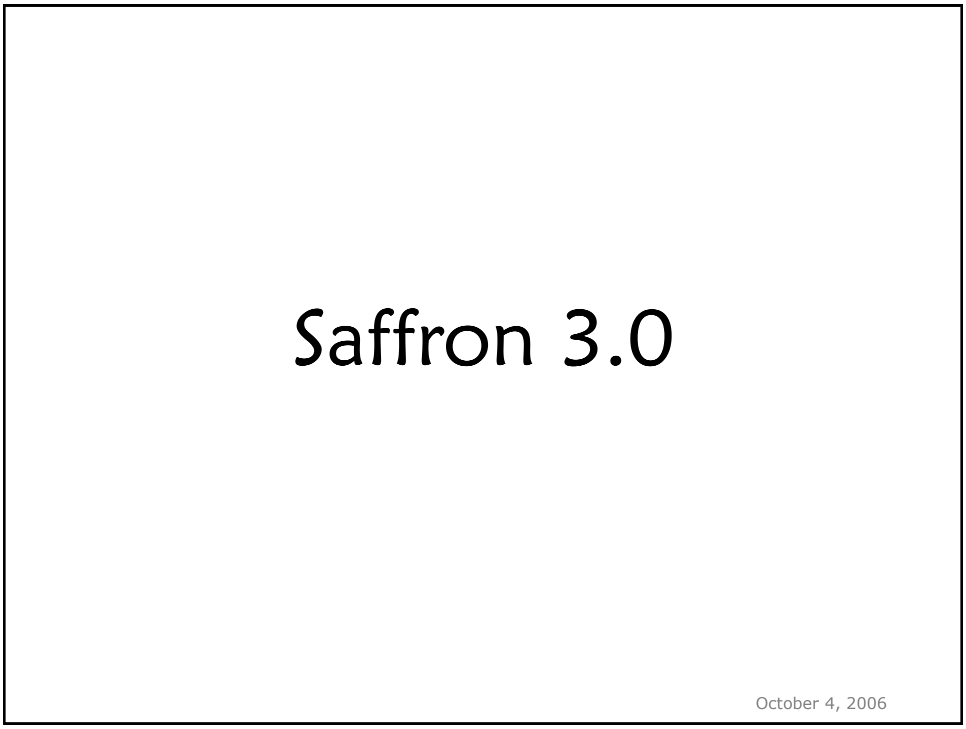
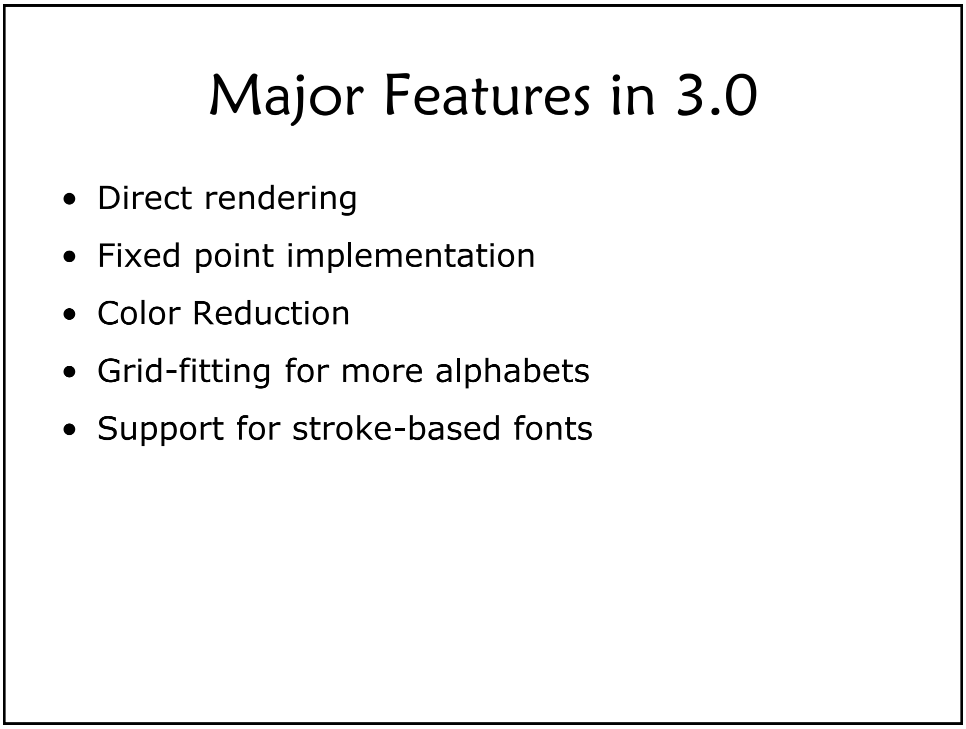


Saffron 3.0

October 4, 2006



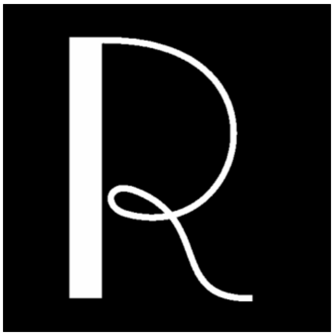
Major Features in 3.0

- Direct rendering
- Fixed point implementation
- Color Reduction
- Grid-fitting for more alphabets
- Support for stroke-based fonts

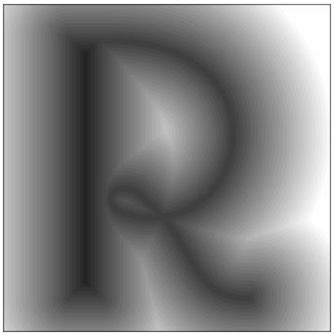
Quick Review

Distance Fields

- What is a distance field?
  - The 2D distance field of a shape represents, for any point in space, the signed minimum distance from that point to the edge (i.e., outline) of the shape

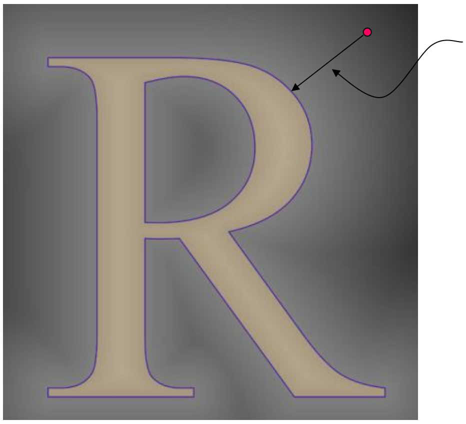


Shape



Shape’s distance field

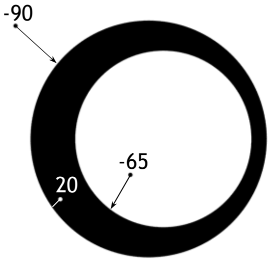
Example



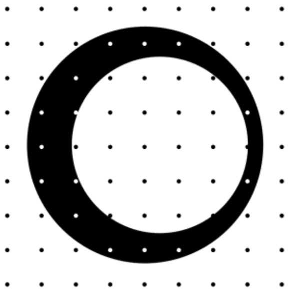
Distance to closest point on outline

Distance Field Representations

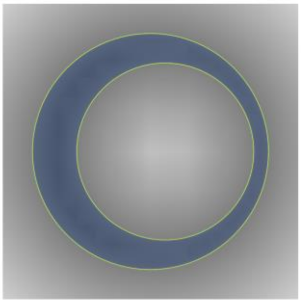
- Distance map using regularly spaced samples
  - Wasteful in both storage and time



