/\*

This is the core graphics library for all our displays, providing a common

set of graphics primitives (points, lines, circles, etc.). It needs to be

paired with a hardware-specific library for each display device we carry

(to handle the lower-level functions).

Adafruit invests time and resources providing this open source code, please

support Adafruit & open-source hardware by purchasing products from Adafruit!

Copyright (c) 2013 Adafruit Industries. All rights reserved.

Redistribution and use in source and binary forms, with or without

modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice,

this list of conditions and the following disclaimer.

- Redistributions in binary form must reproduce the above copyright notice,

this list of conditions and the following disclaimer in the documentation

and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"

AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE

IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE

ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE

LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR

CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF

SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS

INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN

CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)

ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE

POSSIBILITY OF SUCH DAMAGE.

\*/

#include "Adafruit\_GFX.h"

#include "glcdfont.c"

#ifdef \_\_AVR\_\_

#include <avr/pgmspace.h>

#elif defined(ESP8266) || defined(ESP32)

#include <pgmspace.h>

#endif

// Many (but maybe not all) non-AVR board installs define macros

// for compatibility with existing PROGMEM-reading AVR code.

// Do our own checks and defines here for good measure...

#ifndef pgm\_read\_byte

#define pgm\_read\_byte(addr) (\*(const unsigned char \*)(addr))

#endif

#ifndef pgm\_read\_word

#define pgm\_read\_word(addr) (\*(const unsigned short \*)(addr))

#endif

#ifndef pgm\_read\_dword

#define pgm\_read\_dword(addr) (\*(const unsigned long \*)(addr))

#endif

// Pointers are a peculiar case...typically 16-bit on AVR boards,

// 32 bits elsewhere. Try to accommodate both...

#if !defined(\_\_INT\_MAX\_\_) || (\_\_INT\_MAX\_\_ > 0xFFFF)

#define pgm\_read\_pointer(addr) ((void \*)pgm\_read\_dword(addr))

#else

#define pgm\_read\_pointer(addr) ((void \*)pgm\_read\_word(addr))

#endif

inline GFXglyph \*pgm\_read\_glyph\_ptr(const GFXfont \*gfxFont, uint8\_t c) {

#ifdef \_\_AVR\_\_

return &(((GFXglyph \*)pgm\_read\_pointer(&gfxFont->glyph))[c]);

#else

// expression in \_\_AVR\_\_ section may generate "dereferencing type-punned

// pointer will break strict-aliasing rules" warning In fact, on other

// platforms (such as STM32) there is no need to do this pointer magic as

// program memory may be read in a usual way So expression may be simplified

return gfxFont->glyph + c;

#endif //\_\_AVR\_\_

}

inline uint8\_t \*pgm\_read\_bitmap\_ptr(const GFXfont \*gfxFont) {

#ifdef \_\_AVR\_\_

return (uint8\_t \*)pgm\_read\_pointer(&gfxFont->bitmap);

#else

// expression in \_\_AVR\_\_ section generates "dereferencing type-punned pointer

// will break strict-aliasing rules" warning In fact, on other platforms (such

// as STM32) there is no need to do this pointer magic as program memory may

// be read in a usual way So expression may be simplified

return gfxFont->bitmap;

#endif //\_\_AVR\_\_

}

#ifndef min

#define min(a, b) (((a) < (b)) ? (a) : (b))

#endif

#ifndef \_swap\_int16\_t

#define \_swap\_int16\_t(a, b) \

{ \

int16\_t t = a; \

a = b; \

b = t; \

}

#endif

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*!

@brief Instatiate a GFX context for graphics! Can only be done by a

superclass

@param w Display width, in pixels

@param h Display height, in pixels

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Adafruit\_GFX::Adafruit\_GFX(int16\_t w, int16\_t h) : WIDTH(w), HEIGHT(h) {

\_width = WIDTH;

\_height = HEIGHT;

rotation = 0;

cursor\_y = cursor\_x = 0;

textsize\_x = textsize\_y = 1;

textcolor = textbgcolor = 0xFFFF;

wrap = true;

\_cp437 = false;

gfxFont = NULL;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*!

@brief Write a line. Bresenham's algorithm - thx wikpedia

@param x0 Start point x coordinate

@param y0 Start point y coordinate

@param x1 End point x coordinate

@param y1 End point y coordinate

@param color 16-bit 5-6-5 Color to draw with

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Adafruit\_GFX::writeLine(int16\_t x0, int16\_t y0, int16\_t x1, int16\_t y1,

uint16\_t color) {

#if defined(ESP8266)

yield();

#endif

int16\_t steep = abs(y1 - y0) > abs(x1 - x0);

if (steep) {

\_swap\_int16\_t(x0, y0);

\_swap\_int16\_t(x1, y1);

}

if (x0 > x1) {

\_swap\_int16\_t(x0, x1);

\_swap\_int16\_t(y0, y1);

}

int16\_t dx, dy;

dx = x1 - x0;

dy = abs(y1 - y0);

int16\_t err = dx / 2;

int16\_t ystep;

if (y0 < y1) {

ystep = 1;

} else {

ystep = -1;

}

for (; x0 <= x1; x0++) {

if (steep) {

writePixel(y0, x0, color);

} else {

writePixel(x0, y0, color);

}

err -= dy;

if (err < 0) {

y0 += ystep;

err += dx;

}

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*!

@brief Start a display-writing routine, overwrite in subclasses.

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Adafruit\_GFX::startWrite() {}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*!

@brief Write a pixel, overwrite in subclasses if startWrite is defined!

@param x x coordinate

@param y y coordinate

@param color 16-bit 5-6-5 Color to fill with

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Adafruit\_GFX::writePixel(int16\_t x, int16\_t y, uint16\_t color) {

drawPixel(x, y, color);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*!

@brief Write a perfectly vertical line, overwrite in subclasses if

startWrite is defined!

@param x Top-most x coordinate

@param y Top-most y coordinate

@param h Height in pixels

@param color 16-bit 5-6-5 Color to fill with

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Adafruit\_GFX::writeFastVLine(int16\_t x, int16\_t y, int16\_t h,

uint16\_t color) {

// Overwrite in subclasses if startWrite is defined!

// Can be just writeLine(x, y, x, y+h-1, color);

// or writeFillRect(x, y, 1, h, color);

drawFastVLine(x, y, h, color);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*!

@brief Write a perfectly horizontal line, overwrite in subclasses if

startWrite is defined!

@param x Left-most x coordinate

@param y Left-most y coordinate

@param w Width in pixels

@param color 16-bit 5-6-5 Color to fill with

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Adafruit\_GFX::writeFastHLine(int16\_t x, int16\_t y, int16\_t w,

uint16\_t color) {

// Overwrite in subclasses if startWrite is defined!

// Example: writeLine(x, y, x+w-1, y, color);

// or writeFillRect(x, y, w, 1, color);

drawFastHLine(x, y, w, color);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*!

@brief Write a rectangle completely with one color, overwrite in

subclasses if startWrite is defined!

@param x Top left corner x coordinate

@param y Top left corner y coordinate

@param w Width in pixels

@param h Height in pixels

@param color 16-bit 5-6-5 Color to fill with

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Adafruit\_GFX::writeFillRect(int16\_t x, int16\_t y, int16\_t w, int16\_t h,

uint16\_t color) {

// Overwrite in subclasses if desired!

fillRect(x, y, w, h, color);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*!

@brief End a display-writing routine, overwrite in subclasses if

startWrite is defined!

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Adafruit\_GFX::endWrite() {}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*!

@brief Draw a perfectly vertical line (this is often optimized in a

subclass!)

@param x Top-most x coordinate

@param y Top-most y coordinate

@param h Height in pixels

@param color 16-bit 5-6-5 Color to fill with

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Adafruit\_GFX::drawFastVLine(int16\_t x, int16\_t y, int16\_t h,

uint16\_t color) {

startWrite();

writeLine(x, y, x, y + h - 1, color);

endWrite();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*!

@brief Draw a perfectly horizontal line (this is often optimized in a

subclass!)

@param x Left-most x coordinate

@param y Left-most y coordinate

@param w Width in pixels

@param color 16-bit 5-6-5 Color to fill with

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Adafruit\_GFX::drawFastHLine(int16\_t x, int16\_t y, int16\_t w,

uint16\_t color) {

startWrite();

writeLine(x, y, x + w - 1, y, color);

endWrite();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*!

@brief Fill a rectangle completely with one color. Update in subclasses if

desired!

@param x Top left corner x coordinate

@param y Top left corner y coordinate

@param w Width in pixels

@param h Height in pixels

@param color 16-bit 5-6-5 Color to fill with

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Adafruit\_GFX::fillRect(int16\_t x, int16\_t y, int16\_t w, int16\_t h,

uint16\_t color) {

startWrite();

for (int16\_t i = x; i < x + w; i++) {

writeFastVLine(i, y, h, color);

}

endWrite();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*!

@brief Fill the screen completely with one color. Update in subclasses if

desired!

@param color 16-bit 5-6-5 Color to fill with

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Adafruit\_GFX::fillScreen(uint16\_t color) {

fillRect(0, 0, \_width, \_height, color);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*!

@brief Draw a line

@param x0 Start point x coordinate

@param y0 Start point y coordinate

@param x1 End point x coordinate

@param y1 End point y coordinate

@param color 16-bit 5-6-5 Color to draw with

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Adafruit\_GFX::drawLine(int16\_t x0, int16\_t y0, int16\_t x1, int16\_t y1,

uint16\_t color) {

// Update in subclasses if desired!

if (x0 == x1) {

if (y0 > y1)

\_swap\_int16\_t(y0, y1);

drawFastVLine(x0, y0, y1 - y0 + 1, color);

} else if (y0 == y1) {

if (x0 > x1)

\_swap\_int16\_t(x0, x1);

drawFastHLine(x0, y0, x1 - x0 + 1, color);

} else {

startWrite();

writeLine(x0, y0, x1, y1, color);

endWrite();

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*!

