About the speaker

- Working in game/graphics dev since 1994
  - Previously at Sierra, Apple, Naughty Dog

- Current projects:
  - Slug Library, C4 Engine, The 31st, FGED
About this talk

- Unicode
- Glyphs
- TrueType
- Font Rendering
- Typography
Unicode

- Defines character codes
- Originally 16-bit
- Now has range \(0x000000 - 0x10FFFF\)
- Divided into 17 “planes”
Basic Multilingual Plane

- 0x0000 – 0xFFFF
- First 128 code points are ASCII
- Lots of other common scripts/languages
- Lots of common symbols
# Basic Multilingual Plane

<table>
<thead>
<tr>
<th>00</th>
<th>01</th>
<th>02</th>
<th>03</th>
<th>04</th>
<th>05</th>
<th>06</th>
<th>07</th>
<th>08</th>
<th>09</th>
<th>0A</th>
<th>0B</th>
<th>0C</th>
<th>0D</th>
<th>0E</th>
<th>0F</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>1A</td>
<td>1B</td>
<td>1C</td>
<td>1D</td>
<td>1E</td>
<td>1F</td>
</tr>
<tr>
<td>20</td>
<td>21</td>
<td>22</td>
<td>23</td>
<td>24</td>
<td>25</td>
<td>26</td>
<td>27</td>
<td>28</td>
<td>29</td>
<td>2A</td>
<td>2B</td>
<td>2C</td>
<td>2D</td>
<td>2E</td>
<td>2F</td>
</tr>
<tr>
<td>30</td>
<td>31</td>
<td>32</td>
<td>33</td>
<td>34</td>
<td>35</td>
<td>36</td>
<td>37</td>
<td>38</td>
<td>39</td>
<td>3A</td>
<td>3B</td>
<td>3C</td>
<td>3D</td>
<td>3E</td>
<td>3F</td>
</tr>
<tr>
<td>40</td>
<td>41</td>
<td>42</td>
<td>43</td>
<td>44</td>
<td>45</td>
<td>46</td>
<td>47</td>
<td>48</td>
<td>49</td>
<td>4A</td>
<td>4B</td>
<td>4C</td>
<td>4D</td>
<td>4E</td>
<td>4F</td>
</tr>
<tr>
<td>50</td>
<td>51</td>
<td>52</td>
<td>53</td>
<td>54</td>
<td>55</td>
<td>56</td>
<td>57</td>
<td>58</td>
<td>59</td>
<td>5A</td>
<td>5B</td>
<td>5C</td>
<td>5D</td>
<td>5E</td>
<td>5F</td>
</tr>
<tr>
<td>60</td>
<td>61</td>
<td>62</td>
<td>63</td>
<td>64</td>
<td>65</td>
<td>66</td>
<td>67</td>
<td>68</td>
<td>69</td>
<td>6A</td>
<td>6B</td>
<td>6C</td>
<td>6D</td>
<td>6E</td>
<td>6F</td>
</tr>
<tr>
<td>70</td>
<td>71</td>
<td>72</td>
<td>73</td>
<td>74</td>
<td>75</td>
<td>76</td>
<td>77</td>
<td>78</td>
<td>79</td>
<td>7A</td>
<td>7B</td>
<td>7C</td>
<td>7D</td>
<td>7E</td>
<td>7F</td>
</tr>
<tr>
<td>80</td>
<td>81</td>
<td>82</td>
<td>83</td>
<td>84</td>
<td>85</td>
<td>86</td>
<td>87</td>
<td>88</td>
<td>89</td>
<td>8A</td>
<td>8B</td>
<td>8C</td>
<td>8D</td>
<td>8E</td>
<td>8F</td>
</tr>
<tr>
<td>90</td>
<td>91</td>
<td>92</td>
<td>93</td>
<td>94</td>
<td>95</td>
<td>96</td>
<td>97</td>
<td>98</td>
<td>99</td>
<td>9A</td>
<td>9B</td>
<td>9C</td>
<td>9D</td>
<td>9E</td>
<td>9F</td>
</tr>
<tr>
<td>A0</td>
<td>A1</td>
<td>A2</td>
<td>A3</td>
<td>A4</td>
<td>A5</td>
<td>A6</td>
<td>A7</td>
<td>A8</td>
<td>A9</td>
<td>AA</td>
<td>AB</td>
<td>AC</td>
<td>AD</td>
<td>AE</td>
<td>AF</td>
</tr>
<tr>
<td>B0</td>
<td>B1</td>
<td>B2</td>
<td>B3</td>
<td>B4</td>
<td>B5</td>
<td>B6</td>
<td>B7</td>
<td>B8</td>
<td>B9</td>
<td>BA</td>
<td>BB</td>
<td>BC</td>
<td>BD</td>
<td>BE</td>
<td>BF</td>
</tr>
<tr>
<td>C0</td>
<td>C1</td>
<td>C2</td>
<td>C3</td>
<td>C4</td>
<td>C5</td>
<td>C6</td>
<td>C7</td>
<td>C8</td>
<td>C9</td>
<td>CA</td>
<td>CB</td>
<td>CC</td>
<td>CD</td>
<td>CE</td>
<td>CF</td>
</tr>
<tr>
<td>D0</td>
<td>D1</td>
<td>D2</td>
<td>D3</td>
<td>D4</td>
<td>D5</td>
<td>D6</td>
<td>D7</td>
<td>D8</td>
<td>D9</td>
<td>DA</td>
<td>DB</td>
<td>DC</td>
<td>DD</td>
<td>DE</td>
<td>DF</td>
</tr>
<tr>
<td>E0</td>
<td>E1</td>
<td>E2</td>
<td>E3</td>
<td>E4</td>
<td>E5</td>
<td>E6</td>
<td>E7</td>
<td>E8</td>
<td>E9</td>
<td>EA</td>
<td>EB</td>
<td>EC</td>
<td>ED</td>
<td>EE</td>
<td>EF</td>
</tr>
<tr>
<td>F0</td>
<td>F1</td>
<td>F2</td>
<td>F3</td>
<td>F4</td>
<td>F5</td>
<td>F6</td>
<td>F7</td>
<td>F8</td>
<td>F9</td>
<td>FA</td>
<td>FB</td>
<td>FC</td>
<td>FD</td>
<td>FE</td>
<td>FF</td>
</tr>
</tbody>
</table>

