/\*

TrueType to Adafruit\_GFX font converter. Derived from Peter Jakobs'

Adafruit\_ftGFX fork & makefont tool, and Paul Kourany's Adafruit\_mfGFX.

NOT AN ARDUINO SKETCH. This is a command-line tool for preprocessing

fonts to be used with the Adafruit\_GFX Arduino library.

For UNIX-like systems. Outputs to stdout; redirect to header file, e.g.:

./fontconvert ~/Library/Fonts/FreeSans.ttf 18 > FreeSans18pt7b.h

REQUIRES FREETYPE LIBRARY. www.freetype.org

Currently this only extracts the printable 7-bit ASCII chars of a font.

Will eventually extend with some int'l chars a la ftGFX, not there yet.

Keep 7-bit fonts around as an option in that case, more compact.

See notes at end for glyph nomenclature & other tidbits.

\*/

#ifndef ARDUINO

#include <ctype.h>

#include <ft2build.h>

#include <stdint.h>

#include <stdio.h>

#include FT\_GLYPH\_H

#include FT\_MODULE\_H

#include FT\_TRUETYPE\_DRIVER\_H

#include "../gfxfont.h" // Adafruit\_GFX font structures

#define DPI 141 // Approximate res. of Adafruit 2.8" TFT

// Accumulate bits for output, with periodic hexadecimal byte write

void enbit(uint8\_t value) {

static uint8\_t row = 0, sum = 0, bit = 0x80, firstCall = 1;

if (value)

sum |= bit; // Set bit if needed

if (!(bit >>= 1)) { // Advance to next bit, end of byte reached?

if (!firstCall) { // Format output table nicely

if (++row >= 12) { // Last entry on line?

printf(",\n "); // Newline format output

row = 0; // Reset row counter

} else { // Not end of line

printf(", "); // Simple comma delim

}

}

printf("0x%02X", sum); // Write byte value

sum = 0; // Clear for next byte

bit = 0x80; // Reset bit counter

firstCall = 0; // Formatting flag

}

}

int main(int argc, char \*argv[]) {

int i, j, err, size, first = ' ', last = '~', bitmapOffset = 0, x, y, byte;

char \*fontName, c, \*ptr;

FT\_Library library;

FT\_Face face;

FT\_Glyph glyph;

FT\_Bitmap \*bitmap;

FT\_BitmapGlyphRec \*g;

GFXglyph \*table;

uint8\_t bit;

// Parse command line. Valid syntaxes are:

// fontconvert [filename] [size]

// fontconvert [filename] [size] [last char]

// fontconvert [filename] [size] [first char] [last char]

// Unless overridden, default first and last chars are

// ' ' (space) and '~', respectively

if (argc < 3) {

fprintf(stderr, "Usage: %s fontfile size [first] [last]\n", argv[0]);

return 1;

}

size = atoi(argv[2]);

if (argc == 4) {

last = atoi(argv[3]);

} else if (argc == 5) {

first = atoi(argv[3]);

last = atoi(argv[4]);

}

if (last < first) {

i = first;

first = last;

last = i;

}

ptr = strrchr(argv[1], '/'); // Find last slash in filename

if (ptr)

ptr++; // First character of filename (path stripped)

else

ptr = argv[1]; // No path; font in local dir.

// Allocate space for font name and glyph table

if ((!(fontName = malloc(strlen(ptr) + 20))) ||

(!(table = (GFXglyph \*)malloc((last - first + 1) \* sizeof(GFXglyph))))) {

fprintf(stderr, "Malloc error\n");

return 1;

}

// Derive font table names from filename. Period (filename

// extension) is truncated and replaced with the font size & bits.

strcpy(fontName, ptr);

ptr = strrchr(fontName, '.'); // Find last period (file ext)

if (!ptr)

ptr = &fontName[strlen(fontName)]; // If none, append

// Insert font size and 7/8 bit. fontName was alloc'd w/extra

// space to allow this, we're not sprintfing into Forbidden Zone.

sprintf(ptr, "%dpt%db", size, (last > 127) ? 8 : 7);

// Space and punctuation chars in name replaced w/ underscores.

for (i = 0; (c = fontName[i]); i++) {

if (isspace(c) || ispunct(c))

fontName[i] = '\_';

}

// Init FreeType lib, load font

if ((err = FT\_Init\_FreeType(&library))) {

fprintf(stderr, "FreeType init error: %d", err);

return err;

}

// Use TrueType engine version 35, without subpixel rendering.

// This improves clarity of fonts since this library does not

// support rendering multiple levels of gray in a glyph.

// See https://github.com/adafruit/Adafruit-GFX-Library/issues/103

FT\_UInt interpreter\_version = TT\_INTERPRETER\_VERSION\_35;

FT\_Property\_Set(library, "truetype", "interpreter-version",

&interpreter\_version);

if ((err = FT\_New\_Face(library, argv[1], 0, &face))) {

fprintf(stderr, "Font load error: %d", err);

FT\_Done\_FreeType(library);

return err;

}

// << 6 because '26dot6' fixed-point format

FT\_Set\_Char\_Size(face, size << 6, 0, DPI, 0);

// Currently all symbols from 'first' to 'last' are processed.

// Fonts may contain WAY more glyphs than that, but this code

// will need to handle encoding stuff to deal with extracting

// the right symbols, and that's not done yet.