2D shape with sampled distances to the surface



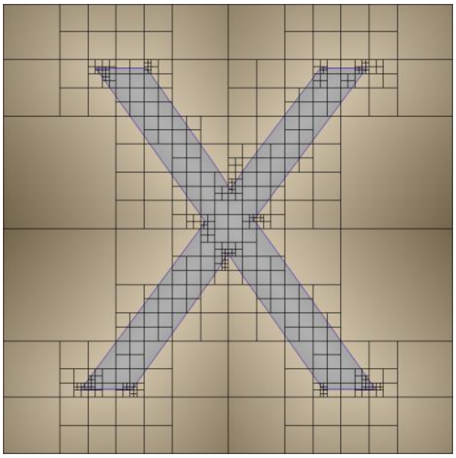
Regularly sampled distance values



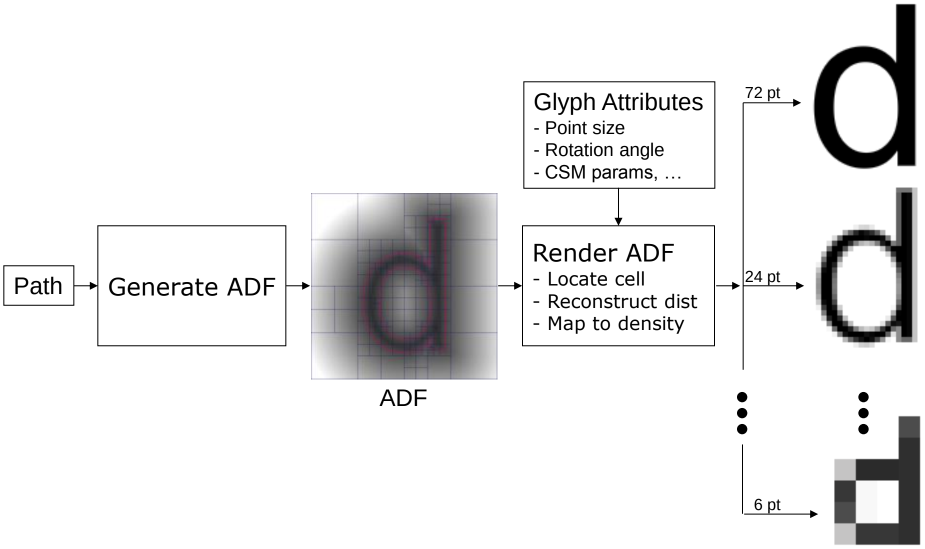
2D distance field

Distance Field Representations

- Adaptively Sampled Distance Fields



Generation and Rendering



Path

Generate ADF

ADF

Glyph Attributes
- Point size
- Rotation angle
- CSM params, …

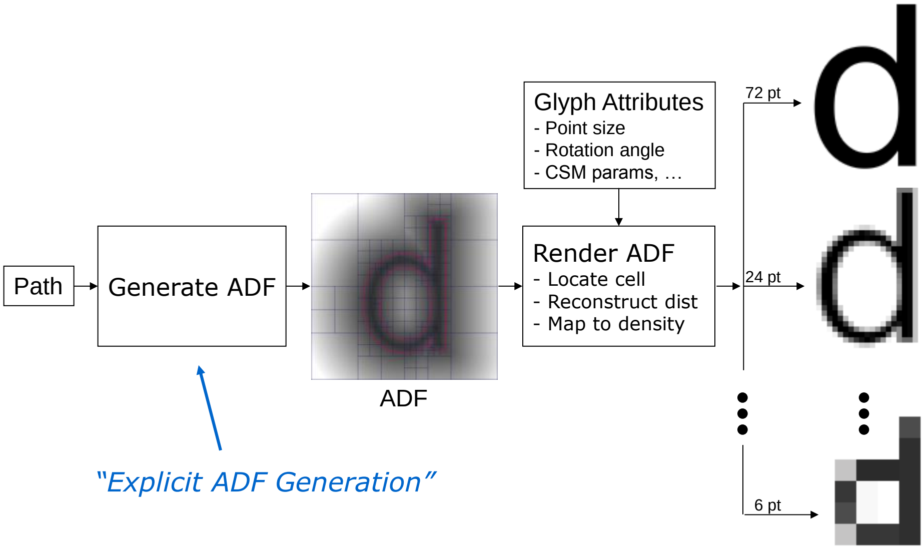
Render ADF
- Locate cell
- Reconstruct dist
- Map to density

72 pt

24 pt

6 pt

Terminology



Path

Generate ADF

ADF

Glyph Attributes
- Point size
- Rotation angle
- CSM params, …

Render ADF
- Locate cell
- Reconstruct dist
- Map to density

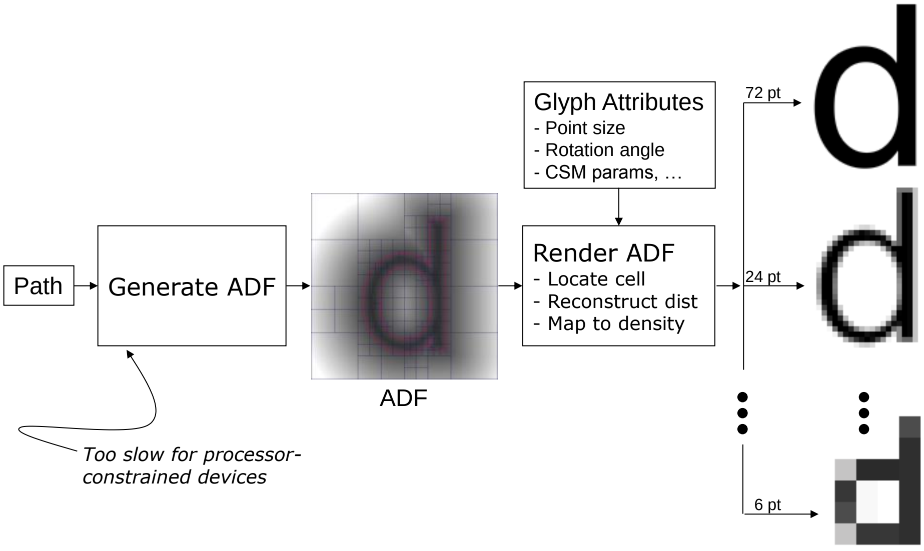
72 pt

24 pt

6 pt

"Explicit ADF Generation"

