Optimizing Glyphs for Real-Time Vector Rendering

Eric Lengyel, Ph.D.

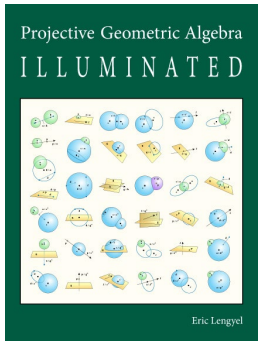
Unicode Technology Workshop

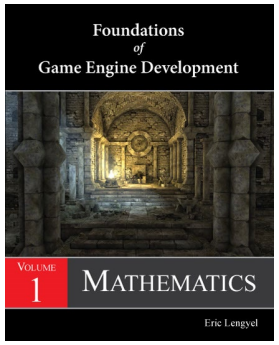
Sunnyvale CA

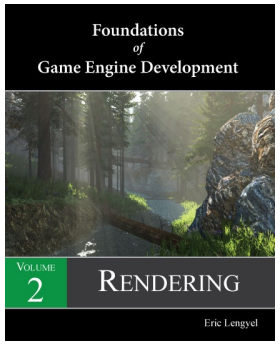
October 22, 2024

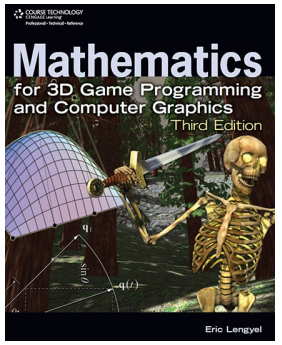
About the Speaker

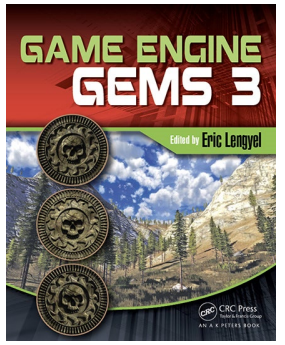
- Computer Scientist / Mathematician
- Working in industry since 1994
- Writes books about math and real-time rendering
- Lifetime member of Unicode Consortium
- Creator of Slug Library for GPU font rendering











Slug Library

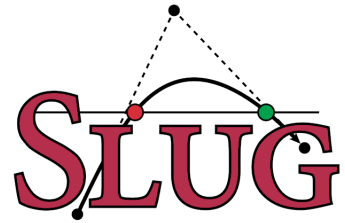
- Renders fonts and vector graphics on the GPU directly from original Bézier curves

- Used across a wide array of applications including VR, CAD, games, industrial modeling, planetariums, video editing

What Slug Does

- Ultra high quality real-time rendering
  - Winding number calculation, antialiasing, dynamic bounds dilation, ...
- Advanced layout and typography
  - Kerning, ligatures, combining marks
  - Emoji, skin tone modifiers, hair styles, ZWJ sequences, ...
  - Alternate substitutions
    - Sub/superscripts, ordinals, small caps, case-sensitive forms, fractions, ...
    - Lining / old style figures, tabular / proportional figures
  - Contextual substitutions
  - Bidirectional layout, cursive joining (Arabic), vertical layout (Japanese)
  - Paragraph layout
    - Indent, justification, line breaking, optional hyphens, ...

Rendering Demo



Rendering and Typographic Features

(Page 1 of 18 — Press space key to advance to next page)

Font Styles

Flexible mapping lets font type codes determine primary fonts and automatic fallback fonts.

Regular *italics* **bold** `{code}` 🐉 🍕 🥳

Stretch and Skew

Glyphs can be transformed in a variety of ways at the character level.

Text stretched

Text *skewed*

Text Decorations

Underline and strikethrough decorations can be applied to any parts of the text.

Text underline

Text ~~strikethrough~~

Tracking

Tracking specifies extra space that is added or subtracted between consecutive glyphs.

Tight (tracking −0.05)

Loose (tracking +0.05)

Multicolor Emoji

Glyphs having multiple color layers are rendered in the same way as ordinary glyphs.

🧑‍🚀 🍕 🎄 😎 🐉

🧑‍🚀 🍕 🎄 😎 🐉

Skin Tone Modifiers

Skin tone modifiers change the colors of a preceding human emoji glyph.

🖐 🖐🏻 🖐🏼 🖐🏽 🖐🏾 🖐🏿

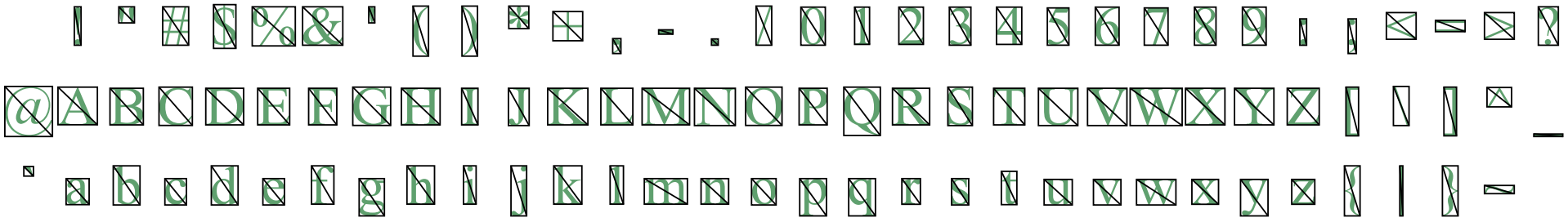
👩‍🔬 👩🏻‍🔬 👩🏼‍🔬 👩🏽‍🔬 👩🏾‍🔬 👩🏿‍🔬

How Slug Works

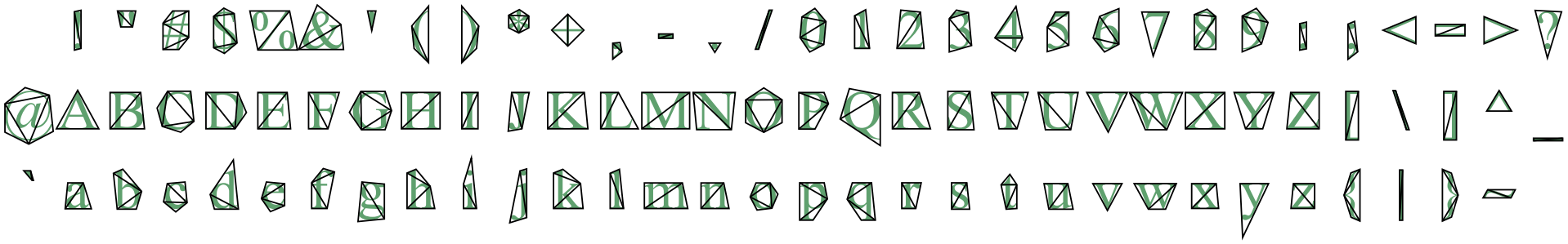
- Builds a small number of triangles per glyph
  - Per layer for color emoji
- Renders with a pixel shader that calculates winding number and fractional coverage value
- Implements speed optimizations
- Solves robustness problems

