3D Printed Daft Punk Helmet

Created by Ruiz Brothers

https://learn.adafruit.com/3d-printed-daft-punk-helmet

Last updated on 2025-02-25 11:02:29 AM EST

Table of Contents

Overview 3

- 3D Printing + Cosplay
- Parts
- Tools & Supplies
- Prerequisite Guides

3D Printing 5

- PLA Filament
- Customize Helmet
- Large Build
- Fuse Filament
- Flexy Plate
- Slicer Settings

Circuit Diagram 7

- TRINKET + NeoPixel Strip
- Slide Switch Adapter

Arduino Code 10

- Side Animation
- Front Animation

CircuitPython Code 14

- Installing NeoPixel Library
- Saving CircuitPython Code
- Side Animation
- Front Animation

Assembly 21

- Clean Up The Print
- Masking Tape
- Cover it Up
- Spray Paint
- Peel it
- Slide Switch
- Measure and Cut NeoPixel Strips
- Solder Trinket
- Solder Strips
- Strip Placement
- Secure LED Strips to Helmet
- Foam Tape
- Wires

Overview

3D Printing + Cosplay

A perfect combination for making great things! This giant helmet is 3d printed in transparent/translucent PLA for a light up wearable, giving you freedom to headbang to your favorite beats.

As a hollowed out shell, you can easily to fit a strip of LEDs inside the helmet for making an epic LED costume. You can even customize the helmet in CAD software to fit your head!

> This guide was written for the original Trinket boards, but can be done with either the Trinket Mini, Trinket Pro or Trinket M0. We recommend the Trinket M0 as it is easier to use and is more compatible with modern computers!

Parts

We have two Trinket (or Pro Trinket) setups - one for front and one for sides, so you'll need to have two 'kits'!

- 2 x Trinket M0 (http://adafru.it/3500) - easier to use and more compatible with modern computers.
- or 2 x Trinket Mini (https://adafru.it/duC) - our demo uses Trinket, but for more complex animations you may want...
- or 2 x Trinket Pro (http://adafru.it/2000) - more code space and pins than a Trinket
- NeoPixel Strip (https://adafru.it/duD) - more LEDs mean more complex animations but more power draw!

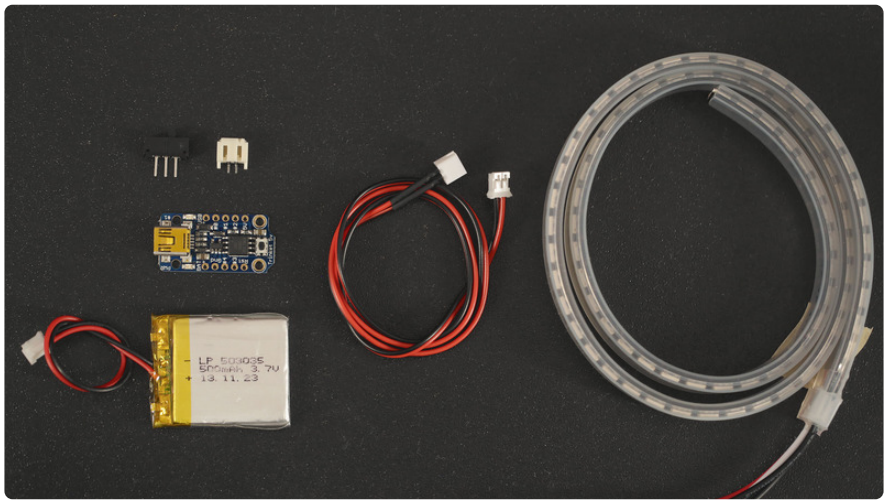
- 2 x Slide Switch (http://adafru.it/805)
- 2 x JST Extension (http://adafru.it/1131)/Connector (http://adafru.it/1769)
- 2 x 1200mAh Lithium Polymer Battery (http://adafru.it/1578) - bigger batteries last longer but take up more space.
- Micro Lipo charger for recharging the batteries (http://adafru.it/1904)

Tools & Supplies

- Solder Iron (http://adafru.it/1204) + Solder (http://adafru.it/734)
- Silicone Wire (http://adafru.it/1877)
- PLA Filament (https://adafru.it/dtp)
- 3D Printer (https://adafru.it/duF)
- Heat shrink tubing (http://adafru.it/344)

Prerequisite Guides

- Trinket M0 (https://adafru.it/Agg)
- Introducing Trinket (https://adafru.it/dhx)
- Introducting Pro Trinket (https://adafru.it/nYa)
- NeoPixel Uberguide (https://adafru.it/dhw)



