INTRODUCTION
- Digital Typography
- Font Types & Formats
- Text Rendering Pipeline

PART I: TEXTURE BASED
- Rasterization
- Fast & Versatile (but ugly)
- Fast & Beautiful (but only 2D)

PART II: DISTANCE BASED
- Signed Distance Fields
- Single Channel
- Arc approximation
- Multiple Channels

PART III: GEOMETRY BASED
- Bézier curves & glyphs
- GPU friendly
- GPU only

CONCLUSION
- Beyond this course
- Questions & answers

25 years of text rendering in computer graphics

Bitmap font (Kilgard, 1994) Stroke font (Kilgard, 1994) Texture font (Kilgard, 1997) Signed Distance Field (Green, 2007)

The Quick Brown Fox Jumps Over The Lazy Dog


Higher 2D Quality (Rougier, 2013) Adaptive Distance Field (Fresken et al., 2000)
INTRODUCTION

Typography is the art of arranging type to make written language legible, readable, and appealing when displayed.

However, for the neophyte, typography is mostly apprehended as the juxtaposition of characters displayed on the screen while for the expert, typography means typeface, scripts, unicode, glyphs, ascender, descender, tracking, hinting, kerning, shaping, weight, slant, etc.

Typography is actually much more than the mere rendering of glyphs and involves many different concepts.
INTRODUCTION
• Digital Typography
• Font Types & Formats
• Text Rendering Pipeline

PART I: TEXTURE BASED
• Rasterization
• Fast & Versatile (but ugly)
• Fast & Beautiful (but only 2D)

PART II: DISTANCE BASED
• Signed Distance Fields
• Single Channel
• Arc approximation
• Multiple Channels

PART III: GEOMETRY BASED
• Bézier curves & glyphs
• GPU friendly
• GPU only

CONCLUSION
• Beyond this course
• Questions & answers

From “Nope, not Arabic” tumbler (https://nopenotarabic.tumblr.com/)
INTRODUCTION
- Digital Typography
- Font Types & Formats
- Text Rendering Pipeline

PART I: TEXTURE BASED
- Rasterization
- Fast & Versatile (but ugly)
- Fast & Beautiful (but only 2D)

PART II: DISTANCE BASED
- Signed Distance Fields
- Single Channel
- Arc approximation
- Multiple Channels

PART III: GEOMETRY BASED
- Bézier curves & glyphs
- GPU friendly
- GPU only

CONCLUSION
- Beyond this course
- Questions & answers
BAD KERNING (keming)

Disaster is a space away... (FLICK, click, CLINT, FINAL)

Part I: Texture Based
- Rasterization
- Fast & Versatile (but ugly)
- Fast & Beautiful (but only 2D)

Part II: Distance Based
- Signed Distance Fields
- Single Channel
- Arc approximation
- Multiple Channels

Part III: Geometry Based
- Bézier curves & glyphs
- GPU friendly
- GPU only

Conclusion
- Beyond this course
- Questions & answers

Click versus dick

Rockstar game removed the extra space

Apple added an extra space
INTRODUCTION
• Digital Typography
• Font Types & Formats
• Text Rendering Pipeline

PART I: TEXTURE BASED
• Rasterization
• Fast & Versatile (but ugly)
• Fast & Beautiful (but only 2D)

PART II: DISTANCE BASED
• Signed Distance Fields
• Single Channel
• Arc approximation
• Multiple Channels

PART III: GEOMETRY BASED
• Bézier curves & glyphs
• GPU friendly
• GPU only

CONCLUSION
• Beyond this course
• Questions & answers

GIBBERISH CAN BE PRETTY...
The Matrix code is actually a sushi recipe...
DIGITAL TYPOGRAPHY

Typography is a vast and complex domain with many rules. You might consider to enforce some basic rules.

How to piss off your designer friends and give them a migraine.

The Anatomy of Typography - Janie Kliever
Unicode is a computing industry standard for the consistent encoding, representation, and handling of text expressed in most of the world’s writing systems. The latest version contains a repertoire of 136,755 characters covering 139 modern and historic scripts, as well as multiple symbol sets.

