HalloWing Light Paintstick

Created by John Park

https://learn.adafruit.com/hallowing-light-paintstick

Last updated on 2024-03-08 03:08:02 PM EST

Table of Contents

Overview 3

- Parts
- Alternative microcontroller and NeoPixel strip option:
- Materials

Build the Light Paintstick 5

- Cut the Yardstick
- Sanding
- Drilling
- Paint the Yardstick
- Connect the Components
- Attach NeoPixel Strip
- Connect the Potentiometer
- Add the Battery
- Attach the HalloWing
- Alternate CPX version

Code with CircuitPython 26

- CircuitPython Setup
- HalloWing Code
- Circuit Playground Express Code
- Image Selection

Make Pixel Art 34

- Images
- Art Specs

Take Long Exposure Photos 36

- Tools
- Action
- Floating Images
- Multiple Stamps

Overview

We love light painting and over the years we have created a number of Learn Guides on the topic, ranging from simple to highly sophisticated (https://adafru.it/wOf).

With this project, painting with light in long exposure photographs has never been easier. You can explore this beautiful artform with a digital camera that has manual setting, or even a smart phone with an app, and build your own Light Paintstick to draw bitmapped images in midair!

Look below each light painting photo in this guide to see the bitmap image that was running on the Light Paintstick.

Parts

1 x Adafruit HalloWing M0 Express
Skull-shaped ATSAMD21 board w 1.44" 128x128 TFT display
https://www.adafruit.com/product/3900

---

1 x NeoPixel Strip 0.5 Meter
with 3-pin JST connector
https://www.adafruit.com/product/3919

---

1 x Lithium Ion Polymer Battery
3.7v 350mAh
https://www.adafruit.com/product/2750

---

1 x Breadboard trim potentiometer
10K Ohm
https://www.adafruit.com/product/356

---

1 x JST PH 3-Pin to Female Socket Cable
200mm
https://www.adafruit.com/product/3894

Alternative microcontroller and NeoPixel strip option:

1 x Circuit Playground Express
Round, Awesome Microcontroller Board
https://www.adafruit.com/product/3333

---

1 x NeoPixel LED Strip w/ Alligator Clips
60 LED/m - 0.5 Meter Long - Black Flex
https://www.adafruit.com/product/3811

Materials

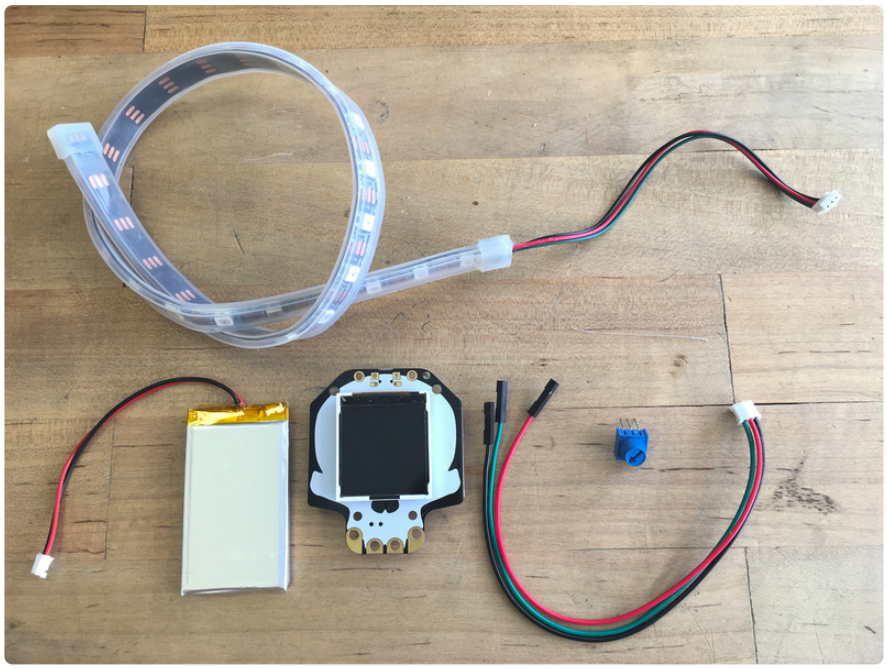
In addition to the parts listed above, you'll need the following materials:

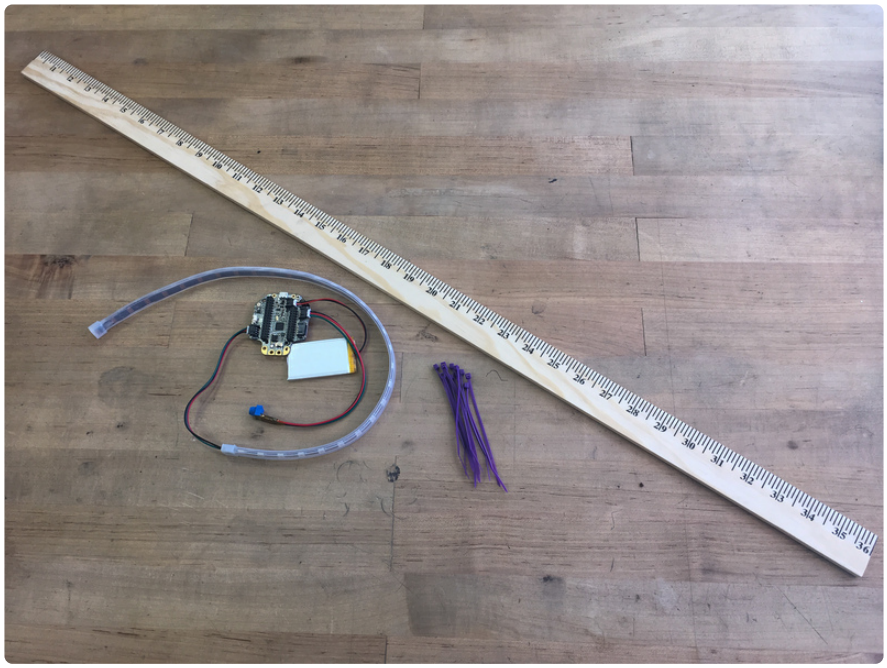
- Wooden yardstick (available at any hardware store)
- Zip ties
- Optional saw to cut the yardstick length down a bit
- Optional sandpaper for shaping the yardstick handle
- Optional matte gray spray paint for the yardstick

And, a camera with manual control of shutter speed and exposure aperture, or a long exposure/"light trails" application for your smart phone, as well as a tripod or other way to secure your camera. It helps (and is lots of fun!) to have a friend to help you out during photo shoots.

Build the Light Paintstick

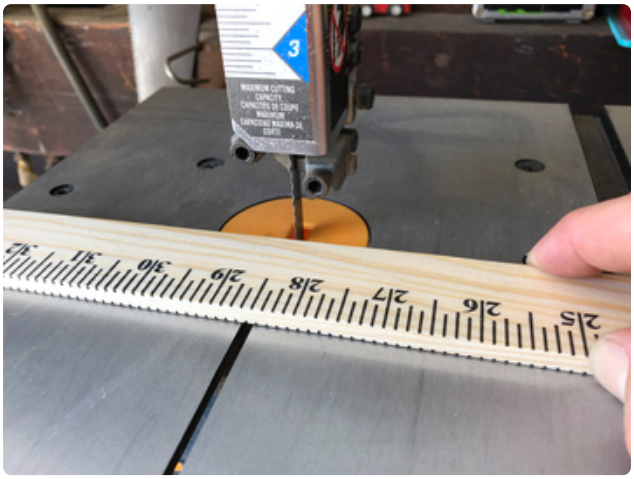
This is a very straightforward build -- all you need is a yardstick and some zip ties to attach the NeoPixel strip, board, battery, and potentiometer!



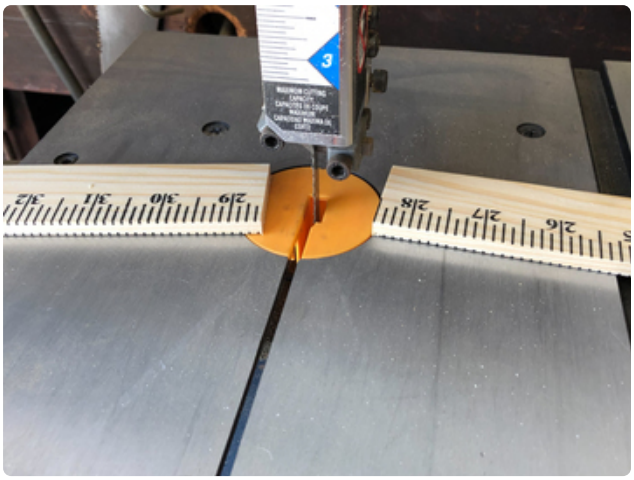


We'll customize the build a little bit to give it a nice fit and finish.

Cut the Yardstick

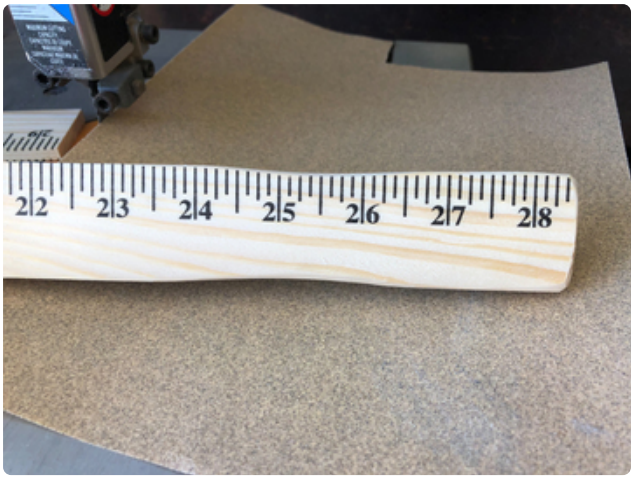


You can skip this step if you like, but it's nice to trim off some of the excess length of the yardstick to make the Light Paintstick a bit more compact and easier to swing around in midair.



No need to measure! Just use a hand saw or power saw (bandsaw, chop saw, miter saw, jig saw, or table saw) to cut the yardstick at the 28-1/2" mark.

Sanding



To make the Light Paintstick more comfortable to hold, you can use some sandpaper to shape the handle section a bit and round off the edges.