Explicit ADF Generation Issue #1



Path

Generate ADF

ADF

Glyph Attributes
- Point size
- Rotation angle
- CSM params, …

Render ADF
- Locate cell
- Reconstruct dist
- Map to density

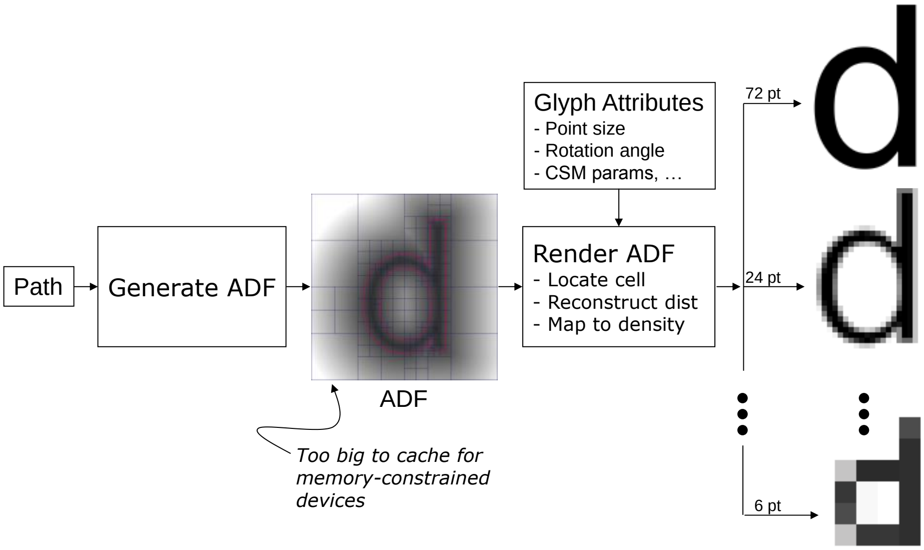
72 pt

24 pt

6 pt

Too slow for processor-constrained devices

Explicit ADF Generation Issue #2



Path

Generate ADF

ADF

Too big to cache for memory-constrained devices

Glyph Attributes
- Point size
- Rotation angle
- CSM params, …

Render ADF
- Locate cell
- Reconstruct dist
- Map to density

Quality Issues With Explicit ADFs

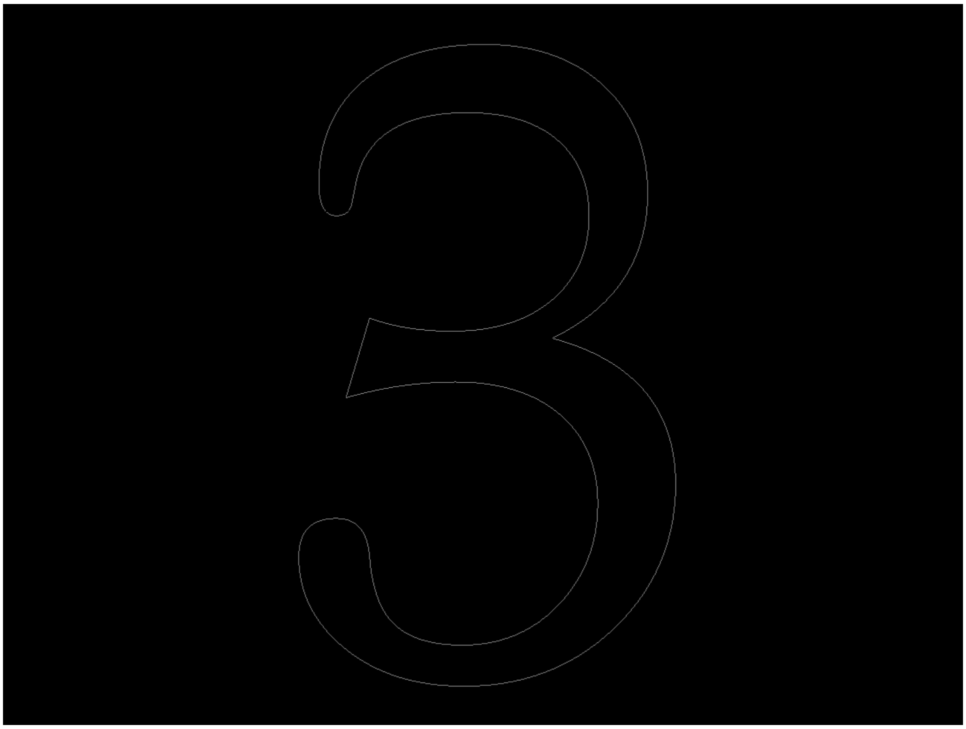
- Accuracy decreases as PPEMs increases
  - L7 ADFs are more accurate than L4 ADFs
  - But L7 ADFs are slower to generate and require more storage
- Artifacts with non-uniform scaling

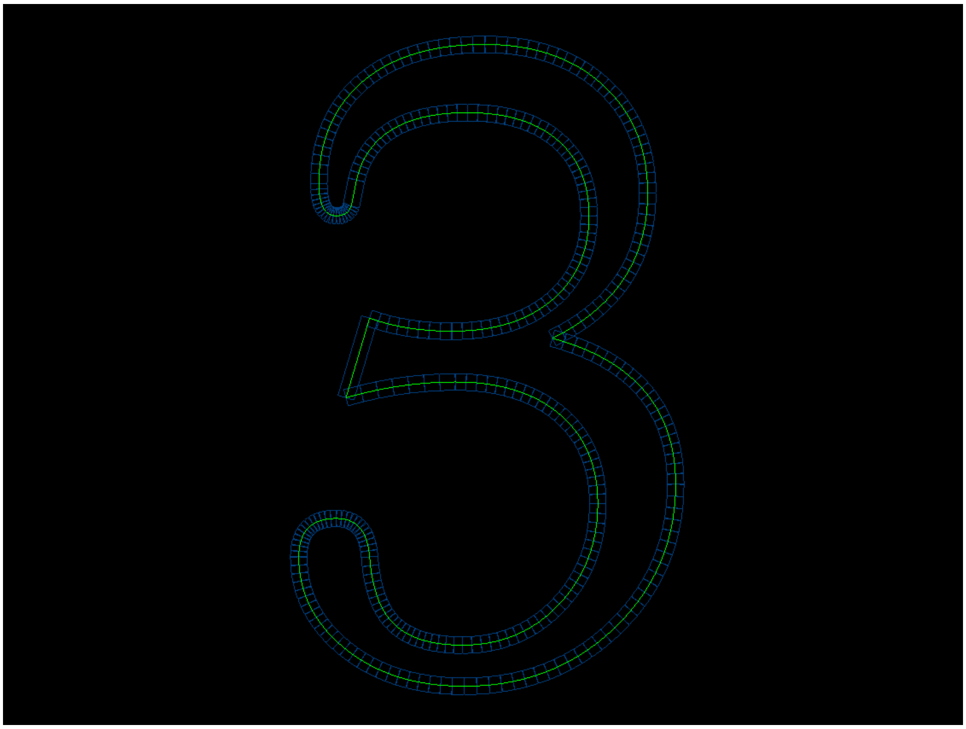
Direct Rendering

- General goals
  - Overcome performance and quality issues with explicit ADFs
  - Target processor-constrained and memory-constrained devices