// fprintf(stderr, "%ld glyphs\n", face->num\_glyphs);

printf("const uint8\_t %sBitmaps[] PROGMEM = {\n ", fontName);

// Process glyphs and output huge bitmap data array

for (i = first, j = 0; i <= last; i++, j++) {

// MONO renderer provides clean image with perfect crop

// (no wasted pixels) via bitmap struct.

if ((err = FT\_Load\_Char(face, i, FT\_LOAD\_TARGET\_MONO))) {

fprintf(stderr, "Error %d loading char '%c'\n", err, i);

continue;

}

if ((err = FT\_Render\_Glyph(face->glyph, FT\_RENDER\_MODE\_MONO))) {

fprintf(stderr, "Error %d rendering char '%c'\n", err, i);

continue;

}

if ((err = FT\_Get\_Glyph(face->glyph, &glyph))) {

fprintf(stderr, "Error %d getting glyph '%c'\n", err, i);

continue;

}

bitmap = &face->glyph->bitmap;

g = (FT\_BitmapGlyphRec \*)glyph;

// Minimal font and per-glyph information is stored to

// reduce flash space requirements. Glyph bitmaps are

// fully bit-packed; no per-scanline pad, though end of

// each character may be padded to next byte boundary

// when needed. 16-bit offset means 64K max for bitmaps,

// code currently doesn't check for overflow. (Doesn't

// check that size & offsets are within bounds either for

// that matter...please convert fonts responsibly.)

table[j].bitmapOffset = bitmapOffset;

table[j].width = bitmap->width;

table[j].height = bitmap->rows;

table[j].xAdvance = face->glyph->advance.x >> 6;

table[j].xOffset = g->left;

table[j].yOffset = 1 - g->top;

for (y = 0; y < bitmap->rows; y++) {

for (x = 0; x < bitmap->width; x++) {

byte = x / 8;

bit = 0x80 >> (x & 7);

enbit(bitmap->buffer[y \* bitmap->pitch + byte] & bit);

}

}

// Pad end of char bitmap to next byte boundary if needed

int n = (bitmap->width \* bitmap->rows) & 7;

if (n) { // Pixel count not an even multiple of 8?

n = 8 - n; // # bits to next multiple

while (n--)

enbit(0);

}

bitmapOffset += (bitmap->width \* bitmap->rows + 7) / 8;

FT\_Done\_Glyph(glyph);

}

printf(" };\n\n"); // End bitmap array

// Output glyph attributes table (one per character)

printf("const GFXglyph %sGlyphs[] PROGMEM = {\n", fontName);

for (i = first, j = 0; i <= last; i++, j++) {

printf(" { %5d, %3d, %3d, %3d, %4d, %4d }", table[j].bitmapOffset,

table[j].width, table[j].height, table[j].xAdvance, table[j].xOffset,

table[j].yOffset);

if (i < last) {

printf(", // 0x%02X", i);

if ((i >= ' ') && (i <= '~')) {

printf(" '%c'", i);

}

putchar('\n');

}

}

printf(" }; // 0x%02X", last);

if ((last >= ' ') && (last <= '~'))

printf(" '%c'", last);

printf("\n\n");

// Output font structure

printf("const GFXfont %s PROGMEM = {\n", fontName);

printf(" (uint8\_t \*)%sBitmaps,\n", fontName);

printf(" (GFXglyph \*)%sGlyphs,\n", fontName);

if (face->size->metrics.height == 0) {

// No face height info, assume fixed width and get from a glyph.

printf(" 0x%02X, 0x%02X, %d };\n\n", first, last, table[0].height);

} else {

printf(" 0x%02X, 0x%02X, %ld };\n\n", first, last,

face->size->metrics.height >> 6);

}

printf("// Approx. %d bytes\n", bitmapOffset + (last - first + 1) \* 7 + 7);

// Size estimate is based on AVR struct and pointer sizes;

// actual size may vary.

FT\_Done\_FreeType(library);

return 0;

}

/\* -------------------------------------------------------------------------

Character metrics are slightly different from classic GFX & ftGFX.

In classic GFX: cursor position is the upper-left pixel of each 5x7

character; lower extent of most glyphs (except those w/descenders)

is +6 pixels in Y direction.

W/new GFX fonts: cursor position is on baseline, where baseline is

'inclusive' (containing the bottom-most row of pixels in most symbols,

except those with descenders; ftGFX is one pixel lower).

Cursor Y will be moved automatically when switching between classic

and new fonts. If you switch fonts, any print() calls will continue

along the same baseline.

...........#####.. -- yOffset

..........######..

..........######..

.........#######..

........#########.

\* = Cursor pos. ........#########.

.......##########.

......#####..####.

......#####..####.

\*.#.. .....#####...####.

.#.#. ....##############

#...# ...###############

#...# ...###############

##### ..#####......#####

#...# .#####.......#####

====== #...# ====== #\*###.........#### ======= Baseline

|| xOffset

glyph->xOffset and yOffset are pixel offsets, in GFX coordinate space

(+Y is down), from the cursor position to the top-left pixel of the

glyph bitmap. i.e. yOffset is typically negative, xOffset is typically

zero but a few glyphs will have other values (even negative xOffsets

sometimes, totally normal). glyph->xAdvance is the distance to move

the cursor on the X axis after drawing the corresponding symbol.

There's also some changes with regard to 'background' color and new GFX

fonts (classic fonts unchanged). See Adafruit\_GFX.cpp for explanation.

\*/

#endif /\* !ARDUINO \*/