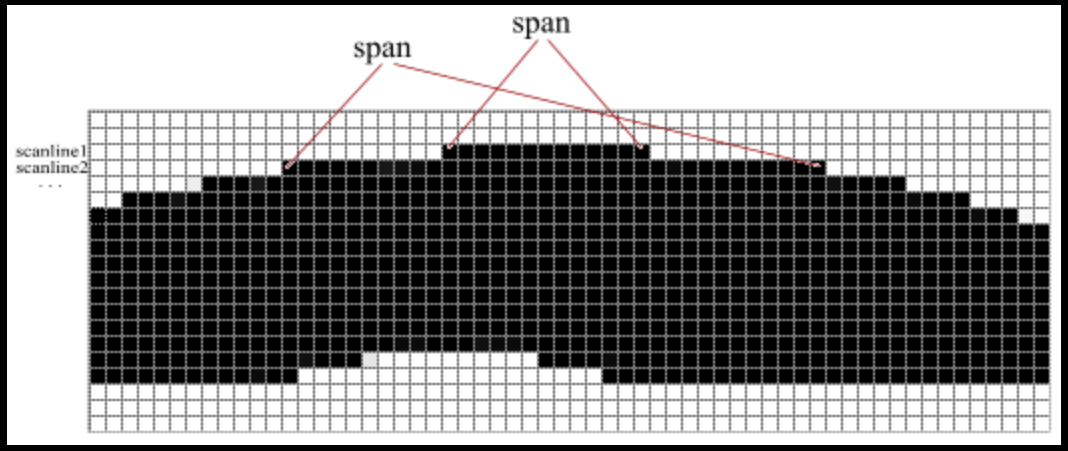
- Each layer is a simple function.
- This example is the privacy mask layer:

```
XDAS_Int32 PrivacyMaskRender(IVIDANALYTICS_Handle biaHandle,
  IVIDEO1_BufDescIn *const bufDesc, void *userData)
{
  int i;
  BIA_RENDER_UDP_YuvColour yuv;

  // Iterate through the privacy masks
  for(i = 0; i < PRIVACY_ZONE_MAX; i++)
  {
    // Get the zone
    const Video_privacyMask * const z =
        gAVSERVER_config.encodeConfig[0].privacyMask + i;
    if(z->enable)
    {
      const XDM_Rect rect = {
         {z->startX, z->startY},
         {z->startX + z->width, z->startY + z->height},
      };

      // Convert the RGB colour to YUV
      const BIA_RENDER_UDP_RgbColour rgb = {z->color_r, z->color_g, z->color_b};
      BIA_RENDER_UDP_Rgb2Yuv(&yuv, &rgb);

      // Render the privacy mask rectangle
      BIA_RENDER_UDP_fill_rect_yuv420sp(biaHandle, bufDesc, &rect, yuv);
    }
  }

  return OSA_SOK;
}
```

How are shapes rendered?

- Speed is important, and the DM368 is not powerful.
- Many lessons have been learned about how to write fast code for IPX.
- I use Bresenham's Algorithm to render Diagonal Lines, Ellipses, Rounded Rectangles etc. without using expensive Divide or Square Root operations.
- Complex shapes are rendered using a scanline-renderer.

Scanline Renderer?



- A Scanline Renderer is a fast and powerful algorithm for painting complex shapes.
- We iterate around the outline of a shape making a list of *guard pixels* on each row of the shape's outline.
- We then quick-sort the list of points top to bottom, left to right.
- We then fill in the spans between the guard pixels.