- **Latin script**
- **Non-Latin European scripts**
- **African scripts**
- **Middle Eastern and Southwest Asian scripts**
- **South and Central Asian scripts**
- **Southeast Asian scripts**
- **East Asian scripts**
- **CJK characters**
- **Indonesian and Oceanic scripts**
- **American scripts**
- **Notational systems**
- **Symbols**
- **Private use**
- **UTF-16 surrogates**
- **Unallocated code points**

As of Unicode 10.0
Supplementary Multilingual Plane

- 0x010000 – 0x01FFFF
- Rare characters from many languages
- Rare scripts like Cuneiform and Hieroglyphs
- Mathematical symbols and bold / italic
- Emoticons 😊
Supplementary Multilingual Plane

As of Unicode 10.0
Supplementary Ideographic Plane

- 0x020000 – 0x02FFFF
- Less common CJK ideographs
Supplementary Ideographic Plane

<table>
<thead>
<tr>
<th>Code Point</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>200-20F</td>
<td>CJK characters</td>
</tr>
<tr>
<td>210-21F</td>
<td>Unallocated code points</td>
</tr>
<tr>
<td>220-22F</td>
<td>As of Unicode 10.0</td>
</tr>
</tbody>
</table>

As of Unicode 10.0
Other Planes

- Planes 0x03 – 0x0D unused
- Plane 0x0E contains special tags and variation selectors
- Planes 0x0F and 0x10 for private use only
Character Encoding

- ASCII
- UCS-2 (Universal Coded Character Set)
  - Always 16 bits per character
- UTF-16
  - 16 bits or 32 bits per character
- UTF-32
  - Always 32 bits per character
UTF-8