Geometry Sent to GPU

- Quads with 4 vertices, 2 triangles

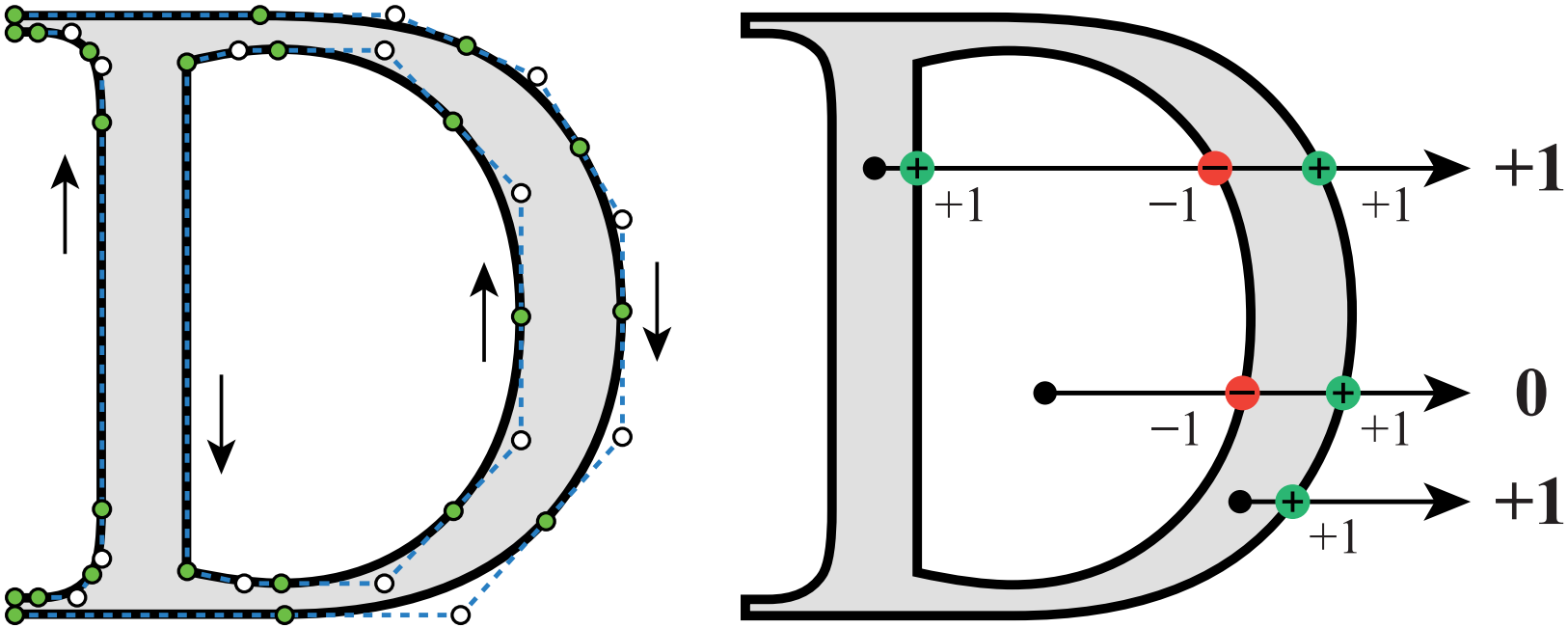


- Or polygons with 3 to 6 vertices, 1 to 4 triangles



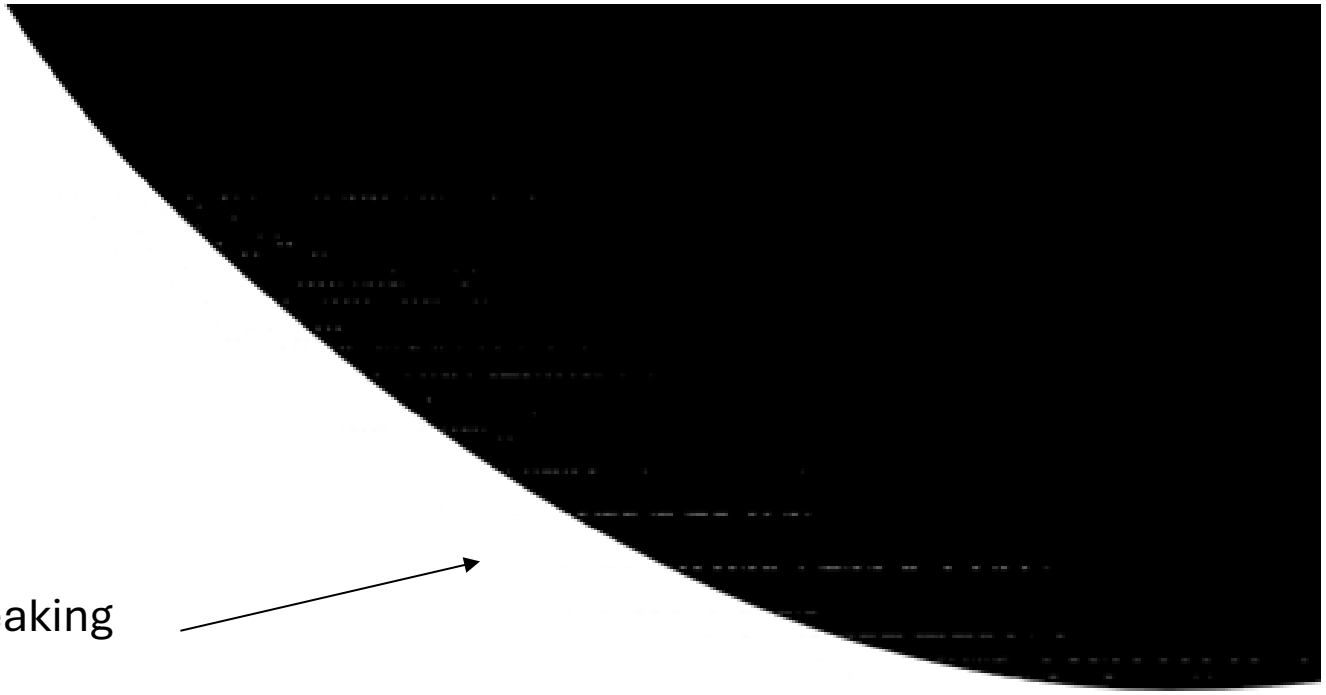
Winding Number Calculation

- Shoot ray from center of pixel being rendered
- Intersect with quadratic Bézier curves



Winding Number Calculation

- Math is straightforward, but direct implementation is not robust
- Suffers from floating-point precision issues



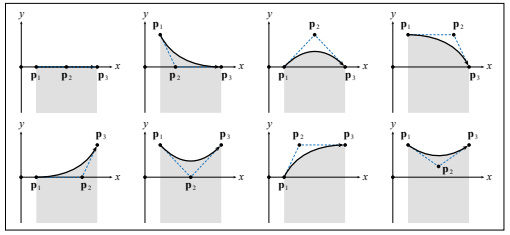
Winding Number Calculation

- Perfect, provable robustness achieved with special equivalence class algorithm
- Source of floating-point errors eliminated

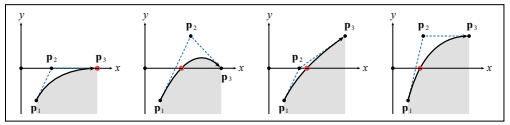
Root Eligibility

Class	$y_3 < 0$	$y_2 < 0$	$y_1 < 0$	Root 2	Root 1
A	0	0	0	0	0
B	0	0	1	1 ⊖	0
C	0	1	0	1 ⊖	1 ⊕
D	0	1	1	1 ⊖	0
E	1	0	0	0	1 ⊕
F	1	0	1	1 ⊖	1 ⊕
G	1	1	0	0	1 ⊕
H	1	1	1	0	0
				0x2E	0x74