Text Rendering

Text Rendering

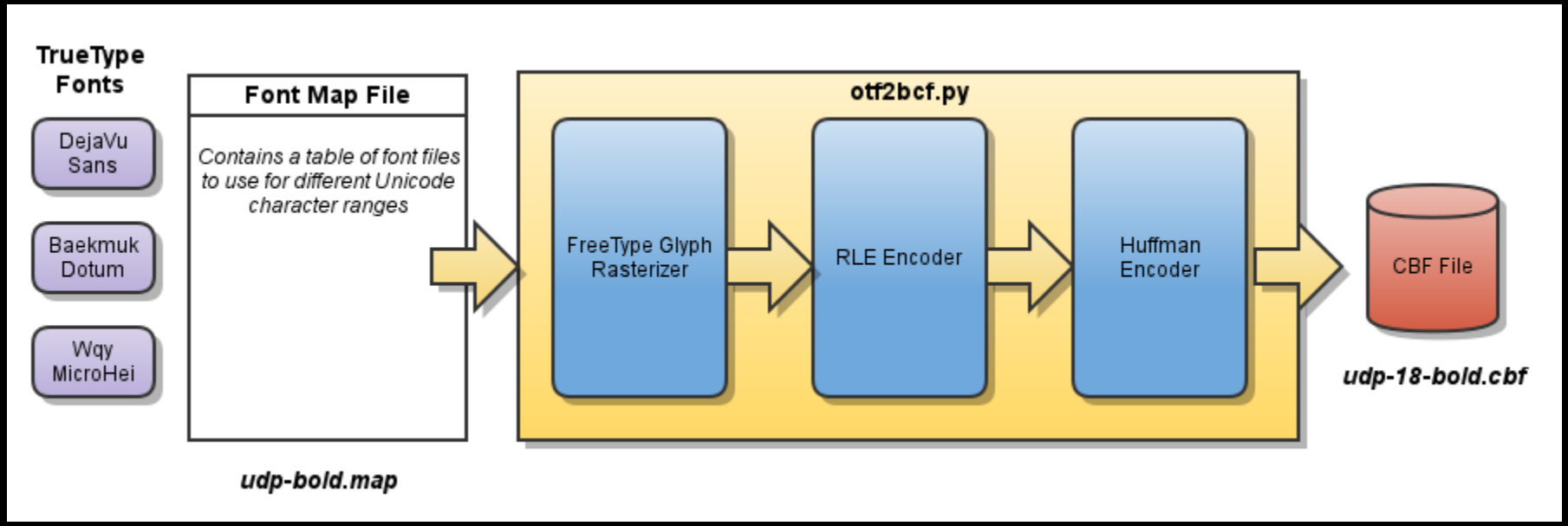
- BIA Text Rendering supports high quality variable-width text layout.
- It supports many languages including Latin Languages (English etc.), Korean, Chinese, Japanese, Greek and Russian with Unicode.
- Text can be rendered solid, or with an outline.

Text Rendering

- High quality text is hard...
  - To support Chinese/Japanese/Korean (CJK), there are many characters >30,000.
  - A CJK TrueType font is a large file >5Mb.
  - The TrueType font libraries (Pango + FreeType + FontConfig) do a good job, but are also large (~3.5Mb), and too slow.
  - We need to store fonts in a small file, but render them very fast on the slow camera.

Text Rendering

- My solution is to convert TrueType fonts into a compressed binary font (CBF) file during the build using a conversion script: *otf2cbf.py*