Direct Rendering Overview

- Avoid explicit ADF generation
- Render images directly from glyph outlines
  - Compute distance field on-the-fly
  - Traverse only important sample locations
  - Minimize distance computations

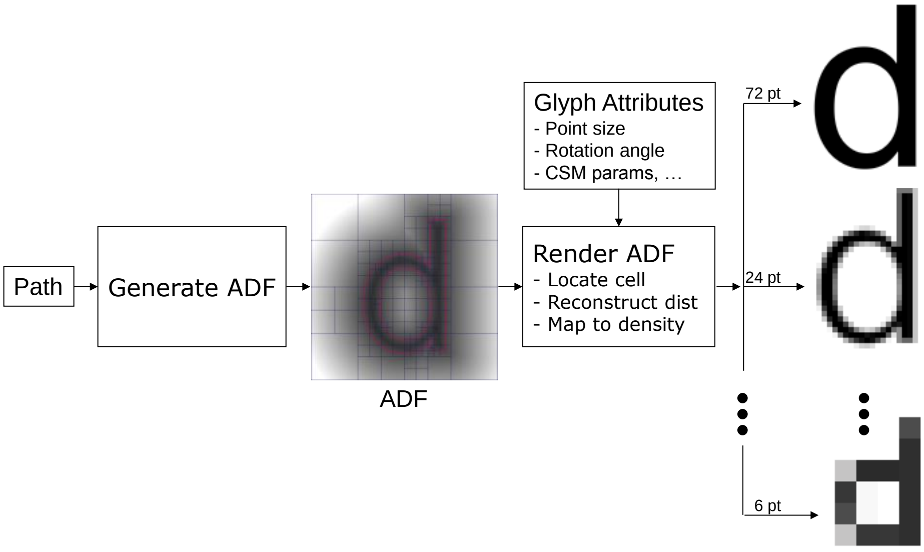




Comparison

- Explicit ADF
  - Computes the distances for ***all potential images***
  - Produces multiple images of the same glyph
- Direct rendering
  - Computes the distances needed for ***only one image***
  - Produces one image for a single set of glyph attributes

Explicit ADF Rendering



Path

Generate ADF

ADF

Glyph Attributes
- Point size
- Rotation angle
- CSM params, …

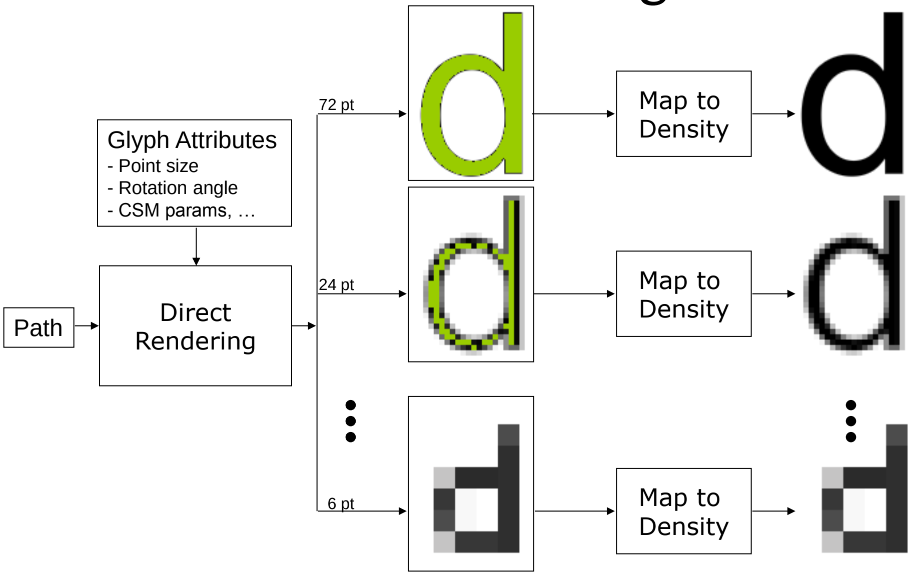
Render ADF
- Locate cell
- Reconstruct dist
- Map to density