Drilling

You may want to mount the Light Paintstick onto a bicycle or scooter to create some large-scale light paintings -- you can add some holes near the base of the handle to accommodate this.

Mark and drill two 1/4" holes as shown.

It's helpful to drill over a sacrificial piece of wood to prevent splintery blow-out on the back-side of the hole as the drill bit breaks through.

20
21
22
24
27
28

Paint the Yardstick

In a well ventilated area, set down some newspaper and a couple of blocks for standoffs.

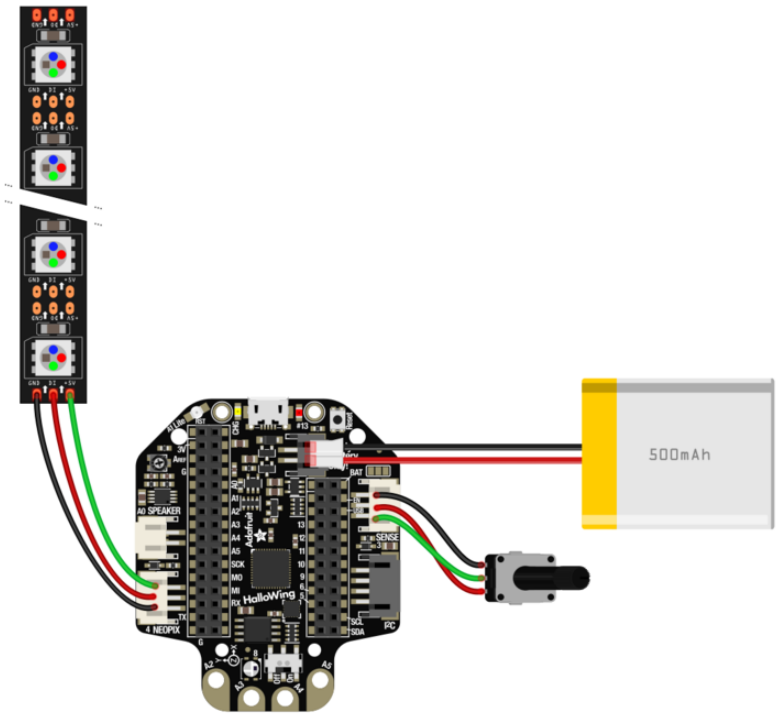
Spray the yardstick with even coats of spray paint.

Allow to dry for 1/2 hour, then flip the stick and paint the other side.

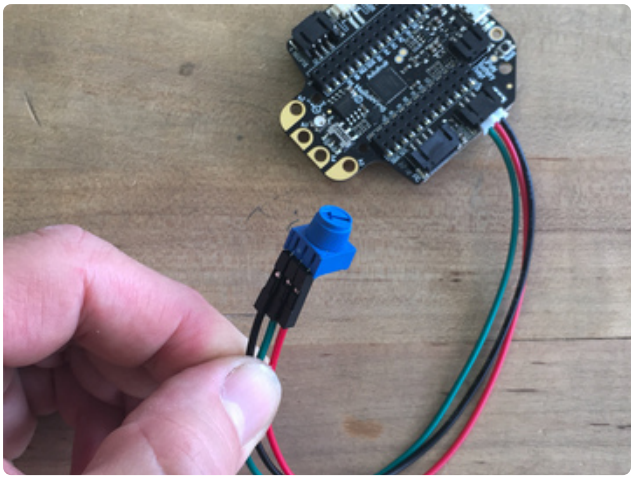
Two full coats should be sufficient.

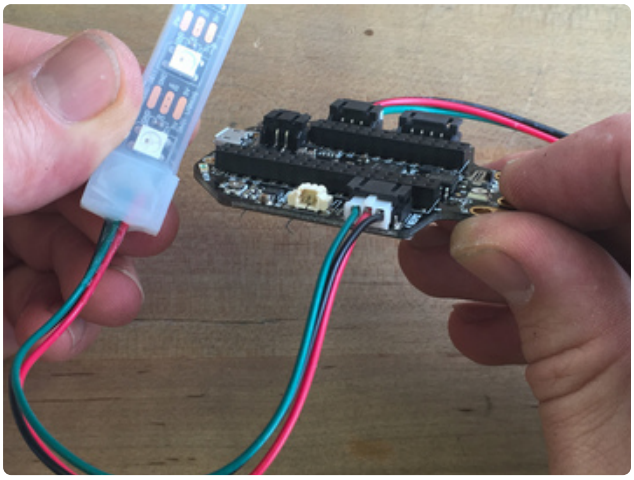
Marvel at how professional and awesome your Light Paintstick is now looking!

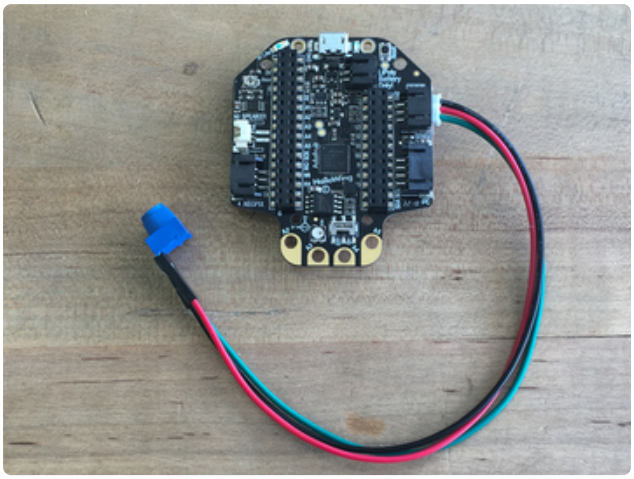
Now that he stick is prepped, it's time to add the electronics!



Connect the Components





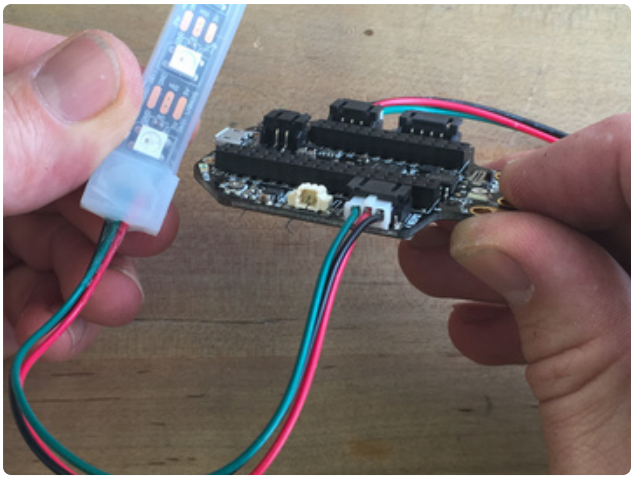


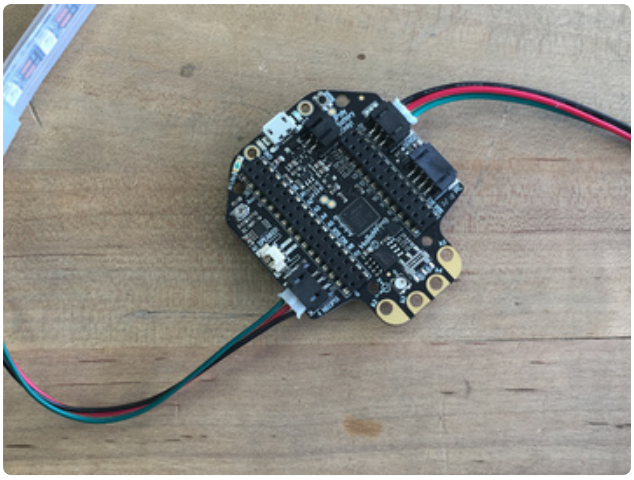
Potentiometer

The potentiometer will be used to control the playback speed of the bitmap image. Connect it to the three sockets of the sensor cable as shown. The green wire goes in the middle to send potentiometer wiper position data, while red and black go to the outer pins for voltage and ground.

Connect the other end of the cable to the JST sensor port on the HalloWing.

This port is marked "**SENSE**".

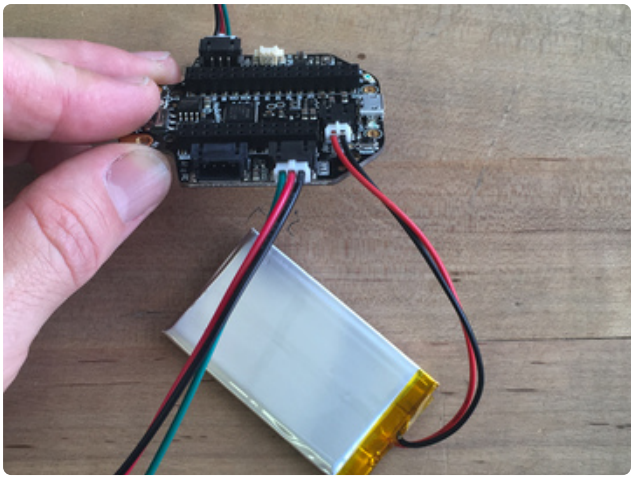


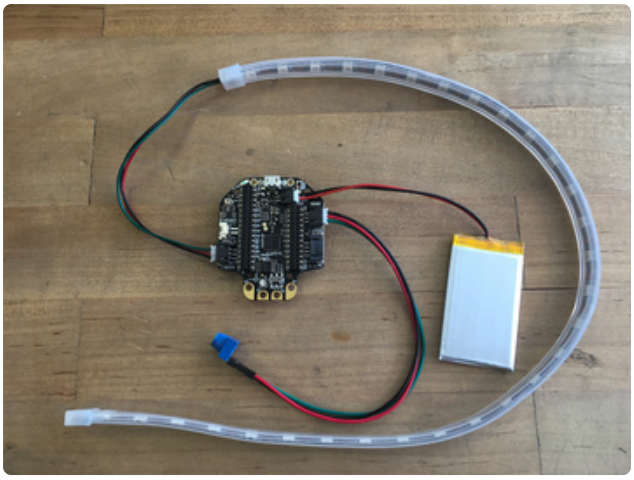


NeoPixel Strip

Plug the NeoPixel strip connector into the NeoPixel JST port on the HalloWing.

This port is marked "**NEOPIX**".