UTF-8 (Unicode Transformation Format, RFC 3629) is a variable width character encoding capable of encoding all 1,112,064 valid code points in Unicode using one to four 8-bit bytes.
# Famous Fonts

<table>
<thead>
<tr>
<th>Font</th>
<th>Year</th>
<th>Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>Garamond</td>
<td>1495</td>
<td></td>
</tr>
<tr>
<td>Caslon</td>
<td>1734</td>
<td>1798</td>
</tr>
<tr>
<td>Baskerville</td>
<td>1757</td>
<td></td>
</tr>
<tr>
<td>Bodoni</td>
<td>1798</td>
<td></td>
</tr>
<tr>
<td>Gill Sans</td>
<td>1926</td>
<td></td>
</tr>
<tr>
<td>Times New Roman</td>
<td>1931</td>
<td></td>
</tr>
<tr>
<td>Helvetica</td>
<td>1957</td>
<td></td>
</tr>
<tr>
<td>Arial</td>
<td>1982</td>
<td></td>
</tr>
</tbody>
</table>
When text is rendered by a computer, sometimes characters are displayed as “tofu”. They are little boxes to indicate your device doesn’t have a font to display the text. Google has been developing a font family called Noto, which aims to support all languages with a harmonious look and feel. Noto is Google’s answer to tofu.
DIGITAL TYPOGRAPHY
N. ROUGIER (INRIA) & B. ESFAHBOD (GOOGLE)

INTRODUCTION
- Digital Typography
- Font Types & Formats
- Text Rendering Pipeline

PART I: TEXTURE BASED
- Rasterization
- Fast & Versatile (but ugly)
- Fast & Beautiful (but only 2D)

PART II: DISTANCE BASED
- Signed Distance Fields
- Single Channel
- Arc approximation
- Multiple Channels

PART III: GEOMETRY BASED
- Bézier curves & glyphs
- GPU friendly
- GPU only

CONCLUSION
- Beyond this course
- Questions & answers

ITSY BITSY BITMAP FONTS

A Free Font
By
Matthew
Welch

永和九年，時在癸丑，暮春之初，會於會稽山陰之蘭亭，修禊事也。群賢畢至，少長咸集。此地有崇山峻嶺（嶺），茂林修竹；又有清流激湍，映帶左右，引以為流觞曲水，列坐其次。雖無絲竹管弦之盛，一觴一詠，亦足以暢叙幽情。是日也，天朗氣清，惠風和暢。仰觀宇宙之大，俯察品类之盛。所以遊目骋懷，足以極视听之娛，信可樂也。夫人之相與，俯仰一世。或取諸懷抱，悟言一室之內；或因寄所託，放浪形骸之外。雖趣舍萬殊，靜躁不同，當其欣於所遇，暫得於己，快然自足，不知老之將至；及其所之既倦，情隨事遷，感慨系之矣。向之所欣，俯仰之間，已為陳迹，猶不能不以之興懷。況修短隨化，終期於盡。古人云：「死生亦大矣。」豈不痛哉！每覽（覽）昔人興感之由，若合一契，未嘗不臨文嗟悼，不能喻之於懷。固知一死生為虚誕，齊彭釐為妄作。後之視今，亦由（由）今之視昔。悲夫！故列數人，錄其所述，雖世殊事異，所以興懷，其致一也。後之盛（盛）者，亦將有感斯文。
**Basic Typography**

**Introduction**
- Digital Typography
- Font Types & Formats
- Text Rendering Pipeline

**Part I: Texture Based**
- Rasterization
- Fast & Versatile (but ugly)
- Fast & Beautiful (but only 2D)

**Part II: Distance Based**
- Signed Distance Fields
- Single Channel
- Arc approximation
- Multiple Channels

**Part III: Geometry Based**
- Bézier curves & glyphs
- GPU friendly
- GPU only

**Conclusion**
- Beyond this course
- Questions & answers

Kerning is the process of adjusting the spacing between characters in a proportional font.