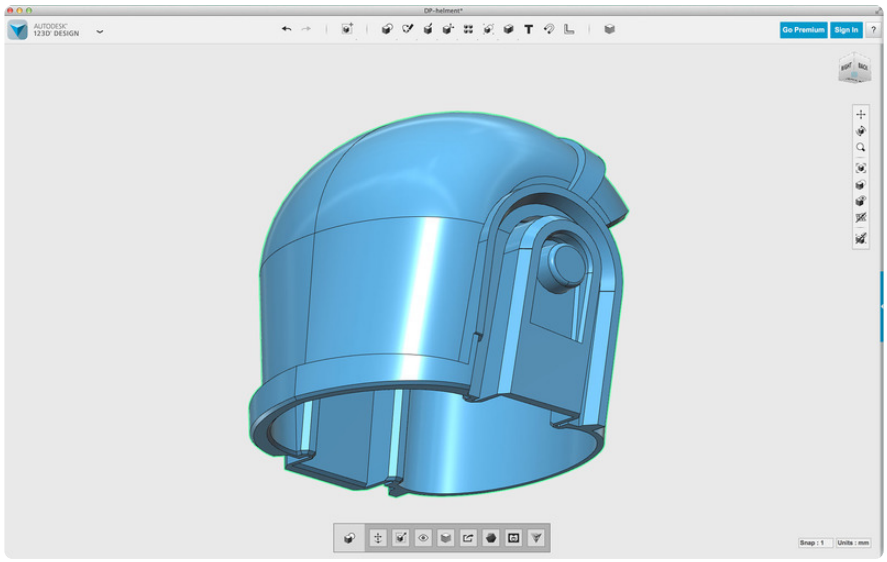
3D Printing

PLA Filament

Optimized for printing in PLA material, due to the large build. PLA is more dimensionally stable than ABS when it comes to big prints. The helmet is shelled out and prints best with no support or raft material.

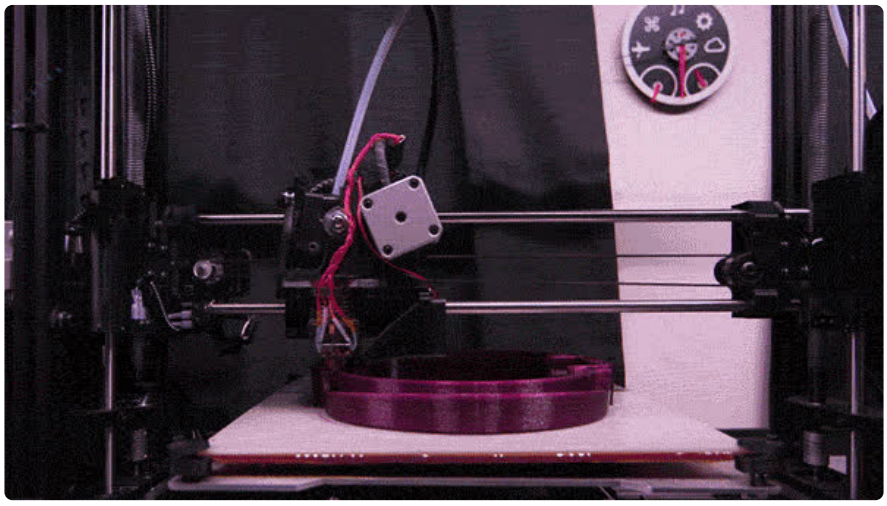
Download Helmet

https://adafru.it/dWM



Customize Helmet

Fit the helmet to your head! Measure around the top part of your head, right above the ears and edit the dimensions inside the DP-helment.123dx file, keep the maximum build size in mind. You can even launch 123D Design right in your browser.



Before you start printing, make sure that you have enough filament to complete the build! You'll need 1.2 pounds of PLA to finish this giant helmet, so it's a good idea to weigh how much filament is on your spool.

Large Build

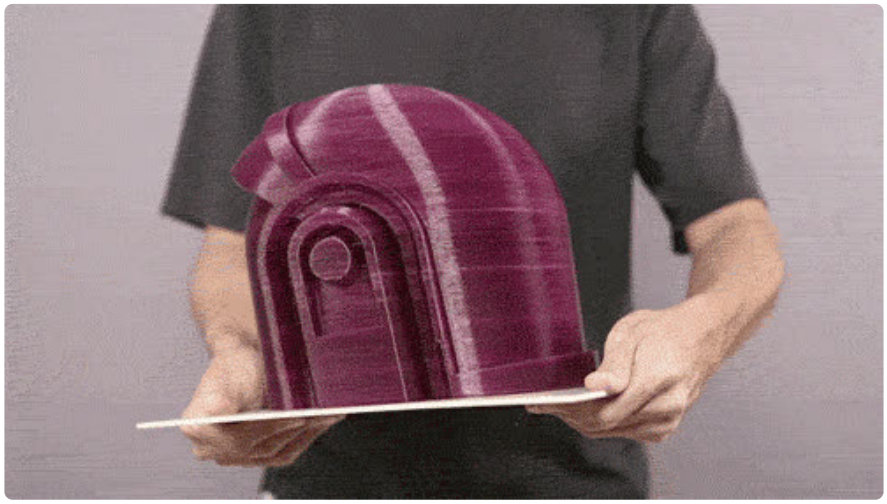
This jumbo helmet measures a massive 246mm x 226mm x 250mm and takes about 49 hours (3 Days) to print!