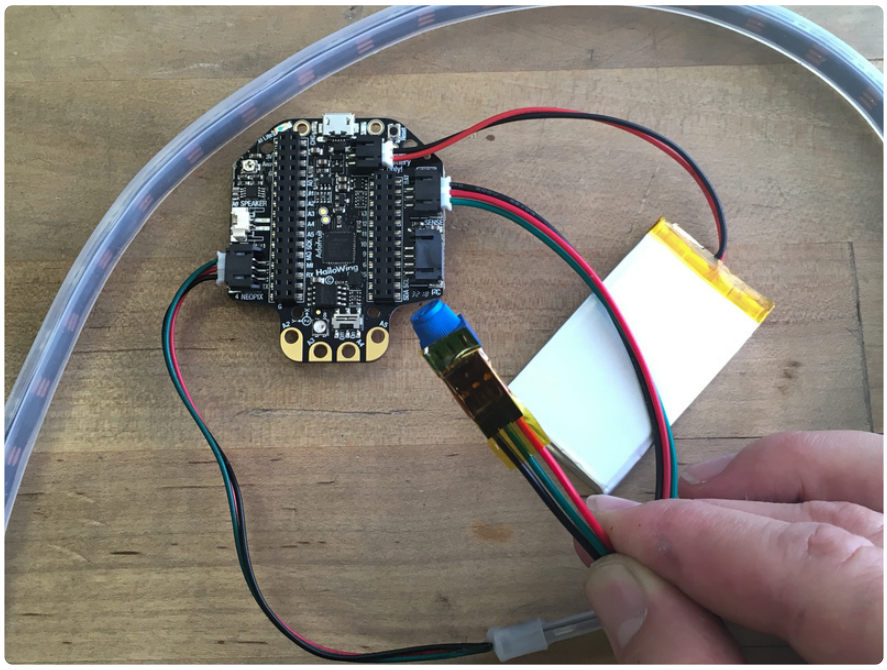
Battery

You'll power the HalloWing Light Paintstick with a 3.7V lithium polymer battery.

The HalloWing even has a charging circuit built in, so you can recharge the battery at any time by simply plugging in a micro USB cable!

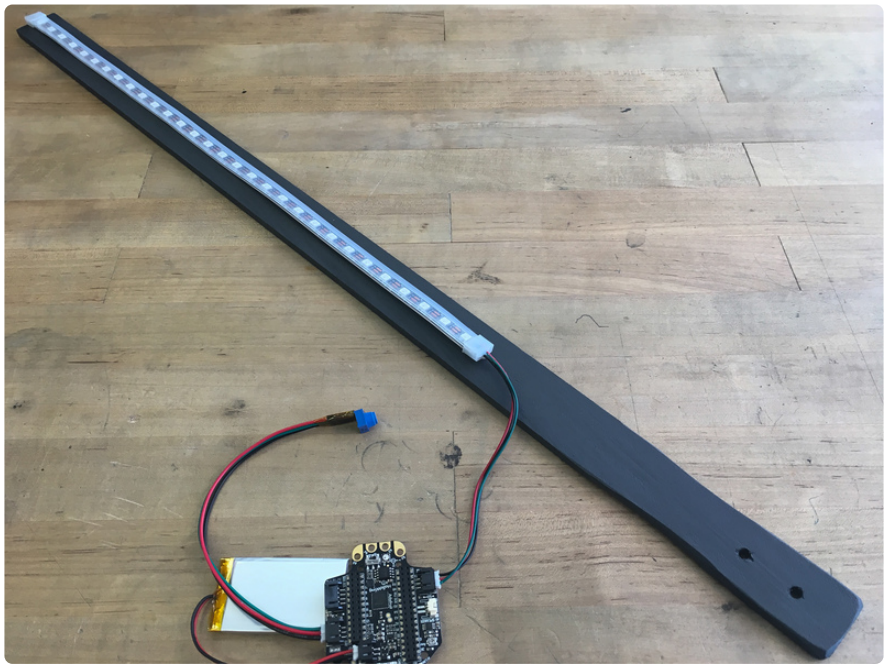
Plug the battery into the port marked "**LiPoly Battery Only!**"

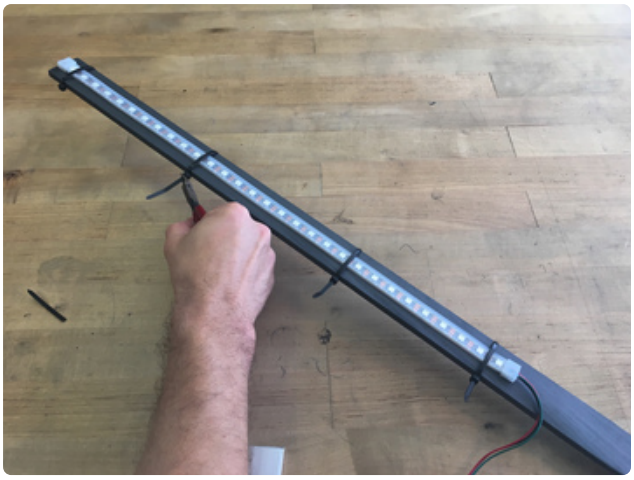
You can secure the potentiometer-to-cable connection with a bit of tape to prevent it from disconnecting.



Now we can connect the HalloWing and components to the stick.

Attach NeoPixel Strip





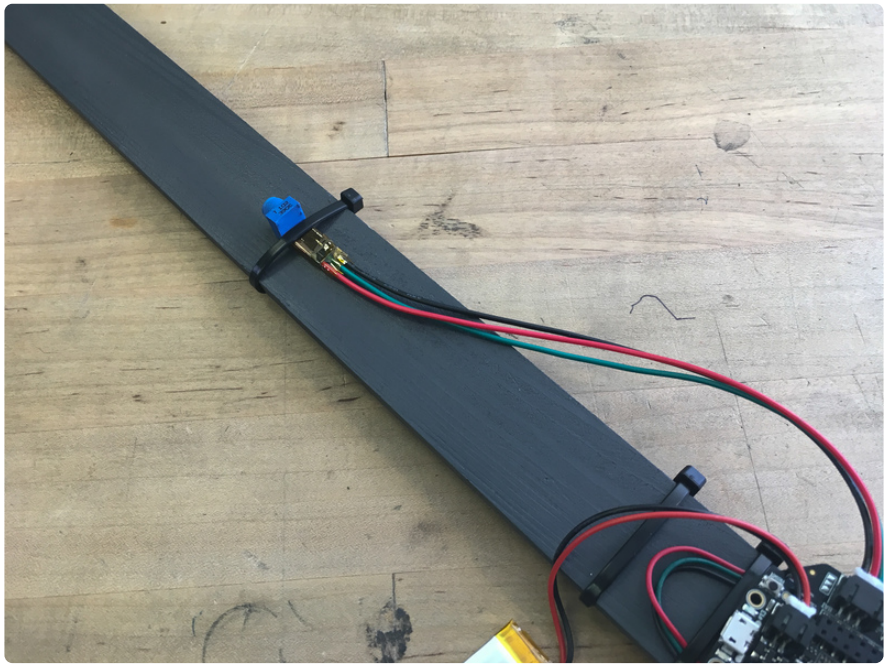
Lay the NeoPixel strip on one side of the stick, with the LEDs facing up.

Use zip ties to secure the strip to the stick.

Trim off the excess zip tie ends with diagonal cutters or scissors.

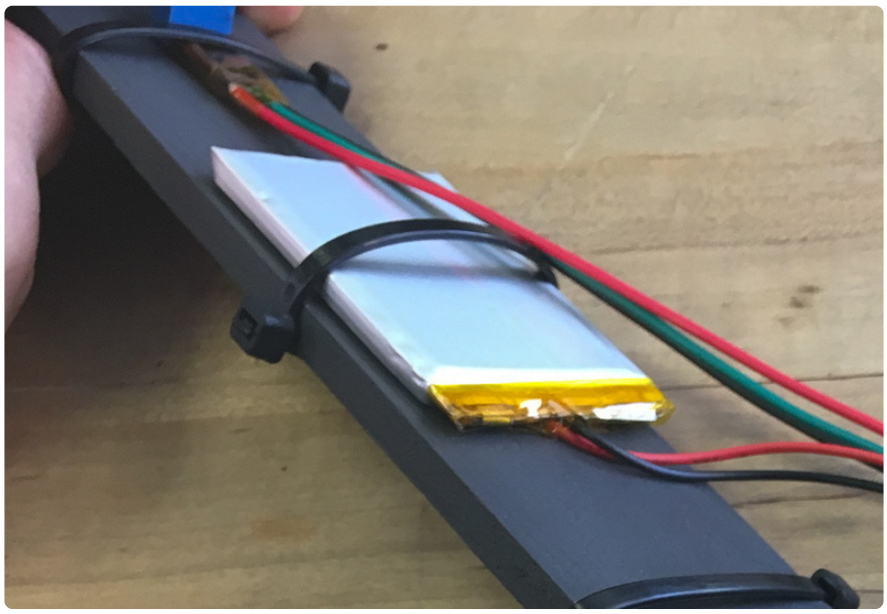
Connect the Potentiometer

Connect the potentiometer to the Light Paintstick with a zip tie as shown.

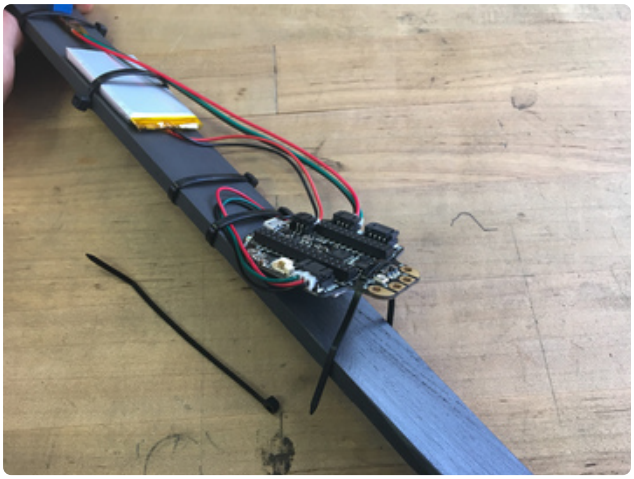


Add the Battery

Secure the battery just below the potentiometer with a zip tie.