@brief Draw a circle outline

@param x0 Center-point x coordinate

@param y0 Center-point y coordinate

@param r Radius of circle

@param color 16-bit 5-6-5 Color to draw with

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Adafruit\_GFX::drawCircle(int16\_t x0, int16\_t y0, int16\_t r,

uint16\_t color) {

#if defined(ESP8266)

yield();

#endif

int16\_t f = 1 - r;

int16\_t ddF\_x = 1;

int16\_t ddF\_y = -2 \* r;

int16\_t x = 0;

int16\_t y = r;

startWrite();

writePixel(x0, y0 + r, color);

writePixel(x0, y0 - r, color);

writePixel(x0 + r, y0, color);

writePixel(x0 - r, y0, color);

while (x < y) {

if (f >= 0) {

y--;

ddF\_y += 2;

f += ddF\_y;

}

x++;

ddF\_x += 2;

f += ddF\_x;

writePixel(x0 + x, y0 + y, color);

writePixel(x0 - x, y0 + y, color);

writePixel(x0 + x, y0 - y, color);

writePixel(x0 - x, y0 - y, color);

writePixel(x0 + y, y0 + x, color);

writePixel(x0 - y, y0 + x, color);

writePixel(x0 + y, y0 - x, color);

writePixel(x0 - y, y0 - x, color);

}

endWrite();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*!

@brief Quarter-circle drawer, used to do circles and roundrects

@param x0 Center-point x coordinate

@param y0 Center-point y coordinate

@param r Radius of circle

@param cornername Mask bit #1 or bit #2 to indicate which quarters of

the circle we're doing

@param color 16-bit 5-6-5 Color to draw with

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Adafruit\_GFX::drawCircleHelper(int16\_t x0, int16\_t y0, int16\_t r,

uint8\_t cornername, uint16\_t color) {

int16\_t f = 1 - r;

int16\_t ddF\_x = 1;

int16\_t ddF\_y = -2 \* r;

int16\_t x = 0;

int16\_t y = r;

while (x < y) {

if (f >= 0) {

y--;

ddF\_y += 2;

f += ddF\_y;

}

x++;

ddF\_x += 2;

f += ddF\_x;

if (cornername & 0x4) {

writePixel(x0 + x, y0 + y, color);

writePixel(x0 + y, y0 + x, color);

}

if (cornername & 0x2) {

writePixel(x0 + x, y0 - y, color);

writePixel(x0 + y, y0 - x, color);

}

if (cornername & 0x8) {

writePixel(x0 - y, y0 + x, color);

writePixel(x0 - x, y0 + y, color);

}

if (cornername & 0x1) {

writePixel(x0 - y, y0 - x, color);

writePixel(x0 - x, y0 - y, color);

}

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*!

@brief Draw a circle with filled color

@param x0 Center-point x coordinate

@param y0 Center-point y coordinate

@param r Radius of circle

@param color 16-bit 5-6-5 Color to fill with

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Adafruit\_GFX::fillCircle(int16\_t x0, int16\_t y0, int16\_t r,

uint16\_t color) {

startWrite();

writeFastVLine(x0, y0 - r, 2 \* r + 1, color);

fillCircleHelper(x0, y0, r, 3, 0, color);

endWrite();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*!

@brief Quarter-circle drawer with fill, used for circles and roundrects

@param x0 Center-point x coordinate

@param y0 Center-point y coordinate

@param r Radius of circle

@param corners Mask bits indicating which quarters we're doing

@param delta Offset from center-point, used for round-rects

@param color 16-bit 5-6-5 Color to fill with

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Adafruit\_GFX::fillCircleHelper(int16\_t x0, int16\_t y0, int16\_t r,

uint8\_t corners, int16\_t delta,

uint16\_t color) {

int16\_t f = 1 - r;

int16\_t ddF\_x = 1;

int16\_t ddF\_y = -2 \* r;

int16\_t x = 0;

int16\_t y = r;

int16\_t px = x;

int16\_t py = y;

delta++; // Avoid some +1's in the loop

while (x < y) {

if (f >= 0) {

y--;

ddF\_y += 2;

f += ddF\_y;

}

x++;

ddF\_x += 2;

f += ddF\_x;

// These checks avoid double-drawing certain lines, important

// for the SSD1306 library which has an INVERT drawing mode.

if (x < (y + 1)) {

if (corners & 1)

writeFastVLine(x0 + x, y0 - y, 2 \* y + delta, color);

if (corners & 2)

writeFastVLine(x0 - x, y0 - y, 2 \* y + delta, color);

}

if (y != py) {

if (corners & 1)

writeFastVLine(x0 + py, y0 - px, 2 \* px + delta, color);

if (corners & 2)

writeFastVLine(x0 - py, y0 - px, 2 \* px + delta, color);

py = y;

}

px = x;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*!

@brief Draw a rectangle with no fill color

@param x Top left corner x coordinate

@param y Top left corner y coordinate

@param w Width in pixels

@param h Height in pixels

@param color 16-bit 5-6-5 Color to draw with

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Adafruit\_GFX::drawRect(int16\_t x, int16\_t y, int16\_t w, int16\_t h,

uint16\_t color) {

startWrite();

writeFastHLine(x, y, w, color);

writeFastHLine(x, y + h - 1, w, color);

writeFastVLine(x, y, h, color);

writeFastVLine(x + w - 1, y, h, color);

endWrite();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*!

@brief Draw a rounded rectangle with no fill color

@param x Top left corner x coordinate

@param y Top left corner y coordinate

@param w Width in pixels

@param h Height in pixels

@param r Radius of corner rounding

@param color 16-bit 5-6-5 Color to draw with

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Adafruit\_GFX::drawRoundRect(int16\_t x, int16\_t y, int16\_t w, int16\_t h,

int16\_t r, uint16\_t color) {

int16\_t max\_radius = ((w < h) ? w : h) / 2; // 1/2 minor axis

if (r > max\_radius)

r = max\_radius;

// smarter version

startWrite();

writeFastHLine(x + r, y, w - 2 \* r, color); // Top

writeFastHLine(x + r, y + h - 1, w - 2 \* r, color); // Bottom

writeFastVLine(x, y + r, h - 2 \* r, color); // Left

writeFastVLine(x + w - 1, y + r, h - 2 \* r, color); // Right

// draw four corners

drawCircleHelper(x + r, y + r, r, 1, color);

drawCircleHelper(x + w - r - 1, y + r, r, 2, color);

drawCircleHelper(x + w - r - 1, y + h - r - 1, r, 4, color);

drawCircleHelper(x + r, y + h - r - 1, r, 8, color);

endWrite();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*!

@brief Draw a rounded rectangle with fill color

@param x Top left corner x coordinate

@param y Top left corner y coordinate

@param w Width in pixels

@param h Height in pixels

@param r Radius of corner rounding

@param color 16-bit 5-6-5 Color to draw/fill with

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Adafruit\_GFX::fillRoundRect(int16\_t x, int16\_t y, int16\_t w, int16\_t h,

int16\_t r, uint16\_t color) {

int16\_t max\_radius = ((w < h) ? w : h) / 2; // 1/2 minor axis

if (r > max\_radius)

r = max\_radius;

// smarter version

startWrite();

writeFillRect(x + r, y, w - 2 \* r, h, color);

// draw four corners

fillCircleHelper(x + w - r - 1, y + r, r, 1, h - 2 \* r - 1, color);

fillCircleHelper(x + r, y + r, r, 2, h - 2 \* r - 1, color);

endWrite();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*!

@brief Draw a triangle with no fill color

@param x0 Vertex #0 x coordinate

@param y0 Vertex #0 y coordinate

@param x1 Vertex #1 x coordinate

@param y1 Vertex #1 y coordinate

@param x2 Vertex #2 x coordinate

@param y2 Vertex #2 y coordinate

@param color 16-bit 5-6-5 Color to draw with

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Adafruit\_GFX::drawTriangle(int16\_t x0, int16\_t y0, int16\_t x1, int16\_t y1,

int16\_t x2, int16\_t y2, uint16\_t color) {

drawLine(x0, y0, x1, y1, color);

drawLine(x1, y1, x2, y2, color);

drawLine(x2, y2, x0, y0, color);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*!

@brief Draw a triangle with color-fill

@param x0 Vertex #0 x coordinate

@param y0 Vertex #0 y coordinate

@param x1 Vertex #1 x coordinate

@param y1 Vertex #1 y coordinate

@param x2 Vertex #2 x coordinate

@param y2 Vertex #2 y coordinate

@param color 16-bit 5-6-5 Color to fill/draw with

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Adafruit\_GFX::fillTriangle(int16\_t x0, int16\_t y0, int16\_t x1, int16\_t y1,

int16\_t x2, int16\_t y2, uint16\_t color) {

int16\_t a, b, y, last;

// Sort coordinates by Y order (y2 >= y1 >= y0)

if (y0 > y1) {

\_swap\_int16\_t(y0, y1);

\_swap\_int16\_t(x0, x1);

}

if (y1 > y2) {

\_swap\_int16\_t(y2, y1);

\_swap\_int16\_t(x2, x1);

}

if (y0 > y1) {

\_swap\_int16\_t(y0, y1);

\_swap\_int16\_t(x0, x1);

}

startWrite();

if (y0 == y2) { // Handle awkward all-on-same-line case as its own thing

a = b = x0;

if (x1 < a)

a = x1;

else if (x1 > b)

b = x1;

if (x2 < a)

a = x2;

else if (x2 > b)

b = x2;

writeFastHLine(a, y0, b - a + 1, color);

endWrite();

return;

}

int16\_t dx01 = x1 - x0, dy01 = y1 - y0, dx02 = x2 - x0, dy02 = y2 - y0,

dx12 = x2 - x1, dy12 = y2 - y1;

int32\_t sa = 0, sb = 0;

// For upper part of triangle, find scanline crossings for segments

// 0-1 and 0-2. If y1=y2 (flat-bottomed triangle), the scanline y1

// is included here (and second loop will be skipped, avoiding a /0

// error there), otherwise scanline y1 is skipped here and handled

// in the second loop...which also avoids a /0 error here if y0=y1

// (flat-topped triangle).

if (y1 == y2)

last = y1; // Include y1 scanline

else

last = y1 - 1; // Skip it

for (y = y0; y <= last; y++) {

a = x0 + sa / dy01;

b = x0 + sb / dy02;

sa += dx01;

sb += dx02;

/\* longhand:

a = x0 + (x1 - x0) \* (y - y0) / (y1 - y0);

b = x0 + (x2 - x0) \* (y - y0) / (y2 - y0);

\*/

if (a > b)

\_swap\_int16\_t(a, b);

writeFastHLine(a, y, b - a + 1, color);

}

// For lower part of triangle, find scanline crossings for segments

// 0-2 and 1-2. This loop is skipped if y1=y2.

sa = (int32\_t)dx12 \* (y - y1);

sb = (int32\_t)dx02 \* (y - y0);

for (; y <= y2; y++) {

a = x1 + sa / dy12;

b = x0 + sb / dy02;

sa += dx12;

sb += dx02;

/\* longhand:

a = x1 + (x2 - x1) \* (y - y1) / (y2 - y1);

b = x0 + (x2 - x0) \* (y - y0) / (y2 - y0);

\*/

if (a > b)

\_swap\_int16\_t(a, b);

writeFastHLine(a, y, b - a + 1, color);

}

endWrite();

}

// BITMAP / XBITMAP / GRAYSCALE / RGB BITMAP FUNCTIONS ---------------------

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*!