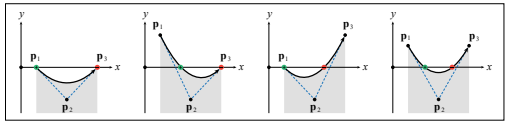
Class A



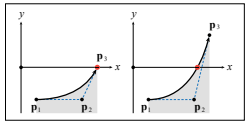
Class B



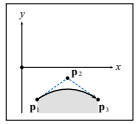
Class C



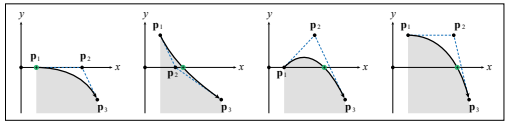
Class D



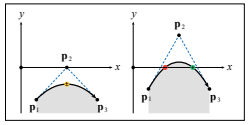
Class H



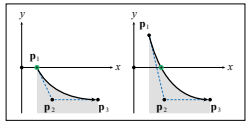
Class E



Class F

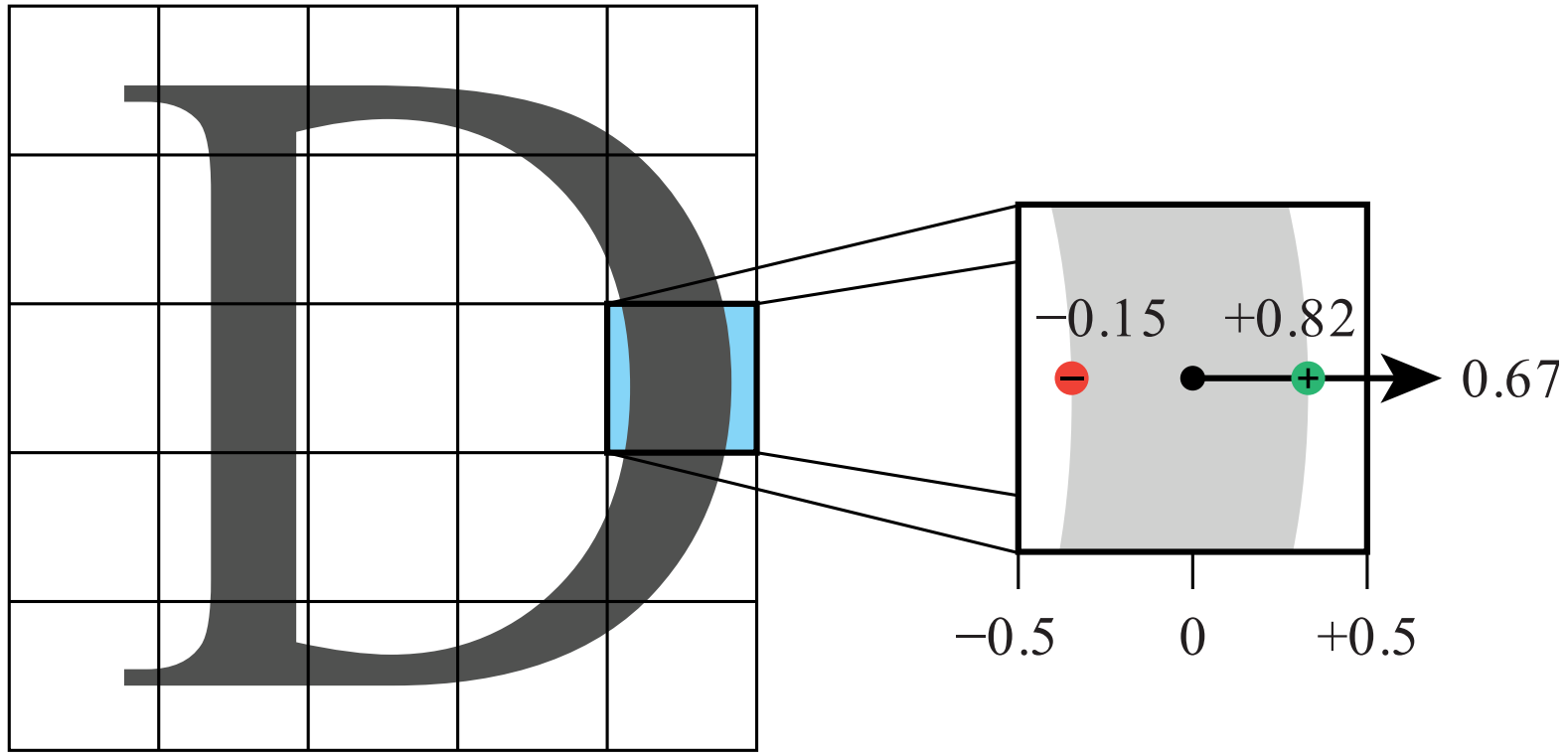


Class G



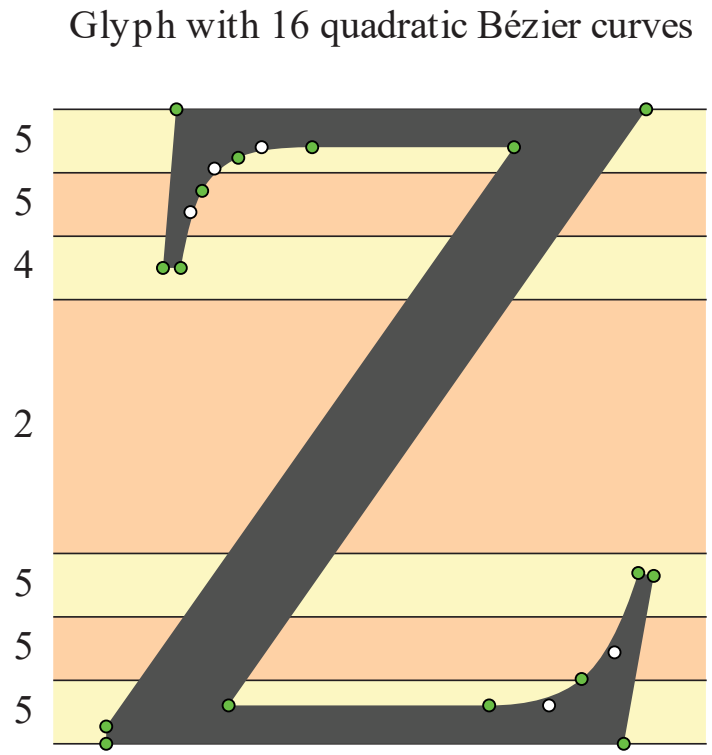
Antialiasing

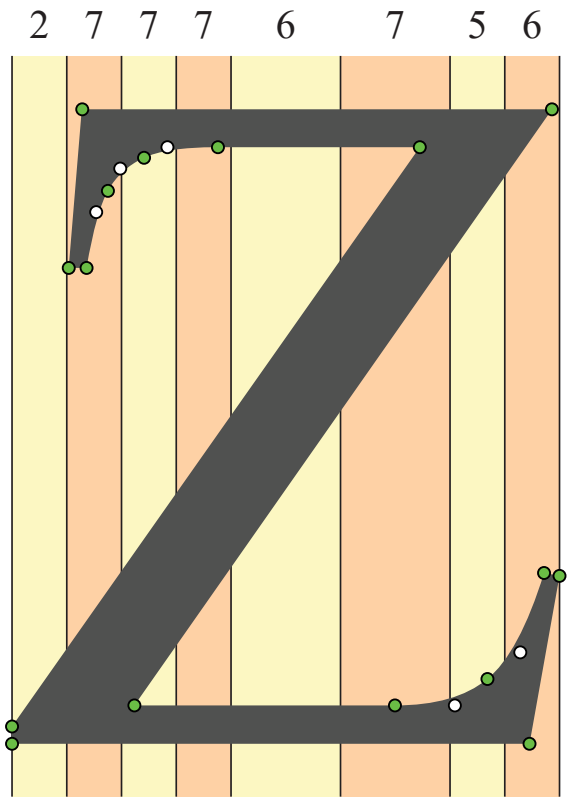
- Partial pixel coverage calculated for horizontal and vertical rays
- Results combined for final coverage value



Banding

- Bézier curves grouped into bands to reduce computation
- Curves in each band are sorted max to min for early out