72 pt

24 pt

6 pt

Direct Rendering



Glyph Attributes
- Point size
- Rotation angle
- CSM params, …

Path

Direct Rendering

72 pt

24 pt

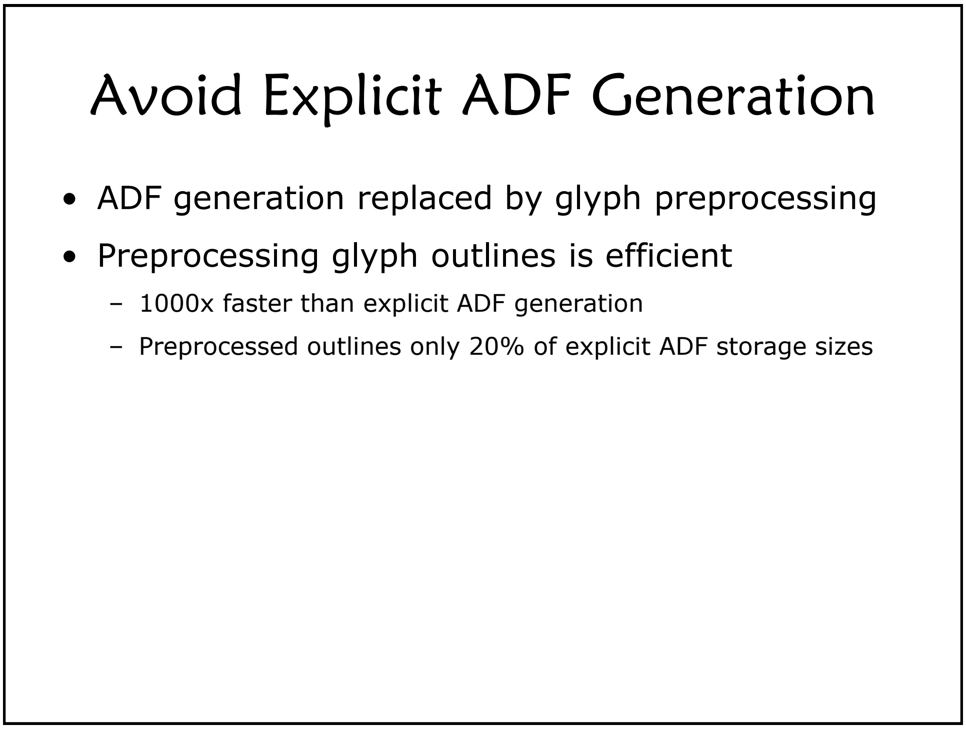
6 pt

Map to Density

Map to Density

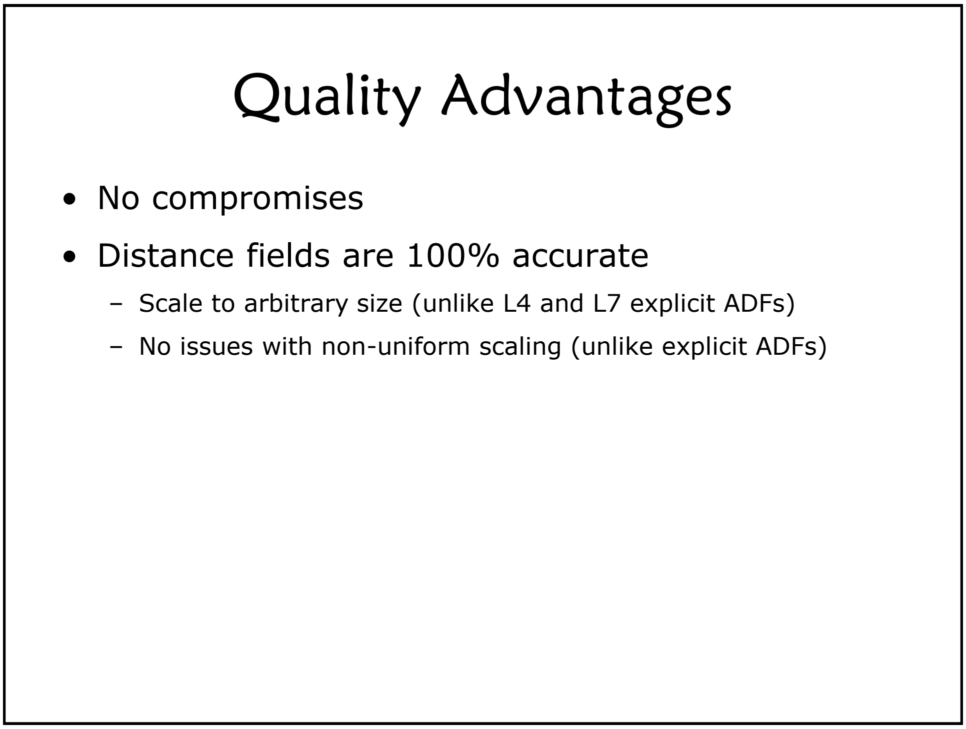
Map to Density

Distance buffer



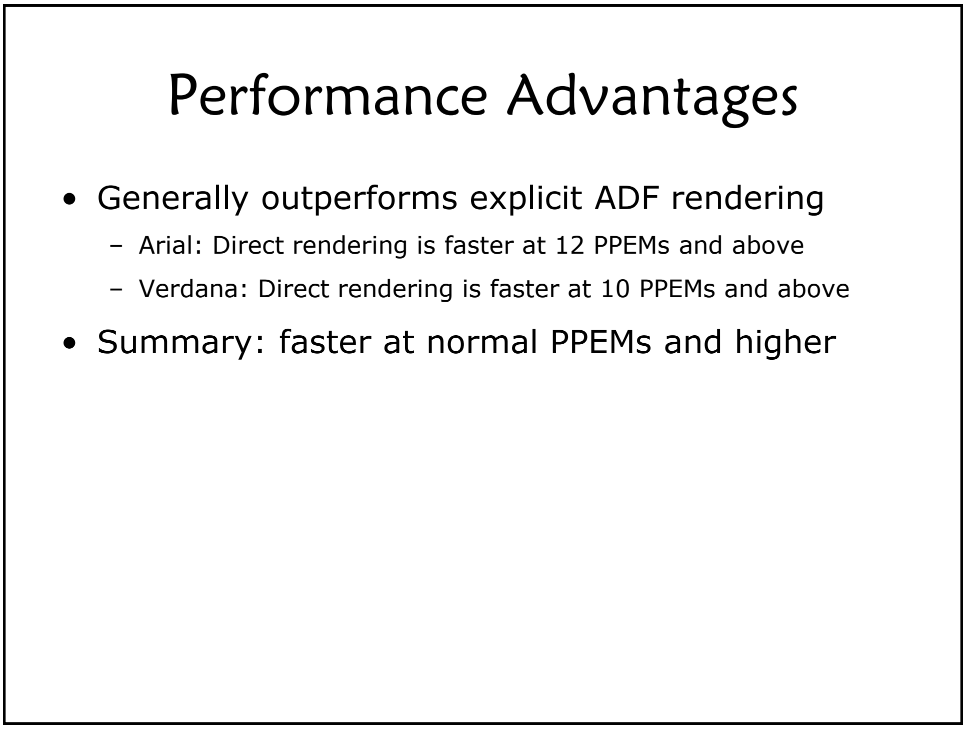
Avoid Explicit ADF Generation

- ADF generation replaced by glyph preprocessing
- Preprocessing glyph outlines is efficient
  - 1000x faster than explicit ADF generation
  - Preprocessed outlines only 20% of explicit ADF storage sizes



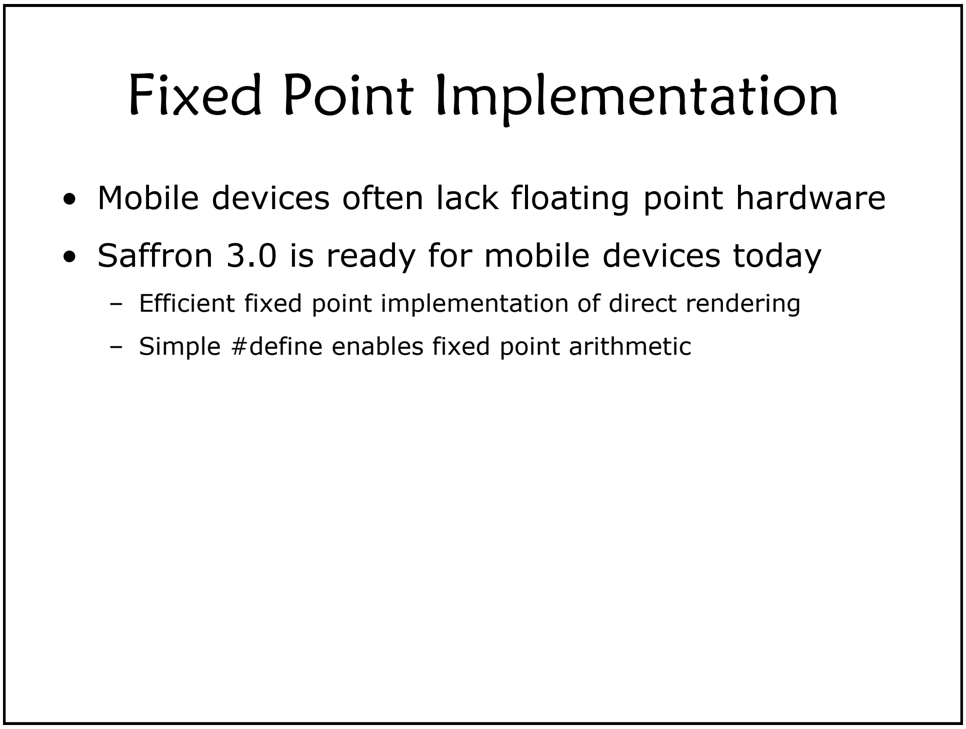
Quality Advantages

- No compromises
- Distance fields are 100% accurate
  - Scale to arbitrary size (unlike L4 and L7 explicit ADFs)
  - No issues with non-uniform scaling (unlike explicit ADFs)



