Adafruitfancyled Library Documentation

Release 1.0

PaintYourDragon

Jan 17, 2020

Contents

FancyLED is a CircuitPython library to assist in creating buttery smooth LED animation. It's loosely inspired by the FastLED library for Arduino, and in fact we have a "helper" library using similar function names to assist with porting of existing Arduino FastLED projects to CircuitPython.

CHAPTER 1

Dependencies

This driver depends on:

- Adafruit CircuitPython

Please ensure all dependencies are available on the CircuitPython filesystem. This is easily achieved by downloading the Adafruit library and driver bundle.

CHAPTER 2

Installing from PyPI

On supported GNU/Linux systems like the Raspberry Pi, you can install the driver locally from PyPI. To install for current user:

```
pip3 install adafruit-circuitpython-fancyled
```

To install system-wide (this may be required in some cases):

```
sudo pip3 install adafruit-circuitpython-fancyled
```

To install in a virtual environment in your current project:

```
mkdir project-name && cd project-name
python3 -m venv .env
source .env/bin/activate
pip3 install adafruit-circuitpython-fancyled
```

CHAPTER 3

Usage Example

See the examples in the examples/ folder.

CHAPTER 4

Contributing

Contributions are welcome! Please read our Code of Conduct before contributing to help this project stay welcoming.

CHAPTER 5

Documentation

For information on building library documentation, please check out this guide.

CHAPTER 6

Table of Contents

6.1 Simple test

Ensure your device works with this simple test.

Listing 1: examples/fancyled_neopixel_rotate_simpletest.py

```
""" Simple FancyLED example for NeoPixel strip
"""

import board
import neopixel
import adafruit_fancyled.adafruit_fancyled as fancy

num_leds = 20

# Declare a 6-element RGB rainbow palette
palette = [fancy.CRGB(1.0, 0.0, 0.0), # Red
           fancy.CRGB(0.5, 0.5, 0.0), # Yellow
           fancy.CRGB(0.0, 1.0, 0.0), # Green
           fancy.CRGB(0.0, 0.5, 0.5), # Cyan
           fancy.CRGB(0.0, 0.0, 1.0), # Blue
           fancy.CRGB(0.5, 0.0, 0.5)] # Magenta

# Declare a NeoPixel object on pin D6 with num_leds pixels, no auto-write.
# Set brightness to max because we'll be using FancyLED's brightness control.
pixels = neopixel.NeoPixel(board.D6, num_leds, brightness=1.0,
                           auto_write=False)

offset = 0  # Positional offset into color palette to get it to 'spin'

while True:
    for i in range(num_leds):
        # Load each pixel's color from the palette using an offset, run it
```

(continues on next page)

(continued from previous page)

```
        # through the gamma function, pack RGB value and assign to pixel.
        color = fancy.palette_lookup(palette, offset + i / num_leds)
        color = fancy.gamma_adjust(color, brightness=0.25)
        pixels[i] = color.pack()
    pixels.show()

    offset += 0.02  # Bigger number = faster spin
```

Listing 2: examples/fancyled_cpx_helper_example.py

```
""" FancyLED example for Circuit Playground Express using fastled_helpers
"""

from adafruit_circuitplayground.express import cpx
import adafruit_fancyled.fastled_helpers as helper

cpx.pixels.auto_write = False  # Refresh pixels only when we say

# A dynamic gradient palette is a compact way of representing a palette with
# non-equal spacing between elements.  This one's a blackbody palette with a
# longer red 'tail'.  The helper functions let us declare this as a list of
# bytes, so they're easier to copy over from existing FastLED projects.
heatmap_gp = bytes([
    0, 255, 255, 255, # White
    64, 255, 255, 0,  # Yellow
    128, 255, 0, 0,   # Red
    255, 0, 0, 0])    # Black

# Convert the gradient palette into a normal palette w/16 elements:
palette = helper.loadDynamicGradientPalette(heatmap_gp, 16)

offset = 0 # Positional offset into color palette to get it to 'spin'

while True:
    for i in range(10):
        # Load each pixel's color from the palette.  FastLED uses 16-step
        # in-between blending...so for a 16-color palette, there's 256
        # steps total.  With 10 pixels, multiply the pixel index by 25.5
        # (and add our offset) to get FastLED-style palette position.
        color = helper.ColorFromPalette(palette, int(offset + i * 25.5),
                                        blend=True)
        # Apply gamma using the FastLED helper syntax
        color = helper.applyGamma_video(color)
        # 'Pack' color and assign to NeoPixel #i
        cpx.pixels[i] = color.pack()
    cpx.pixels.show()

    offset += 8  # Bigger number = faster spin
```

Listing 3: examples/fancyled_cpx_rotate.py

```
""" Simple FancyLED example for Circuit Playground Express
"""

from adafruit_circuitplayground.express import cpx
import adafruit_fancyled.adafruit_fancyled as fancy
```

(continues on next page)