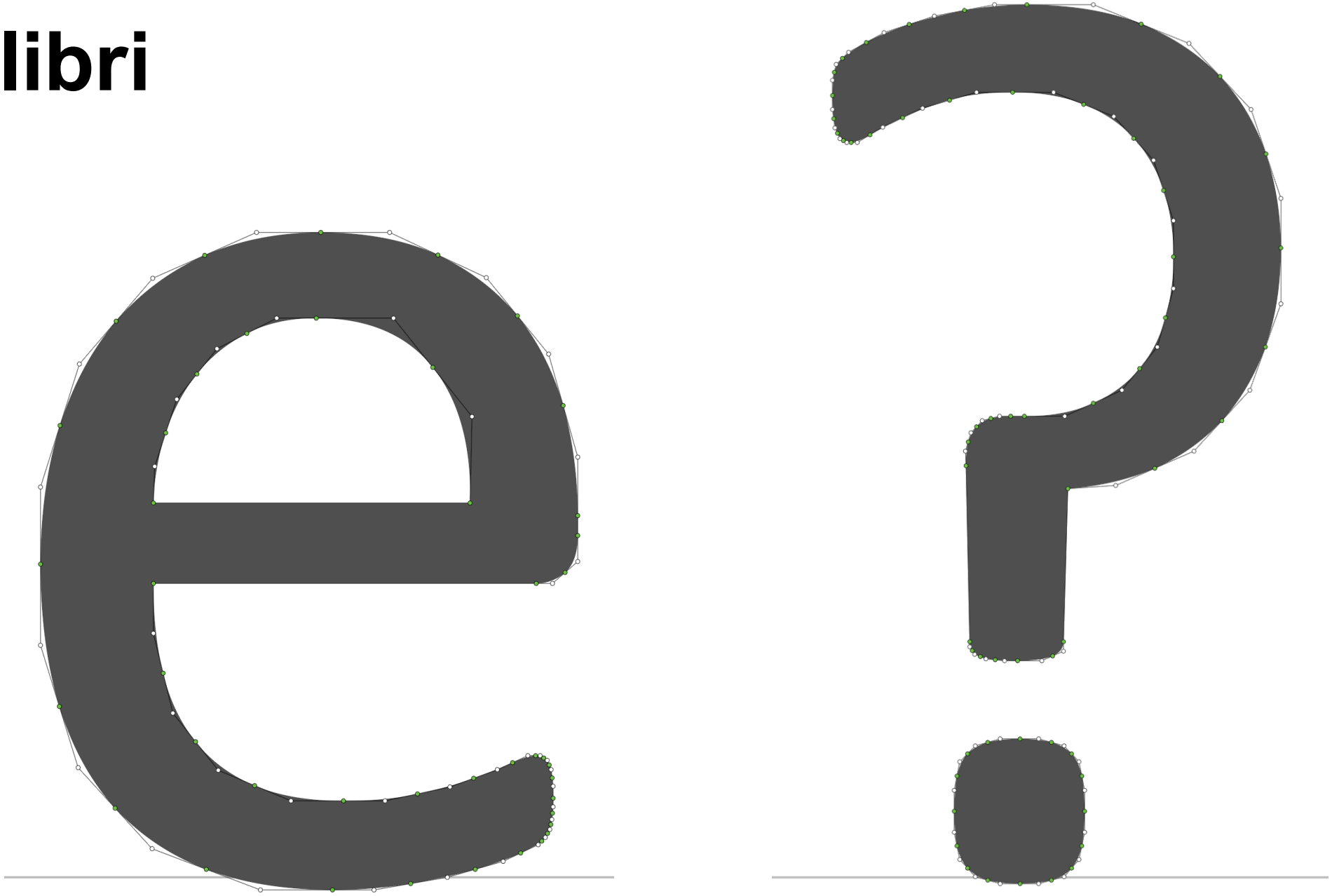
Optimization Goals

- Performance
  - Minimize fill area in hidden parts
  - Eliminate unnecessary Bézier curves
  - Minimize number of curves in worst band
  - Prefer exact horizontal / vertical lines where pertinent
    - These cost half as much
- Appearance
  - Eliminate unwanted wiggles, cusps, concavities, stray control points
  - Ensure multi-layer control point alignment in emoji
  - Avoid shared boundaries for best antialiasing
    - In particular, across emoji layers

Excess Control Points

- Often the case that more control points than necessary are used in a glyph’s outline
- This degrades performance, especially if they’re clustered
- It usually degrades visual appearance as well under extreme magnification
- Puts unnecessary pressure on GPU caches

Calibri

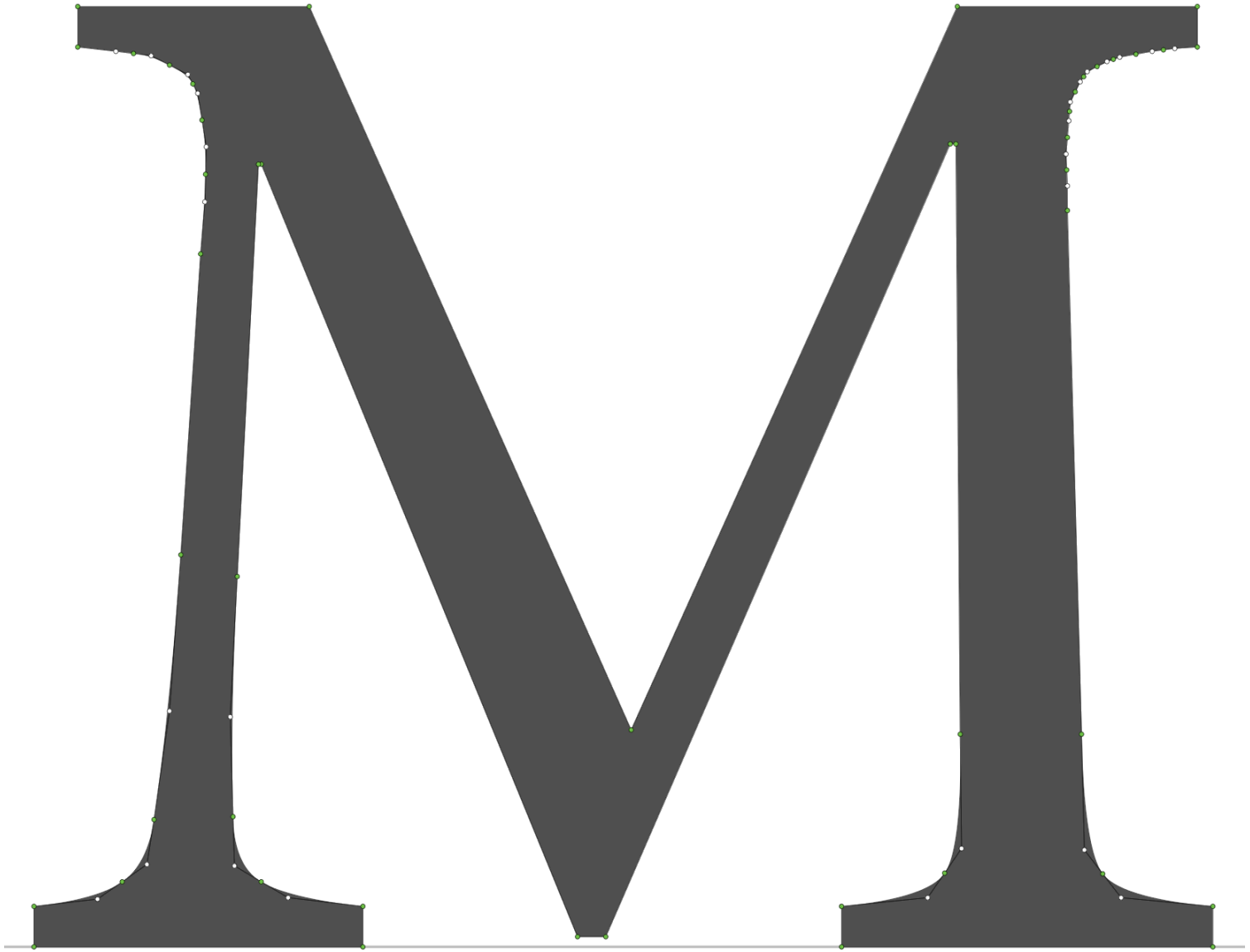


Visual Artifacts

- Unwanted concavities
- Discontinuous tangents
- Layer misalignments
- Tiny cusps

- These can all be highly magnified in 3D environment

Minion Pro



Comic Sans Bold

- Microscopic cusp example
- Wreaks havoc on expanded outlines
- This appears to be a case of automated boldening

ABCDEFGHIJKLMN
OPQRSTUVWXYZ
abcdefghijklmnopqrs
tuvwxyz
123456789 $+\frac{1}{2}\frac{1}{4}\frac{3}{4}$

ABCDEFGHIJKLMN

OPQRSTUVWXYZ

abcdefghijklmnopqrs

tuvwxyz

123456789

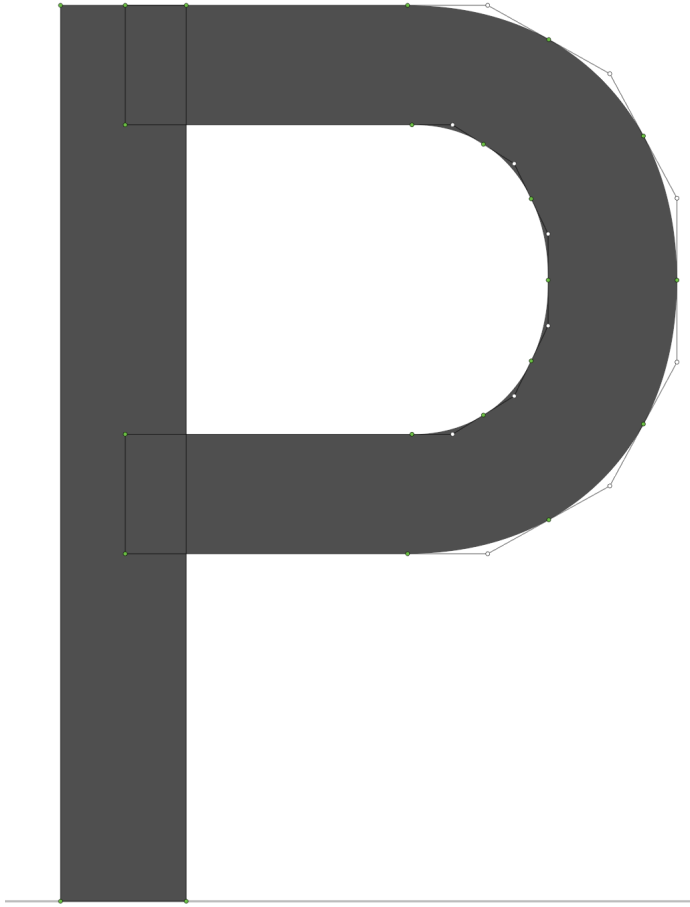
$$+\frac{1}{2}\frac{1}{4}\frac{3}{4}$$

Coincident Edges

- These cause antialiasing problems and should be avoided
- Can be eliminated by using min or max blend mode if background is pure black or pure white
- Otherwise, usual alpha blending will over-cover background
- For emoji, antialiased edges can also have wrong color