Performance Advantages

- Generally outperforms explicit ADF rendering
  - Arial: Direct rendering is faster at 12 PPEMs and above
  - Verdana: Direct rendering is faster at 10 PPEMs and above
- Summary: faster at normal PPEMs and higher



Fixed Point Implementation

- Mobile devices often lack floating point hardware
- Saffron 3.0 is ready for mobile devices today
  - Efficient fixed point implementation of direct rendering
  - Simple #define enables fixed point arithmetic

Demos

- System
  - Dell Axim x51v
  - 640x480 16-bit LCD display (~170 dpi)

Direct Rendering Summary

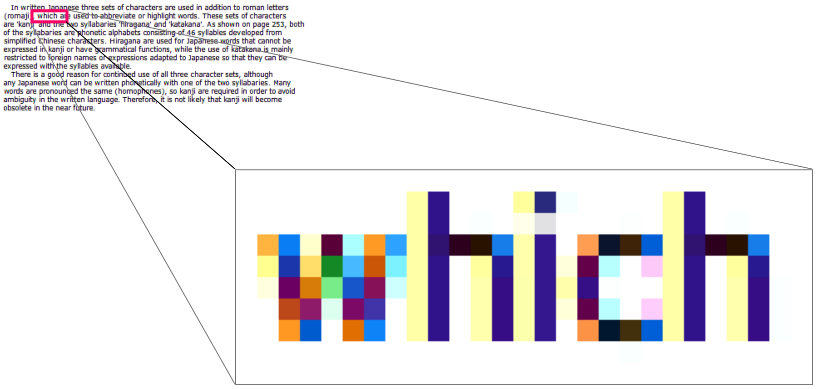
- Overcomes speed/memory issues of explicit ADFs
- Higher rendering quality than explicit ADFs
- Higher rendering performance than explicit ADFs
- Efficient fixed point implementation
- Saffron 3.0 runs in real-time on mobile devices
- Recompile only (no code changes)

Why Color Reduction?

- Principle of LCD rendering
  - Use separate samples at red, green, blue pixel locations
  - Trade off color accuracy for spatial resolution

Why Color Reduction?

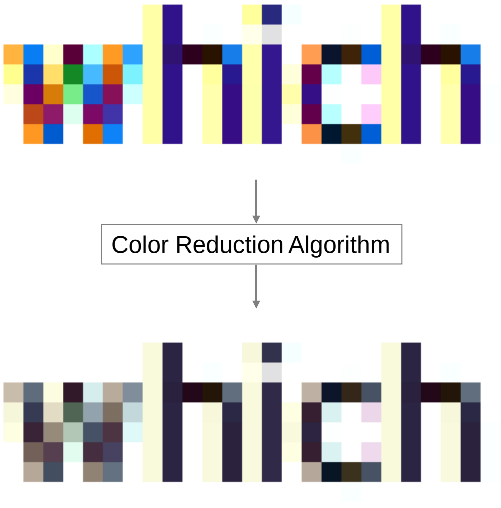
- Problem: LCD rendering produces color fringes



In written Japanese three sets of characters are used in addition to roman letters (romaji), which are used to abbreviate or highlight words. These sets of characters are 'kanji' and the two syllabaries 'hiragana' and 'katakana'. As shown on page 253, both of the syllabaries are phonetic alphabets consisting of 46 syllables developed from simplified Chinese characters. Hiragana are used for Japanese words that cannot be expressed in kanji or have grammatical functions, while the use of katakana is mainly restricted to foreign names or expressions adapted to Japanese so that they can be expressed with the syllables available.

There is a good reason for continued use of all three character sets, although any Japanese word can be written phonetically with one of the two syllabaries. Many words are pronounced the same (homophones), so kanji are required in order to avoid ambiguity in the written language. Therefore, it is not likely that kanji will become obsolete in the near future.

Color Reduction



Color Reduction Algorithm

Color Reduction Benefits

- Minimizes color fringing
- Makes tuning CSM parameters much easier
- Can be used in fully automatic mode
  - No changes required for existing .swf files
- API allows fine-tuning of color reduction
  - Not necessary in our experience

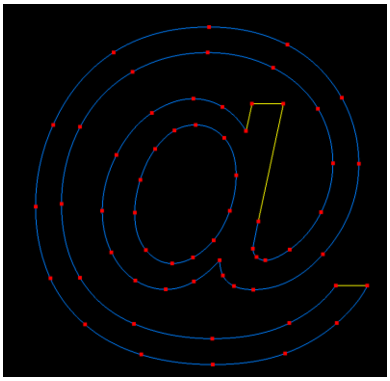
Grid-Fitting

- Alignment support for the following alphabets
  - Latin
  - Arabic
  - Devanagari
  - Hebrew
  - Thai

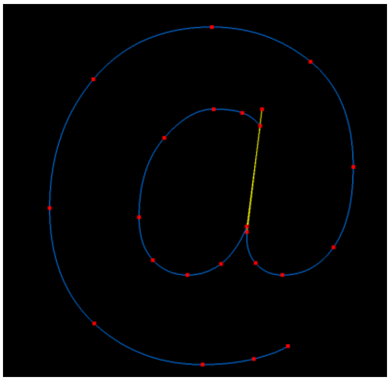
कम्प्यूटर, मूल रूप से, नंबरों से सम्बंध रखते हैं। ये प्रत्येक
अक्षर और वर्ण के लिए एक नंबर निर्धारित करके अक्षर और वर्ण
संग्रहित करते हैं। यूनिकोड का आविष्कार होने से पहले, ऐसे नंबर
देने के लिए सैंकडों विभिन्न संकेत लिपि प्रणालियां थीं। किसी एक
संकेत लिपि में पर्याप्त अक्षर नहीं हो सकते हैं : उदाहरण के लिए,
यूरोपियि संघ को अकेले ही, अपनी सभी भाषाओं को कवर करने के लिए अनेक
विभिन्न संकेत लिपियों की आवश्यकता होती है। अंग्रेजी जैसी भाषा के
लिए भी, सभी अक्षरों, विरामचिन्हों और सामान्य प्रयोग के तकनीकी
प्रतीकों हेतु एक ही संकेत लिपि पर्याप्त नहीं थी।
प्रतीकों हेतु एक ही संकेत लिपि पर्याप्त नहीं थी। के तकनीकी