A ligature occurs where two or more graphemes or letters are joined as a single glyph.

A diacritic mark is a glyph added to a letter, or basic glyph.

A various number of face metrics are defined for all glyphs in a given font.

Kerning is the process of adjusting the spacing between characters in a proportional font.

A ligature occurs where two or more graphemes or letters are joined as a single glyph.

A diacritic mark is a glyph added to a letter, or basic glyph.

A various number of face metrics are defined for all glyphs in a given font.
Text shaping is the process of converting Unicode text to glyph indices and positions.

Stylistic alternate allows to replace a glyph by some variant.

The set of rules applied to produce the correct order at the time of display are described by the Unicode Bidirectional Algorithm.
Digital Typography

N. Rougier (INRIA) & B. Esfahbod (GOOGLE)

Introduction
- Digital Typography
- Font Types & Formats
- Text Rendering Pipeline

Part I: Texture Based
- Rasterization
- Fast & Versatile (but ugly)
- Fast & Beautiful (but only 2D)

Part II: Distance Based
- Signed Distance Fields
- Single Channel
- Arc approximation
- Multiple Channels

Part III: Geometry Based
- Bézier curves & glyphs
- GPU friendly
- GPU only

Conclusion
- Beyond this course
- Questions & answers

Advanced Typography

Anatomy of the Hindi Font by Aditya Dipankar.
OTHER FORMS OF COMPLEX LAYOUT

\[ \sum_{n=1}^{k} \frac{1}{n} > \int_{1}^{k+1} \frac{1}{x} \, dx = \ln(k + 1) \]

\[ \sum_{n=0}^{\infty} \frac{(-1)^n}{2n + 1} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \cdots = \frac{\pi}{4} \]

\[ \int_{a}^{b} f(x) \, dx = F(b) - F(a) \]

\[ f'(a) = \lim_{h \to 0} \frac{f(a + h) - f(a)}{h} \]

\[ (z/w) = \bar{z}/\bar{w} \]

\[ r = |z| = \sqrt{x^2 + y^2} \]

\[ \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 3 \end{bmatrix} \]

Made with LaTeX (what else?)
**FONT FORMATS**

**Types**
- Type 1 (.pfb, .pfr): glyphs are described with cubic Bézier curves
- True Type (.ttf): glyphs are described with quadratic Bézier curves
- Open Type (.otf): glyphs are described with quadratic or cubic Bézier curves
- Web Open Font (.woff): compressed TrueType or Open Type
- And many more actually...

---

**Part I: Texture Based**
- Rasterization
- Fast & Versatile (but ugly)
- Fast & Beautiful (but only 2D)

**Part II: Distance Based**
- Signed Distance Fields
- Single Channel
- Arc approximation
- Multiple Channels

**Part III: Geometry Based**
- Bézier curves & glyphs
- GPU friendly
- GPU only

**Conclusion**
- Beyond this course
- Questions & answers
A PRIMER ON BÉZIER CURVES

Pomax (2017) A primer on Bézier curves

Sergei Natanovich Bernstein (1912) / Paul de Casteljau (1959) / Pierre Bézier (1962)

A Bézier curve is defined by a set of control points \( P_0 \) through \( P_n \), where \( n \) is called its order. The first and last control points are always the end points of the curve.

Linear (n=1): \[ B_{P_0/P_1}(t) = (1-t)P_0 + tP_1 \]

Quadratic (n=2): \[ B_{P_0/P_1/P_2}(t) = (1-t)B_{P_0/P_1}(t) + tB_{P_1/P_2}(t) \]

Cubic(n=3): \[ B_{P_0/P_1/P_2/P_3}(t) = (1-t)B_{P_0/P_1/P_2}(t) + tB_{P_1/P_2/P_3}(t) \]

Some related problems: signed distance, thick curves, subdivision, bounding box, linear speed, self-intersection, curve splitting, arc length, approximation of a cubic with quadratics, etc.
A PRIMER ON BÉZIER CURVES