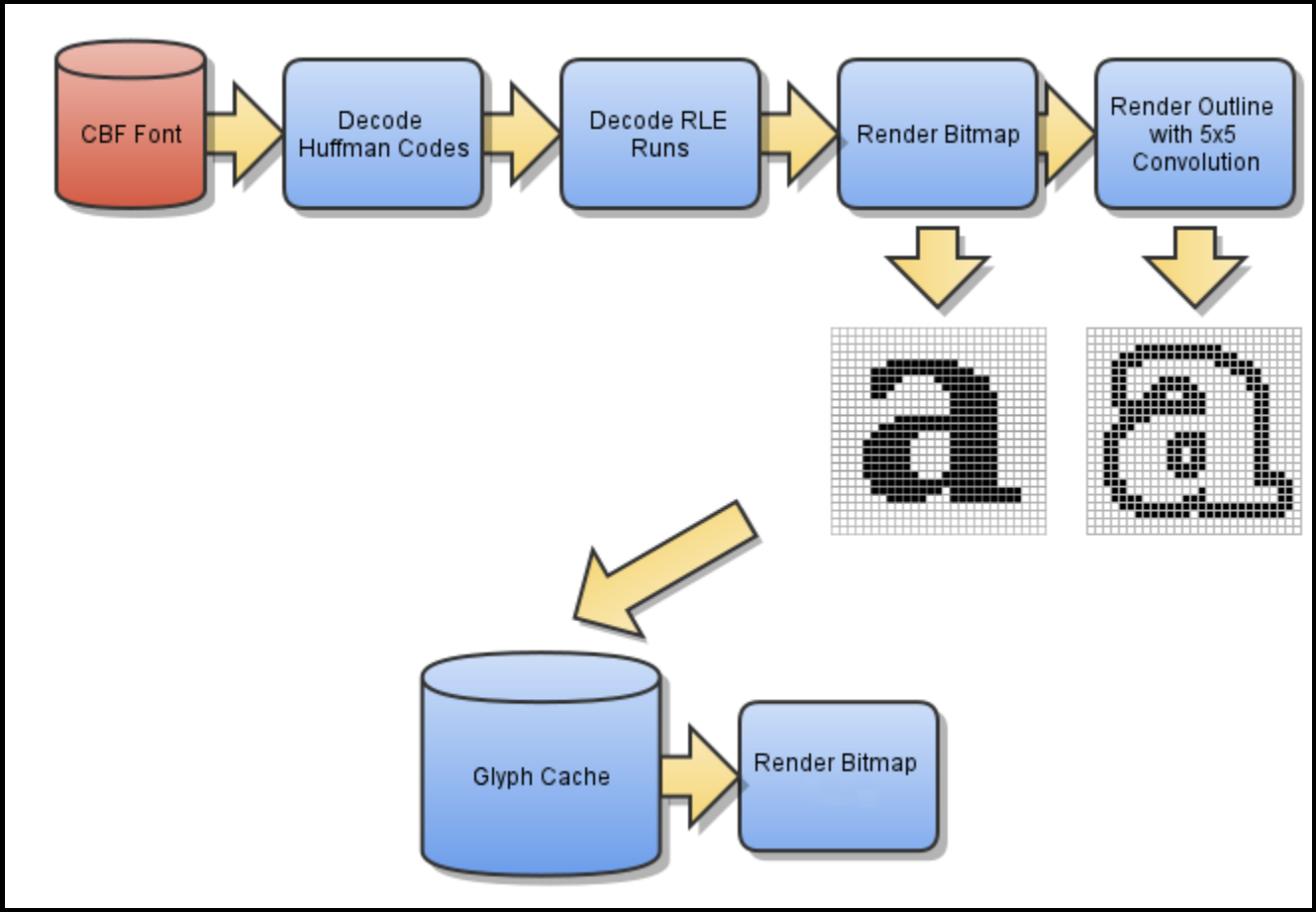
Text Rendering

- The map file describes which characters are included, and from which font:

```
# Font File                       Start        End


# Cyrillic
fonts/dejavu-sans.ttf,            0x00000400, 0x000004FF     # Cyrillic
fonts/dejavu-sans.ttf,            0x00000500, 0x0000052F     # Cyrillic Supplement

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

# Greek
fonts/dejavu-sans.ttf,            0x00001F00, 0x00001FFF     # Greek Extended

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

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

Text Rendering

- Glyphs are decoded on demand into a cache



Text Rendering

- Rendering performance depends on font size, but is similar to SWOSD Text.

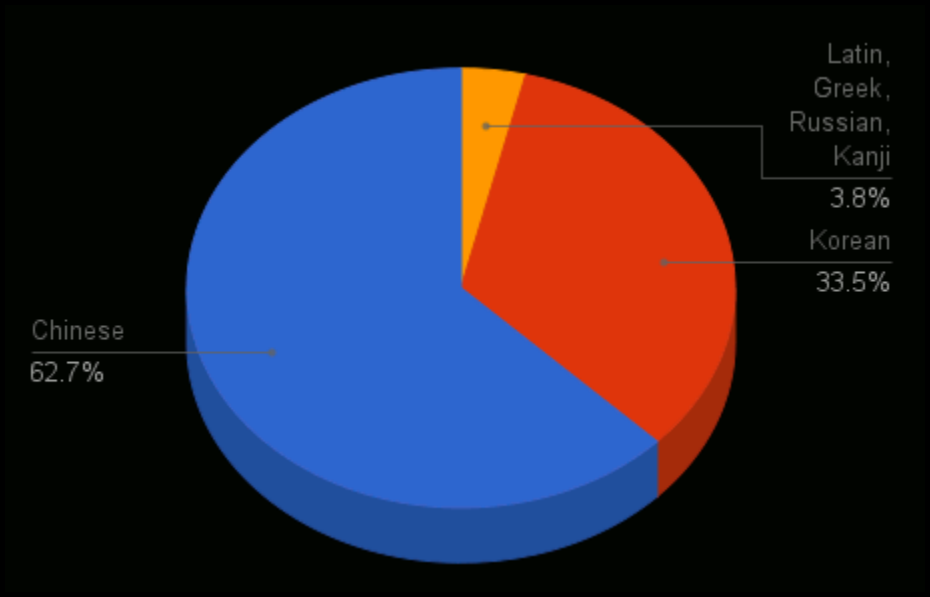
UDP BIT	The quick brown fox jumps over the lazy dog
12px Bold	The quick brown fox jumps over the lazy dog
14px Bold	The quick brown fox jumps over the lazy dog
18px Regular	The quick brown fox jumps over the lazy dog

	Test String Painting Time (ms)				
	The quick brown fox jumps over the lazy dog	The quick brown fox	The quick	The	AAAAAAAAAA
UDP's BIT Implementation	**0.406**	**0.201**	**0.112**	**0.045**	**0.398**
BIA Render Size 12 Bold	0.361	0.182	0.102	0.047	0.517
BIA Render Size 14 Bold	0.417	0.204	0.112	0.05	0.577
BIA Render Size 18	0.525	0.251	0.135	0.061	0.763

Text Rendering

- Each text size must have a different CBF file
  - udp-12-bold.cbf (635kB)
  - udp-14-bold.cbf (735kB)
  - udp-18-regular.cbf (975kB)
- Font sizes must be chosen carefully not to take up too much space.
- Giant font sizes have large files:
  - udp-48-regular.cbf (3.0Mb)