Fuse Filament

If you notice your spool running low, you can heat up two filament ends together with a heating element like a nozzle from a 3D Printing Pen to fuse the ends.

Flexy Plate

Easy remove large print off the platform by using a flexable build plate (http://adafru.it/2132).



Slicer Settings

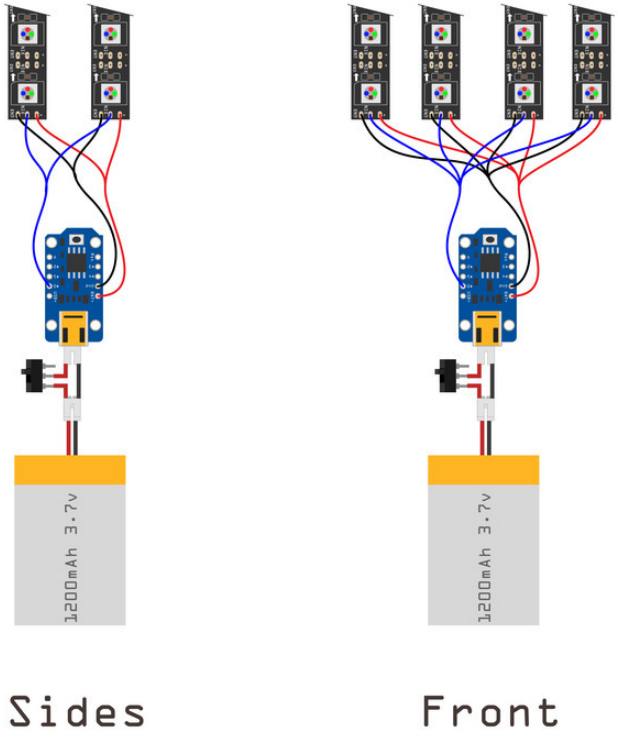
For the best quality when printing with PLA, we recommend the following slice settings:

- Retraction Speed: 800mm
- Retraction Distance: 1.5mm
- Speeds: 60/80
- Shells: 2
- Extruder Temp: 225c
- Infill 10%
- Support: Off
- No Heated Bed

Circuit Diagram

For our demo we're keeping it simple with two low cost Trinkets + neopixel strips that share the same data pin. However, this is just to get you started! Customize your own helmet with your own designs. You can also upgrade to a Pro Trinket which has way more flash space for more complex designs. (http://adafru.it/2000)

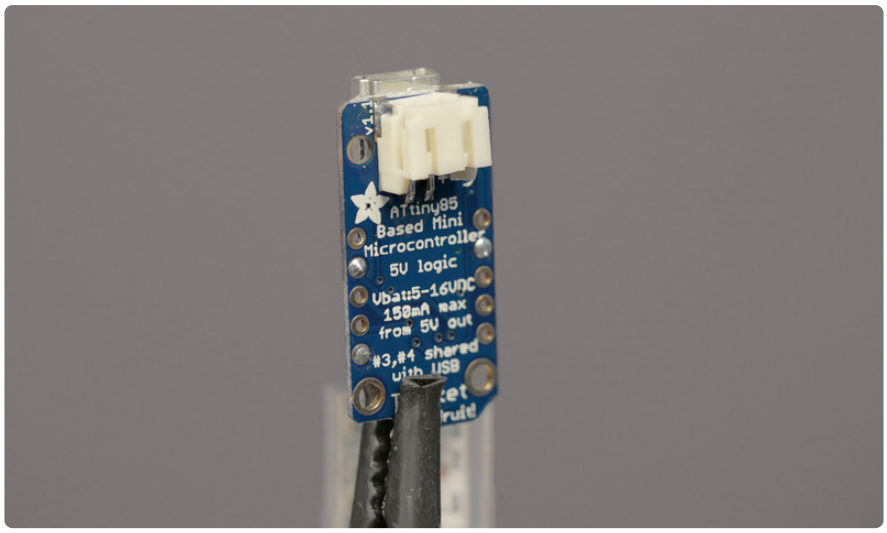
This diagram uses the original Trinket but you can also use the Trinket M0 with the exact same wiring!



Follow the illustration above as a reference for wiring the components. The trinket micro-controller is wired to the neopixel strip using a Y connection. Each strip is sharing a single connection to the trinket's data, power and ground pins.