@brief Draw a PROGMEM-resident 1-bit image at the specified (x,y)

position, using the specified foreground color (unset bits are transparent).

@param x Top left corner x coordinate

@param y Top left corner y coordinate

@param bitmap byte array with monochrome bitmap

@param w Width of bitmap in pixels

@param h Height of bitmap in pixels

@param color 16-bit 5-6-5 Color to draw with

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Adafruit\_GFX::drawBitmap(int16\_t x, int16\_t y, const uint8\_t bitmap[],

int16\_t w, int16\_t h, uint16\_t color) {

int16\_t byteWidth = (w + 7) / 8; // Bitmap scanline pad = whole byte

uint8\_t b = 0;

startWrite();

for (int16\_t j = 0; j < h; j++, y++) {

for (int16\_t i = 0; i < w; i++) {

if (i & 7)

b <<= 1;

else

b = pgm\_read\_byte(&bitmap[j \* byteWidth + i / 8]);

if (b & 0x80)

writePixel(x + i, y, color);

}

}

endWrite();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*!

@brief Draw a PROGMEM-resident 1-bit image at the specified (x,y)

position, using the specified foreground (for set bits) and background (unset

bits) colors.

@param x Top left corner x coordinate

@param y Top left corner y coordinate

@param bitmap byte array with monochrome bitmap

@param w Width of bitmap in pixels

@param h Height of bitmap in pixels

@param color 16-bit 5-6-5 Color to draw pixels with

@param bg 16-bit 5-6-5 Color to draw background with

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Adafruit\_GFX::drawBitmap(int16\_t x, int16\_t y, const uint8\_t bitmap[],

int16\_t w, int16\_t h, uint16\_t color,

uint16\_t bg) {

int16\_t byteWidth = (w + 7) / 8; // Bitmap scanline pad = whole byte

uint8\_t b = 0;

startWrite();

for (int16\_t j = 0; j < h; j++, y++) {

for (int16\_t i = 0; i < w; i++) {

if (i & 7)

b <<= 1;

else

b = pgm\_read\_byte(&bitmap[j \* byteWidth + i / 8]);

writePixel(x + i, y, (b & 0x80) ? color : bg);

}

}

endWrite();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*!

@brief Draw a RAM-resident 1-bit image at the specified (x,y) position,

using the specified foreground color (unset bits are transparent).

@param x Top left corner x coordinate

@param y Top left corner y coordinate

@param bitmap byte array with monochrome bitmap

@param w Width of bitmap in pixels

@param h Height of bitmap in pixels

@param color 16-bit 5-6-5 Color to draw with

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Adafruit\_GFX::drawBitmap(int16\_t x, int16\_t y, uint8\_t \*bitmap, int16\_t w,

int16\_t h, uint16\_t color) {

int16\_t byteWidth = (w + 7) / 8; // Bitmap scanline pad = whole byte

uint8\_t b = 0;

startWrite();

for (int16\_t j = 0; j < h; j++, y++) {

for (int16\_t i = 0; i < w; i++) {

if (i & 7)

b <<= 1;

else

b = bitmap[j \* byteWidth + i / 8];

if (b & 0x80)

writePixel(x + i, y, color);

}

}

endWrite();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*!

@brief Draw a RAM-resident 1-bit image at the specified (x,y) position,

using the specified foreground (for set bits) and background (unset bits)

colors.

@param x Top left corner x coordinate

@param y Top left corner y coordinate

@param bitmap byte array with monochrome bitmap

@param w Width of bitmap in pixels

@param h Height of bitmap in pixels

@param color 16-bit 5-6-5 Color to draw pixels with

@param bg 16-bit 5-6-5 Color to draw background with

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Adafruit\_GFX::drawBitmap(int16\_t x, int16\_t y, uint8\_t \*bitmap, int16\_t w,

int16\_t h, uint16\_t color, uint16\_t bg) {

int16\_t byteWidth = (w + 7) / 8; // Bitmap scanline pad = whole byte

uint8\_t b = 0;

startWrite();

for (int16\_t j = 0; j < h; j++, y++) {

for (int16\_t i = 0; i < w; i++) {

if (i & 7)

b <<= 1;

else

b = bitmap[j \* byteWidth + i / 8];

writePixel(x + i, y, (b & 0x80) ? color : bg);

}

}

endWrite();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*!

@brief Draw PROGMEM-resident XBitMap Files (\*.xbm), exported from GIMP.

Usage: Export from GIMP to \*.xbm, rename \*.xbm to \*.c and open in editor.

C Array can be directly used with this function.

There is no RAM-resident version of this function; if generating bitmaps

in RAM, use the format defined by drawBitmap() and call that instead.

@param x Top left corner x coordinate

@param y Top left corner y coordinate

@param bitmap byte array with monochrome bitmap

@param w Width of bitmap in pixels

@param h Height of bitmap in pixels

@param color 16-bit 5-6-5 Color to draw pixels with

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Adafruit\_GFX::drawXBitmap(int16\_t x, int16\_t y, const uint8\_t bitmap[],

int16\_t w, int16\_t h, uint16\_t color) {

int16\_t byteWidth = (w + 7) / 8; // Bitmap scanline pad = whole byte

uint8\_t b = 0;

startWrite();

for (int16\_t j = 0; j < h; j++, y++) {

for (int16\_t i = 0; i < w; i++) {

if (i & 7)

b >>= 1;

else

b = pgm\_read\_byte(&bitmap[j \* byteWidth + i / 8]);

// Nearly identical to drawBitmap(), only the bit order

// is reversed here (left-to-right = LSB to MSB):

if (b & 0x01)

writePixel(x + i, y, color);

}

}

endWrite();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*!

@brief Draw a PROGMEM-resident 8-bit image (grayscale) at the specified

(x,y) pos. Specifically for 8-bit display devices such as IS31FL3731; no

color reduction/expansion is performed.

@param x Top left corner x coordinate

@param y Top left corner y coordinate

@param bitmap byte array with grayscale bitmap

@param w Width of bitmap in pixels

@param h Height of bitmap in pixels

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Adafruit\_GFX::drawGrayscaleBitmap(int16\_t x, int16\_t y,

const uint8\_t bitmap[], int16\_t w,

int16\_t h) {

startWrite();

for (int16\_t j = 0; j < h; j++, y++) {

for (int16\_t i = 0; i < w; i++) {

writePixel(x + i, y, (uint8\_t)pgm\_read\_byte(&bitmap[j \* w + i]));

}

}

endWrite();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*!

@brief Draw a RAM-resident 8-bit image (grayscale) at the specified (x,y)

pos. Specifically for 8-bit display devices such as IS31FL3731; no color

reduction/expansion is performed.

@param x Top left corner x coordinate

@param y Top left corner y coordinate

@param bitmap byte array with grayscale bitmap

@param w Width of bitmap in pixels

@param h Height of bitmap in pixels

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Adafruit\_GFX::drawGrayscaleBitmap(int16\_t x, int16\_t y, uint8\_t \*bitmap,

int16\_t w, int16\_t h) {

startWrite();

for (int16\_t j = 0; j < h; j++, y++) {

for (int16\_t i = 0; i < w; i++) {

writePixel(x + i, y, bitmap[j \* w + i]);

}

}

endWrite();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*!

@brief Draw a PROGMEM-resident 8-bit image (grayscale) with a 1-bit mask

(set bits = opaque, unset bits = clear) at the specified (x,y) position.

BOTH buffers (grayscale and mask) must be PROGMEM-resident.

Specifically for 8-bit display devices such as IS31FL3731; no color

reduction/expansion is performed.

@param x Top left corner x coordinate

@param y Top left corner y coordinate

@param bitmap byte array with grayscale bitmap

@param mask byte array with mask bitmap

@param w Width of bitmap in pixels

@param h Height of bitmap in pixels

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Adafruit\_GFX::drawGrayscaleBitmap(int16\_t x, int16\_t y,

const uint8\_t bitmap[],

const uint8\_t mask[], int16\_t w,

int16\_t h) {

int16\_t bw = (w + 7) / 8; // Bitmask scanline pad = whole byte

uint8\_t b = 0;

startWrite();

for (int16\_t j = 0; j < h; j++, y++) {

for (int16\_t i = 0; i < w; i++) {

if (i & 7)

b <<= 1;

else

b = pgm\_read\_byte(&mask[j \* bw + i / 8]);

if (b & 0x80) {

writePixel(x + i, y, (uint8\_t)pgm\_read\_byte(&bitmap[j \* w + i]));

}

}

}

endWrite();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*!

@brief Draw a RAM-resident 8-bit image (grayscale) with a 1-bit mask

(set bits = opaque, unset bits = clear) at the specified (x,y) position.

BOTH buffers (grayscale and mask) must be RAM-residentt, no mix-and-match

Specifically for 8-bit display devices such as IS31FL3731; no color

reduction/expansion is performed.

@param x Top left corner x coordinate

@param y Top left corner y coordinate

@param bitmap byte array with grayscale bitmap

@param mask byte array with mask bitmap

@param w Width of bitmap in pixels

@param h Height of bitmap in pixels

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Adafruit\_GFX::drawGrayscaleBitmap(int16\_t x, int16\_t y, uint8\_t \*bitmap,

uint8\_t \*mask, int16\_t w, int16\_t h) {

int16\_t bw = (w + 7) / 8; // Bitmask scanline pad = whole byte

uint8\_t b = 0;

startWrite();

for (int16\_t j = 0; j < h; j++, y++) {

for (int16\_t i = 0; i < w; i++) {

if (i & 7)

b <<= 1;

else

b = mask[j \* bw + i / 8];

if (b & 0x80) {

writePixel(x + i, y, bitmap[j \* w + i]);

}

}

}

endWrite();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*!

@brief Draw a PROGMEM-resident 16-bit image (RGB 5/6/5) at the specified

(x,y) position. For 16-bit display devices; no color reduction performed.