Bahnschrift

- Variable font with overlapping components



Coincident Edges

- The antialiasing problem can be avoided by “notching”
- We’ll see examples in color emoji

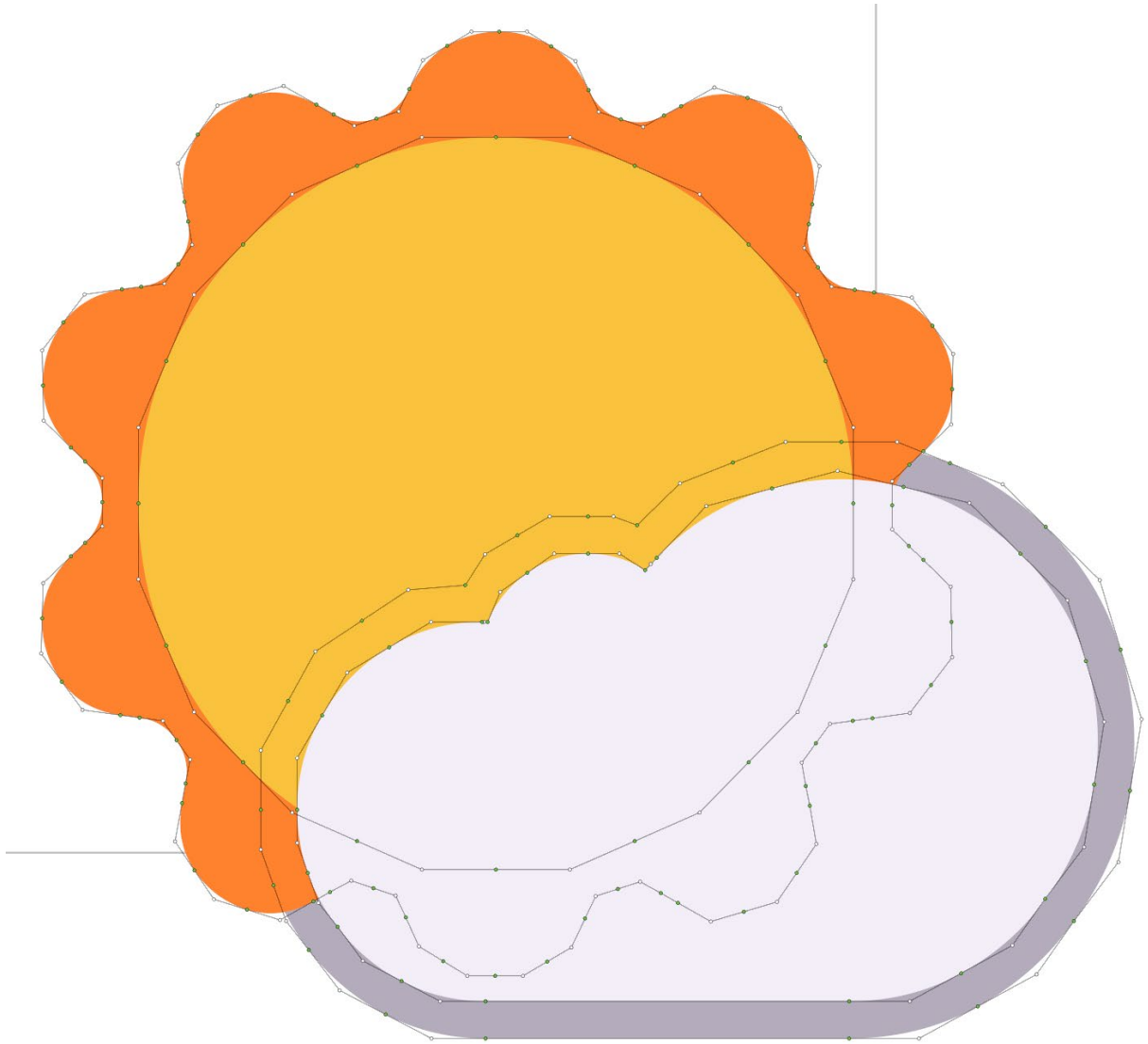
Hidden Geometry

- Many emoji contain geometry in background layers that can never be seen
- This wastes space and hurts performance for no reason
- Easy fix -- just delete it

Twemoji U+01F92C

Segoe UI Emoji U+01F927

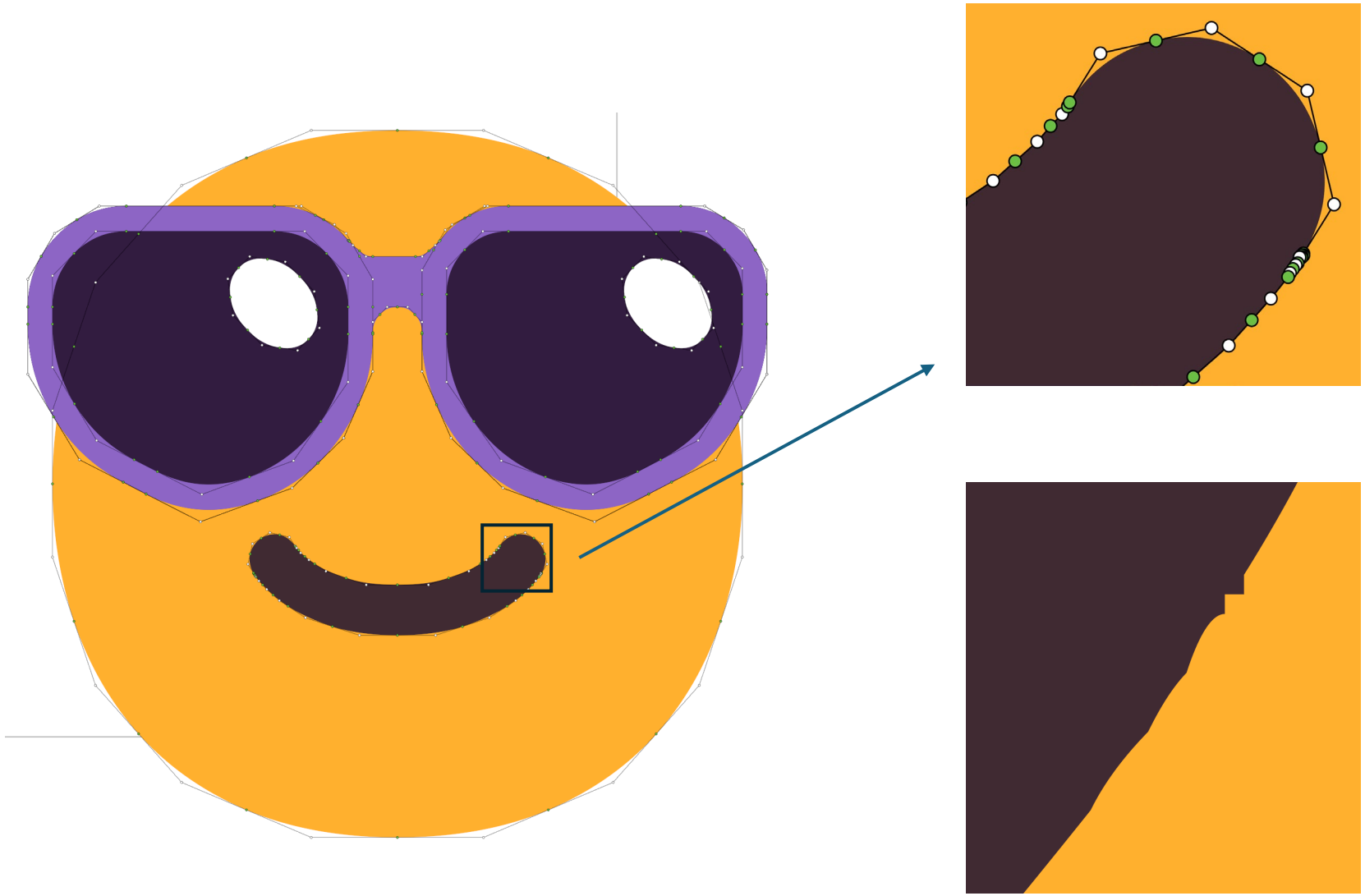
Segoe UI Emoji U+01F324



Excess / Sloppy Curves

- Lots of emoji have many excess control points
- Those control points often belong to ugly curves
- Fixing probably requires lots of manual adjustments