(continued from previous page)

```
6
7  cpx.pixels.auto_write = False  # Refresh pixels only when we say
8  cpx.pixels.brightness = 1.0    # We'll use FancyLED's brightness controls
9
10 # Declare a 4-element color palette, this one happens to be a
11 # 'blackbody' palette -- good for heat maps and firey effects.
12 palette = [fancy.CRGB(1.0, 1.0, 1.0), # White
13            fancy.CRGB(1.0, 1.0, 0.0), # Yellow
14            fancy.CRGB(1.0, 0.0, 0.0), # Red
15            fancy.CRGB(0.0, 0.0, 0.0)] # Black
16
17 offset = 0  # Positional offset into color palette to get it to 'spin'
18 levels = (0.25, 0.3, 0.15)  # Color balance / brightness for gamma function
19
20 while True:
21     for i in range(10):
22         # Load each pixel's color from the palette using an offset, run it
23         # through the gamma function, pack RGB value and assign to pixel.
24         color = fancy.palette_lookup(palette, offset + i / 10)
25         color = fancy.gamma_adjust(color, brightness=levels)
26         cpx.pixels[i] = color.pack()
27     cpx.pixels.show()
28
29     offset += 0.033  # Bigger number = faster spin
```

6.2 adafruit_fancyled.adafruit_fancyled

FancyLED is a CircuitPython library to assist in creating buttery smooth LED animation. It's loosely inspired by the FastLED library for Arduino, and in fact we have a "helper" library using similar function names to assist with porting of existing Arduino FastLED projects to CircuitPython.

- Author(s): PaintYourDragon

class adafruit_fancyled.adafruit_fancyled.**CHSV**(*h*, *s=1.0*, *v=1.0*)
Color stored in Hue, Saturation, Value color space.

Accepts hue as float (any range) or integer (0-256 -> 0.0-1.0) with no clamping performed (hue can 'wrap around'), saturation and value as float (0.0 to 1.0) or integer (0 to 255), both are clamped and stored internally in the normalized (float) format. Latter two are optional, can pass juse hue and saturation/value will default to 1.0.

Unlike *CRGB* (which can take a *CHSV* as input), there's currently no equivalent RGB-to-HSV conversion, mostly because it's a bit like trying to reverse a hash... there may be multiple HSV solutions for a given RGB input.

This might be OK as long as conversion precedence is documented, but otherwise (and maybe still) could cause confusion as certain HSV->RGB->HSV translations won't have the same input and output.

pack()
'Pack' a *CHSV* color into a 24-bit RGB integer.

Returns 24-bit integer a la `0x00RRGGBB`.

class adafruit_fancyled.adafruit_fancyled.**CRGB**(*red*, *green=0.0*, *blue=0.0*)
Color stored in Red, Green, Blue color space.

One of two ways: separate red, gren, blue values (either as integers (0 to 255 range) or floats (0.0 to 1.0 range), either type is 'clamped' to valid range and stored internally in the normalized (float) format), OR can accept a

CHSV color as input, which will be converted and stored in RGB format.

Following statements are equivalent - all return red:

```
c = CRGB(255, 0, 0)
c = CRGB(1.0, 0.0, 0.0)
c = CRGB(CHSV(0.0, 1.0, 1.0))
```

pack()
'Pack' a *CRGB* color into a 24-bit RGB integer.

Returns 24-bit integer a la `0x00RRGGBB`.

adafruit_fancyled.adafruit_fancyled.**clamp**(*val*, *lower*, *upper*)
Constrain value within a numeric range (inclusive).

adafruit_fancyled.adafruit_fancyled.**denormalize**(*val*, *inplace=False*)
Convert normalized (0.0 to 1.0) value to 8-bit (0 to 255) value

Accepts float, 0.0 to 1.0 range or a list or tuple of floats. In list case, 'inplace' can be used to control whether the original list is modified (True) or a new list is generated and returned (False).

Returns integer, 0 to 255 range, or list of integers (or None if inplace).

adafruit_fancyled.adafruit_fancyled.**expand_gradient**(*gradient*, *length*)
Convert gradient palette into standard equal-interval palette.

Parameters **gradient** (*sequence*) – List or tuple of of 2-element lists/tuples containing position (0.0 to 1.0) and color (packed int, CRGB or CHSV). It's OK if the list/tuple elements are either lists OR tuples, but don't mix and match lists and tuples – use all one or the other.

Returns CRGB list, can be used with palette_lookup() function.

adafruit_fancyled.adafruit_fancyled.**gamma_adjust**(*val*, *gamma_value=None*, *brightness=1.0*, *inplace=False*)
Provides gamma adjustment for single values, *CRGB* and *CHSV* types and lists of any of these.

Works in one of three ways:

1. Accepts a single normalized level (0.0 to 1.0) and optional gamma-adjustment factor (float usu. > 1.0, default if unspecified is GFACTOR) and brightness (float 0.0 to 1.0, default is 1.0). Returns a single normalized gamma-corrected brightness level (0.0 to 1.0).
2. Accepts a single *CRGB* or *CHSV* type, optional single gamma factor OR a (R,G,B) gamma tuple (3 values usu. > 1.0), optional single brightness factor OR a (R,G,B) brightness tuple. The input tuples are RGB even when a *CHSV* color is passed. Returns a normalized gamma-corrected *CRGB* type (NOT *CHSV*!).
3. Accept a list or tuple of normalized levels, *CRGB* or *CHSV* types (and optional gamma and brightness levels or tuples applied to all). Returns a list of gamma-corrected values or *CRGB* types (NOT *CHSV*!).