Pomax (2017) A primer on Bézier curves

Cubic Bézier curves have a lot of corner cases: inflection points, cusp, loop, overlap
Vector fonts are collections of vector images, consisting of lines and curves defining the boundary of glyphs (wikipedia). Type 1 and Type 3 Postscript fonts are described with cubic Bézier curves. Truetype fonts are described with quadratic Bézier curves.
Font Tables

Open Type Tables
- BASE: Baseline data
- CMAP: Character to glyph mapping
- GDEF: Glyph definition data
- GSUB: Glyph substitution data
  - Single: Replaces one glyph with one glyph.
  - Multiple: Replaces one glyph with more than one glyph.
  - Alternate: Replaces one glyph with one of many glyphs.
  - Ligature: Replaces multiple glyphs with one glyph.
  - Context: Replaces one or more glyphs in context.
  - Chaining: Replaces one or more glyphs in chained context.
- GPOS: Glyph positioning data
- JSTF: Justification data

and many more...

Libraries
- STB_truetype (single header file, basic support)
  → https://github.com/nothings/stb
- FreeType (standard support without text shaping)
  → https://www.freetype.org
- HarfBuzz (advanced support with text shaping)
  → https://harfbuzz.github.io
Text Rendering Pipeline

**Introduction**
- Digital Typography
- Font Types & Formats
- Text Rendering Pipeline

**Part I: Texture Based**
- Rasterization
- Fast & Versatile (but ugly)
- Fast & Beautiful (but only 2D)

**Part II: Distance Based**
- Signed Distance Fields
- Single Channel
- Arc approximation
- Multiple Channels

**Part III: Geometry Based**
- Bézier curves & glyphs
- GPU friendly
- GPU only

**Conclusion**
- Beyond this course
- Questions & answers
Near the end of the last century (1997), Mark Kilgard introduced a simple OpenGL-based API for texture mapped text. The method packed many rasterized glyphs into a single alpha-only texture map and used a lookup table to assign texture coordinates to a quadrilateral to extract a glyph when rendering.
2D RASTERIZATION

Same input, different outputs

FreeType Native CFF Rasterizer

FreeType Light Auto Hint Rasterizer

FreeType using the New Adobe CFF Rasterizer (2013)

Quartz Rendering

ClearType Rendering
INTRODUCTION
- Digital Typography
- Font Types & Formats
- Text Rendering Pipeline

PART I: TEXTURE BASED
- Rasterization
- Fast & Versatile (but ugly)
- Fast & Beautiful (but only 2D)

PART II: DISTANCE BASED
- Signed Distance Fields
- Single Channel
- Arc approximation
- Multiple Channels

PART III: GEOMETRY BASED
- Bézier curves & glyphs
- GPU friendly
- GPU only

CONCLUSION
- Beyond this course
- Questions & answers

2D RASTERIZATION
GDI rendering of FacitWeb (above) and Minion Pro (below) with no antialiasing.
(Source Adobe Typekit Blog)
GDI rendering of FacitWeb (above) and Minion Pro (below) with standard antialiasing.
(Source Adobe Typekit Blog)
2D Rasterization

Core text rendering of FacitWeb (above) and Minion Pro (below).
(Source Adobe Typekit Blog)
2D RASTERIZATION

DirectWrite rendering of FacitWeb (above) and Minion Pro (below).
(Source Adobe Typekit Blog)
INTRODUCTION
- Digital Typography
- Font Types & Formats
- Text Rendering Pipeline

PART I: TEXTURE BASED
- Rasterization
- Fast & Versatile (but ugly)
- Fast & Beautiful (but only 2D)

PART II: DISTANCE BASED
- Signed Distance Fields
- Single Channel
- Arc approximation
- Multiple Channels

PART III: GEOMETRY BASED
- Bézier curves & glyphs
- GPU friendly
- GPU only

CONCLUSION
- Beyond this course
- Questions & answers

GDI rendering of FacitWeb (above) and Minion Pro (below) with ClearType enabled.
(Source Adobe Typekit Blog)
2D RASTERIZATION

Gamma correction, hinting, energy distribution, etc.