Attach the HalloWing

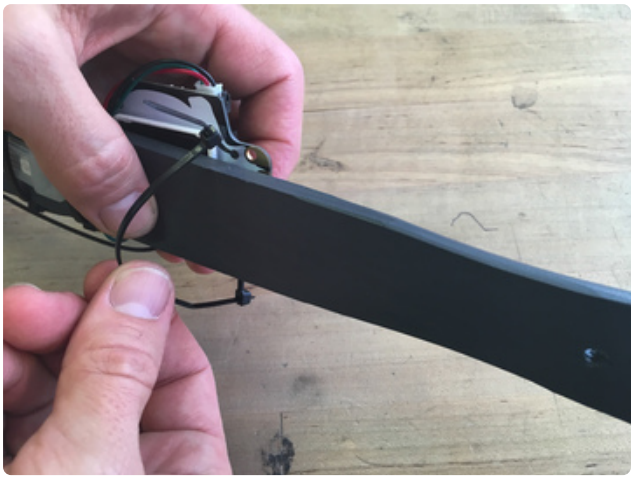


Use four thin zip ties to secure the board to the stick.

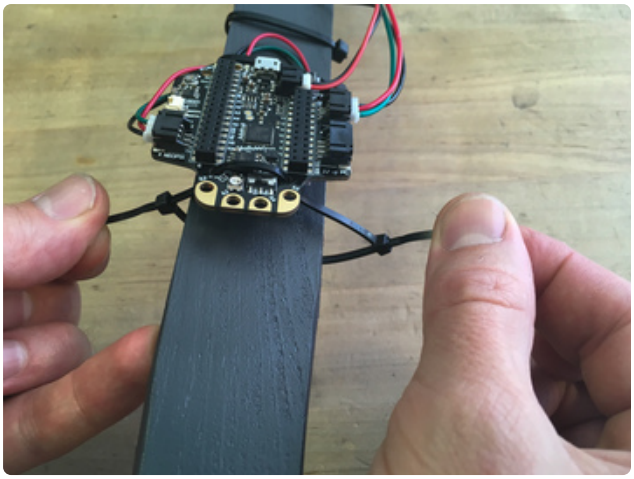
Feed one zip tie through the two mounting holes as shown.

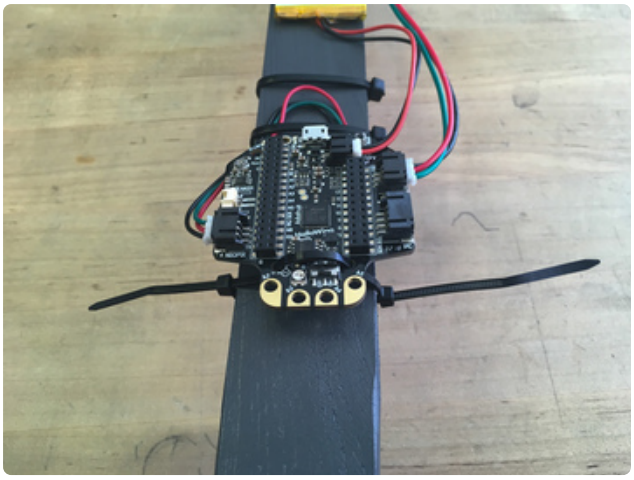
Use a second zip tie connected on both ends to the first zip tie on the underside of the board to join them.

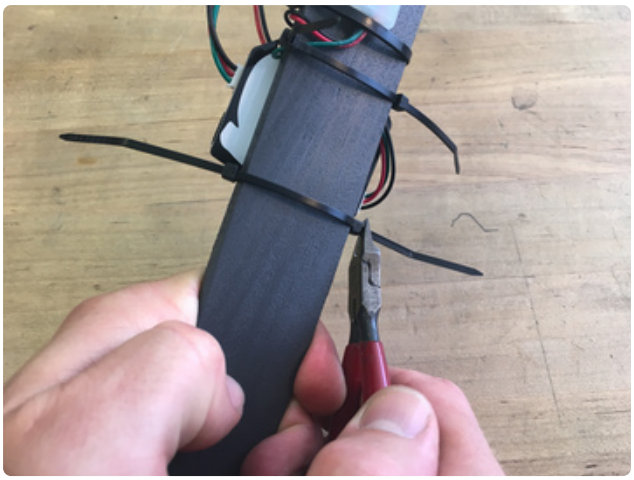
Pull them snug but not too tight (don't want to put too much pressure on the TFT display!).



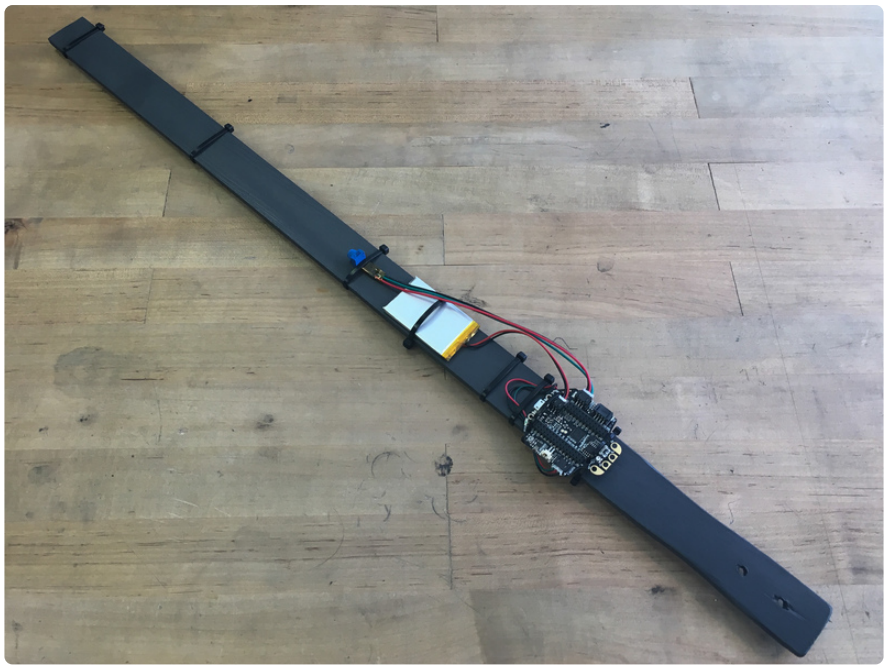
Trim the excess ends.





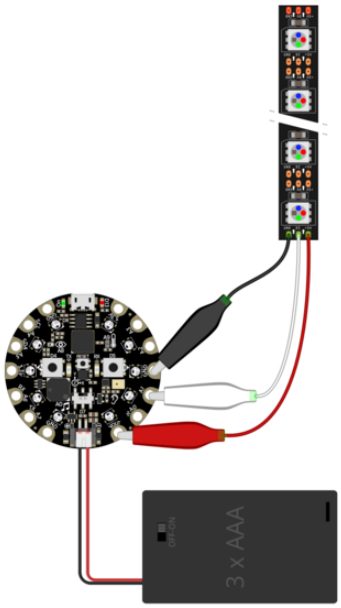


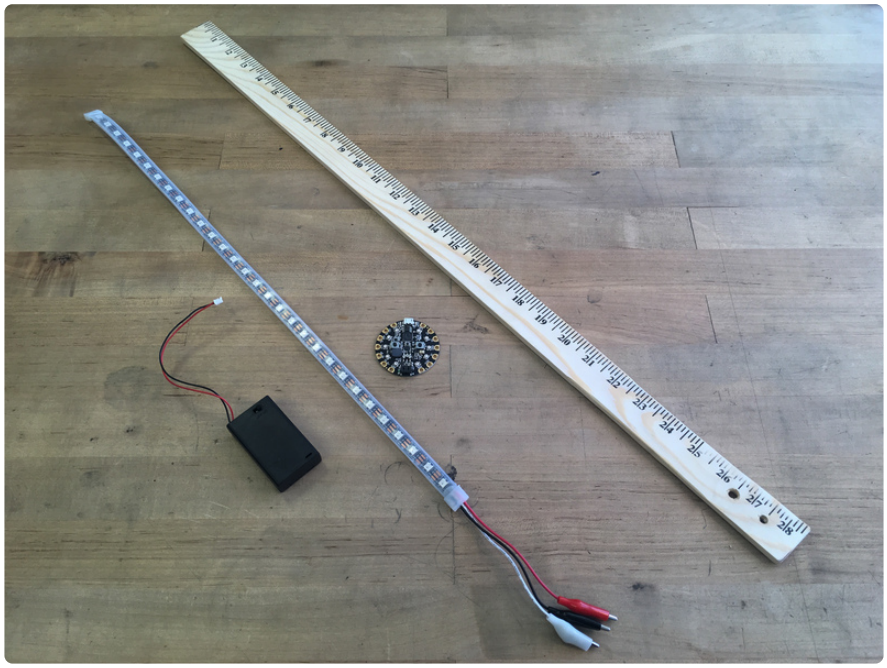
Here's what the finished HalloWing Light Paintstick looks like!