@param x Top left corner x coordinate

@param y Top left corner y coordinate

@param bitmap byte array with 16-bit color bitmap

@param w Width of bitmap in pixels

@param h Height of bitmap in pixels

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Adafruit\_GFX::drawRGBBitmap(int16\_t x, int16\_t y, const uint16\_t bitmap[],

int16\_t w, int16\_t h) {

startWrite();

for (int16\_t j = 0; j < h; j++, y++) {

for (int16\_t i = 0; i < w; i++) {

writePixel(x + i, y, pgm\_read\_word(&bitmap[j \* w + i]));

}

}

endWrite();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*!

@brief Draw a RAM-resident 16-bit image (RGB 5/6/5) at the specified (x,y)

position. For 16-bit display devices; no color reduction performed.

@param x Top left corner x coordinate

@param y Top left corner y coordinate

@param bitmap byte array with 16-bit color bitmap

@param w Width of bitmap in pixels

@param h Height of bitmap in pixels

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Adafruit\_GFX::drawRGBBitmap(int16\_t x, int16\_t y, uint16\_t \*bitmap,

int16\_t w, int16\_t h) {

startWrite();

for (int16\_t j = 0; j < h; j++, y++) {

for (int16\_t i = 0; i < w; i++) {

writePixel(x + i, y, bitmap[j \* w + i]);

}

}

endWrite();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*!

@brief Draw a PROGMEM-resident 16-bit image (RGB 5/6/5) with a 1-bit mask

(set bits = opaque, unset bits = clear) at the specified (x,y) position. BOTH

buffers (color and mask) must be PROGMEM-resident. For 16-bit display

devices; no color reduction performed.

@param x Top left corner x coordinate

@param y Top left corner y coordinate

@param bitmap byte array with 16-bit color bitmap

@param mask byte array with monochrome mask bitmap

@param w Width of bitmap in pixels

@param h Height of bitmap in pixels

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Adafruit\_GFX::drawRGBBitmap(int16\_t x, int16\_t y, const uint16\_t bitmap[],

const uint8\_t mask[], int16\_t w, int16\_t h) {

int16\_t bw = (w + 7) / 8; // Bitmask scanline pad = whole byte

uint8\_t b = 0;

startWrite();

for (int16\_t j = 0; j < h; j++, y++) {

for (int16\_t i = 0; i < w; i++) {

if (i & 7)

b <<= 1;

else

b = pgm\_read\_byte(&mask[j \* bw + i / 8]);

if (b & 0x80) {

writePixel(x + i, y, pgm\_read\_word(&bitmap[j \* w + i]));

}

}

}

endWrite();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*!

@brief Draw a RAM-resident 16-bit image (RGB 5/6/5) with a 1-bit mask (set

bits = opaque, unset bits = clear) at the specified (x,y) position. BOTH

buffers (color and mask) must be RAM-resident. For 16-bit display devices; no

color reduction performed.

@param x Top left corner x coordinate

@param y Top left corner y coordinate

@param bitmap byte array with 16-bit color bitmap

@param mask byte array with monochrome mask bitmap

@param w Width of bitmap in pixels

@param h Height of bitmap in pixels

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Adafruit\_GFX::drawRGBBitmap(int16\_t x, int16\_t y, uint16\_t \*bitmap,

uint8\_t \*mask, int16\_t w, int16\_t h) {

int16\_t bw = (w + 7) / 8; // Bitmask scanline pad = whole byte

uint8\_t b = 0;

startWrite();

for (int16\_t j = 0; j < h; j++, y++) {

for (int16\_t i = 0; i < w; i++) {

if (i & 7)

b <<= 1;

else

b = mask[j \* bw + i / 8];

if (b & 0x80) {

writePixel(x + i, y, bitmap[j \* w + i]);

}

}

}

endWrite();

}

// TEXT- AND CHARACTER-HANDLING FUNCTIONS ----------------------------------

// Draw a character

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*!

@brief Draw a single character

@param x Bottom left corner x coordinate

@param y Bottom left corner y coordinate

@param c The 8-bit font-indexed character (likely ascii)

@param color 16-bit 5-6-5 Color to draw chraracter with

@param bg 16-bit 5-6-5 Color to fill background with (if same as color,

no background)

@param size Font magnification level, 1 is 'original' size

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Adafruit\_GFX::drawChar(int16\_t x, int16\_t y, unsigned char c,

uint16\_t color, uint16\_t bg, uint8\_t size) {

drawChar(x, y, c, color, bg, size, size);

}

// Draw a character

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*!

@brief Draw a single character

@param x Bottom left corner x coordinate

@param y Bottom left corner y coordinate

@param c The 8-bit font-indexed character (likely ascii)

@param color 16-bit 5-6-5 Color to draw chraracter with

@param bg 16-bit 5-6-5 Color to fill background with (if same as color,

no background)

@param size\_x Font magnification level in X-axis, 1 is 'original' size

@param size\_y Font magnification level in Y-axis, 1 is 'original' size

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Adafruit\_GFX::drawChar(int16\_t x, int16\_t y, unsigned char c,

uint16\_t color, uint16\_t bg, uint8\_t size\_x,

uint8\_t size\_y) {

if (!gfxFont) { // 'Classic' built-in font

if ((x >= \_width) || // Clip right

(y >= \_height) || // Clip bottom

((x + 6 \* size\_x - 1) < 0) || // Clip left

((y + 8 \* size\_y - 1) < 0)) // Clip top

return;

if (!\_cp437 && (c >= 176))

c++; // Handle 'classic' charset behavior

startWrite();

for (int8\_t i = 0; i < 5; i++) { // Char bitmap = 5 columns

uint8\_t line = pgm\_read\_byte(&font[c \* 5 + i]);

for (int8\_t j = 0; j < 8; j++, line >>= 1) {

if (line & 1) {

if (size\_x == 1 && size\_y == 1)

writePixel(x + i, y + j, color);

else

writeFillRect(x + i \* size\_x, y + j \* size\_y, size\_x, size\_y,

color);

} else if (bg != color) {

if (size\_x == 1 && size\_y == 1)

writePixel(x + i, y + j, bg);

else

writeFillRect(x + i \* size\_x, y + j \* size\_y, size\_x, size\_y, bg);

}

}

}

if (bg != color) { // If opaque, draw vertical line for last column

if (size\_x == 1 && size\_y == 1)

writeFastVLine(x + 5, y, 8, bg);

else

writeFillRect(x + 5 \* size\_x, y, size\_x, 8 \* size\_y, bg);

}

endWrite();

} else { // Custom font

// Character is assumed previously filtered by write() to eliminate

// newlines, returns, non-printable characters, etc. Calling

// drawChar() directly with 'bad' characters of font may cause mayhem!

c -= (uint8\_t)pgm\_read\_byte(&gfxFont->first);

GFXglyph \*glyph = pgm\_read\_glyph\_ptr(gfxFont, c);

uint8\_t \*bitmap = pgm\_read\_bitmap\_ptr(gfxFont);

uint16\_t bo = pgm\_read\_word(&glyph->bitmapOffset);

uint8\_t w = pgm\_read\_byte(&glyph->width), h = pgm\_read\_byte(&glyph->height);

int8\_t xo = pgm\_read\_byte(&glyph->xOffset),

yo = pgm\_read\_byte(&glyph->yOffset);

uint8\_t xx, yy, bits = 0, bit = 0;

int16\_t xo16 = 0, yo16 = 0;

if (size\_x > 1 || size\_y > 1) {

xo16 = xo;

yo16 = yo;

}

// Todo: Add character clipping here

// NOTE: THERE IS NO 'BACKGROUND' COLOR OPTION ON CUSTOM FONTS.

// THIS IS ON PURPOSE AND BY DESIGN. The background color feature

// has typically been used with the 'classic' font to overwrite old

// screen contents with new data. This ONLY works because the

// characters are a uniform size; it's not a sensible thing to do with

// proportionally-spaced fonts with glyphs of varying sizes (and that

// may overlap). To replace previously-drawn text when using a custom

// font, use the getTextBounds() function to determine the smallest

// rectangle encompassing a string, erase the area with fillRect(),

// then draw new text. This WILL infortunately 'blink' the text, but

// is unavoidable. Drawing 'background' pixels will NOT fix this,

// only creates a new set of problems. Have an idea to work around

// this (a canvas object type for MCUs that can afford the RAM and

// displays supporting setAddrWindow() and pushColors()), but haven't

// implemented this yet.

startWrite();

for (yy = 0; yy < h; yy++) {

for (xx = 0; xx < w; xx++) {

if (!(bit++ & 7)) {

bits = pgm\_read\_byte(&bitmap[bo++]);

}

if (bits & 0x80) {

if (size\_x == 1 && size\_y == 1) {

writePixel(x + xo + xx, y + yo + yy, color);

} else {

writeFillRect(x + (xo16 + xx) \* size\_x, y + (yo16 + yy) \* size\_y,

size\_x, size\_y, color);

}

}

bits <<= 1;

}

}

endWrite();

} // End classic vs custom font

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*!

@brief Print one byte/character of data, used to support print()

@param c The 8-bit ascii character to write

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

size\_t Adafruit\_GFX::write(uint8\_t c) {

if (!gfxFont) { // 'Classic' built-in font

if (c == '\n') { // Newline?

cursor\_x = 0; // Reset x to zero,

cursor\_y += textsize\_y \* 8; // advance y one line

} else if (c != '\r') { // Ignore carriage returns

if (wrap && ((cursor\_x + textsize\_x \* 6) > \_width)) { // Off right?

cursor\_x = 0; // Reset x to zero,

cursor\_y += textsize\_y \* 8; // advance y one line

}

drawChar(cursor\_x, cursor\_y, c, textcolor, textbgcolor, textsize\_x,

textsize\_y);

cursor\_x += textsize\_x \* 6; // Advance x one char

}

} else { // Custom font

if (c == '\n') {

cursor\_x = 0;

cursor\_y +=

(int16\_t)textsize\_y \* (uint8\_t)pgm\_read\_byte(&gfxFont->yAdvance);

} else if (c != '\r') {

uint8\_t first = pgm\_read\_byte(&gfxFont->first);

if ((c >= first) && (c <= (uint8\_t)pgm\_read\_byte(&gfxFont->last))) {

GFXglyph \*glyph = pgm\_read\_glyph\_ptr(gfxFont, c - first);

uint8\_t w = pgm\_read\_byte(&glyph->width),

h = pgm\_read\_byte(&glyph->height);

if ((w > 0) && (h > 0)) { // Is there an associated bitmap?

int16\_t xo = (int8\_t)pgm\_read\_byte(&glyph->xOffset); // sic

if (wrap && ((cursor\_x + textsize\_x \* (xo + w)) > \_width)) {

cursor\_x = 0;

cursor\_y += (int16\_t)textsize\_y \*

(uint8\_t)pgm\_read\_byte(&gfxFont->yAdvance);

}

drawChar(cursor\_x, cursor\_y, c, textcolor, textbgcolor, textsize\_x,

textsize\_y);

}

cursor\_x +=

(uint8\_t)pgm\_read\_byte(&glyph->xAdvance) \* (int16\_t)textsize\_x;

}

}

}

return 1;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*!