This switch is only for about ~500mA of current (so only maybe 10 or 20 LEDs on at once). If you want to have way more LEDs and higher draw, please use this switch which is rated for 2 Amps (http://adafru.it/1092)

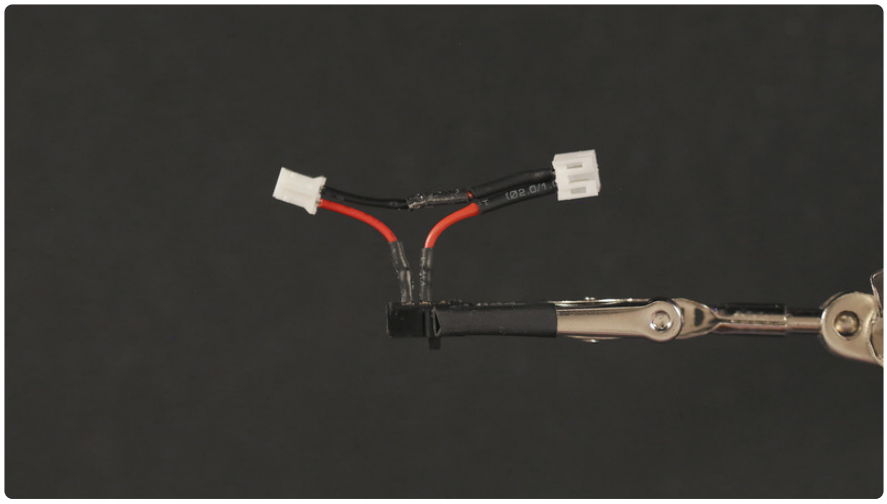
Be sure to wire the NeoPixels to the correct end! Remember it's the one with the arrow pointing to the right.



TRINKET + NeoPixel Strip

In this circuit diagram, you will need to solder a JST female connector (http://adafru.it/1769) to the bottom of the Trinket where the postive+ and negative- terminals are exposed.

The NeoPixel Strip **IN** pin is wired to **D0** on Trinket. **Postive+** pin on NeoPixel Strip is wired to **BAT** pin on Trinket. The **Negative-** pin on the strip is wired to **GND** pin on Trinket.



Slide Switch Adapter

Shorten a JST extension cable to about 10mm long by cutting the positive and negative cables with wire cutters. Use wire stripers to strip the ends of the positive and negative wires. Apply a bit of rosin to the stripped ends and tin the tips of the wires. Add a piece of shrink tubing to the negative wire and solder them together by holding them in place with a third-helping-hand.

Arduino Code

Make sure to to download the NeoPixel (https://adafru.it/aZU) Arduino library. Below is front and side code that will change the color of the NeoPixel strip - copy it into your Adafruit Arduino IDE as-is and then mod the LED Pins and number of pixels to make it your own. Remember that to program Trinket you need to download the special Adafruit version of the Arduino IDE from the Introduction to Trinket guide. (https://adafru.it/rBF)

The Arduino code presented below works equally well on the Trinket Mini and Trinket Pro. But if you have a Trinket M0 board you will need to use the CircuitPython code on the next page of this guide, no Arduino IDE required!