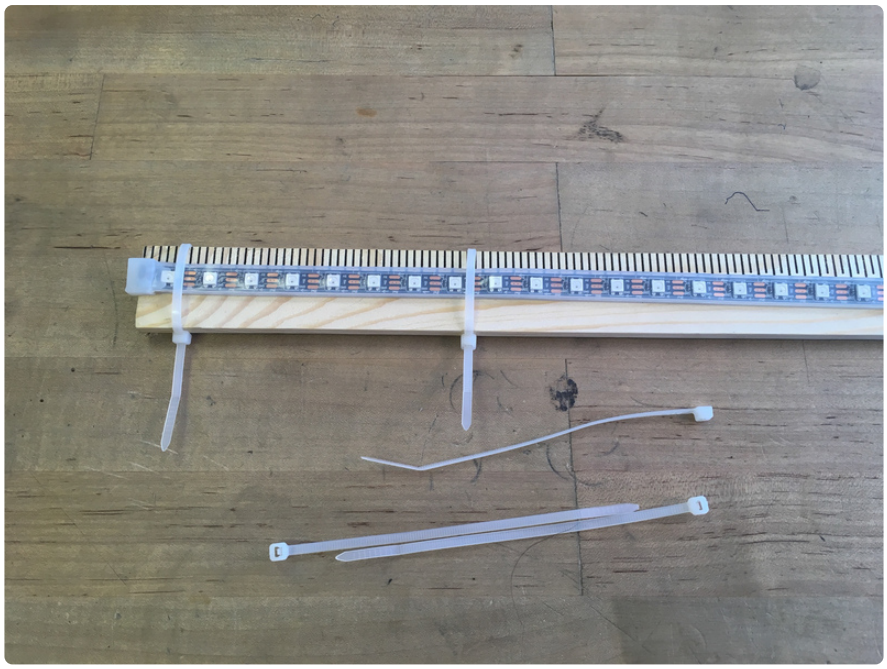


Alternate CPX version

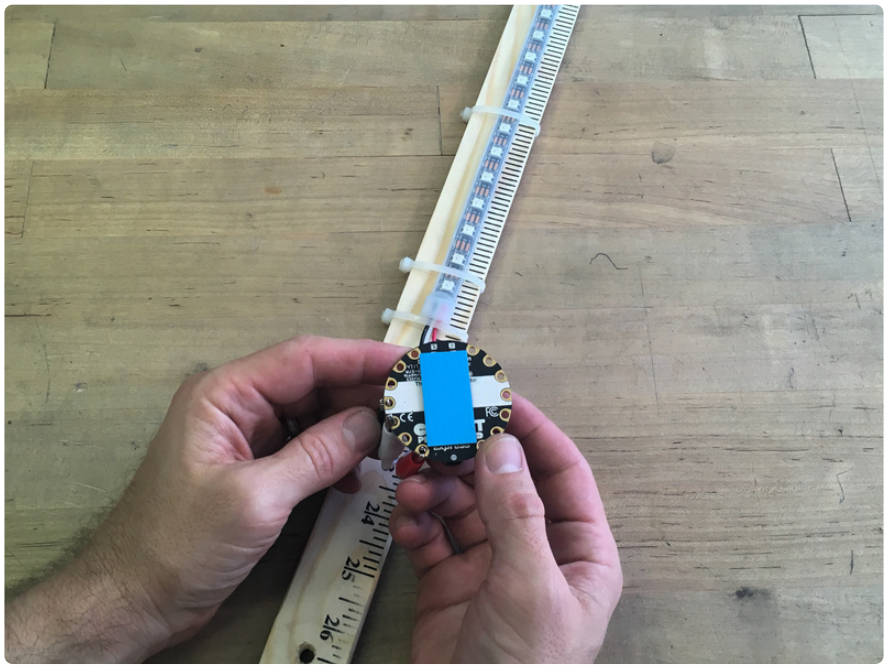
You can create a Circuit Playground Express Light Paintstick instead if you like! For this, we'll use a 3x AAA battery box instead of a LiPoly, a NeoPixel strip with alligator clips, and we won't use a potentiometer, instead controlling playback speed with the CPX buttons.

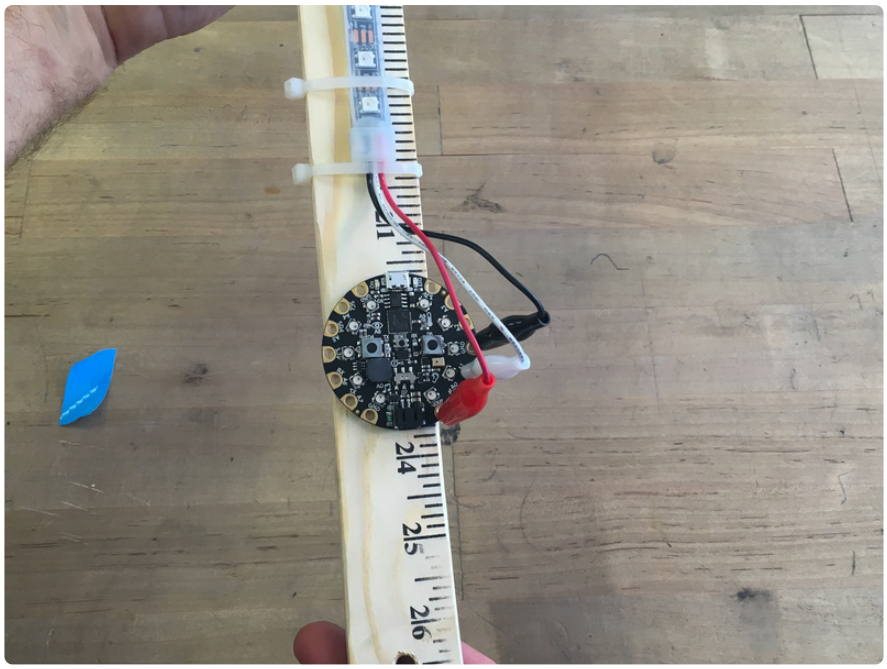

3 x AAA

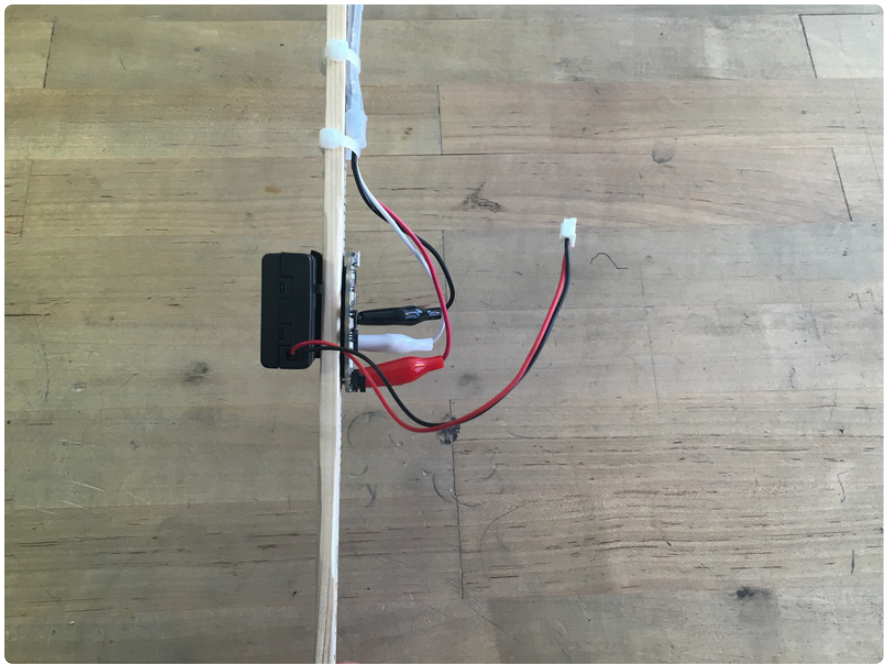


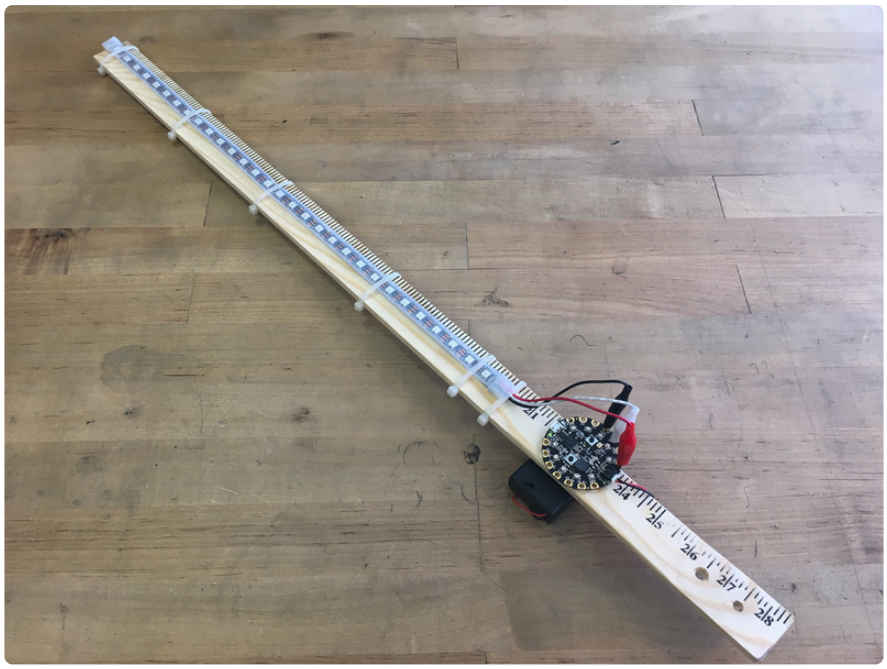


Use double-stick foam tape to connect the CPX to the stick, and to connect the battery box to the back side of the stick, behind the CPX.









Code with CircuitPython

The CircuitPython code we'll use was designed to display light painting art based upon bitmap images. There are two display modes: looping and one-shot.

Looping will run the image continuously. This is great for streaking images all around, or for "stamping" copies of your image into mid-air.

One-shot images will only display once each time you trigger them by pressing the capacitive touch sensor assigned -- in the case of HalloWing this is pad A2.

CircuitPython Setup

To get started, you'll want to set up your HalloWing or Circuit Playground Express for use with CircuitPython by following this guide for HalloWing (https://adafru.it/CmJ) or by following this guide (https://adafru.it/AFI) for CPX. When you're ready, and can upload code to the board return here.

Adafruit really likes using the Mu editor to edit the CircuitPython code. See this guide on loading and using Mu (https://adafru.it/ANO).