- 1 – 4 bytes per character
  - Using variable-length encoding

- Values 0x00 – 0x7F identical to ASCII

- High bit set indicates part of multi-byte sequence
UTF-8

- 1 byte: 0x00 – 0x7F
- 2 bytes: 0x0080 – 0x07FF
- 3 bytes: 0x0800 – 0xFFFF
- 4 bytes: 0x010000 – 0x10FFFF
Glyphs

- Fonts contain glyphs
- Glyphs have font-specific internal numbering
- Fonts contain tables that map character codes (Unicode values) to glyph indexes
Glyphs

- Fonts typically contain many more glyphs that are not directly mapped from characters
  - Type variations
  - Alternate styles
  - Ligatures, ZWJ sequences
  - Initial, medial, final forms (Arabic)
- More about these later
TrueType

- Contains resolution-independent representations of glyph outlines
- Has character-to-glyph mappings
- Usually contains several other tables with typographic information (e.g., kerning)
Glyph Outline

- Glyph defined by one or more closed contours
- Each contour defined by continuous sequence of quadratic Bézier curves
- Winding number determines whether a given point is inside the glyph
Winding Number

- Contours defining outer edge of glyph wound in one direction (either CW or CCW is okay)
- Contours defining a hole in the glyph wound in the opposite direction
Winding Number

- Count number of positive loops for outer contours
- Subtract number of negative loops for inner contours
- Nonzero means point inside glyph boundary
Glyph Outline / Winding Number
Font Rendering in Games

- Text rendered in lots of places
  - GUI: Buttons, menus, ...
  - HUD: Score, health, ammo, ...
  - In scene: Signs, labels, computer screens, ...
  - Debug info: Console, stats, timings, ...
GPU Font Rendering: Current State of the Art

May 7, 2018
Irvine, California
Basic GPU Font Rendering

- Rasterize each glyph on CPU and store results in a texture map called an “atlas”

- Can be done for multiple font sizes at once

- Packing methods can vary in sophistication
Font Atlases
Font Atlases

- Render one quad for each glyph
- Texture map the glyph’s image from the atlas
- Very simple and stupid fast
Font Atlases

- Very limited quality
- Only looks good at originally rendered size
- Magnification looks terrible
Font Atlases

- Minification also problematic

- Mipmaps work to a degree

- Glyphs must be surrounded by empty space in atlas to prevent bleeding into neighbors
Signed Distance Fields

- Instead of storing glyph images in atlas, store distance to glyph outline at each point.

Image credit: Konstantin Käfer, "Drawing Text with Signed Distance Fields in Mapbox GL", 2014.
Signed Distance Fields

- Render linear coverage by scaling distance to pixel units and clamping
- Requires derivatives in pixel shader and extra computation
- Still very fast
Signed Distance Fields

- Addresses magnification problem
- Also allows good perspective rendering
Signed Distance Fields

- Need high resolution to capture glyph details
- Sharp corners always rounded off
  - Can be addressed with multiple distance channels
- Minification becomes bigger problem
  - Because one distance value can’t account for multiple curves in scaled-down field
Signed Distance Fields

Resolution Independence

- Render directly from original outline data
  - Control points for quadratic Bézier curves

- No more texture atlases!
  - No resolution-dependent approximation
  - Impossible to lose detail
Loop-Blinn Method

- Creates a triangulation for each glyph using its outline control points
- Each triangle corresponds to one Bézier curve
- Simple calculation based on interpolated texture coordinates yields inside/outside state
Loop-Blinn Method

Loop-Blinn Method

- Needs further subdivision for interior triangles so they never border more than one curve

- Correct antialiasing also requires more triangles in the exterior
  - Consider a pixel intersecting the outline but without its center covered by a triangle
Loop-Blinn Method

Loop-Blinn Method

- Requires a large number of triangles for each glyph
- More complex glyphs could require 1000s!
- Calculation of triangles is complex
Loop-Blinn Method

- Produces high-quality magnification

- However, minification is poor
  - Any pixel is covered by at most one triangle
  - Each triangle corresponds to only one curve
  - Thus, it’s impossible for one pixel to consider contribution from multiple nearby curves
Dobbie Method