The quick brown fox jumps over the lazy dog.
The quick brown fox jumps over the lazy dog.
The quick brown fox jumps over the lazy dog.
The quick brown fox jumps over the lazy dog.

No hinting

Native hinting
Texture mapping is well suited for rendering text because textures can be rendered quickly with current 3D hardware and even via clever programming of today's fast CPUs. Textures can be stretched, rotated, scaled, and even projected (assuming the texture mapping is perspective correct) so that texture mapped text looks reasonable in 3D scenes.


https://github.com/rougier/freetype-gl
Several algorithms that can be used to solve the problem of packing rectangles into two-dimensional finite bins. Most of the algorithms have well been studied in literature, but some of the variants are less known and some are apparently regarded as "folklore" and no previous reference is known. Different variants are presented and compared.
When displaying text on low-resolution devices (DPI < 150), one typically has to decide if one wants to respect the pixel grid (e.g., Cleartype technology / Microsoft / native hinting) for crisp rendering or, to privilege glyph shapes (Quartz technology / Apple / no hinting) at the cost of blurring. There is, however, a third way that may combine the best of the two technologies (vertical hinting).
TEXTURE FONT II

Rougier (2013) Higher Quality 2D Text Rendering

i. Use horizontal RGB sub-pixel anti-aliasing for LCD flat panels.
ii. Use vertical hinting only and completely discard the horizontal one.
iii. Use accurate glyph advance values from unhinted glyph
iv. Use accurate, high resolution values from the kerning table.

Rougier (2013) Higher Quality 2D Text Rendering

i. Use horizontal RGB sub-pixel anti-aliasing for LCD flat panels.
ii. Use vertical hinting only and completely discard the horizontal one.
iii. Use accurate glyph advance values from unhinted glyph
iv. Use accurate, high resolution values from the kerning table.
TEXTURE FONT II

Rougier (2013) Higher Quality 2D Text Rendering

Subpixel rendering on the CPU
Subpixel positioning on the GPU

\[
\begin{align*}
0 < s \leq 1/3, t = 3s & \quad R_{out} = tB_{left} + (1-t)R \\
G_{out} = tR + (1-t)G \\
B_{out} = tG + (1-t)B \\

1/3 < s \leq 2/3, t = 3s - 1 & \quad R_{out} = tG_{left} + (1-t)B_{left} \\
G_{out} = tB_{left} + (1-t)R \\
B_{out} = tR + (1-t)G \\

2/3 < s < 1, t = 3s - 2 & \quad R_{out} = tR_{left} + (1-t)G_{left} \\
G_{out} = tG_{left} + (1-t)B_{left} \\
B_{out} = tB_{left} + (1-t)R
\end{align*}
\]

uniform sampler2D texture;
uniform vec2 pixel;
varying float shift;
void main()
{
    vec2 uv = gl_TexCoord[0].xy;
    vec4 current = texture2D(texture, uv);
    vec4 previous = texture2D(texture, uv+vec2(1,0) * pixel);
    float r = current.r;
    float g = current.g;
    float b = current.b;
    float a = current.a;
    if(shift < 1.0/3.0)
    {
        float z = 3.0*shift;
        r = mix(current.r, previous.b, z);
        g = mix(current.g, current.r, z);
        b = mix(current.b, current.g, z);
    }
    else if(shift < 2.0/3.0)
    {
        float z = 3.0*shift-1.0;
        r = mix(previous.b, previous.g, z);
        g = mix(current.r, previous.b, z);
        b = mix(current.b, current.g, z);
    }
    else if(shift < 1.0)
    {
        float z = 3.0*shift-2.0;
        r = mix(previous.g, previous.r, z);
        g = mix(previous.b, previous.g, z);
        b = mix(current.r, previous.b, z);
    }
    gl_FragColor = vec3(r,g,b,a);
}
PART II

Distance based rendering takes advantage of a signed distance function (that can be approximated) to compute the individual coverage for each pixel.