@brief Set text 'magnification' size. Each increase in s makes 1 pixel

that much bigger.

@param s Desired text size. 1 is default 6x8, 2 is 12x16, 3 is 18x24, etc

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Adafruit\_GFX::setTextSize(uint8\_t s) { setTextSize(s, s); }

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*!

@brief Set text 'magnification' size. Each increase in s makes 1 pixel

that much bigger.

@param s\_x Desired text width magnification level in X-axis. 1 is default

@param s\_y Desired text width magnification level in Y-axis. 1 is default

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Adafruit\_GFX::setTextSize(uint8\_t s\_x, uint8\_t s\_y) {

textsize\_x = (s\_x > 0) ? s\_x : 1;

textsize\_y = (s\_y > 0) ? s\_y : 1;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*!

@brief Set rotation setting for display

@param x 0 thru 3 corresponding to 4 cardinal rotations

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Adafruit\_GFX::setRotation(uint8\_t x) {

rotation = (x & 3);

switch (rotation) {

case 0:

case 2:

\_width = WIDTH;

\_height = HEIGHT;

break;

case 1:

case 3:

\_width = HEIGHT;

\_height = WIDTH;

break;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*!

@brief Set the font to display when print()ing, either custom or default

@param f The GFXfont object, if NULL use built in 6x8 font

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Adafruit\_GFX::setFont(const GFXfont \*f) {

if (f) { // Font struct pointer passed in?

if (!gfxFont) { // And no current font struct?

// Switching from classic to new font behavior.

// Move cursor pos down 6 pixels so it's on baseline.

cursor\_y += 6;

}

} else if (gfxFont) { // NULL passed. Current font struct defined?

// Switching from new to classic font behavior.

// Move cursor pos up 6 pixels so it's at top-left of char.

cursor\_y -= 6;

}

gfxFont = (GFXfont \*)f;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*!

@brief Helper to determine size of a character with current font/size.

Broke this out as it's used by both the PROGMEM- and RAM-resident

getTextBounds() functions.

@param c The ASCII character in question

@param x Pointer to x location of character. Value is modified by

this function to advance to next character.

@param y Pointer to y location of character. Value is modified by

this function to advance to next character.

@param minx Pointer to minimum X coordinate, passed in to AND returned

by this function -- this is used to incrementally build a

bounding rectangle for a string.

@param miny Pointer to minimum Y coord, passed in AND returned.

@param maxx Pointer to maximum X coord, passed in AND returned.

@param maxy Pointer to maximum Y coord, passed in AND returned.

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Adafruit\_GFX::charBounds(unsigned char c, int16\_t \*x, int16\_t \*y,

int16\_t \*minx, int16\_t \*miny, int16\_t \*maxx,

int16\_t \*maxy) {

if (gfxFont) {

if (c == '\n') { // Newline?

\*x = 0; // Reset x to zero, advance y by one line

\*y += textsize\_y \* (uint8\_t)pgm\_read\_byte(&gfxFont->yAdvance);

} else if (c != '\r') { // Not a carriage return; is normal char

uint8\_t first = pgm\_read\_byte(&gfxFont->first),

last = pgm\_read\_byte(&gfxFont->last);

if ((c >= first) && (c <= last)) { // Char present in this font?

GFXglyph \*glyph = pgm\_read\_glyph\_ptr(gfxFont, c - first);

uint8\_t gw = pgm\_read\_byte(&glyph->width),

gh = pgm\_read\_byte(&glyph->height),

xa = pgm\_read\_byte(&glyph->xAdvance);

int8\_t xo = pgm\_read\_byte(&glyph->xOffset),

yo = pgm\_read\_byte(&glyph->yOffset);

if (wrap && ((\*x + (((int16\_t)xo + gw) \* textsize\_x)) > \_width)) {

\*x = 0; // Reset x to zero, advance y by one line

\*y += textsize\_y \* (uint8\_t)pgm\_read\_byte(&gfxFont->yAdvance);

}

int16\_t tsx = (int16\_t)textsize\_x, tsy = (int16\_t)textsize\_y,

x1 = \*x + xo \* tsx, y1 = \*y + yo \* tsy, x2 = x1 + gw \* tsx - 1,

y2 = y1 + gh \* tsy - 1;

if (x1 < \*minx)

\*minx = x1;

if (y1 < \*miny)

\*miny = y1;

if (x2 > \*maxx)

\*maxx = x2;

if (y2 > \*maxy)

\*maxy = y2;

\*x += xa \* tsx;

}

}

} else { // Default font

if (c == '\n') { // Newline?

\*x = 0; // Reset x to zero,

\*y += textsize\_y \* 8; // advance y one line

// min/max x/y unchaged -- that waits for next 'normal' character

} else if (c != '\r') { // Normal char; ignore carriage returns

if (wrap && ((\*x + textsize\_x \* 6) > \_width)) { // Off right?

\*x = 0; // Reset x to zero,

\*y += textsize\_y \* 8; // advance y one line

}

int x2 = \*x + textsize\_x \* 6 - 1, // Lower-right pixel of char

y2 = \*y + textsize\_y \* 8 - 1;

if (x2 > \*maxx)

\*maxx = x2; // Track max x, y

if (y2 > \*maxy)

\*maxy = y2;

if (\*x < \*minx)

\*minx = \*x; // Track min x, y

if (\*y < \*miny)

\*miny = \*y;

\*x += textsize\_x \* 6; // Advance x one char

}

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*!

@brief Helper to determine size of a string with current font/size.

Pass string and a cursor position, returns UL corner and W,H.

@param str The ASCII string to measure

@param x The current cursor X

@param y The current cursor Y

@param x1 The boundary X coordinate, returned by function

@param y1 The boundary Y coordinate, returned by function

@param w The boundary width, returned by function

@param h The boundary height, returned by function

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Adafruit\_GFX::getTextBounds(const char \*str, int16\_t x, int16\_t y,

int16\_t \*x1, int16\_t \*y1, uint16\_t \*w,

uint16\_t \*h) {

uint8\_t c; // Current character

int16\_t minx = 0x7FFF, miny = 0x7FFF, maxx = -1, maxy = -1; // Bound rect

// Bound rect is intentionally initialized inverted, so 1st char sets it

\*x1 = x; // Initial position is value passed in

\*y1 = y;

\*w = \*h = 0; // Initial size is zero

while ((c = \*str++)) {

// charBounds() modifies x/y to advance for each character,

// and min/max x/y are updated to incrementally build bounding rect.

charBounds(c, &x, &y, &minx, &miny, &maxx, &maxy);

}

if (maxx >= minx) { // If legit string bounds were found...

\*x1 = minx; // Update x1 to least X coord,

\*w = maxx - minx + 1; // And w to bound rect width

}

if (maxy >= miny) { // Same for height

\*y1 = miny;

\*h = maxy - miny + 1;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*!

@brief Helper to determine size of a string with current font/size. Pass

string and a cursor position, returns UL corner and W,H.

@param str The ascii string to measure (as an arduino String() class)

@param x The current cursor X

@param y The current cursor Y

@param x1 The boundary X coordinate, set by function

@param y1 The boundary Y coordinate, set by function

@param w The boundary width, set by function

@param h The boundary height, set by function

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Adafruit\_GFX::getTextBounds(const String &str, int16\_t x, int16\_t y,

int16\_t \*x1, int16\_t \*y1, uint16\_t \*w,

uint16\_t \*h) {

if (str.length() != 0) {

getTextBounds(const\_cast<char \*>(str.c\_str()), x, y, x1, y1, w, h);

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*!

@brief Helper to determine size of a PROGMEM string with current

font/size. Pass string and a cursor position, returns UL corner and W,H.

@param str The flash-memory ascii string to measure

@param x The current cursor X

@param y The current cursor Y

@param x1 The boundary X coordinate, set by function

@param y1 The boundary Y coordinate, set by function

@param w The boundary width, set by function

@param h The boundary height, set by function

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Adafruit\_GFX::getTextBounds(const \_\_FlashStringHelper \*str, int16\_t x,

int16\_t y, int16\_t \*x1, int16\_t \*y1,

uint16\_t \*w, uint16\_t \*h) {

uint8\_t \*s = (uint8\_t \*)str, c;

\*x1 = x;

\*y1 = y;

\*w = \*h = 0;

int16\_t minx = \_width, miny = \_height, maxx = -1, maxy = -1;

while ((c = pgm\_read\_byte(s++)))

charBounds(c, &x, &y, &minx, &miny, &maxx, &maxy);

if (maxx >= minx) {

\*x1 = minx;

\*w = maxx - minx + 1;

}

if (maxy >= miny) {

\*y1 = miny;

\*h = maxy - miny + 1;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*!

@brief Invert the display (ideally using built-in hardware command)

@param i True if you want to invert, false to make 'normal'

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Adafruit\_GFX::invertDisplay(bool i) {

// Do nothing, must be subclassed if supported by hardware

(void)i; // disable -Wunused-parameter warning

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*!

@brief Create a simple drawn button UI element

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Adafruit\_GFX\_Button::Adafruit\_GFX\_Button(void) { \_gfx = 0; }

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*!

@brief Initialize button with our desired color/size/settings

@param gfx Pointer to our display so we can draw to it!

@param x The X coordinate of the center of the button

@param y The Y coordinate of the center of the button

@param w Width of the buttton

@param h Height of the buttton

@param outline Color of the outline (16-bit 5-6-5 standard)

@param fill Color of the button fill (16-bit 5-6-5 standard)

@param textcolor Color of the button label (16-bit 5-6-5 standard)

@param label Ascii string of the text inside the button

@param textsize The font magnification of the label text

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Classic initButton() function: pass center & size

void Adafruit\_GFX\_Button::initButton(Adafruit\_GFX \*gfx, int16\_t x, int16\_t y,

uint16\_t w, uint16\_t h, uint16\_t outline,

uint16\_t fill, uint16\_t textcolor,

char \*label, uint8\_t textsize) {

// Tweak arguments and pass to the newer initButtonUL() function...

initButtonUL(gfx, x - (w / 2), y - (h / 2), w, h, outline, fill, textcolor,

label, textsize);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*!