- Covers each glyph with a single quad
- Pixel shader considers subset of all Bézier curves to determine winding number
- Basically ray tracing glyphs
Dobbie Method

- For a given point, shoot a ray outward and count curve intersections
- An intersection makes a positive or negative contribution based on its winding direction
- Nonzero total means inside glyph boundary
Dobbie Method

- Antialiasing possible along ray direction

- If intersection occurs within pixel, it makes a fractional contribution

- Test rays in multiple directions and average to get isotropic antialiasing
Dobbie Method
Dobbie Method

- Very slow to test all Bézier curves defining the glyph for each ray
- Dobbie method divides glyph’s bounding box into cells
- Each cell has list of intersecting curves
Dobbie Method

Dobbie Method

- Pixel footprint could overlap multiple cells
  - Have to sort that out in pixel shader

- Need to precompute whether cell center inside or outside glyph boundary
  - Then trace extra ray from pixel location to cell center to fix up winding number
Dobbie Method

- There’s a serious problem:
- Numerical robustness
- Floating-point round-off error causes rendering artifacts
Dobbie Method

Sparkle / streaking artifacts
Glyphy

- Similar to Dobbie method in that a glyph is covered by a single quad
- Pixel shader determines distance to nearest Bézier curve
Glyphy

- Original outlines not preserved
- Also has numerical robustness problems
GPU Font Rendering: Current State of the Art

May 7, 2018
Irvine, California

Glyphy

Straight lines rounded

Sparkle artifacts
GPU Font Rendering: Current State of the Art

May 7, 2018
Irvine, California
Slug Library

- The result of my own research begun in 2016
- Uses one quad per glyph
- Calculates winding number in pixel shader
- Has *perfect* numerical robustness
Numerical Robustness

- Round-off errors in previous methods:
  - Generally come from determining whether roots of ray-curve intersections fall in \([0,1]\) range
  - Problems typically occur at the endpoints
  - Especially bad when ray nearly tangent to curve
  - Hacks like using epsilons or perturbing coordinates just shift the problem cases around
Numerical Robustness

- Only way to solve is to completely eliminate the [0,1] range test

- Slug introduces an equivalence class algorithm
  - Equivalence class represents control point state
  - Same actions taken for all cases in same class
Equivalence Classes

- With respect to a given ray, a particular quadratic Bézier curve is classified into one of 8 possible equivalence classes.

- Based on which side of ray each of three control points falls, positive or negative:
  - Exactly on ray is considered positive.
Equivalence Classes

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Equivalence Classes A" /></td>
<td><img src="image2.png" alt="Equivalence Classes B" /></td>
</tr>
<tr>
<td><img src="image3.png" alt="Equivalence Classes C" /></td>
<td><img src="image4.png" alt="Equivalence Classes D" /></td>
</tr>
<tr>
<td><img src="image5.png" alt="Equivalence Classes E" /></td>
<td><img src="image6.png" alt="Equivalence Classes F" /></td>
</tr>
<tr>
<td><img src="image7.png" alt="Equivalence Classes G" /></td>
<td><img src="image8.png" alt="Equivalence Classes H" /></td>
</tr>
</tbody>
</table>
Equivalence Classes

- For each Bézier curve, always calculate roots

\[(y_1 - 2y_2 + y_3)t^2 - 2(y_1 - y_2)t + y_1\]

\[t_1 = \frac{b - \sqrt{b^2 - ac}}{a}\quad \text{and} \quad t_2 = \frac{b + \sqrt{b^2 - ac}}{a}\]

\[a = y_1 - 2y_2 + y_3 \quad b = y_1 - y_2 \quad c = y_1\]
Equivalence Classes

- A 16-bit LUT tells us what to do with roots for each equivalence class (8 classes x 2 roots)

- Action taken only when x coordinate positive at a root, meaning intersection was on ray
Winding Number

- 1 in LUT for first root means add one
- 1 in LUT for second root means subtract one

- Total after considering all curves is winding number at pixel location
- Fractional values used when roots within pixel distance of ray origin
Antialiasing