The coverage can be selected to implement various effects (stroke, thinner, thicker, shadow, etc).
Distance based rendering takes advantage of a signed distance function (that can be approximated) to compute the individual coverage for each pixel. The coverage can be selected to implement various effects (stroke, thinner, thicker, shadow, etc).
Computing SDF

**Gustavson & Strand (2011) Anti-Aliased Euclidean distance transform**

We present a modified distance measure for use with distance transforms of anti-aliased, area sampled grayscale images of arbitrary binary contours. The modified measure can be used in any vector-propagation Euclidean distance transform. Our test implementation in the traditional SSED8 algorithm shows a considerable improvement in accuracy and homogeneity of the distance field compared to a traditional binary image transform.

See also [http://contourtextures.wikidot.com](http://contourtextures.wikidot.com)
A distance field is generated from a high resolution image, and then stored into a channel of a lower-resolution texture. In the simplest case, this texture can then be rendered simply by using the alpha-testing and alpha-thresholding feature of modern GPUs, without a custom shader. This allows the technique to be used on even the lowest-end 3D graphics hardware. With the use of programmable shading, the technique is extended to perform various special effect renderings, including soft edges, outlining, drop shadows, multi-colored images, and sharp corners.
SDF: Multiple Channels

Chlumsky (2015) Shape Decomposition for Multi-channel Distance Fields

- **Part I: Texture Based**
  - Rasterization
  - Fast & Versatile (but ugly)
  - Fast & Beautiful (but only 2D)

- **Part II: Distance Based**
  - Signed Distance Fields
  - Single Channel
  - Arc approximation
  - Multiple Channels

- **Part III: Geometry Based**
  - Bézier curves & glyphs
  - GPU friendly
  - GPU only

**Conclusion**
- Beyond this course
- Questions & answers
PART I: TEXTURE BASED
• Rasterization
• Fast & Versatile (but ugly)
  • Fast & Beautiful (but only 2D)

PART II: DISTANCE BASED
• Signed Distance Fields
  • Single Channel
  • Arc approximation
  • Multiple Channels

PART III: GEOMETRY BASED
• Bézier curves & glyphs
  • GPU friendly
  • GPU only

CONCLUSION
• Beyond this course
• Questions & answers

SDF: MULTIPLE CHANNELS

Chlumsky (2015) Shape Decomposition for Multi-channel Distance Fields

Texture Single channel SDF Multi channel SDF
Since distance to arbitrary Bézier curves is hard, we can instead first convert a glyph into (approximated) circular arc splines and upload this converted vector glyph to the GPU. Distance are then computed onto the GPU. Corner cases are overlapping contours, tangent arcs and float precision.
**ARC APPROXIMATION**

Esfahbod (2012) Glyph

SDF-based antialiasing + subpixel positioning + random access (coarse grid).
ARC APPROXIMATION

Esfahbod (2012) Glyphy

Drawback: Memory and speed are font dependent
PART III

Charles Loop and Jim Blinn introduced in 2005 a new approach for resolution-independent rendering of quadratic and cubic spline curves.

By tessellating a glyph the proper way, they offered de facto a method for resolution independent rendering of a glyph with good rendering quality.
We present a method for resolution independent rendering of paths and bounded regions, defined by quadratic and cubic spline curves, that leverages the parallelism of programmable graphics hardware to achieve high performance. Our result is a mechanism for rendering vector geometry that has the following properties: resolution independence, compact geometric representation, high performance.
We determine if the pixel is inside or outside the curve by evaluating $f(u, v) = u^2 - v$ in a pixel shader program. If $f(u, v) < 0$ then the pixel is inside the curve, otherwise it is outside.