In cases 2 and 3, if the input is a list (NOT a tuple!), the 'inplace' flag determines whether a new tuple/list is calculated and returned, or the existing value is modified in-place. By default this is 'False'. If you try to inplace-modify a tuple, an exception is raised.

In cases 2 and 3, there is NO return value if 'inplace' is True – the original values are modified.

adafruit_fancyled.adafruit_fancyled.**mix**(*color1*, *color2*, *weight2=0.5*)
Blend between two colors using given ratio. Accepts two colors (each may be *CRGB*, *CHSV* or packed integer), and weighting (0.0 to 1.0) of second color.

Returns *CRGB* color in most cases, *CHSV* if both inputs are *CHSV*.

adafruit_fancyled.adafruit_fancyled.**normalize**(*val*, *inplace=False*)
Convert 8-bit (0 to 255) value to normalized (0.0 to 1.0) value.

Accepts integer, 0 to 255 range (input is clamped) or a list or tuple of integers. In list case, 'inplace' can be used to control whether the original list is modified (True) or a new list is generated and returned (False).

Returns float, 0.0 to 1.0 range, or list of floats (or None if inplace).

adafruit_fancyled.adafruit_fancyled.**palette_lookup**(*palette*, *position*)
Fetch color from color palette, with interpolation.

Parameters

- **palette** – color palette (list of CRGB, CHSV and/or packed integers)
- **position** (*float*) – palette position (0.0 to 1.0, wraps around).

Returns *CRGB* or *CHSV* instance, no gamma correction applied.

adafruit_fancyled.adafruit_fancyled.**unpack**(*val*)
'Unpack' a 24-bit color into a *CRGB* instance.

Parameters **val** (*int*) – 24-bit integer a la `0x00RRGGBB`.

Returns CRGB color.

Return type *CRGB*

6.3 adafruit_fancyled.fastled_helpers

CircuitPython "helper" library based on the Arduino FastLED library. Uses similar function names to assist with porting of existing Arduino FastLED projects to CircuitPython.

- Author(s): PaintYourDragon

adafruit_fancyled.fastled_helpers.**ColorFromPalette**(*pal*, *pos*, *brightness=255*, *blend=False*)
Approximates the FastLED ColorFromPalette() function

ACCEPTS: color palette (list of CRGB, CSHV and/or packed ints), palette index (x16) + blend factor of next index (0-15) – e.g. pass 32 to retrieve palette index 2, or 40 for an interpolated value between palette index 2 and 3, optional brightness (0-255), optional blend flag (True/False)

RETURNS: CRGB color, no gamma correction

adafruit_fancyled.fastled_helpers.**applyGamma_video**(*n*, *g_r=2.5*, *g_g=None*, *g_b=None*, *inplace=False*)
Approximates various invocations of FastLED's many-ways-overloaded applyGamma_video() function.

ACCEPTS: One of three ways:

1. A single brightness level (0-255) and optional gamma-correction factor (float usu. > 1.0, default if unspecified is 2.5).
2. A single CRGB, CHSV or packed integer type and optional gamma factor or separate R, G, B gamma values.
3. A list of CRGB, CHSV or packed integer types (and optional gamma(s)).

In the tuple/list cases, the 'inplace' flag determines whether a new tuple/list is calculated and returned, or the existing value is modified in-place. By default this is 'False'. Can also use the napplyGamma_video() function to more directly approximate FastLED syntax/behavior.

RETURNS: Corresponding to above cases:

1. Single gamma-corrected brightness level (0-255).
2. A gamma-corrected CRGB value (even if input is CHSV or packed).
3. A list of gamma-corrected CRGB values.

In the tuple/list cases, there is NO return value if 'inplace' is true – the original values are modified.

adafruit_fancyled.fastled_helpers.**hsv2rgb_spectrum**(*hue*, *sat*, *val*)

This is named the same thing as FastLED's simpler HSV to RGB function (spectrum, vs rainbow) but implementation is a bit different for the sake of getting something running (adapted from some NeoPixel code).

ACCEPTS: hue, saturation, value in range 0 to 255 RETURNS: CRGB color.

adafruit_fancyled.fastled_helpers.**loadDynamicGradientPalette**(*src*, *size*)

Kindasorta like FastLED's loadDynamicGradientPalette() function, with some gotchas.

ACCEPTS: Gradient palette data as a 'bytes' type (makes it easier to copy over gradient palettes from existing FastLED Arduino sketches)... each palette entry is four bytes: a relative position (0-255) within the overall resulting palette (whatever its size), and 3 values for R, G and B... and a length for a new palette list to be allocated.

RETURNS: list of CRGB colors.

adafruit_fancyled.fastled_helpers.**napplyGamma_video**(*n*, *g_r=2.5*, *g_g=None*, *g_b=None*)

In-place version of applyGamma_video() (to mimic FastLED function name). This is for RGB tuples and tuple lists (not the prior function's integer case)

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

a

Index