Side Animation

```
// SPDX-FileCopyrightText: 2014 Phil Burgess for Adafruit Industries
//
// SPDX-License-Identifier: MIT

// Adafruit Trinket+NeoPixel animation for Daft Punk-inspired helmet.
// Contains some ATtiny85-specific stuff; won't run as-is on Uno, etc.

// Operates in HSV (hue, saturation, value) colorspace rather than RGB.
// Animation is an interference pattern between two waves; one controls
// saturation, the other controls value (brightness).  The wavelength,
// direction, speed and type (square vs triangle wave) for each is randomly
// selected every few seconds.  Hue is always linear, but other parameters
// are similarly randomized.

#include <Adafruit_NeoPixel.h>
#include <avr/power.h>

// GLOBAL STUFF ------------------------------------------------------------

#define N_LEDS 29
#define PIN     0

Adafruit_NeoPixel    pixels = Adafruit_NeoPixel(N_LEDS, PIN);
volatile uint16_t    count  = 1; // Countdown to next animation change
extern const uint8_t gamma[];    // Big table at end of this code

volatile struct {
  uint8_t type,      // 0 = square wave, 1 = triangle wave
          value[2];  // 0 = start-of-frame value, 1 = pixel-to-pixel value
  int8_t  inc[2];    // 0 = frame-to-frame increment, 1 = pixel-to-pixel inc
} wave[3];           // 0 = Hue, 1 = Saturation, 2 = Value (brightness)

#define WAVE_H 0     // Array indices for wave[]
#define WAVE_S 1
#define WAVE_V 2
#define FRAME  0     // Array indices for value[] and inc[]
#define PIXEL  1


// INITIALIZATION ----------------------------------------------------------
```

```
void setup() {
  pixels.begin();
  randomSeed(analogRead(0)); // Seed random() from a floating pin (D2)

  // Timer/Counter 1 is used to generate a steady ~50 Hz frame rate.
#if F_CPU == 16000000L
  clock_prescale_set(clock_div_1);
  TCCR1  = _BV(PWM1A) | _BV(CS13) | _BV(CS12); // 1:2048 prescale
  OCR1C  = F_CPU / 2048 / 50 - 1;
#else
  TCCR1  = _BV(PWM1A) | _BV(CS13) | _BV(CS11) | _BV(CS10); // 1:1024
  OCR1C  = F_CPU / 1024 / 50 - 1;
#endif
  GTCCR  = 0;          // No PWM out
  TIMSK |= _BV(TOIE1); // Enable overflow interrupt
}

void loop() { } // Not used here -- everything's in interrupt below


// 50 HZ LOOP ------------------------------------------------------------------

ISR(TIMER1_OVF_vect) {
  uint8_t  w, i, n, s, v, r, g, b;
  uint16_t v1, s1;

  if(!(--count)) {              // Time for new animation?
    count = 250 + random(250);  // New effect will run for 5-10 sec.
    for(w=0; w<3; w++) {        // Three waves (H,S,V)...
      wave[w].type = random(2); // Assign random type (square/triangle)
      for(i=0; i<2; i++) {      // For frame and pixel increments...
        while(!(wave[w].inc[i] = random(15) - 7)); // Set non-zero random
        // wave value is never initialized; it's allowed to carry over
      }
      wave[w].value[PIXEL] = wave[w].value[FRAME];
    }
    wave[WAVE_S].inc[PIXEL] *= 16; // Make saturation & value
    wave[WAVE_V].inc[PIXEL] *= 16; // blinkier along strip
  } else { // Continue current animation; update waves
    for(w=0; w<3; w++) {
      wave[w].value[FRAME] += wave[w].inc[FRAME];   // OK if this wraps!
      wave[w].value[PIXEL]  = wave[w].value[FRAME];
    }
  }

  // Render current animation frame.  COGNITIVE HAZARD: fixed point math.

  for(i=0; i<N_LEDS; i++) { // For each LED along strip...