@brief Initialize button with our desired color/size/settings

@param gfx Pointer to our display so we can draw to it!

@param x The X coordinate of the center of the button

@param y The Y coordinate of the center of the button

@param w Width of the buttton

@param h Height of the buttton

@param outline Color of the outline (16-bit 5-6-5 standard)

@param fill Color of the button fill (16-bit 5-6-5 standard)

@param textcolor Color of the button label (16-bit 5-6-5 standard)

@param label Ascii string of the text inside the button

@param textsize\_x The font magnification in X-axis of the label text

@param textsize\_y The font magnification in Y-axis of the label text

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Classic initButton() function: pass center & size

void Adafruit\_GFX\_Button::initButton(Adafruit\_GFX \*gfx, int16\_t x, int16\_t y,

uint16\_t w, uint16\_t h, uint16\_t outline,

uint16\_t fill, uint16\_t textcolor,

char \*label, uint8\_t textsize\_x,

uint8\_t textsize\_y) {

// Tweak arguments and pass to the newer initButtonUL() function...

initButtonUL(gfx, x - (w / 2), y - (h / 2), w, h, outline, fill, textcolor,

label, textsize\_x, textsize\_y);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*!

@brief Initialize button with our desired color/size/settings, with

upper-left coordinates

@param gfx Pointer to our display so we can draw to it!

@param x1 The X coordinate of the Upper-Left corner of the button

@param y1 The Y coordinate of the Upper-Left corner of the button

@param w Width of the buttton

@param h Height of the buttton

@param outline Color of the outline (16-bit 5-6-5 standard)

@param fill Color of the button fill (16-bit 5-6-5 standard)

@param textcolor Color of the button label (16-bit 5-6-5 standard)

@param label Ascii string of the text inside the button

@param textsize The font magnification of the label text

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Adafruit\_GFX\_Button::initButtonUL(Adafruit\_GFX \*gfx, int16\_t x1,

int16\_t y1, uint16\_t w, uint16\_t h,

uint16\_t outline, uint16\_t fill,

uint16\_t textcolor, char \*label,

uint8\_t textsize) {

initButtonUL(gfx, x1, y1, w, h, outline, fill, textcolor, label, textsize,

textsize);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*!

@brief Initialize button with our desired color/size/settings, with

upper-left coordinates

@param gfx Pointer to our display so we can draw to it!

@param x1 The X coordinate of the Upper-Left corner of the button

@param y1 The Y coordinate of the Upper-Left corner of the button

@param w Width of the buttton

@param h Height of the buttton

@param outline Color of the outline (16-bit 5-6-5 standard)

@param fill Color of the button fill (16-bit 5-6-5 standard)

@param textcolor Color of the button label (16-bit 5-6-5 standard)

@param label Ascii string of the text inside the button

@param textsize\_x The font magnification in X-axis of the label text

@param textsize\_y The font magnification in Y-axis of the label text

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Adafruit\_GFX\_Button::initButtonUL(Adafruit\_GFX \*gfx, int16\_t x1,

int16\_t y1, uint16\_t w, uint16\_t h,

uint16\_t outline, uint16\_t fill,

uint16\_t textcolor, char \*label,

uint8\_t textsize\_x, uint8\_t textsize\_y) {

\_x1 = x1;

\_y1 = y1;

\_w = w;

\_h = h;

\_outlinecolor = outline;

\_fillcolor = fill;

\_textcolor = textcolor;

\_textsize\_x = textsize\_x;

\_textsize\_y = textsize\_y;

\_gfx = gfx;

strncpy(\_label, label, 9);

\_label[9] = 0; // strncpy does not place a null at the end.

// When 'label' is >9 characters, \_label is not terminated.

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*!

@brief Draw the button on the screen

@param inverted Whether to draw with fill/text swapped to indicate

'pressed'

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Adafruit\_GFX\_Button::drawButton(bool inverted) {

uint16\_t fill, outline, text;

if (!inverted) {

fill = \_fillcolor;

outline = \_outlinecolor;

text = \_textcolor;

} else {

fill = \_textcolor;

outline = \_outlinecolor;

text = \_fillcolor;

}

uint8\_t r = min(\_w, \_h) / 4; // Corner radius

\_gfx->fillRoundRect(\_x1, \_y1, \_w, \_h, r, fill);

\_gfx->drawRoundRect(\_x1, \_y1, \_w, \_h, r, outline);

\_gfx->setCursor(\_x1 + (\_w / 2) - (strlen(\_label) \* 3 \* \_textsize\_x),

\_y1 + (\_h / 2) - (4 \* \_textsize\_y));

\_gfx->setTextColor(text);

\_gfx->setTextSize(\_textsize\_x, \_textsize\_y);

\_gfx->print(\_label);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*!

@brief Helper to let us know if a coordinate is within the bounds of the

button

@param x The X coordinate to check

@param y The Y coordinate to check

@returns True if within button graphics outline

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

bool Adafruit\_GFX\_Button::contains(int16\_t x, int16\_t y) {

return ((x >= \_x1) && (x < (int16\_t)(\_x1 + \_w)) && (y >= \_y1) &&

(y < (int16\_t)(\_y1 + \_h)));

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*!

@brief Query whether the button was pressed since we last checked state

@returns True if was not-pressed before, now is.

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

bool Adafruit\_GFX\_Button::justPressed() { return (currstate && !laststate); }

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*!

@brief Query whether the button was released since we last checked state

@returns True if was pressed before, now is not.

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

bool Adafruit\_GFX\_Button::justReleased() { return (!currstate && laststate); }

// -------------------------------------------------------------------------

// GFXcanvas1, GFXcanvas8 and GFXcanvas16 (currently a WIP, don't get too

// comfy with the implementation) provide 1-, 8- and 16-bit offscreen

// canvases, the address of which can be passed to drawBitmap() or

// pushColors() (the latter appears only in a couple of GFX-subclassed TFT

// libraries at this time). This is here mostly to help with the recently-

// added proportionally-spaced fonts; adds a way to refresh a section of the

// screen without a massive flickering clear-and-redraw...but maybe you'll

// find other uses too. VERY RAM-intensive, since the buffer is in MCU

// memory and not the display driver...GXFcanvas1 might be minimally useful

// on an Uno-class board, but this and the others are much more likely to

// require at least a Mega or various recent ARM-type boards (recommended,

// as the text+bitmap draw can be pokey). GFXcanvas1 requires 1 bit per

// pixel (rounded up to nearest byte per scanline), GFXcanvas8 is 1 byte

// per pixel (no scanline pad), and GFXcanvas16 uses 2 bytes per pixel (no

// scanline pad).

// NOT EXTENSIVELY TESTED YET. MAY CONTAIN WORST BUGS KNOWN TO HUMANKIND.

#ifdef \_\_AVR\_\_

// Bitmask tables of 0x80>>X and ~(0x80>>X), because X>>Y is slow on AVR

const uint8\_t PROGMEM GFXcanvas1::GFXsetBit[] = {0x80, 0x40, 0x20, 0x10,

0x08, 0x04, 0x02, 0x01};

const uint8\_t PROGMEM GFXcanvas1::GFXclrBit[] = {0x7F, 0xBF, 0xDF, 0xEF,

0xF7, 0xFB, 0xFD, 0xFE};

#endif

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*!

@brief Instatiate a GFX 1-bit canvas context for graphics

@param w Display width, in pixels

@param h Display height, in pixels

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

GFXcanvas1::GFXcanvas1(uint16\_t w, uint16\_t h) : Adafruit\_GFX(w, h) {

uint32\_t bytes = ((w + 7) / 8) \* h;

if ((buffer = (uint8\_t \*)malloc(bytes))) {

memset(buffer, 0, bytes);

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*!

@brief Delete the canvas, free memory

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

GFXcanvas1::~GFXcanvas1(void) {

if (buffer)

free(buffer);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*!

@brief Draw a pixel to the canvas framebuffer

@param x x coordinate

@param y y coordinate

@param color Binary (on or off) color to fill with

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void GFXcanvas1::drawPixel(int16\_t x, int16\_t y, uint16\_t color) {

if (buffer) {

if ((x < 0) || (y < 0) || (x >= \_width) || (y >= \_height))

return;

int16\_t t;

switch (rotation) {

case 1:

t = x;

x = WIDTH - 1 - y;

y = t;

break;

case 2:

x = WIDTH - 1 - x;

y = HEIGHT - 1 - y;

break;

case 3:

t = x;

x = y;

y = HEIGHT - 1 - t;

break;

}

uint8\_t \*ptr = &buffer[(x / 8) + y \* ((WIDTH + 7) / 8)];

#ifdef \_\_AVR\_\_

if (color)

\*ptr |= pgm\_read\_byte(&GFXsetBit[x & 7]);

else

\*ptr &= pgm\_read\_byte(&GFXclrBit[x & 7]);

#else

if (color)

\*ptr |= 0x80 >> (x & 7);

else

\*ptr &= ~(0x80 >> (x & 7));

#endif

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*!

@brief Get the pixel color value at a given coordinate

@param x x coordinate

@param y y coordinate

@returns The desired pixel's binary color value, either 0x1 (on) or 0x0

(off)

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

bool GFXcanvas1::getPixel(int16\_t x, int16\_t y) const {

int16\_t t;

switch (rotation) {

case 1:

t = x;

x = WIDTH - 1 - y;

y = t;

break;

case 2:

x = WIDTH - 1 - x;

y = HEIGHT - 1 - y;

break;

case 3:

t = x;

x = y;

y = HEIGHT - 1 - t;

break;

}

return getRawPixel(x, y);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*!

@brief Get the pixel color value at a given, unrotated coordinate.

This method is intended for hardware drivers to get pixel value

in physical coordinates.

@param x x coordinate

@param y y coordinate

@returns The desired pixel's binary color value, either 0x1 (on) or 0x0

(off)

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

bool GFXcanvas1::getRawPixel(int16\_t x, int16\_t y) const {

if ((x < 0) || (y < 0) || (x >= WIDTH) || (y >= HEIGHT))

return 0;

if (buffer) {

uint8\_t \*ptr = &buffer[(x / 8) + y \* ((WIDTH + 7) / 8)];

#ifdef \_\_AVR\_\_

return ((\*ptr) & pgm\_read\_byte(&GFXsetBit[x & 7])) != 0;

#else

return ((\*ptr) & (0x80 >> (x & 7))) != 0;

#endif

}

return 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*!