See also
"Rendering Vector Art on the GPU" (Charles Loop & Jim Blinn, GPU Gems 3, Chapter 25)
"Easy Scalable Text Rendering on the GPU" (Evan Wallace Medium)
This sketch presents a new method for resolution independent rendering of vector images suitable for programmable graphics hardware. We have enhanced a previous method [Loop and Blinn 2005] by using a stencil buffer and transparency multisampling.
This sketch presents a new method for resolution independent rendering of vector images suitable for programmable graphics hardware. We have enhanced a previous method [Loop and Blinn 2005] by using a stencil buffer and transparency multisampling.

We present a vector graphics representation suitable for real-time rendering on GPUs. Our representation can be used in place of a texture map, and renders precise antialiased edges at any magnification. A combination of texture data and procedural computation is used to evaluate an exact signed distance to a contour and its gradient.

Anisotropic antialiasing technique + GPU-based representation of contours + Packed grid accelerator structure based + Sprite mapping technique + Special effects
We introduce a two-step “Stencil, then Cover” (StC) programming interface. Our GPU-based approach builds upon existing techniques for curve rendering using the stencil buffer, but we explicitly decouple in our programming interface the stencil step to determine a path's filled or stroked coverage from the subsequent cover step to rasterize conservative geometry intended to test and reset the coverage determinations of the first step while shading color samples within the path.
This paper describes a method for rendering antialiased text directly from glyph outline data on the GPU without the use of any precomputed texture images or distance fields. This capability is valuable for text displayed inside a 3D scene because, in addition to a perspective projection, the transform applied to the text is constantly changing with a dynamic camera view. Our method overcomes numerical precision problems that produced artefacts in previously published techniques and promotes high GPU utilization with an implementation that naturally avoids divergent branching.

See demo at sluglibrary.com
See also GPU-Centered font rendering & GPU font rendering

<table>
<thead>
<tr>
<th>Font</th>
<th>Sample</th>
<th>Complexity</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arial</td>
<td>ABCDEFG</td>
<td>20</td>
<td>1.1 ms</td>
</tr>
<tr>
<td>Centaur</td>
<td>ABCDEFG</td>
<td>48</td>
<td>1.3 ms</td>
</tr>
<tr>
<td>Halloween</td>
<td>ABCDEFG</td>
<td>72</td>
<td>3.8 ms</td>
</tr>
<tr>
<td>Wildwood</td>
<td>ABCDEFG</td>
<td>546</td>
<td>13.3 ms</td>
</tr>
</tbody>
</table>
Walton (2017) Pathfinder, a fast GPU-based font rasterizer in Rust

“Today I’m pleased to announce Pathfinder, a Rust library for OpenType font rendering. The goal is nothing less than to be the fastest vector graphics renderer in existence, and the results so far are extremely encouraging. Not only is it very fast according to the traditional metric of raw rasterization performance, it’s practical, featuring very low setup time (end-to-end time superior to the best CPU rasterizers), best-in-class rasterization performance even at small glyph sizes, minimal memory consumption (both on CPU and GPU), compatibility with existing font formats, portability to most graphics hardware manufactured in the past five years (DirectX 10 level), and security/safety.”
Walton (2017) PathFinder, a fast GPU-based font rasterizer in Rust

PathFinder algorithm takes advantage of compute shaders (GL 4.3) & post-transform cache and rasterization occurs on the fly on the GPU. CPU setup time is minimal. Subpixel antialiasing & positioning, 2D, 3D, outline, etc.

It is mostly a (highly efficient) translation of CPU rasterization onto the GPU.
BEYOND MERE GEOMETRY

Maharik et al (2011) Digital Micrography

We present an algorithm for creating digital micrography images, or micrograms, a special type of calligrams created from minuscule text. These attractive text-art works successfully combine beautiful images with readable meaningful text.
CONCLUSION

Many available techniques. Choice is dependent of usage (2D or 3D, dynamic or static, memory vs speed vs accuracy, etc).

PathFinder and Slug library seem to be the main players in 2018. Loop & Blinn still competitive.

Hinting and anti-aliasing still needed until we (all) get screen with dpi > 600.