    // Coarse (8-bit) HSV-to-RGB conversion, hue first:
    n   = (wave[WAVE_H].value[PIXEL] % 43) * 6; // Angle within sextant; 0-255
    switch(wave[WAVE_H].value[PIXEL] / 43) {    // Sextant number; 0-5
      case 0 : r = 255    ; g =   n    ; b =   0    ; break; // R to Y
      case 1 : r = 254 - n; g = 255    ; b =   0    ; break; // Y to G
      case 2 : r =   0    ; g = 255    ; b =   n    ; break; // G to C
      case 3 : r =   0    ; g = 254 - n; b = 255    ; break; // C to B
      case 4 : r =   n    ; g =   0    ; b = 255    ; break; // B to M
      default: r = 255    ; g =   0    ; b = 254 - n; break; // M to R
    }

    // Saturation = 1-256 to allow >>8 instead of /255
    s = wave[WAVE_S].value[PIXEL];
    if(wave[WAVE_S].type) {     // Triangle wave?
      if(s & 0x80) {            // Downslope
        s    = (s & 0x7F) << 1;
        s1   = 256 - s;
      } else {                  // Upslope
        s  <<= 1;
```

```
        s1   = 1   + s;
        s    = 255 - s;
      }
    } else {                    // Square wave
      if(s & 0x80) {            // 100% saturation
        s1 = 256;
        s  =   0;
      } else {                  // 0% saturation (white)
        s1 =   1;
        s  = 255;
      }
    }

    // Value (brightness) = 1-256 for similar reasons
    v  = wave[WAVE_V].value[PIXEL];
    v1 = (wave[WAVE_V].type) ?                      // Triangle wave?
         ((v & 0x80) ? 64 - ((v & 0x7F) << 1) :    // Downslope
                       1 + ( v         << 1)  ) : // Upslope
         ((v & 0x80) ? 256 : 1);                    // Square wave; on/off

    pixels.setPixelColor(i,
      pgm_read_byte(&gamma[((((r * s1) >> 8) + s) * v1) >> 8]),
      pgm_read_byte(&gamma[((((g * s1) >> 8) + s) * v1) >> 8]),
      pgm_read_byte(&gamma[((((b * s1) >> 8) + s) * v1) >> 8]));

    // Update wave values along length of strip (values may wrap, is OK!)
    for(w=0; w<3; w++) wave[w].value[PIXEL] += wave[w].inc[PIXEL];
  }

  pixels.show();
}

// Gamma correction improves appearance of midrange colors.
// This table is positioned down here because it's a big annoying
// distraction.  The 'extern' near the top lets us reference it earlier.
const uint8_t gamma[] PROGMEM = {
    0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
    0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  1,  1,  1,  1,
    1,  1,  1,  1,  1,  1,  1,  1,  1,  2,  2,  2,  2,  2,  2,  2,
    2,  3,  3,  3,  3,  3,  3,  3,  4,  4,  4,  4,  4,  5,  5,  5,
    5,  6,  6,  6,  6,  7,  7,  7,  7,  8,  8,  8,  9,  9,  9, 10,
   10, 10, 11, 11, 11, 12, 12, 13, 13, 13, 14, 14, 15, 15, 16, 16,
   17, 17, 18, 18, 19, 19, 20, 20, 21, 21, 22, 22, 23, 24, 24, 25,
   25, 26, 27, 27, 28, 29, 29, 30, 31, 32, 32, 33, 34, 35, 35, 36,
   37, 38, 39, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 50,
   51, 52, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 66, 67, 68,
   69, 70, 72, 73, 74, 75, 77, 78, 79, 81, 82, 83, 85, 86, 87, 89,
   90, 92, 93, 95, 96, 98, 99,101,102,104,105,107,109,110,112,114,
  115,117,119,120,122,124,126,127,129,131,133,135,137,138,140,142,
  144,146,148,150,152,154,156,158,160,162,164,167,169,171,173,175,
  177,180,182,184,186,189,191,193,196,198,200,203,205,208,210,213,
  215,218,220,223,225,228,231,233,236,239,241,244,247,249,252,255 };
```