Segoe UI Emoji U+01F60E



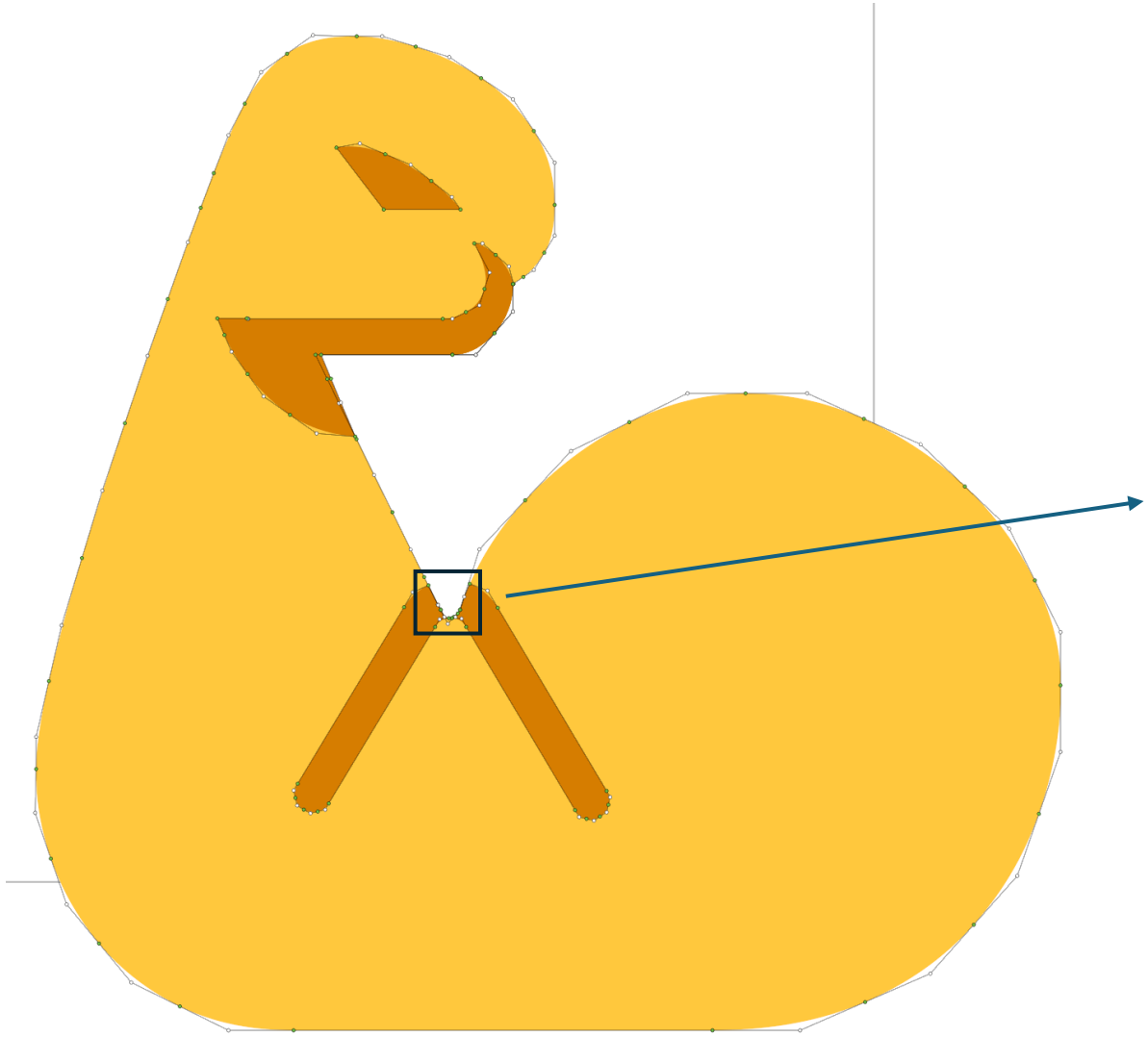
Segoe UI Emoji U+01F393

Segoe UI Emoji U+01F4AF

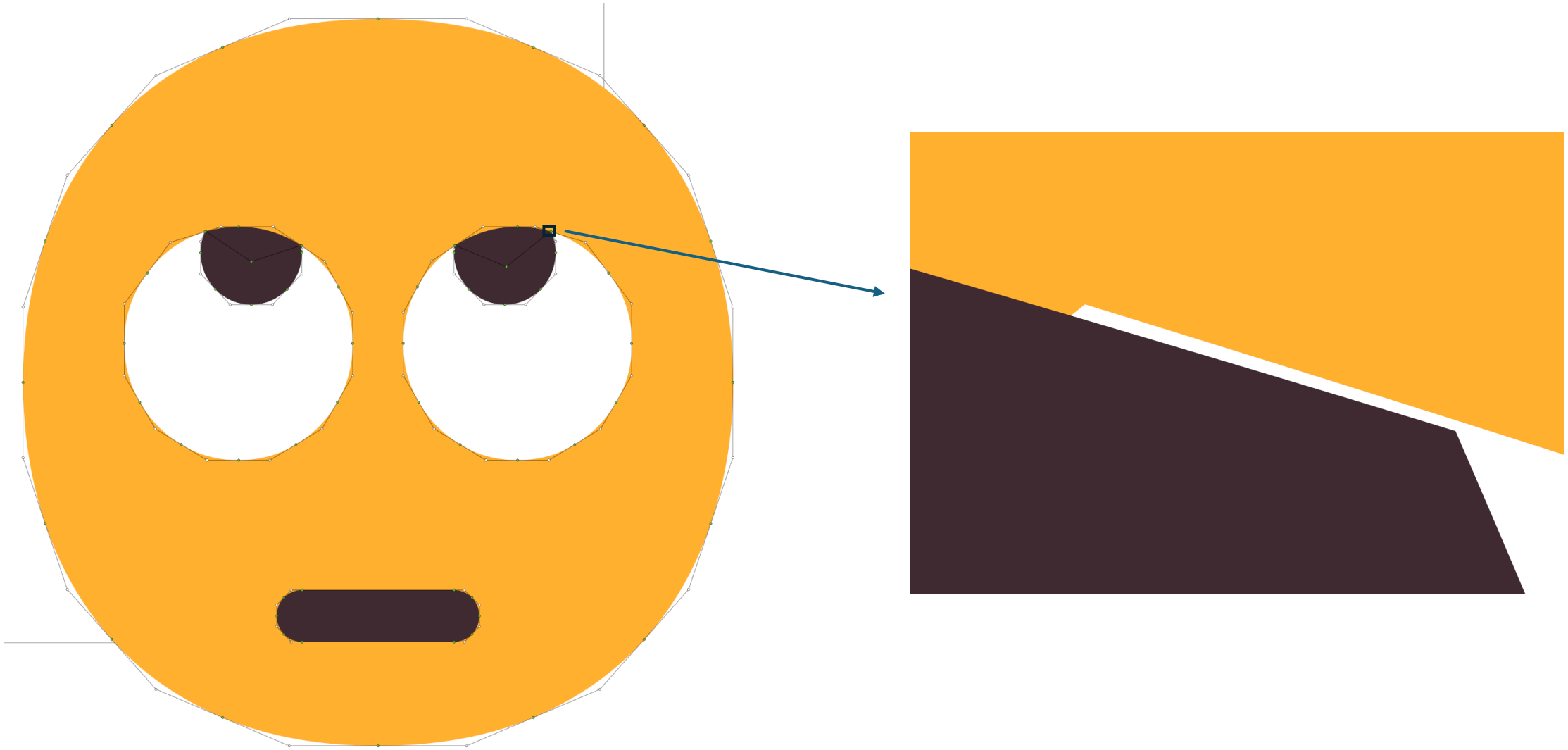
Layer Alignment

- Many emoji contain layers with misaligned control points
- Difficult to see at normal font sizes
- Easy to see when font rendered at equivalent of 5000 pt size!
- Since control points lie on integer grid, fix is simple