HalloWing Code

You can copy the code here and then paste it into Mu. Save it to your HalloWing as **code.py**

```
# SPDX-FileCopyrightText: 2018 John Edgar Park for Adafruit Industries
#
# SPDX-License-Identifier: MIT

"""HalloWing Light Paintbrush"""
# Single images only. Filename is set in code,
# potentiometer is used to tune playback SPEED
# images should be 30px high, up to 100px wide, 24-bit .bmp files

import gc
import time
import board
import touchio
import digitalio
from analogio import AnalogIn
from neopixel_write import neopixel_write

# uncomment one line only here to select bitmap
FILENAME = "bats.bmp" # BMP file to load from flash filesystem
#FILENAME = "digikey.bmp"
#FILENAME = "burger.bmp"
#FILENAME = "afbanner.bmp"
#FILENAME = "blinka.bmp"
#FILENAME = "ghost04.bmp"
#FILENAME = "ghost07.bmp"
#FILENAME = "ghost02.bmp"
#FILENAME = "helix-32x30.bmp"
#FILENAME = "wales2-107x30.bmp"
#FILENAME = "pumpkin.bmp"
#FILENAME = "rainbow.bmp"
#FILENAME = "rainbowRoad.bmp"
#FILENAME = "rainbowZig.bmp"
#FILENAME = "skull.bmp"
#FILENAME = "adabot.bmp"
#FILENAME = "green_stripes.bmp"
#FILENAME = "red_blue.bmp"
#FILENAME = "minerva.bmp"

TOUCH = touchio.TouchIn(board.A2) # Rightmost capacitive touch pad
ANALOG = AnalogIn(board.SENSE)    # Potentiometer on SENSE pin
BRIGHTNESS = 1.0                  # NeoPixel brightness 0.0 (min) to 1.0 (max)
GAMMA = 2.7                       # Adjusts perceived brightness linearity
```

```
NUM_PIXELS = 30                  # NeoPixel strip length (in pixels)
LOOP = False  #set to True for looping
# Switch off onboard NeoPixel...
NEOPIXEL_PIN = digitalio.DigitalInOut(board.NEOPIXEL)
NEOPIXEL_PIN.direction = digitalio.Direction.OUTPUT
neopixel_write(NEOPIXEL_PIN, bytearray(3))
# ...then assign NEOPIXEL_PIN to the external NeoPixel connector:
NEOPIXEL_PIN = digitalio.DigitalInOut(board.EXTERNAL_NEOPIXEL)
NEOPIXEL_PIN.direction = digitalio.Direction.OUTPUT
neopixel_write(NEOPIXEL_PIN, bytearray(NUM_PIXELS * 3))

def read_le(value):
    """Interpret multi-byte value from file as little-endian value"""
    result = 0
    shift = 0
    for byte in value:
        result += byte << shift
        shift += 8
    return result

class BMPError(Exception):
    """Error handler for BMP-loading function"""

def load_bmp(filename):
    """Load BMP file, return as list of column buffers"""
    # pylint: disable=too-many-locals, too-many-branches
    try:
        print("Loading", filename)
        with open("/" + filename, "rb") as bmp:
            print("File opened")
            if bmp.read(2) != b'BM':  # check signature
                raise BMPError("Not BitMap file")

            bmp.read(8) # Read & ignore file size and creator bytes

            bmp_image_offset = read_le(bmp.read(4)) # Start of image data
            bmp.read(4) # Read & ignore header size
            bmp_width = read_le(bmp.read(4))
            bmp_height = read_le(bmp.read(4))
            # BMPs are traditionally stored bottom-to-top.
            # If bmp_height is negative, image is in top-down order.
            # This is not BMP canon but has been observed in the wild!
            flip = True
            if bmp_height < 0:
                bmp_height = -bmp_height
                flip = False

            print("WxH: (%d,%d)" % (bmp_width, bmp_height))

            if read_le(bmp.read(2)) != 1:
                raise BMPError("Not single-plane")
            if read_le(bmp.read(2)) != 24: # bits per pixel
                raise BMPError("Not 24-bit")
            if read_le(bmp.read(2)) != 0:
                raise BMPError("Compressed file")

            print("Image format OK, reading data...")

            row_size = (bmp_width * 3 + 3) & ~3  # 32-bit line boundary

            # Constrain rows loaded to pixel strip length
            clipped_height = min(bmp_height, NUM_PIXELS)

            # Allocate per-column pixel buffers, sized for NeoPixel strip:
            columns = [bytearray(NUM_PIXELS * 3) for _ in range(bmp_width)]

            # Image is displayed at END (not start) of NeoPixel strip,
            # this index works incrementally backward in column buffers...
            idx = (NUM_PIXELS - 1) * 3
```

```
            for row in range(clipped_height):  # For each scanline...
                if flip:  # Bitmap is stored bottom-to-top order (normal BMP)
                    pos = bmp_image_offset + (bmp_height - 1 - row) * row_size
                else:  # Bitmap is stored top-to-bottom
                    pos = bmp_image_offset + row * row_size
                bmp.seek(pos) # Start of scanline
                for column in columns: # For each pixel of scanline...
                    # BMP files use BGR color order
                    blue, green, red = bmp.read(3)
                    # Rearrange into NeoPixel strip's color order,
                    # while handling brightness & gamma correction:
                    column[idx] = int(pow(green / 255, GAMMA) * BRIGHTNESS * 255 +
0.5)
                    column[idx+1] = int(pow(red / 255, GAMMA) * BRIGHTNESS * 255 +
0.5)
                    column[idx+2] = int(pow(blue / 255, GAMMA) * BRIGHTNESS * 255 +
0.5)
                idx -= 3  # Advance (back) one pixel

            # Add one more column with no color data loaded.  This is used
            # to turn the strip off at the end of the painting operation.
            if not LOOP:
                columns.append(bytearray(NUM_PIXELS * 3))

            print("Loaded OK!")
            gc.collect()  # Garbage-collect now so playback is smoother
            return columns

    except OSError as err:
        if err.args[0] == 28:
            raise OSError("OS Error 28 0.25") from err
        raise OSError("OS Error 0.5") from err
    except BMPError as err:
        raise BMPError("Failed to parse BMP: " + err.args[0]) from err


# Load BMP image, return 'COLUMNS' array:
COLUMNS = load_bmp(FILENAME)

print("Mem free:", gc.mem_free())

COLUMN_DELAY = ANALOG.value / 65535.0 / 10.0  # 0.0 to 0.1 seconds
while LOOP:
    for COLUMN in COLUMNS:
        neopixel_write(NEOPIXEL_PIN, COLUMN)
        time.sleep(COLUMN_DELAY)

while True:
    # Wait for touch pad input:
    while not TOUCH.value:
        continue

    COLUMN_DELAY = ANALOG.value / 65535.0 / 10.0  # 0.0 to 0.1 seconds
    # print(COLUMN_DELAY)

    # Play back color data loaded into each column:
    for COLUMN in COLUMNS:
        neopixel_write(NEOPIXEL_PIN, COLUMN)
        time.sleep(COLUMN_DELAY)
        # Last column is all 0's, no need to explicitly clear strip

    # Wait for touch pad release, just in case:
    while TOUCH.value:
        continue
```