Complex text layout is (really, really) difficult, you don’t want to do it yourself: github.com/HOST-Oman/libraqm

Lot of patents around!
FURTHER READING

Web

- Texts Rasterization Exposures - Maxim Shemanarev (2006)
- The Technology of Text - Kevin Larson (2007)
- Treatise on Font Rasterisation - Freddie Witherden (2010)
- CS 354 Typography - Mark Kilgard (2012)
- Android's Font Renderer - Romain Guy (2014)
- Complex Text on Simple Devices - Pedro Navarro (2016)
- Introducing DirectWrite - Microsoft (2017)
- A Primer on Bézier curves - Pomax (2017)

Papers

- Ron Maharik et al. Digital Micrography (2011)
- Stefan Gustavson. 2D Shape Rendering by Distance Fields (2012)
- Mark Kilgard and Jeff Bolz. GPU-Accelerated Path Rendering (2012)
- Viktor Chlumsky. Shape Decomposition for Multi-channel Distance Fields (2015)
- Eric Lengyel. GPU-Centered Font Rendering Directly from Glyph Outlines (2017)
- Patrick Walton. Path Finder (2017)
INTRODUCTION

• Digital Typography
• Font Types & Formats
• Text Rendering Pipeline

PART I: TEXTURE BASED

• Rasterization
• Fast & Versatile (but ugly)
• Fast & Beautiful (but only 2D)

PART II: DISTANCE BASED

• Signed Distance Fields
• Single Channel
• Arc approximation
• Multiple Channels

PART III: GEOMETRY BASED

• Bézier curves & glyphs
• GPU friendly
• GPU only

CONCLUSION

• Beyond this course
• Questions & answers

RELATED SHADERTOYS

Hello Shadertoy this is my proportional font. I hope you like it. Feel free to use it.
abcdefghijklmnopqrstuvwxyz
ABCDEFGHIJKLMNOPQRSTUVWXYZ
1234567890.!?:-+=*/<>&@#$%^&*_(){}[]

www.shadertoy.com/view/MdycWh
www.shadertoy.com/view/Mt2GMD
www.shadertoy.com/view/4s3XDn

www.shadertoy.com/view/4sVyWh
www.shadertoy.com/view/ldXyRn
www.shadertoy.com/view/4dfXDn

www.shadertoy.com/view/4ts3DB
www.shadertoy.com/view/4s3XDn
www.shadertoy.com/view/4dfXDn
Questions?

- Where to find good fonts? dafont.com, fontsquirrel.com
- When does cleartype patent end? Not clear (ha ha)
- Are hinting and aliasing still needed? Yes
- What are the most complete font families? Noto, DejaVu
- How many languages in the world? Around 7000
- How many scripts in latest unicode (11.0)? 146
- How many characters in latest unicode (11.0)? 137,439
- Is there any forthcoming support for text in Vulkan? No
- Should I tell my colleagues I love Comic Sans? No
- Are there any open source tools to design fonts? FontForge
- Why “ff” & “fi” disappear when I copy text? Ligatures
- Should I apply AA before or after gamma correction? Before
- Should I stay or should I go? stay (a few more minutes)

Any other questions?
DIGITAL TYPOGRAPHY
N. ROUGIER (INRIA) & B. ESFAHBOD (GOOGLE)

INTRODUCTION
• Digital Typography
• Font Types & Formats
• Text Rendering Pipeline

PART I: TEXTURE BASED
• Rasterization
• Fast & Versatile (but ugly)
• Fast & Beautiful (but only 2D)

PART II: DISTANCE BASED
• Signed Distance Fields
• Single Channel
• Arc approximation
• Multiple Channels

PART III: GEOMETRY BASED
• Bézier curves & glyphs
• GPU friendly
• GPU only

CONCLUSION
• Beyond this course
• Questions & answers

DIGITAL TYPOGRAPHY
Between art & design.

Meet the cast:
ABCD
EFGHIJK
LMNOP
QRSTU
VWX

Now see the movie:
Helvetica
A documentary film by Gary Hustwit
Gary Hustwit

Ragheed Abu Hamdan

Unknown artist

devanagari