@brief Fill the framebuffer completely with one color

@param color Binary (on or off) color to fill with

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void GFXcanvas1::fillScreen(uint16\_t color) {

if (buffer) {

uint32\_t bytes = ((WIDTH + 7) / 8) \* HEIGHT;

memset(buffer, color ? 0xFF : 0x00, bytes);

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*!

@brief Speed optimized vertical line drawing

@param x Line horizontal start point

@param y Line vertical start point

@param h Length of vertical line to be drawn, including first point

@param color Color to fill with

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void GFXcanvas1::drawFastVLine(int16\_t x, int16\_t y, int16\_t h,

uint16\_t color) {

if (h < 0) { // Convert negative heights to positive equivalent

h \*= -1;

y -= h - 1;

if (y < 0) {

h += y;

y = 0;

}

}

// Edge rejection (no-draw if totally off canvas)

if ((x < 0) || (x >= width()) || (y >= height()) || ((y + h - 1) < 0)) {

return;

}

if (y < 0) { // Clip top

h += y;

y = 0;

}

if (y + h > height()) { // Clip bottom

h = height() - y;

}

if (getRotation() == 0) {

drawFastRawVLine(x, y, h, color);

} else if (getRotation() == 1) {

int16\_t t = x;

x = WIDTH - 1 - y;

y = t;

x -= h - 1;

drawFastRawHLine(x, y, h, color);

} else if (getRotation() == 2) {

x = WIDTH - 1 - x;

y = HEIGHT - 1 - y;

y -= h - 1;

drawFastRawVLine(x, y, h, color);

} else if (getRotation() == 3) {

int16\_t t = x;

x = y;

y = HEIGHT - 1 - t;

drawFastRawHLine(x, y, h, color);

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*!

@brief Speed optimized horizontal line drawing

@param x Line horizontal start point

@param y Line vertical start point

@param w Length of horizontal line to be drawn, including first point

@param color Color to fill with

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void GFXcanvas1::drawFastHLine(int16\_t x, int16\_t y, int16\_t w,

uint16\_t color) {

if (w < 0) { // Convert negative widths to positive equivalent

w \*= -1;

x -= w - 1;

if (x < 0) {

w += x;

x = 0;

}

}

// Edge rejection (no-draw if totally off canvas)

if ((y < 0) || (y >= height()) || (x >= width()) || ((x + w - 1) < 0)) {

return;

}

if (x < 0) { // Clip left

w += x;

x = 0;

}

if (x + w >= width()) { // Clip right

w = width() - x;

}

if (getRotation() == 0) {

drawFastRawHLine(x, y, w, color);

} else if (getRotation() == 1) {

int16\_t t = x;

x = WIDTH - 1 - y;

y = t;

drawFastRawVLine(x, y, w, color);

} else if (getRotation() == 2) {

x = WIDTH - 1 - x;

y = HEIGHT - 1 - y;

x -= w - 1;

drawFastRawHLine(x, y, w, color);

} else if (getRotation() == 3) {

int16\_t t = x;

x = y;

y = HEIGHT - 1 - t;

y -= w - 1;

drawFastRawVLine(x, y, w, color);

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*!

@brief Speed optimized vertical line drawing into the raw canvas buffer

@param x Line horizontal start point

@param y Line vertical start point

@param h length of vertical line to be drawn, including first point

@param color Binary (on or off) color to fill with

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void GFXcanvas1::drawFastRawVLine(int16\_t x, int16\_t y, int16\_t h,

uint16\_t color) {

// x & y already in raw (rotation 0) coordinates, no need to transform.

int16\_t row\_bytes = ((WIDTH + 7) / 8);

uint8\_t \*ptr = &buffer[(x / 8) + y \* row\_bytes];

if (color > 0) {

#ifdef \_\_AVR\_\_

uint8\_t bit\_mask = pgm\_read\_byte(&GFXsetBit[x & 7]);

#else

uint8\_t bit\_mask = (0x80 >> (x & 7));

#endif

for (int16\_t i = 0; i < h; i++) {

\*ptr |= bit\_mask;

ptr += row\_bytes;

}

} else {

#ifdef \_\_AVR\_\_

uint8\_t bit\_mask = pgm\_read\_byte(&GFXclrBit[x & 7]);

#else

uint8\_t bit\_mask = ~(0x80 >> (x & 7));

#endif

for (int16\_t i = 0; i < h; i++) {

\*ptr &= bit\_mask;

ptr += row\_bytes;

}

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*!

@brief Speed optimized horizontal line drawing into the raw canvas buffer

@param x Line horizontal start point

@param y Line vertical start point

@param w length of horizontal line to be drawn, including first point

@param color Binary (on or off) color to fill with

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void GFXcanvas1::drawFastRawHLine(int16\_t x, int16\_t y, int16\_t w,

uint16\_t color) {

// x & y already in raw (rotation 0) coordinates, no need to transform.

int16\_t rowBytes = ((WIDTH + 7) / 8);

uint8\_t \*ptr = &buffer[(x / 8) + y \* rowBytes];

size\_t remainingWidthBits = w;

// check to see if first byte needs to be partially filled

if ((x & 7) > 0) {

// create bit mask for first byte

uint8\_t startByteBitMask = 0x00;

for (int8\_t i = (x & 7); ((i < 8) && (remainingWidthBits > 0)); i++) {

#ifdef \_\_AVR\_\_

startByteBitMask |= pgm\_read\_byte(&GFXsetBit[i]);

#else

startByteBitMask |= (0x80 >> i);

#endif

remainingWidthBits--;

}

if (color > 0) {

\*ptr |= startByteBitMask;

} else {

\*ptr &= ~startByteBitMask;

}

ptr++;

}

// do the next remainingWidthBits bits

if (remainingWidthBits > 0) {

size\_t remainingWholeBytes = remainingWidthBits / 8;

size\_t lastByteBits = remainingWidthBits % 8;

uint8\_t wholeByteColor = color > 0 ? 0xFF : 0x00;

memset(ptr, wholeByteColor, remainingWholeBytes);

if (lastByteBits > 0) {

uint8\_t lastByteBitMask = 0x00;

for (size\_t i = 0; i < lastByteBits; i++) {

#ifdef \_\_AVR\_\_

lastByteBitMask |= pgm\_read\_byte(&GFXsetBit[i]);

#else

lastByteBitMask |= (0x80 >> i);

#endif

}

ptr += remainingWholeBytes;

if (color > 0) {

\*ptr |= lastByteBitMask;

} else {

\*ptr &= ~lastByteBitMask;

}

}

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*!

@brief Instatiate a GFX 8-bit canvas context for graphics

@param w Display width, in pixels

@param h Display height, in pixels

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

GFXcanvas8::GFXcanvas8(uint16\_t w, uint16\_t h) : Adafruit\_GFX(w, h) {

uint32\_t bytes = w \* h;

if ((buffer = (uint8\_t \*)malloc(bytes))) {

memset(buffer, 0, bytes);

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*!

@brief Delete the canvas, free memory

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

GFXcanvas8::~GFXcanvas8(void) {

if (buffer)

free(buffer);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*!

@brief Draw a pixel to the canvas framebuffer

@param x x coordinate

@param y y coordinate

@param color 8-bit Color to fill with. Only lower byte of uint16\_t is used.

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void GFXcanvas8::drawPixel(int16\_t x, int16\_t y, uint16\_t color) {

if (buffer) {

if ((x < 0) || (y < 0) || (x >= \_width) || (y >= \_height))

return;

int16\_t t;

switch (rotation) {

case 1:

t = x;

x = WIDTH - 1 - y;

y = t;

break;

case 2:

x = WIDTH - 1 - x;

y = HEIGHT - 1 - y;

break;

case 3:

t = x;

x = y;

y = HEIGHT - 1 - t;

break;

}

buffer[x + y \* WIDTH] = color;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*!

@brief Get the pixel color value at a given coordinate

@param x x coordinate

@param y y coordinate

@returns The desired pixel's 8-bit color value

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

uint8\_t GFXcanvas8::getPixel(int16\_t x, int16\_t y) const {

int16\_t t;

switch (rotation) {

case 1:

t = x;

x = WIDTH - 1 - y;

y = t;

break;

case 2:

x = WIDTH - 1 - x;

y = HEIGHT - 1 - y;

break;

case 3:

t = x;

x = y;

y = HEIGHT - 1 - t;

break;

}

return getRawPixel(x, y);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*!

@brief Get the pixel color value at a given, unrotated coordinate.

This method is intended for hardware drivers to get pixel value

in physical coordinates.

@param x x coordinate

@param y y coordinate

@returns The desired pixel's 8-bit color value

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

uint8\_t GFXcanvas8::getRawPixel(int16\_t x, int16\_t y) const {

if ((x < 0) || (y < 0) || (x >= WIDTH) || (y >= HEIGHT))

return 0;

if (buffer) {

return buffer[x + y \* WIDTH];

}

return 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*!

@brief Fill the framebuffer completely with one color

@param color 8-bit Color to fill with. Only lower byte of uint16\_t is used.

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void GFXcanvas8::fillScreen(uint16\_t color) {

if (buffer) {

memset(buffer, color, WIDTH \* HEIGHT);

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*!

@brief Speed optimized vertical line drawing

@param x Line horizontal start point

@param y Line vertical start point

@param h Length of vertical line to be drawn, including first point

@param color 8-bit Color to fill with. Only lower byte of uint16\_t is

used.

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void GFXcanvas8::drawFastVLine(int16\_t x, int16\_t y, int16\_t h,

uint16\_t color) {

if (h < 0) { // Convert negative heights to positive equivalent

h \*= -1;

y -= h - 1;

if (y < 0) {

h += y;

y = 0;

}

}

// Edge rejection (no-draw if totally off canvas)

if ((x < 0) || (x >= width()) || (y >= height()) || ((y + h - 1) < 0)) {

return;

}

if (y < 0) { // Clip top

h += y;

y = 0;

}

if (y + h > height()) { // Clip bottom

h = height() - y;

}

if (getRotation() == 0) {

drawFastRawVLine(x, y, h, color);

} else if (getRotation() == 1) {

int16\_t t = x;

x = WIDTH - 1 - y;

y = t;

x -= h - 1;

drawFastRawHLine(x, y, h, color);

} else if (getRotation() == 2) {

x = WIDTH - 1 - x;

y = HEIGHT - 1 - y;

y -= h - 1;

drawFastRawVLine(x, y, h, color);

} else if (getRotation() == 3) {

int16\_t t = x;

x = y;

y = HEIGHT - 1 - t;

drawFastRawHLine(x, y, h, color);

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*!