Segoe UI Emoji U+01F4AA



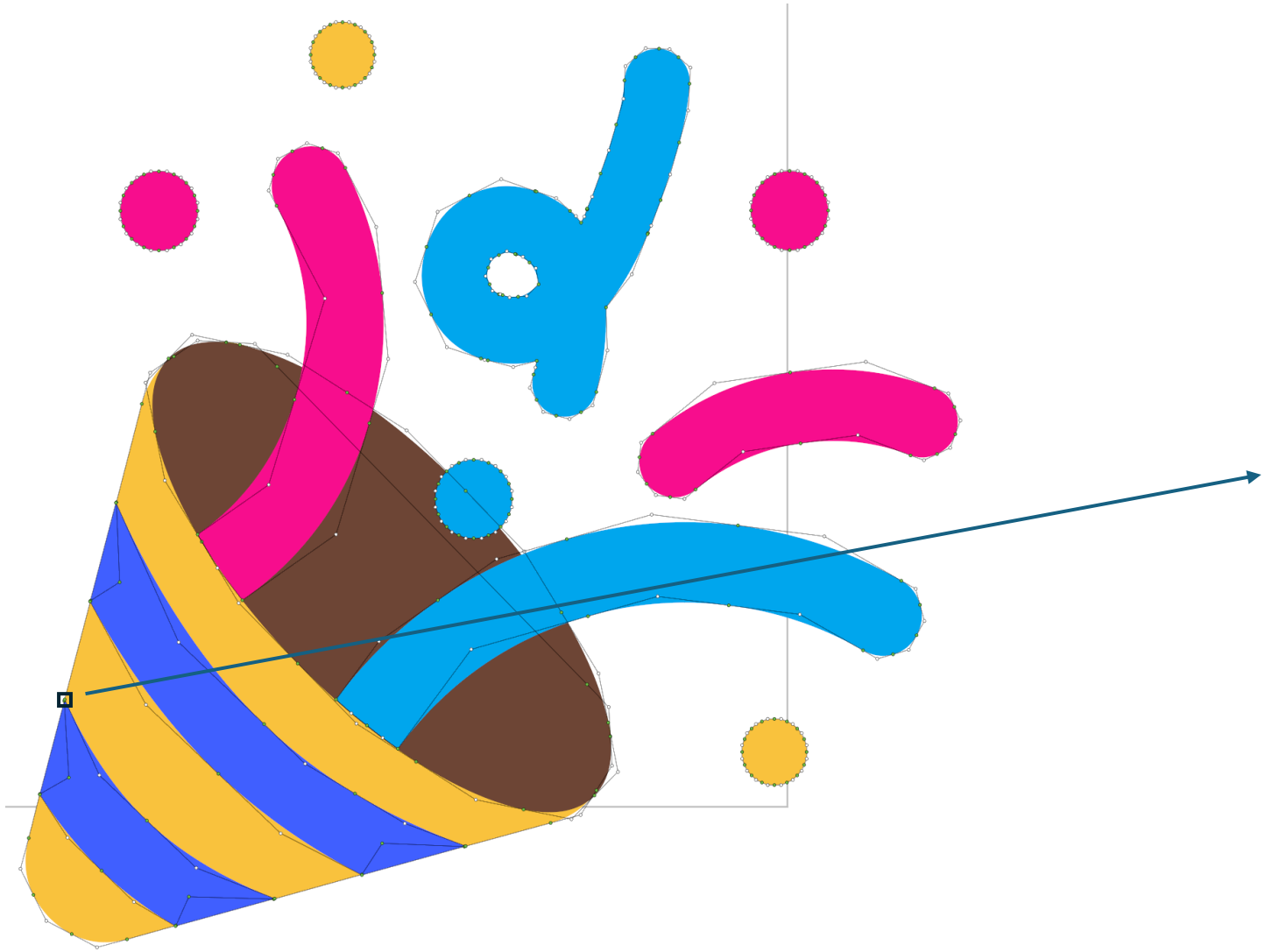
Segoe UI Emoji U+01F644

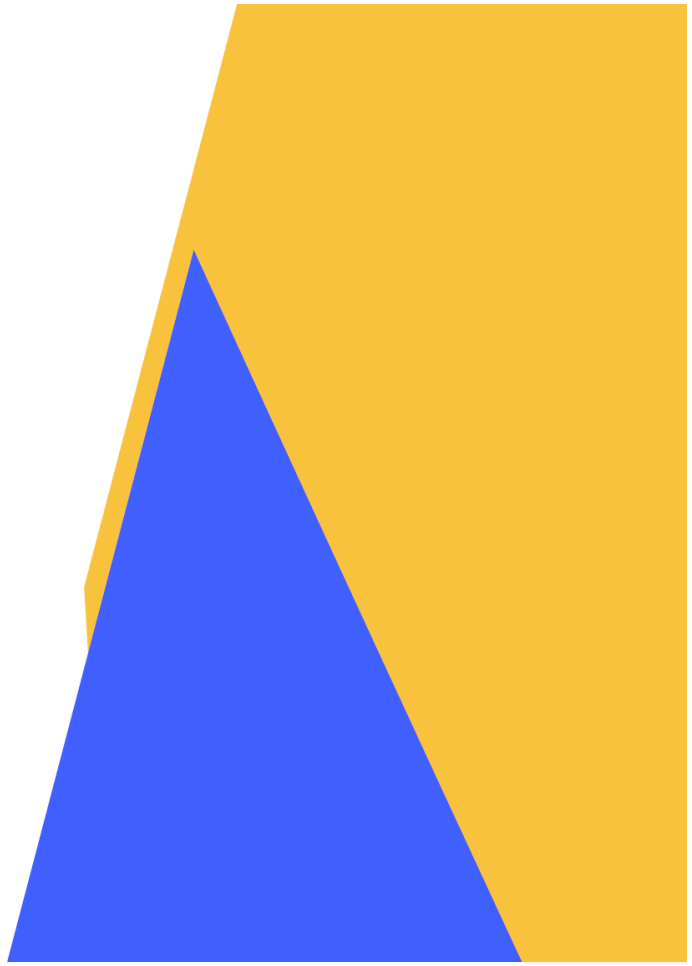


Notching

- If two layers have coincident edges, then antialiasing doesn’t work well
- Imagine two coincident layer boundaries covering 50% of pixel
  - Final result should be 50% background color / 50% top layer color
  - Actual result is 25% background / 25% bottom layer / 50% top layer
- Solution is to cut notches out of bottom layer
- Emoji fonts do this, but often sloppily