Stroke-Based Fonts

- Saffron 3.0 supports stroke-based fonts

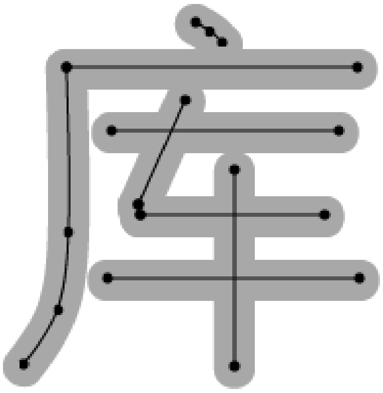


Outline-based glyph



Stroke-based glyph

Example



Benefits of Stroke-Based Fonts

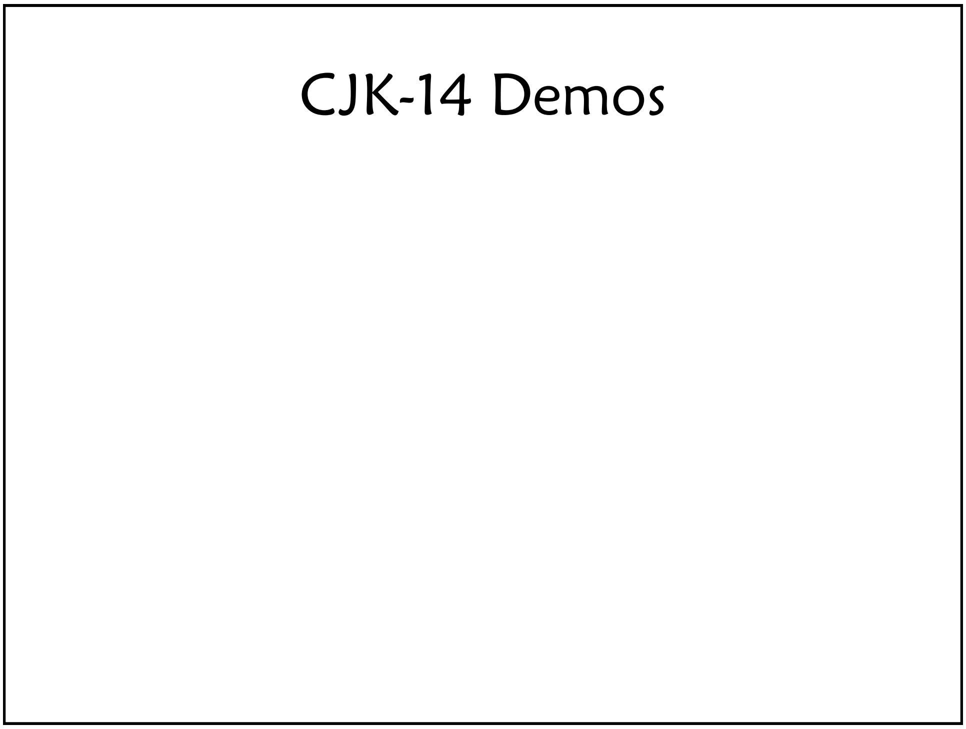
- Enormous space savings
  - GB 2312 Simplified Chinese outline typeface: ~2.7 MB
  - Equivalent stroke-based typeface: < 256 KB
- Faster rendering (about 2x)

Supported Stroke Formats

- Current release (3.0)
  - Uniform-width stroke fonts (e.g., MTI sticks)
  - Application-hinted stroke data (e.g., MTI CJK-14)
- In development
  - Stylized Stroke Fonts (developed at MERL)
- API can accept strokes from multiple sources

Example: CJK-14

- What is CJK-14?
  - Stroke-based typeface for CJK
  - Contains both alignment and simplification hints
- Exceptional clarity down to 14 ppems



CJK-14 Demos

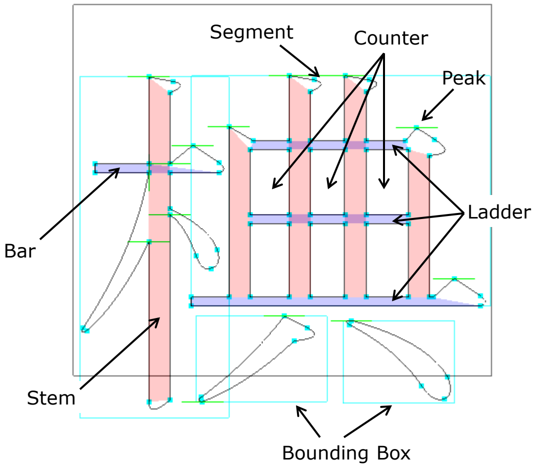
In Development

Multiple Alignment Zones

- Goal: better CJK rendering from outlines
- Saffron 3.0 (and earlier) do not grid-fit CJK

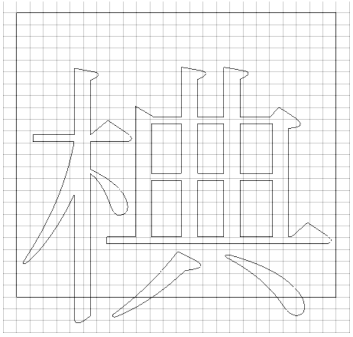
Multiple Alignment Zones

- Analysis of CJK-oriented features in outlines

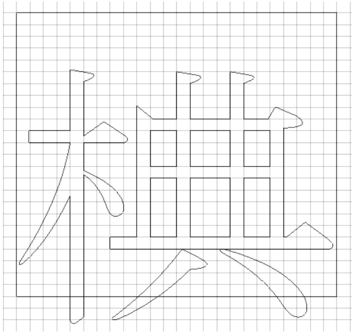


Multiple Alignment Zones

- Convert to hints and then grid-fit

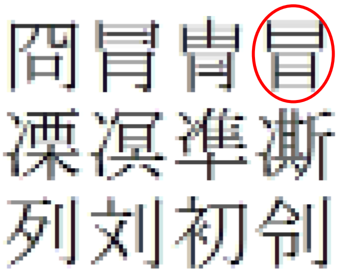


Unaligned glyph outlines

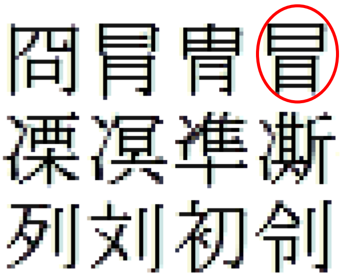


After grid-fitting

Example



Without Grid-Fitting



With grid-fitting

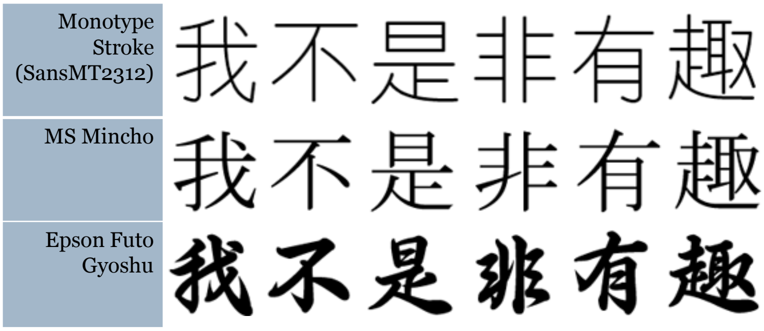
Live Examples

More Details

- Feature extraction is fully automatic
- Extraction requires < 30 sec for ~11,000 glyphs
  - Research code (unoptimized)
  - Can likely be sped up a lot
- Hints occupy ~60 bytes/glyph on average
  - Average glyph outline has ~120 points (~500 bytes)