- Result is coverage value with perfect one-dimensional antialiasing
- Evaluate horizontal and vertical rays
- Combine to produce 2D antialiasing
Banding

- For best performance, we want to minimize number of curves tested

- Cells don’t work well
  - Pixel footprint can cover multiple cells
  - Pixels get larger as font size decreases
Banding

- Instead of cells, use horizontal and vertical bands that extend to infinity
Banding

- Bézier curves are sorted into the bands
  - A curve can belong to multiple bands
  - When rendering, band selected based on ray origin

- Doesn’t matter how large pixel footprints get
  - Only matters in ray direction
  - Band parallel to ray extends forever
Banding

- Curves in each band are sorted to allow early exit in pixel shader

- Once right-pointing ray’s origin is beyond maximum curve x coordinate, we’re done
Banding

- Curves sorted in both directions
- Ray points left or right depending on pixel position within a band
- Reduces number of curves tested
Banding

- We want worst-case band to contain fewest curves possible

- GPU thread coherence will make shader wait for longest number of loop iterations in a group of pixels (32 or 64)
Banding

- Use large number of bands
- Merge those with equal subsets of Bézier curves
Minification

- High-quality minification achieved with adaptive supersampling
  - Based on screen-space derivatives

- Already have perfect 1D antialiasing
  - Take $n$ samples in $x$ and $y$ directions
  - Produces better than $n \times n$ supersampling
Minification
Font Data

- Two texture maps, data only (no images)
- Curve texture, 4 x 16-bit float
  - Contains all Bézier curves
- Band texture, 4 x 16-bit integer
  - Contains curve subsets for all bands
Multicolor Glyphs

- Microsoft fonts use vector data for color emoji
- Layered glyphs with color palette
- Easy to handle with loop in pixel shader
Typography

- Slug algorithm can make individual glyphs look great at any scale or from any perspective

- Higher-level:
  Make entire lines of text look good
Metrics

- advance width
- bounding box
- em square

Illustration of a letter A with coordinates (0,0) for the bottom left corner and (1,1) for the top right corner.
Metrics

- **em size**
- **baseline**
- **leading**
- **baseline**
- **cap height**
- **ascent**
- **ex height**
- **descent**
Kerning

- Some pairs of glyphs appear to the eye to have too much space in between
- Fonts usually contain kerning tables to improve overall appearance
Kerning

“Too Wavy.”

“Too Wavy.”

Kerning off

Kerning on
Ligatures

- Replaces a sequence of glyphs with one new glyph that looks better

- In some languages, ligatures that change appearance are required for correctness
Ligatures

The firefly craft.
The firefly craft.

Normal text

With ligatures
ZWJ Sequences

- Unicode has control character “zero-width joiner” (ZWJ)
- Often used by fonts for combining several glyphs into single ligature
ZWJ Sequences
Combining Marks

- Unicode defines many accents and other symbols that are designed to combine with a preceding base character.

- Fonts determine how this combination works by defining attachment points.
Combining Marks

\[ \text{ä} \quad \hat{\text{a}} \]
Alternate Substitution

- OpenType fonts define a large array of substitution features
- Independent of Unicode
- Not directly accessible through characters
Alternate Substitution

- Small caps
- Subscripts and superscripts
- Case-sensitive forms
- Stylistic alternates
- Tabular and proportional figures
- Lining and old-style figures
Small Caps

Text

Text

Small caps alternates

Scaled glyphs
Lining and Old-style Figures

0123456789

0123456789
Cursive Joining

- In languages like Arabic, letters have multiple forms depending on position in word

- Isolated, initial, medial, final forms

- Do not have separate character codes
Cursive Joining

متقدمة تقدمية الخط وتخطيط النص

تقدم تقنيم الخط وتخطيط النص
Materials

- Rendering glyphs outputs coverage value
  - (Plus color for multi-color emoji)

- Can be combined with other materials in game
Materials
Contact / More Info

- lengyel@terathon.com
- Twitter: @EricLengyel
- sluglibrary.com