Front Animation

```
// SPDX-FileCopyrightText: 2014 Phil Burgess for Adafruit Industries
//
// SPDX-License-Identifier: MIT
//
// Fiery demon horns (rawr!) for Adafruit Trinket/Gemma.
// Adafruit invests time and resources providing this open source code,
// please support Adafruit and open-source hardware by purchasing
// products from Adafruit!
#include <Adafruit_NeoPixel.h>
#include <avr/power.h>

#define N_HORNS 1
```

```
#define N_LEDS 30 // Per horn
#define PIN     0
Adafruit_NeoPixel pixels = Adafruit_NeoPixel(N_HORNS * N_LEDS, PIN);

//      /\  ->   Fire-like effect is the sum of multiple triangle
// ____/  \____  waves in motion, with a 'warm' color map applied.
#define N_WAVES 6     // Number of simultaneous waves (per horn)
// Coordinate space for waves is 16x the pixel spacing,
// allowing fixed-point math to be used instead of floats.
struct {
  int16_t  lower;     // Lower bound of wave
  int16_t  upper;     // Upper bound of wave
  int16_t  mid;       // Midpoint (peak) ((lower+upper)/2)
  uint8_t  vlower;    // Velocity of lower bound
  uint8_t  vupper;    // Velocity of upper bound
  uint16_t intensity; // Brightness at peak
} wave[N_HORNS][N_WAVES];
long fade; // Decreases brightness as wave moves

// Gamma correction improves appearance of midrange colors
const uint8_t gamma[] PROGMEM = {
    0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
    0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  1,  1,  1,  1,
    1,  1,  1,  1,  1,  1,  1,  1,  1,  2,  2,  2,  2,  2,  2,  2,
    2,  3,  3,  3,  3,  3,  3,  3,  4,  4,  4,  4,  4,  5,  5,  5,
    5,  6,  6,  6,  6,  7,  7,  7,  7,  8,  8,  8,  9,  9,  9, 10,
   10, 10, 11, 11, 11, 12, 12, 13, 13, 13, 14, 14, 15, 15, 16, 16,
   17, 17, 18, 18, 19, 19, 20, 20, 21, 21, 22, 22, 23, 24, 24, 25,
   25, 26, 27, 27, 28, 29, 29, 30, 31, 32, 32, 33, 34, 35, 35, 36,
   37, 38, 39, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 50,
   51, 52, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 66, 67, 68,
   69, 70, 72, 73, 74, 75, 77, 78, 79, 81, 82, 83, 85, 86, 87, 89,
   90, 92, 93, 95, 96, 98, 99,101,102,104,105,107,109,110,112,114,
  115,117,119,120,122,124,126,127,129,131,133,135,137,138,140,142,
  144,146,148,150,152,154,156,158,160,162,164,167,169,171,173,175,
  177,180,182,184,186,189,191,193,196,198,200,203,205,208,210,213,
  215,218,220,223,225,228,231,233,236,239,241,244,247,249,252,255 };

static void random_wave(uint8_t h,uint8_t w) {          // Randomize one wave struct
  wave[h][w].upper     = -1;                            // Always start just below
head of strip
  wave[h][w].lower     = -16 * (3 + random(4));         // Lower end starts ~3-7
pixels back
  wave[h][w].mid       = (wave[h][w].lower + wave[h][w].upper) / 2;
  wave[h][w].vlower    = 3 + random(4);                 // Lower end moves at ~1/8
to 1/4 pixel/frame
  wave[h][w].vupper    = wave[h][w].vlower + random(4); // Upper end moves a bit
faster, spreading wave
  wave[h][w].intensity = 300 + random(600);
}

void setup() {
  uint8_t h, w;

  randomSeed(analogRead(1));
  pixels.begin();
  for(h=0; h<N_HORNS; h++) {
    for(w=0; w<N_WAVES; w++) random_wave(h, w);
  }
  fade = 234 + N_LEDS / 2;
  if(fade > 255) fade = 255;

  // A ~100 Hz timer interrupt on Timer/Counter1 makes everything run
  // at regular intervals, regardless of current amount of motion.
#if F_CPU == 16000000L
  clock_prescale_set(clock_div_1);
  TCCR1  = _BV(PWM1A) | _BV(CS13) | _BV(CS11) | _BV(CS10); // 1:1024 prescale
  OCR1C  = F_CPU / 1024 / 100 - 1;
#else
```

```
  TCCR1  = _BV(PWM1A) | _BV(CS13) | _BV(CS11); // 1:512 prescale
  OCR1C  = F_CPU / 512 / 100 - 1;
#endif
  GTCCR  = 0;          // No PWM out
  TIMSK |= _BV(TOIE1); // Enable overflow interrupt
}

void loop() { } // Not used -- everything's in interrupt below

ISR(TIMER1_OVF_vect) {
  uint8_t  h, w, i, r, g, b;
  int16_t  x;
  uint16_t sum;

  for(h=0; h<N_HORNS; h++) {              // For each horn...
    for(x=7, i=0; i<N_LEDS; i++, x+=16) { // For each LED along horn...
      for(sum=w=0; w<N_WAVES; w++) {      // For each wave of horn...
        if((x < wave[h][w].lower) || (x > wave[h][w].upper)) continue; // Out of
range
        if(x <= wave[h][w].mid) { // Lower half of wave (ramping up to peak
brightness)
          sum += wave[h][w].intensity * (x - wave[h][w].lower) / (wave[h][w].mid -
wave[h][w].lower);
        } else {               // Upper half of wave (ramping down from peak)
          sum += wave[h][w].intensity * (wave[h][w].upper - x) / (wave[h][w].upper -
wave[h][w].mid);
        }
      }
      // Now the magnitude (sum) is remapped to color for the LEDs.
      // A blackbody palette is used - fades white-yellow-red-black.
      if(sum < 255) {        // 0-254 = black to red-1
        r = pgm_read_byte(&gamma[sum]);
        g = b = 0;
      } else if(sum < 510) { // 255-509 = red to yellow-1
        r = 255;
        g = pgm_read_byte(&gamma[sum - 255]);
        b = 0;
      } else if(sum < 765) { // 510-764 = yellow to white-1
        r = g = 255;
        b = pgm_read_byte(&gamma[sum - 510]);
      } else {               // 765+ = white
        r = g = b = 255;
      }
      pixels.setPixelColor(h * N_LEDS + i, r, g, b);
    }

    for(w=0; w<N_WAVES; w++) { // Update wave positions for each horn
      wave[h][w].lower += wave[h][w].vlower;  // Advance lower position
      if(wave[h][w].lower >= (N_LEDS * 16)) { // Off end of strip?
        random_wave(h, w);                   // Yes, 'reboot' wave
      } else {                               // No, adjust other values...
        wave[h][w].upper    +=  wave[h][w].vupper;
        wave[h][w].mid       = (wave[h][w].lower + wave[h][w].upper) / 2;
        wave[h][w].intensity = (wave[h][w].intensity * fade) / 256; // Dimmer
      }
    }
  }
  pixels.show();
}
```