Circuit Playground Express Code

If you're using the Circuit Playground Express, copy and paste this code instead.

```
# SPDX-FileCopyrightText: 2018 John Edgar Park for Adafruit Industries
#
# SPDX-License-Identifier: MIT

"""Circuit Playground Express Light Paintbrush"""
# Single images only. Filename is set in code,
# CPX buttons A&B are used to increase and decrease playback speed
# Touch pad A5 to trigger playback in one-shot mode
# Switch on CPX goes from one-shot to looping mode, requires reset
# images should be 30px high, up to 100px wide, 24-bit .bmp files

import gc
import time
import board
import touchio
import digitalio
from digitalio import DigitalInOut, Direction, Pull
from neopixel_write import neopixel_write

# uncomment one line only here to select bitmap
FILENAME = "bats.bmp" # BMP file to load from flash filesystem
#FILENAME = "jpw01.bmp"
#FILENAME = "digikey.bmp"
#FILENAME = "burger.bmp"
#FILENAME = "afbanner.bmp"
#FILENAME = "blinka.bmp"
#FILENAME = "ghost.bmp"
#FILENAME = "helix-32x30.bmp"
#FILENAME = "wales2-107x30.bmp"
#FILENAME = "pumpkin.bmp"
#FILENAME = "rainbow.bmp"
#FILENAME = "rainbowRoad.bmp"
#FILENAME = "rainbowZig.bmp"
#FILENAME = "skull.bmp"
#FILENAME = "adabot.bmp"
#FILENAME = "green_stripes.bmp"
#FILENAME = "red_blue.bmp"
#FILENAME = "minerva.bmp"

TOUCH = touchio.TouchIn(board.A5) #  capacitive touch pad
SPEED = 10000
SPEED_ADJUST = 2500               # This value changes the increment for button
speed adjustments
BRIGHTNESS = 1.0                  # Set brightness here, NOT in NeoPixel constructor
GAMMA = 2.7                       # Adjusts perceived brightness linearity
NUM_PIXELS = 30                   # NeoPixel strip length (in pixels)
NEOPIXEL_PIN = board.A1           # Pin where NeoPixels are connected
DELAY_TIME = 0.01                 # Timer delay before it starts
LOOP = False                      # Set to True for looping

# button setup
button_a = DigitalInOut(board.BUTTON_A)
button_a.direction = Direction.INPUT
button_a.pull = Pull.DOWN

button_b = DigitalInOut(board.BUTTON_B)
button_b.direction = Direction.INPUT
button_b.pull = Pull.DOWN

# switch setup
switch = DigitalInOut(board.SLIDE_SWITCH)
```

```
switch.direction = Direction.INPUT
switch.pull = Pull.UP

#status led setup
led = DigitalInOut(board.D13)
led.direction = Direction.OUTPUT


if switch.value:
    LOOP = True
else:
    LOOP = False


# Enable NeoPixel pin as output and clear the strip
NEOPIXEL_PIN = digitalio.DigitalInOut(NEOPIXEL_PIN)
NEOPIXEL_PIN.direction = digitalio.Direction.OUTPUT
neopixel_write(NEOPIXEL_PIN, bytearray(NUM_PIXELS * 3))

def read_le(value):
    """Interpret multi-byte value from file as little-endian value"""
    result = 0
    shift = 0
    for byte in value:
        result += byte << shift
        shift += 8
    return result

class BMPError(Exception):
    """Error handler for BMP-loading function"""

def load_bmp(filename):
    """Load BMP file, return as list of column buffers"""
    # pylint: disable=too-many-locals, too-many-branches
    try:
        print("Loading", filename)
        with open("/" + filename, "rb") as bmp:
            print("File opened")
            if bmp.read(2) != b'BM':  # check signature
                raise BMPError("Not BitMap file")

            bmp.read(8) # Read & ignore file size and creator bytes

            bmp_image_offset = read_le(bmp.read(4)) # Start of image data
            bmp.read(4) # Read & ignore header size
            bmp_width = read_le(bmp.read(4))
            bmp_height = read_le(bmp.read(4))
            # BMPs are traditionally stored bottom-to-top.
            # If bmp_height is negative, image is in top-down order.
            # This is not BMP canon but has been observed in the wild!
            flip = True
            if bmp_height < 0:
                bmp_height = -bmp_height
                flip = False

            print("WxH: (%d,%d)" % (bmp_width, bmp_height))

            if read_le(bmp.read(2)) != 1:
                raise BMPError("Not single-plane")
            if read_le(bmp.read(2)) != 24: # bits per pixel
                raise BMPError("Not 24-bit")
            if read_le(bmp.read(2)) != 0:
                raise BMPError("Compressed file")

            print("Image format OK, reading data...")

            row_size = (bmp_width * 3 + 3) & ~3  # 32-bit line boundary

            # Constrain rows loaded to pixel strip length
```

```
            clipped_height = min(bmp_height, NUM_PIXELS)

            # Allocate per-column pixel buffers, sized for NeoPixel strip:
            columns = [bytearray(NUM_PIXELS * 3) for _ in range(bmp_width)]

            # Image is displayed at END (not start) of NeoPixel strip,
            # this index works incrementally backward in column buffers...
            idx = (NUM_PIXELS - 1) * 3
            for row in range(clipped_height):  # For each scanline...
                if flip:  # Bitmap is stored bottom-to-top order (normal BMP)
                    pos = bmp_image_offset + (bmp_height - 1 - row) * row_size
                else:  # Bitmap is stored top-to-bottom
                    pos = bmp_image_offset + row * row_size
                bmp.seek(pos) # Start of scanline
                for column in columns: # For each pixel of scanline...
                    # BMP files use BGR color order
                    blue, green, red = bmp.read(3)
                    # Rearrange into NeoPixel strip's color order,
                    # while handling brightness & gamma correction:
                    column[idx] = int(pow(green / 255, GAMMA) * BRIGHTNESS * 255 +
0.5)
                    column[idx+1] = int(pow(red / 255, GAMMA) * BRIGHTNESS * 255 +
0.5)
                    column[idx+2] = int(pow(blue / 255, GAMMA) * BRIGHTNESS * 255 +
0.5)
                idx -= 3  # Advance (back) one pixel

            # Add one more column with no color data loaded.  This is used
            # to turn the strip off at the end of the painting operation.
            if not LOOP:
                columns.append(bytearray(NUM_PIXELS * 3))

            print("Loaded OK!")
            gc.collect()  # Garbage-collect now so playback is smoother
            return columns

    except OSError as err:
        if err.args[0] == 28:
            raise OSError("OS Error 28 0.25") from err
        raise OSError("OS Error 0.5") from err
    except BMPError as err:
        raise BMPError("Failed to parse BMP: " + err.args[0]) from err



# Load BMP image, return 'columns' array:
COLUMNS = load_bmp(FILENAME)

print("Mem free:", gc.mem_free())

COLUMN_DELAY = SPEED / 65535.0 / 10.0  # 0.0 to 0.1 seconds
# print(COLUMN_DELAY)

led.value = True

while LOOP:
    for COLUMN in COLUMNS:
        neopixel_write(NEOPIXEL_PIN, COLUMN)
        time.sleep(COLUMN_DELAY)

while True:

    # Wait for touch pad input:
    # buttons increase and decrease speed of playback
    while not TOUCH.value:
        if button_a.value:
            if SPEED >= SPEED_ADJUST:
                led.value = False
                SPEED = SPEED - SPEED_ADJUST
```

```
                COLUMN_DELAY = SPEED / 65535.0 / 10.0  # 0.0 to 0.1 seconds
                time.sleep(0.25)  #debounce
                led.value = True
                # print(SPEED)
        if button_b.value:
            if SPEED < 100000:
                led.value = False
                SPEED = SPEED + SPEED_ADJUST
                COLUMN_DELAY = SPEED / 65535.0 / 10.0  # 0.0 to 0.1 seconds
                time.sleep(0.25)  #debounce
                led.value = True
                # print(SPEED)
        continue

    time.sleep(DELAY_TIME)

# Play back color data loaded into each column:
    for COLUMN in COLUMNS:
        neopixel_write(NEOPIXEL_PIN, COLUMN)
        time.sleep(COLUMN_DELAY)
        # Last column is all 0's, no need to explicitly clear strip

    # Wait for touch pad release, just in case:
    while TOUCH.value:
        continue
```

For the CPX version, you can hit the on board buttons to increase or decrease the speed of playback. Additionally, flipping the switch to the left and restarting the board puts into looping animation mode, flip to the right and restart for one-shot mode where pressing A5 runs the image one time.

Image Selection

The are only two sections of code you'll need to adjust: `FILENAME` variable assignment and the image looping.

By default, it will play the **bats.bmp** file. If you create your own bitmap named **unicorn.bmp** and load it onto the HalloWing, you'll want to adjust the FILENAME to match, such as:

```
FILENAME = "unicorn.bmp"
```

If you want your image to loop automatically to create streaks or repeating stamps, change this line:

```
LOOP = False
```

to this:

```
LOOP = True
```

On the next page we'll add images to the board.

Make Pixel Art

You'll need some art to display with your HalloWing Light Paintstick. You can get started by downloading this collection, and then unzipping it.

PixelArt.zip

https://adafru.it/COb

Images

The Light Paintstick plays back images stored in flash memory on the HalloWing or Circuit Playground Express. To add images, plug in your board via USB to your computer. When the board mounts, you'll see a new drive named **CIRCUITPY** appear. Simply drag your image files onto the board with your computer file manager.-

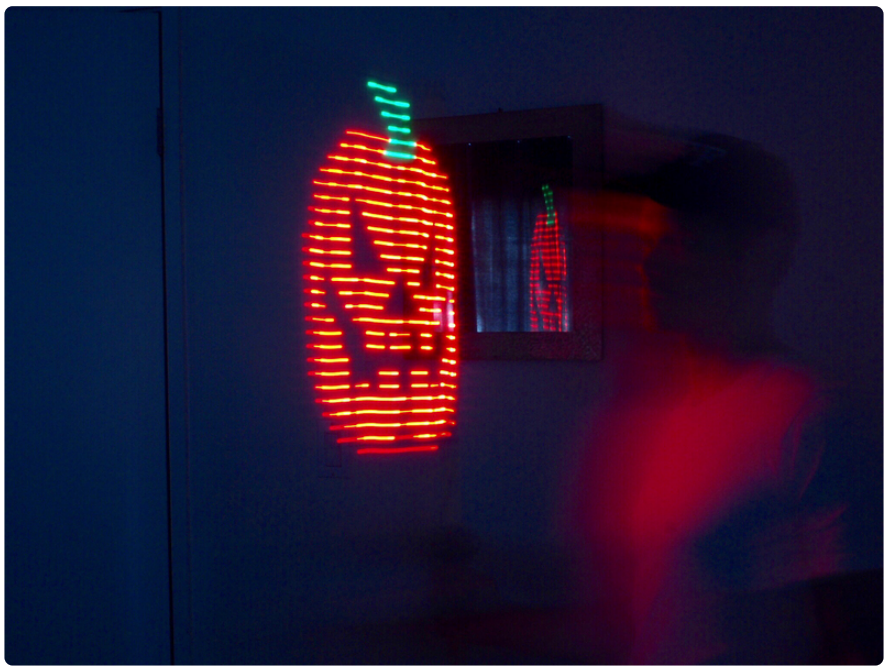
Art Specs

If you want to create your own artwork for display, these are the specifications to follow:

- Images should be a maximum of 30 pixels high (the same as the number of NeoPixels on our strip)
- Images can be up to 100 pixels wide
- Colors are 24-bit (8-bits per channel) RGB
- Save files as .bmp format

We've found that crisp images (not too much antialiasing) work best. The colors look best when using full primaries, such as 255 red, green, or blue, and mixes of those, but you can definitely try any color you like.

You can use nearly any paint program, including online pixel art generators that run in the browser!



Take Long Exposure Photos

Now we come to the fun part of making art that can hover in space! You'll freeze temporal, sequential art into what appears to be a single instant! How is this done? The trick is long exposure photography.

Typically, we take photographs that only expose the sensor (or film) for a very tiny fraction of a second. This is a simplification that ignores many factors, but you can think of it as: the shorter the exposure (also called shutter speed) the sharper the image. This is because any subjects that are moving will create a blur if the exposure is long, since they will occupy more than one point in space during the time that light is exposing onto the sensor.

Additionally, if we use typical aperture sizes, or f-stops, (think of it as the size of the hole letting light pass onto the shutter) we can let in a lot of light during that very quick exposure. If the shutter is open too long, then too much light will hit the sensor and the shot will be over exposed.

Long exposure photography flips these conventions on their head! We'll use very long exposures -- anywhere from 4 to 30 seconds for our Light Paintstick images -- so that our subject (the NeoPixels) will occupy many different points in space during the time that the shutter is open. But, to avoid over exposing the sensor and creating a blindingly bright image with no details, we'll use a very small aperture. This means that only very bright objects (such as our NeoPixel LEDs) will send enough light to the sensor to be exposed on the final image.

The size of the aperture is expressed in f-stop values. It can be confusing at first because the lower the f-stop number the larger the opening. This is because the f-number is a ratio of focal length to aperture diameter.

Typical settings for "normal" daylight photography are fast shutters -- 1/250 second for example, and wide open apertures -- f/5.6 for example. Long exposure photographs taken in dark settings will use slow shutter speeds such as 4" (seconds) to 30" and small apertures such as f/22.

Tools

Ideally, you will want to use a good camera with manual control over the settings, mounted on a tripod. Any mirrorless system, DSLR, or higher-end point-and-shoot should give you the control you need. The camera will need to allow you to shoot either long exposures or in "bulb" mode where the shutter stays open indefinitely until you release it.

Alternately, you can use a smart phone and dedicated apps. Search for the terms "long exposure" and "light trails" to find some options.