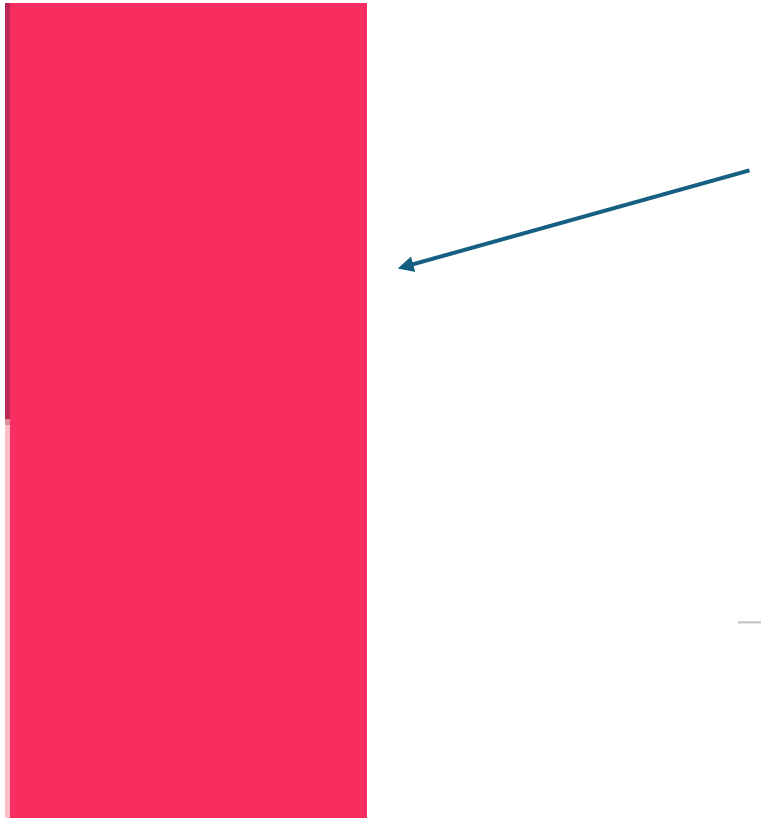
Segoe UI Emoji U+01F389





Segoe UI Emoji U+01F309

- Notched, but misaligned
- Also has unwanted cusps

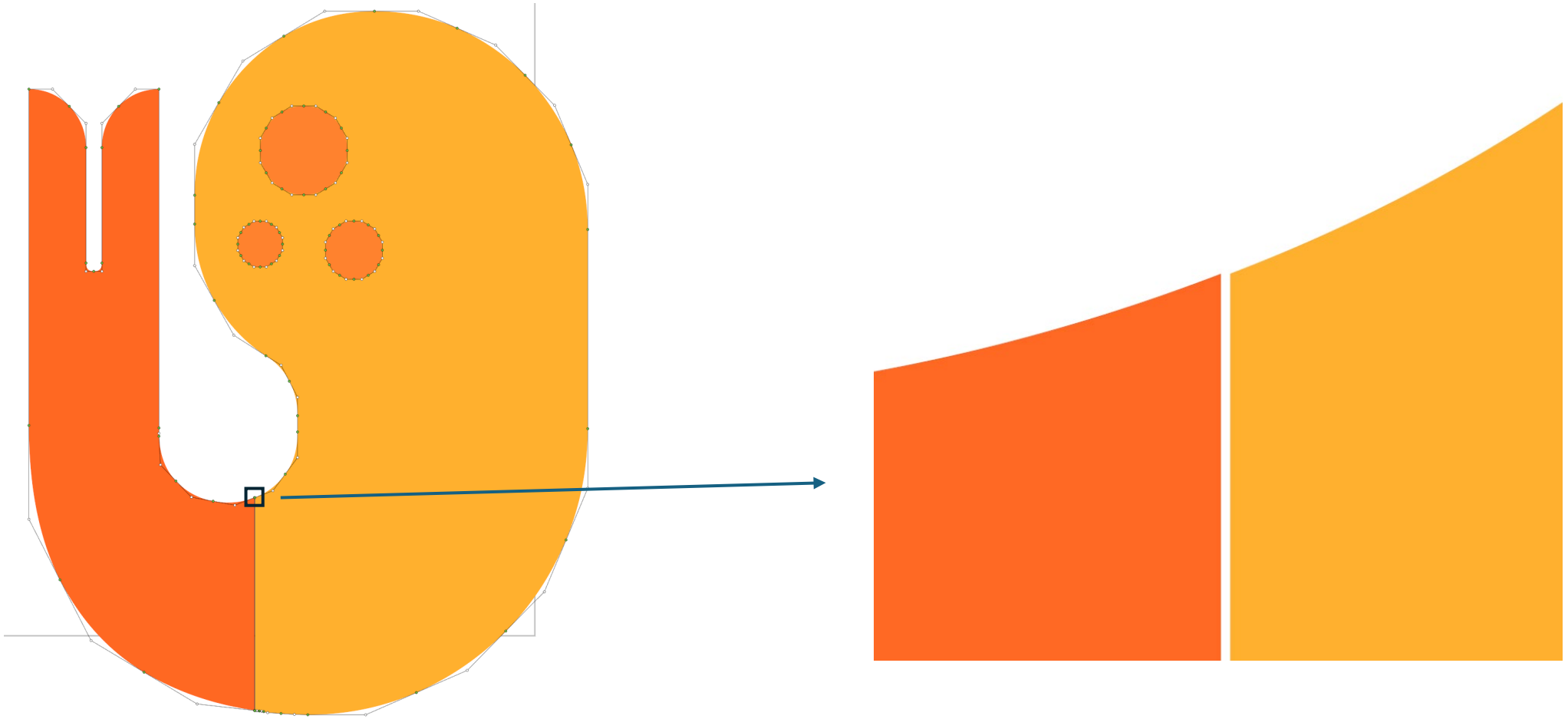


Segoe UI Emoji U+01F3FA

- Perfect notching
- Still has hidden geometry, but just a little

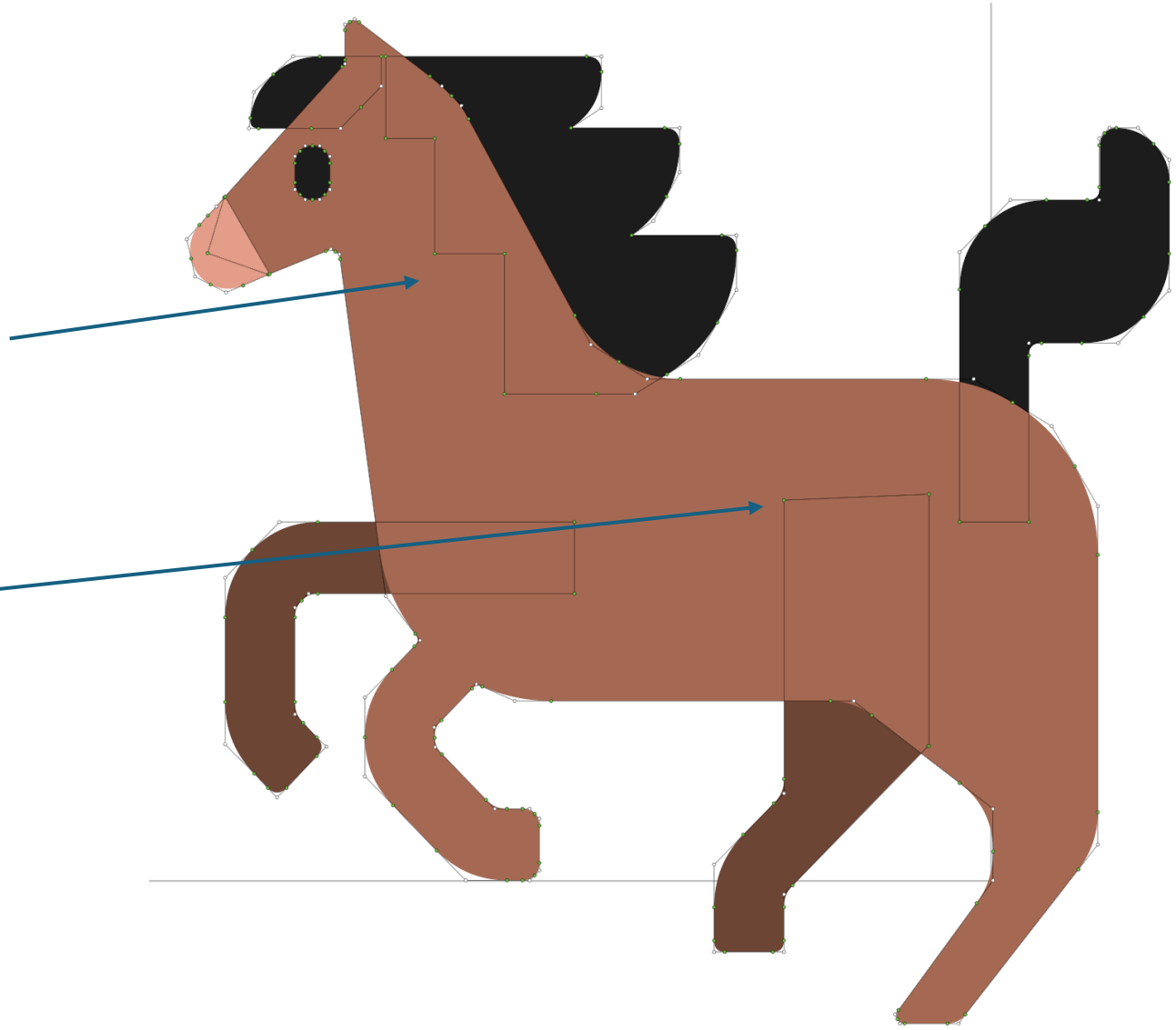
Segoe UI Emoji U+01F364

- When layers are adjacent, extend one beneath the other



Segoe UI Emoji U+01F40E

- Area can be reduced
- Staircase should be one diagonal line
- Exact horizontal line should be preferred for rear right leg



Segoe UI Emoji U+01F3C4

- Excessive curves
- Bad alignment
- Hidden geometry
- Coincident boundaries
- Inconsistent

Optimization Summary

- Minimize number of control points
- Avoid control point clustering
- Prefer exact horizontal or vertical lines
- Prefer quadratic curves in .ttf format instead of cubic curves in .otf format
- In emoji, keep layers with same color together
- Minimize area of polygonal bounds of each color layer

Contact

- lengyel@terathon.com
- Twitter: @EricLengyel
- Bluesky: @ericlengyel.bsky.social
- LinkedIn: www.linkedin.com/in/eric-lengyel