Stylized Stroke Fonts

- Problems with uniform-width stroke-based fonts
  - Bland and unexpressive
  - Lacking cultural acceptance



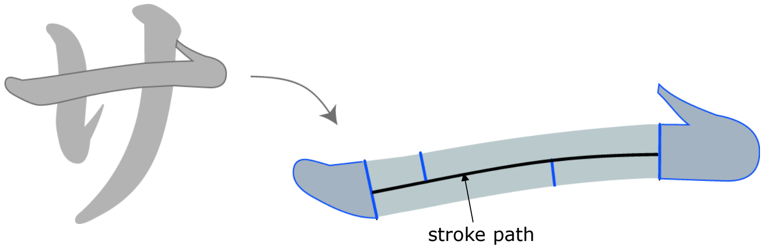
Monotype Stroke (SansMT2312)	我不是非有趣
MS Mincho	我不是非有趣
Epson Futo Gyoshu	我不是非有趣

Stylized Stroke Fonts

- Goal: Best of both worlds
  - Expressiveness of outline-based fonts
  - Memory requirements of stroke-based fonts

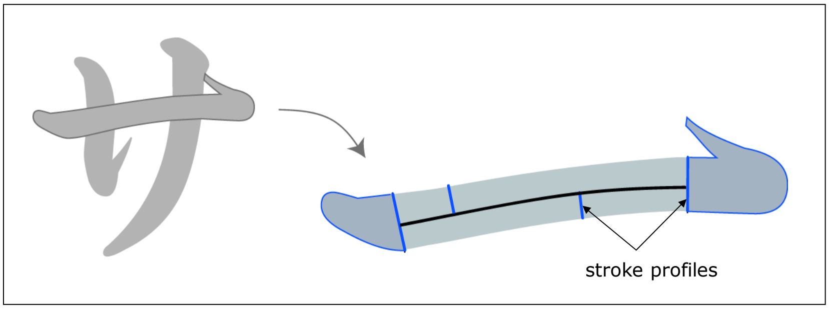
Components

- Stroke path
  - Composed of line segments and Bezier curves
  - Typically runs along the centerline of the stroke



Components

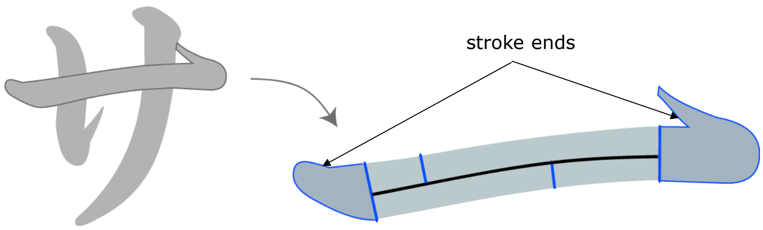
- Stroke profiles
  - Define the shape of the stroke
  - Specify the distance from the stroke path to a stroke edge



Stroke profiles can be one-sided or two-sided

Components

- Stroke end
  - Represented as an outline
  - Determines the shape at each end of a stroke

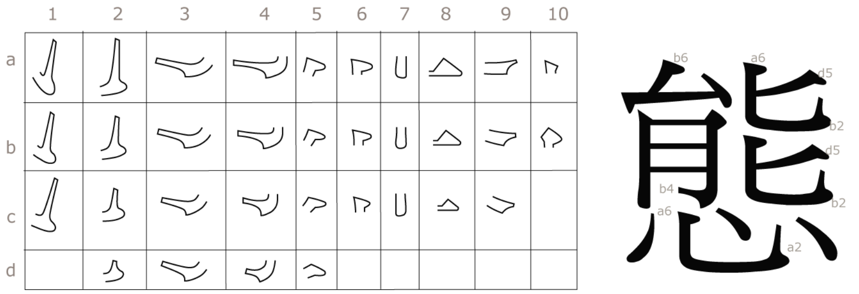


Compression

- How do Stylized Stroke Fonts save memory?

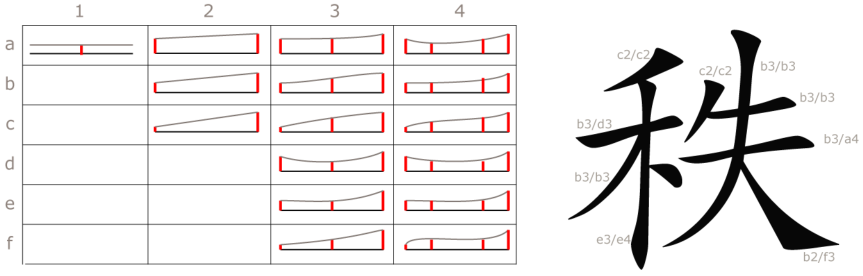
Compression

- Reuse end caps throughout the typeface
  - Transform end caps to match a given stroke body



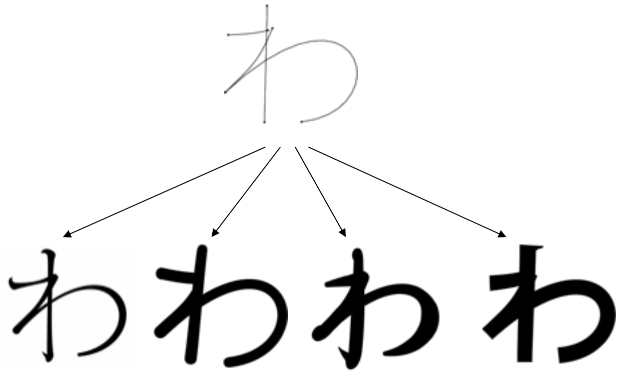
More Compression

- Reuse profile sets throughout the typeface



Even More Compression

- Reuse stroke paths across multiple typefaces



The same stroke path can be used for multiple typefaces

Memory Cost Estimate

- Memory costs for storing a simplified Chinese typeface with 7,663 characters

Representation	Size	Example
Outlines	2.5 mb	
Uniform Stroke Fonts	250 kb	
Stylized Stroke Fonts	338 kb	

Stylized Stroke Fonts add ~25% to uniform stroke fonts for end caps and profile indices and 25kb for storing end caps and profile representations

Demo and Example Images

- Only 1 unique profile
- Only 1 unique endcap
- Memory cost: stroke-based font + a few bytes