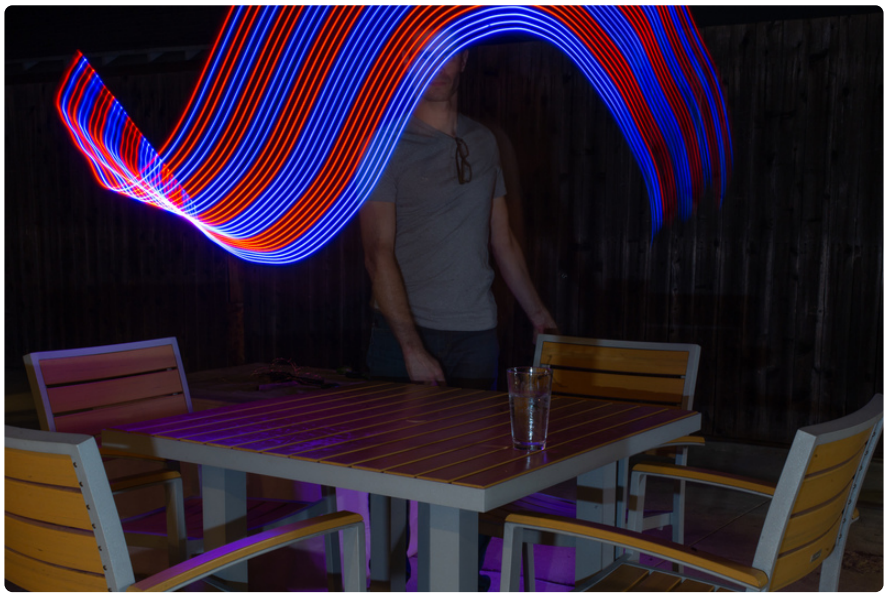
Don't forget that the HalloWing has an ON/OFF switch!

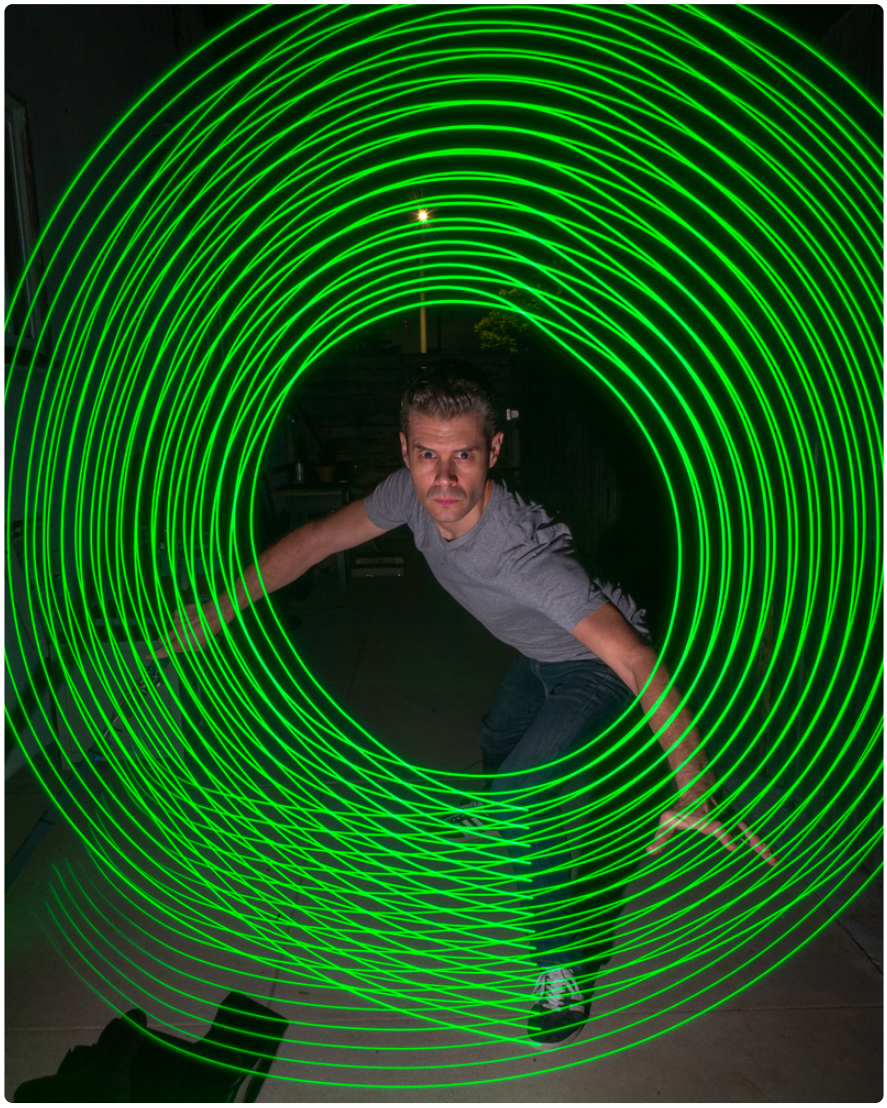
Action

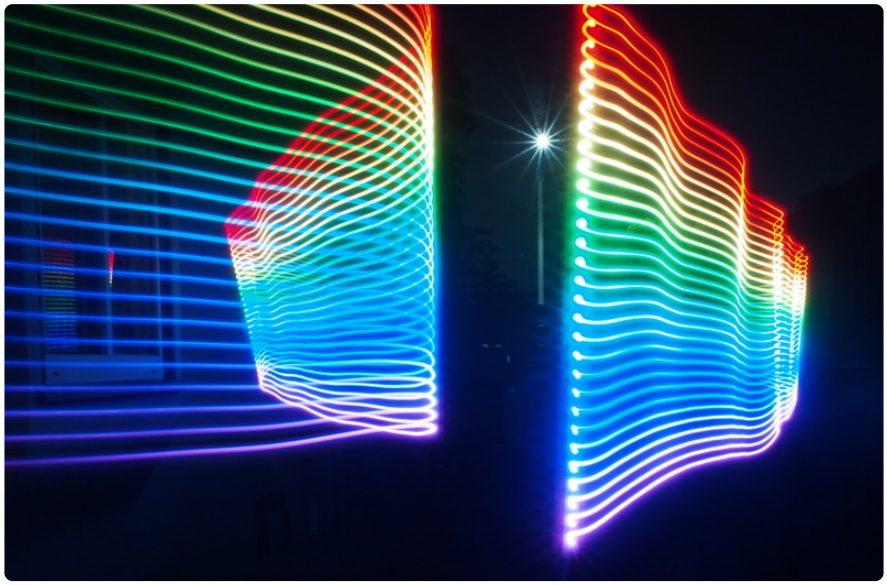
Now, you get to start experimenting! Start off simply, with a rainbow pattern. In a dark environment, set up your camera, trigger the shutter, get in front of the lens, turn on your Light Paintstick, and sweep an arc shape over your head.

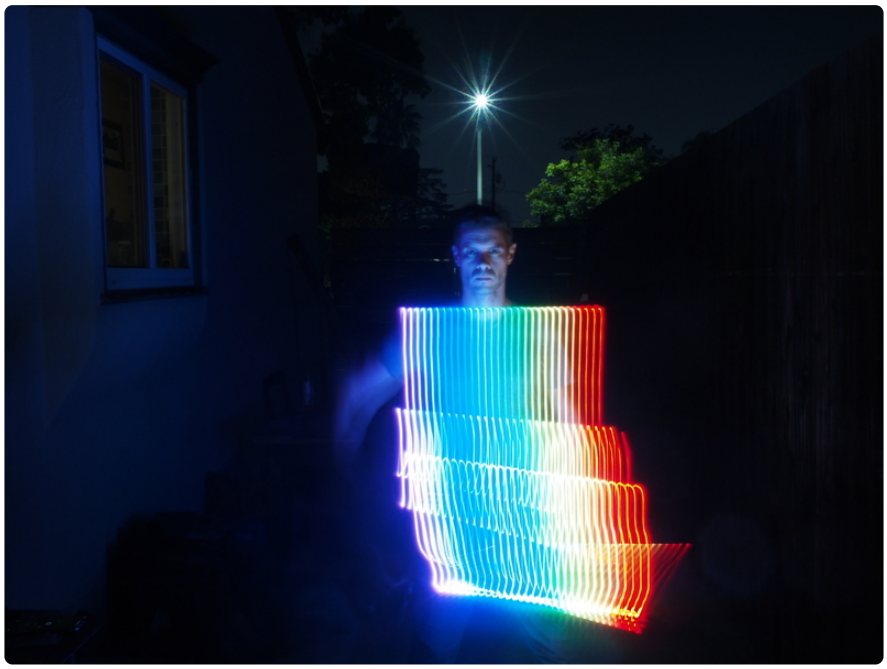
Release the shutter and check out your photo! You can now start to tune the settings to dial things in.

Next, try some longer exposures and run around with your Light Paintstick. Get creative! It's also fun to have some context in you photos, so try tuning the exposure settings on your camera so that some of your environment is visible, not just LED streaks against black.









Floating Images

Now, you can try stamping an image into midair! Switch the CircuitPython code `LOOP = True` to `LOOP = False` and re-save the **code.py** file onto the board so that you're displaying one of the individual bitmaps, such as the pumpkin.

You will want to tune the speed of the play back so that the image draws in about 3-4 seconds. Set your camera for a 5-6 second exposure. Trigger the shutter and then move the Light Paintstick in a straight line parallel to the camera.

You can draw logos in midair, too!

Bats!

Multiple Stamps

You can trigger your image multiple times during a single exposure, just try to not overlap! Also notice that you can create a "backwards" image by moving the want from left-to-right instead of right to left as with the bottom pumpkin shown here.

You can also switch the code back to looping `LOOP = True` so that as long as you move the wand the image will repeat.

Here's an example of this on a playground merry-go-round.

If your image is squashed, try turning the potentiometer to the right a bit so that it draws the raster a bit slower. If the image is too wide, turn the pot to the left to speed things up. You'll want to still move the wand at the same speed so that you only adjust a single variable at a time.

Here's an example of the same image being played back at different speeds by tuning the potentiometer between takes.

Rotate your arm in a big circle!

Have fun with your light painting! You can even start to get fancy and include yourself in the photos -- just draw your images as usual in the air, and then at the end, hold very still and point your Light Paintbrush at your own face for a few seconds to add it to the exposed portion of the frame!

So, have fun experimenting with different artwork and techniques as you explore the fascinating art of long exposure light painting with your HalloWing Light Paintstick!

Note, you can create some creepy outtakes while you're at it.

Here's the ghost of photographer Joel looking highly weird. This is what happens when you test the long exposure + flash theory. NOTE: He is not wearing a mask.