  - If we need giant fonts in the future we may need to reduce the number of characters, or design a compressed vector format for large fonts.

Text Rendering



- Most of the size is caused by the number of Korean (11172) and Chinese (20900) characters.
- Is it possible to filter the list of characters?

What next?

What next?

- BIA Render is nearly ready to integrate.
  - More testing, bug-fixing, completing is needed.
- Once integrated, it will upgrade the following features...

Burnt-in-Text

PTZ Position Display

BIA Render allows us to display the PTZ status with high quality indicators, so the user can see small position changes.

Privacy Mask

Obselete Code

- Many modules are now obsolete and have been deleted in my branch.
  - Swosd (3806 LOC)
  - Swosd Fonts (43737 LOC)
  - avServerUiMenu (302 LOC)
  - libfreetype
  - Logo support*

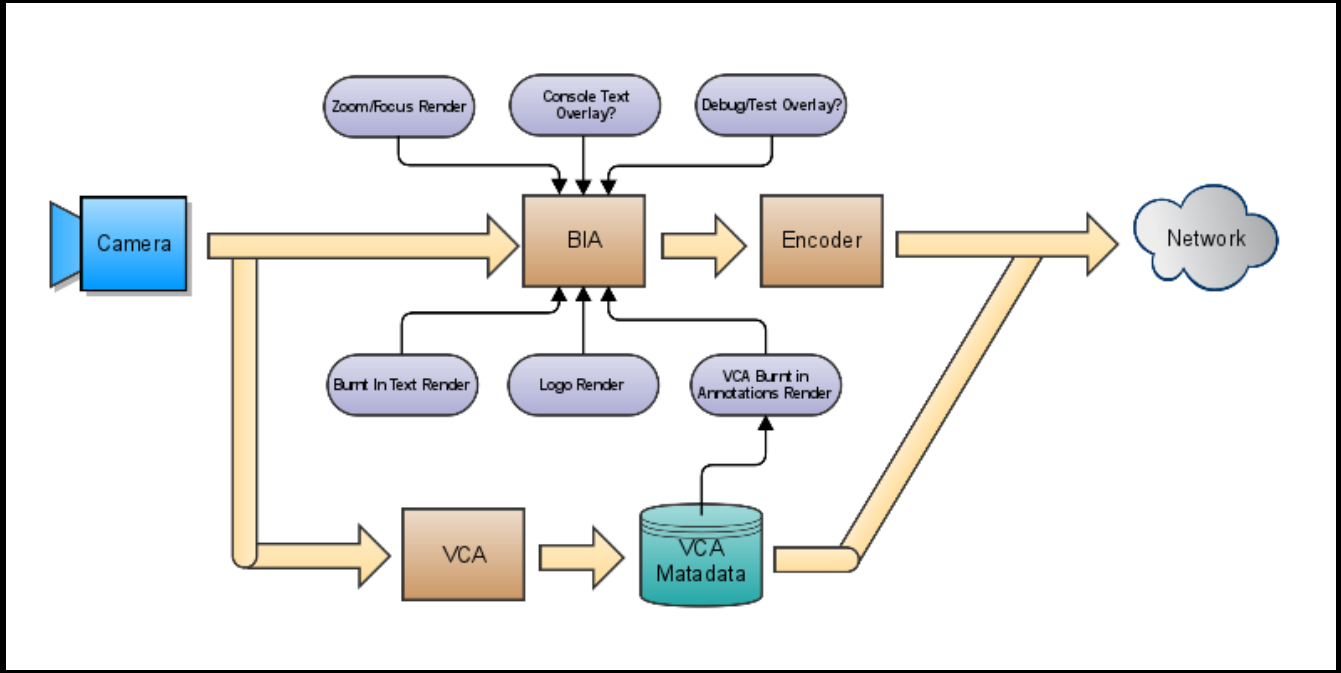
*If this is needed, it will need to be reimplemented with BIA Render.

What next?

- The code is available to test via BitBucket:
  - https://bitbucket.org/jhol/ipx/overview
- If you have git, and a UDP BitBucket account, you can download it with the following command:
  - `$ git clone https://username@bitbucket.org/jhol/ipx.git`

What next?

- Once this branch is merged, VCA Burnt-in Annotation can be implemented.
- They will be implemented as another layer.



Future Work

- Add more commands
  - Polygons, Fat Lines
  - Giant Text
- Accelerated Blitting/Text with DMA/VICP
- Support RTL Languages (Arabic & Hebrew)
- Logo Rendering

Summary

- BIA Render is the new rendering framework for IPX.
- It will be merged to trunk in the next few days.
- It will upgrade BIT, PTZ Position and Privacy Mask rendering.


BIA Render
Coming soon