@brief Speed optimized horizontal line drawing

@param x Line horizontal start point

@param y Line vertical start point

@param w Length of horizontal line to be drawn, including 1st point

@param color 8-bit Color to fill with. Only lower byte of uint16\_t is

used.

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void GFXcanvas8::drawFastHLine(int16\_t x, int16\_t y, int16\_t w,

uint16\_t color) {

if (w < 0) { // Convert negative widths to positive equivalent

w \*= -1;

x -= w - 1;

if (x < 0) {

w += x;

x = 0;

}

}

// Edge rejection (no-draw if totally off canvas)

if ((y < 0) || (y >= height()) || (x >= width()) || ((x + w - 1) < 0)) {

return;

}

if (x < 0) { // Clip left

w += x;

x = 0;

}

if (x + w >= width()) { // Clip right

w = width() - x;

}

if (getRotation() == 0) {

drawFastRawHLine(x, y, w, color);

} else if (getRotation() == 1) {

int16\_t t = x;

x = WIDTH - 1 - y;

y = t;

drawFastRawVLine(x, y, w, color);

} else if (getRotation() == 2) {

x = WIDTH - 1 - x;

y = HEIGHT - 1 - y;

x -= w - 1;

drawFastRawHLine(x, y, w, color);

} else if (getRotation() == 3) {

int16\_t t = x;

x = y;

y = HEIGHT - 1 - t;

y -= w - 1;

drawFastRawVLine(x, y, w, color);

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*!

@brief Speed optimized vertical line drawing into the raw canvas buffer

@param x Line horizontal start point

@param y Line vertical start point

@param h length of vertical line to be drawn, including first point

@param color 8-bit Color to fill with. Only lower byte of uint16\_t is

used.

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void GFXcanvas8::drawFastRawVLine(int16\_t x, int16\_t y, int16\_t h,

uint16\_t color) {

// x & y already in raw (rotation 0) coordinates, no need to transform.

uint8\_t \*buffer\_ptr = buffer + y \* WIDTH + x;

for (int16\_t i = 0; i < h; i++) {

(\*buffer\_ptr) = color;

buffer\_ptr += WIDTH;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*!

@brief Speed optimized horizontal line drawing into the raw canvas buffer

@param x Line horizontal start point

@param y Line vertical start point

@param w length of horizontal line to be drawn, including first point

@param color 8-bit Color to fill with. Only lower byte of uint16\_t is

used.

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void GFXcanvas8::drawFastRawHLine(int16\_t x, int16\_t y, int16\_t w,

uint16\_t color) {

// x & y already in raw (rotation 0) coordinates, no need to transform.

memset(buffer + y \* WIDTH + x, color, w);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*!

@brief Instatiate a GFX 16-bit canvas context for graphics

@param w Display width, in pixels

@param h Display height, in pixels

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

GFXcanvas16::GFXcanvas16(uint16\_t w, uint16\_t h) : Adafruit\_GFX(w, h) {

uint32\_t bytes = w \* h \* 2;

if ((buffer = (uint16\_t \*)malloc(bytes))) {

memset(buffer, 0, bytes);

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*!

@brief Delete the canvas, free memory

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

GFXcanvas16::~GFXcanvas16(void) {

if (buffer)

free(buffer);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*!

@brief Draw a pixel to the canvas framebuffer

@param x x coordinate

@param y y coordinate

@param color 16-bit 5-6-5 Color to fill with

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void GFXcanvas16::drawPixel(int16\_t x, int16\_t y, uint16\_t color) {

if (buffer) {

if ((x < 0) || (y < 0) || (x >= \_width) || (y >= \_height))

return;

int16\_t t;

switch (rotation) {

case 1:

t = x;

x = WIDTH - 1 - y;

y = t;

break;

case 2:

x = WIDTH - 1 - x;

y = HEIGHT - 1 - y;

break;

case 3:

t = x;

x = y;

y = HEIGHT - 1 - t;

break;

}

buffer[x + y \* WIDTH] = color;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*!

@brief Get the pixel color value at a given coordinate

@param x x coordinate

@param y y coordinate

@returns The desired pixel's 16-bit 5-6-5 color value

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

uint16\_t GFXcanvas16::getPixel(int16\_t x, int16\_t y) const {

int16\_t t;

switch (rotation) {

case 1:

t = x;

x = WIDTH - 1 - y;

y = t;

break;

case 2:

x = WIDTH - 1 - x;

y = HEIGHT - 1 - y;

break;

case 3:

t = x;

x = y;

y = HEIGHT - 1 - t;

break;

}

return getRawPixel(x, y);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*!

@brief Get the pixel color value at a given, unrotated coordinate.

This method is intended for hardware drivers to get pixel value

in physical coordinates.

@param x x coordinate

@param y y coordinate

@returns The desired pixel's 16-bit 5-6-5 color value

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

uint16\_t GFXcanvas16::getRawPixel(int16\_t x, int16\_t y) const {

if ((x < 0) || (y < 0) || (x >= WIDTH) || (y >= HEIGHT))

return 0;

if (buffer) {

return buffer[x + y \* WIDTH];

}

return 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*!

@brief Fill the framebuffer completely with one color

@param color 16-bit 5-6-5 Color to fill with

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void GFXcanvas16::fillScreen(uint16\_t color) {

if (buffer) {

uint8\_t hi = color >> 8, lo = color & 0xFF;

if (hi == lo) {

memset(buffer, lo, WIDTH \* HEIGHT \* 2);

} else {

uint32\_t i, pixels = WIDTH \* HEIGHT;

for (i = 0; i < pixels; i++)

buffer[i] = color;

}

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*!

@brief Reverses the "endian-ness" of each 16-bit pixel within the

canvas; little-endian to big-endian, or big-endian to little.

Most microcontrollers (such as SAMD) are little-endian, while

most displays tend toward big-endianness. All the drawing

functions (including RGB bitmap drawing) take care of this

automatically, but some specialized code (usually involving

DMA) can benefit from having pixel data already in the

display-native order. Note that this does NOT convert to a

SPECIFIC endian-ness, it just flips the bytes within each word.

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void GFXcanvas16::byteSwap(void) {

if (buffer) {

uint32\_t i, pixels = WIDTH \* HEIGHT;

for (i = 0; i < pixels; i++)

buffer[i] = \_\_builtin\_bswap16(buffer[i]);

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*!

@brief Speed optimized vertical line drawing

@param x Line horizontal start point

@param y Line vertical start point

@param h length of vertical line to be drawn, including first point

@param color color 16-bit 5-6-5 Color to draw line with

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void GFXcanvas16::drawFastVLine(int16\_t x, int16\_t y, int16\_t h,

uint16\_t color) {

if (h < 0) { // Convert negative heights to positive equivalent

h \*= -1;

y -= h - 1;

if (y < 0) {

h += y;

y = 0;

}

}

// Edge rejection (no-draw if totally off canvas)

if ((x < 0) || (x >= width()) || (y >= height()) || ((y + h - 1) < 0)) {

return;

}

if (y < 0) { // Clip top

h += y;

y = 0;

}

if (y + h > height()) { // Clip bottom

h = height() - y;

}

if (getRotation() == 0) {

drawFastRawVLine(x, y, h, color);

} else if (getRotation() == 1) {

int16\_t t = x;

x = WIDTH - 1 - y;

y = t;

x -= h - 1;

drawFastRawHLine(x, y, h, color);

} else if (getRotation() == 2) {

x = WIDTH - 1 - x;

y = HEIGHT - 1 - y;

y -= h - 1;

drawFastRawVLine(x, y, h, color);

} else if (getRotation() == 3) {

int16\_t t = x;

x = y;

y = HEIGHT - 1 - t;

drawFastRawHLine(x, y, h, color);

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*!

@brief Speed optimized horizontal line drawing

@param x Line horizontal start point

@param y Line vertical start point

@param w Length of horizontal line to be drawn, including 1st point

@param color Color 16-bit 5-6-5 Color to draw line with

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void GFXcanvas16::drawFastHLine(int16\_t x, int16\_t y, int16\_t w,

uint16\_t color) {

if (w < 0) { // Convert negative widths to positive equivalent

w \*= -1;

x -= w - 1;

if (x < 0) {

w += x;

x = 0;

}

}

// Edge rejection (no-draw if totally off canvas)

if ((y < 0) || (y >= height()) || (x >= width()) || ((x + w - 1) < 0)) {

return;

}

if (x < 0) { // Clip left

w += x;

x = 0;

}

if (x + w >= width()) { // Clip right

w = width() - x;

}

if (getRotation() == 0) {

drawFastRawHLine(x, y, w, color);

} else if (getRotation() == 1) {

int16\_t t = x;

x = WIDTH - 1 - y;

y = t;

drawFastRawVLine(x, y, w, color);

} else if (getRotation() == 2) {

x = WIDTH - 1 - x;

y = HEIGHT - 1 - y;

x -= w - 1;

drawFastRawHLine(x, y, w, color);

} else if (getRotation() == 3) {

int16\_t t = x;

x = y;

y = HEIGHT - 1 - t;

y -= w - 1;

drawFastRawVLine(x, y, w, color);

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*!

@brief Speed optimized vertical line drawing into the raw canvas buffer

@param x Line horizontal start point

@param y Line vertical start point

@param h length of vertical line to be drawn, including first point

@param color color 16-bit 5-6-5 Color to draw line with

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void GFXcanvas16::drawFastRawVLine(int16\_t x, int16\_t y, int16\_t h,

uint16\_t color) {

// x & y already in raw (rotation 0) coordinates, no need to transform.

uint16\_t \*buffer\_ptr = buffer + y \* WIDTH + x;

for (int16\_t i = 0; i < h; i++) {

(\*buffer\_ptr) = color;

buffer\_ptr += WIDTH;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*!

@brief Speed optimized horizontal line drawing into the raw canvas buffer

@param x Line horizontal start point

@param y Line vertical start point

@param w length of horizontal line to be drawn, including first point

@param color color 16-bit 5-6-5 Color to draw line with

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void GFXcanvas16::drawFastRawHLine(int16\_t x, int16\_t y, int16\_t w,

uint16\_t color) {

// x & y already in raw (rotation 0) coordinates, no need to transform.

uint32\_t buffer\_index = y \* WIDTH + x;

for (uint32\_t i = buffer\_index; i < buffer\_index + w; i++) {

buffer[i] = color;

}

}