SD4 Assignment 4: Fancy Fonts

# Summary

* Support for variable-width font rendering (e.g. WWW is wider than iii)
* Support for fonts using BMFont-generated font metadata (arbitrary layout, data-driven glyph info)
* Support for SDF-style fonts rendered with custom shaders, which test font texture alphas vs. threshold value(s) rather than drawing literally as standard alpha translucency, yielding fonts with crisp edges at arbitrary magnification scales.
* Support for custom font vertex format which conveys meta-information about each glyph’s vertex, i.e. where the vertex is relative to its glyph-quad, string-area, and character position.
* Demonstration cases of font(s) and font effects which take advantage of all supported features.

# Requirements (maximum possible score is 110/100):

***Note: No points will be awarded for any feature that is not specifically demonstrated in your demo/video.***

* (10) “**Tier 1**” font rendering *or better*: SD1 Fonts (basic “conchars” style)
  + Fixed-width, fixed-layout (16x16 square grid) bitmap fonts (SD1-style)

*Note: you get automatic full points for this item if you have Tier 2 or better font support!*

* (20) “**Tier 2**” font rendering *or better*: Proportional Auto-Width Bitmap Fonts (no metadata)
  + Proportional-width, fixed-layout (16x16 square grid) bitmap fonts with no additional metadata file(s); glyph vary in width (‘W’ vs. ‘i’), and widths are inferred by scanning font image glyph data directly

*Note: you get automatic full points for this item if you have Tier 3 or better font support!*

* (30) “**Tier 3**” font rendering *or better*: Fonts with metadata (e.g. BMFont) and arbitrary sheet layouts
  + Proportional-width, arbitrary layout bitmap fonts with accompanying metadata file(s), as exported by BMFont or similar program
  + Full adherence to, and correct rendering of, font glyphs with regard to their UV and quad information as provided by the metadata file(s)
  + (3) Support for / adherence to any letter-pair “kerning” rules provided in the font metadata

*Note: you get automatic full points for this item if you have Tier 4 or better font support!*

* (15) “**Tier 4**” font rendering *or better*: Fonts with blurred edges and custom shader for edge-threshold detection
  + Font is exported (from BMFont, etc.) with a few texels (3+) of extra “Padding” on all sides of each glyph
  + Font is blurred, either in a paint program (using a “Gaussian blur”) or a custom SDF cooker pipeline tool
  + Font texture is sampled with bilinear (smoothed/linear, not pixelated/nearest/point) sampling
  + Font is drawn with a standard vertex array, using a basic vertex format (e.g. VertexPCU/VertexUnlit)
  + Font is rendered with a custom shader, which:
    - Samples the font glyph texture sheet;
    - Tests the sampled texel’s alpha value against a “minimum threshold” for acceptance
    - Discards the pixel if the alpha value is below the minimum threshold
    - Outputs a fully-opaque pixel otherwise (if alpha >= threshold)
    - Output color RGB = font texture’s RGB x drawn-triangle’s RGB
    - Output alpha = 1.0

*Note: you get automatic full points for this item if you have Tier 5 or better font support!*

* (15) “**Tier 5**” font rendering: Fonts drawn with custom vertex format and custom shader for font effects
  + Same as Tier 4, except:
  + Font uses a custom vertex format (VertexFont), which provides the following additional data per vertex:
    - Vec2 glyphPosition; // normalized [0,1] coordinates of where this vert lies within this glyph-quad
    - Vec2 textPosition; // normalized coordinates of where this vert is within this overall text block
    - int characterIndex; // index of which character in this text-string this vert belongs to (same at all four corners of each glyph-quad)

and at least one additional piece of data per vertex (your choice – can be anything); ideas include:

* + - float weight; // thin-normal-bold, either with 1.0 as normal, or use standard font weights (400+-)
    - float oblique; // italic-ness; either with 1.0 as normal, or use standard font oblique degrees (0+-)
    - Vec4 specialEffects; // four generic floats, each triggers/parameterizes a different “special effect”
  + Font uses a custom shader which leverages these additional vertex data to perform cool text effects.
    - Wavy, color changing, typewriter effect
* (5) Bonus / cool font effect(s)
  + Make anything really cool, clever, and original using any of the above tools. For example:
  + Sample a Perlin Noise texture and add that (scaled by maybe 0.1) to the threshold to get “curvy edges”
  + Use time to animate – either one-off (appear/disappear) or cyclic (ongoing) effects
    - Typewriter and noise based wiggle
  + Manipulate font vertexes in the vertex shader based on custom uniform and/or vertex data (e.g. wiggle)
  + Support simple normal-maps for font glyphs, with positional lights that shade & shine off of fake-3D fonts
  + Hidden text that appears in a spotlight when the mouse hovers over
    - Sent the mouse cords as special effects and then clipped based on distance of pixel to mouse

It is also valuable to keep previous font “Tiers” intact, as they have many benefits (including simplicity of font-authorship, etc.) The following bullet points give credit for maintaining font pipelines and keeping options intact:

* (5) Tier 1 font pipeline remains intact and functional (i.e. neither skipped, nor broken, nor deprecated)

*Note: you do NOT get points for this item if you have Tier 2/3/4/5 but cannot support Tier 1*

* (3) Tier 2 font pipeline is intact and functional (i.e. neither skipped, nor broken, nor deprecated)

*Note: you do NOT get points for this item if you have Tier 1/3/4/5 but cannot support Tier 2*

* (5) Tier 3 font pipeline is intact and functional (i.e. neither skipped, nor broken, nor deprecated)

*Note: you do NOT get points for this item if you have Tier 1/2/4/5 but cannot support Tier 3*

* (2) Tier 4 font pipeline is intact and functional (i.e. neither skipped, nor broken, nor deprecated)

*Note: you do NOT get points for this item if you have Tier 1/2/3/5 but cannot support Tier 4*

# Submission

* Use your own Guildhall “SD” C++ engine – maintained, buddy-tested, and submitted via Perforce.
  + A committed changelist in P4 with submission comment “SD4-A4: COMPLETE”.
  + Be sure to include an updated ReadMe, a current Release-built .EXE, and all required code & data files.
  + *See notes from Professor Service’s DFS2 class regarding maintenance of your code across P4 / GitHub.*
* Submit a **.zip file** to Canvas named **C29\_SD4\_*A4*\_*p4username*.zip** (*for example:* ***C29\_SD4\_A4\_beiserloh.zip***) which contains:
  + A very short (informal) voice-narrated **video** demonstrating the full functionality of your project;
  + A copy of **this document**, with completed items highlighted cyan, omitted items highlighted red, and partially completed items highlighted yellow (with inserted bullets-text underneath explaining).