CircuitPython Code

If you haven't already, follow this guide (https://adafru.it/ABS) to preparing the Trinket M0, including updating it with the latest version of CircuitPython.

After prepping the Trinket M0 to run CircuitPython we'll also need to add a NeoPixel library. This guide tells you how (https://adafru.it/zcp), as well as providing a good primer on using NeoPixels on the Trinket M0 with CircuitPython.

Installing NeoPixel Library

Download the latest **adafruit-circuitpython-bundle-xxxx.zip** (or newer) from the releases directory (https://adafru.it/zB-)and then unzip it somewhere easy to find, such as your desktop. Then, copy the **neopixel.mpy** file to your **CIRCUITPY lib** directory on the Trinket M0.

Finally, if you had a **neopixel.py** file that was already in that same **lib** directory you can delete it (the mpy is a compressed version).

Saving CircuitPython Code

Once you've got things working, you can edit the **code.py** file on the Trinket M0 to adjust what it actually does. No need for a software IDE, complaining tools, or flashing the chip -- when you code with CircuitPython all you need is a text editor. Edit the code, save it to the Trinket M0, and it immediately runs!

Below is side and front code that will change the color of the NeoPixel strip.

Side Animation

```
# SPDX-FileCopyrightText: 2014 Phil Burgess for Adafruit Industries
#
# SPDX-License-Identifier: MIT

# Adafruit Trinket+NeoPixel animation for Daft Punk-inspired helmet.
# Contains some ATtiny85-specific stuff; won't run as-is on Uno, etc.

# Operates in HSV (hue, saturation, value) colorspace rather than RGB.
# Animation is an interference pattern between two waves; one controls
# saturation, the other controls value (brightness).  The wavelength,
# direction, speed and type (square vs triangle wave) for each is randomly
# selected every few seconds.  Hue is always linear, but other parameters
# are similarly randomized.

import random
import board
import neopixel
from analogio import AnalogIn

n_leds = 29             # number of LEDs per horn
led_pin = board.D0      # which pin your pixels are connected to

# initialize neopixel strip
pixels = neopixel.NeoPixel(led_pin, n_leds, brightness=1, auto_write=False)
count = 1               # countdown to next animation change

# Gamma-correction table
gamma = [
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1,
```

```
    1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2,
    2, 3, 3, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 5, 5, 5,
    5, 6, 6, 6, 6, 7, 7, 7, 7, 8, 8, 8, 9, 9, 9, 10,
    10, 10, 11, 11, 11, 12, 12, 13, 13, 13, 14, 14, 15, 15, 16, 16,
    17, 17, 18, 18, 19, 19, 20, 20, 21, 21, 22, 22, 23, 24, 24, 25,
    25, 26, 27, 27, 28, 29, 29, 30, 31, 32, 32, 33, 34, 35, 35, 36,
    37, 38, 39, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 50,
    51, 52, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 66, 67, 68,
    69, 70, 72, 73, 74, 75, 77, 78, 79, 81, 82, 83, 85, 86, 87, 89,
    90, 92, 93, 95, 96, 98, 99, 101, 102, 104, 105, 107, 109, 110,
    112, 114, 115, 117, 119, 120, 122, 124, 126, 127, 129, 131, 133,
    135, 137, 138, 140, 142, 144, 146, 148, 150, 152, 154, 156, 158,
    160, 162, 164, 167, 169, 171, 173, 175, 177, 180, 182, 184, 186,
    189, 191, 193, 196, 198, 200, 203, 205, 208, 210, 213, 215, 218,
    220, 223, 225, 228, 231, 233, 236, 239, 241, 244, 247, 249, 252,
    255
]

# initialize 3D list
wave = [0] * 5, [0] * 5, [0] * 5

wave_type = 0   # 0 = square wave, 1 = triangle wave
value_frame = 1 # start-of-frame value
value_pixel = 2 # pixel-to-pixel value
inc_frame = 3   # frame-to-frame increment
inc_pixel = 4   # pixel-to-pixel inc

wave_h = 0      # hue
wave_s = 1      # saturation
wave_v = 2      # brightness

# Random number generator is seeded from an unused 'floating'
# analog input - this helps ensure the random color choices
# aren't always the same order.
pin = AnalogIn(board.A0)
random.seed(pin.value)
pin.deinit()

# generate a non-zero random number for frame and pixel increments
def nz_random():
    random_number = 0

    while random_number <= 0:
        random_number = random.randint(0,15) - 7

    return random_number

while True:

    w = i = n = s = v = r = g = b = v1 = s1 = 0

    if count <= 0:                                  # time for new animation
        count = 250 + random.randint(0,250)         # effect run for 5-10 sec.

        for w in range(3):                          # three waves (H,S,V)
            wave[w][wave_type] = random.randint(0,2)# square vs triangle
            wave[w][inc_frame] = nz_random()        # frame increment
            wave[w][inc_pixel] = nz_random()        # pixel increment
            wave[w][value_pixel] = wave[w][value_frame]

        wave[wave_s][inc_pixel] *= 16               # make saturation & value
        wave[wave_v][inc_pixel] *= 16               # blinkier along strip

    else:                                           # continue animation
        count -= 1
        for w in range(3):
            wave[w][value_frame] += wave[w][inc_frame]
            wave[w][value_pixel] = wave[w][value_frame]
```

```
    # Render current animation frame.  COGNITIVE HAZARD: fixed point math.
    for i in range(n_leds):                             # for each LED along
strip...
        # Coarse (8-bit) HSV-to-RGB conversion, hue first:
        n = (wave[wave_h][value_pixel] % 43) * 6   # angle within sextant

        sextant = wave[wave_h][value_pixel] / 43        # sextant number 0-5

        # R to Y
        if sextant == 0:
            r = 255
            g = n
            b = 0
        # Y to G
        elif sextant == 1:
            r = 254 - n
            g = 255
            b = 0
        # G to C
        elif sextant == 2:
            r = 0
            g = 255
            b = n
        # C to B
        elif sextant == 3:
            r = 0
            g = 254 - n
            b = 255
        # B to M
        elif sextant == 4:
            r = n
            g = 0
            b = 255
        # M to R
        else:
            r = 255
            g = 0
            b = 254 - n

        # Saturation = 1-256 to allow >>8 instead of /255
        s = wave[wave_s][value_pixel]

        if wave[wave_s][wave_type]:     # triangle wave?
            if s & 0x80:                # downslope
                s = (s & 0x7F) << 1
                s1 = 256 - s
            else:                       # upslope
                s = s<<1
                s1 = 1 + s
                s = 255 - s
        else:
            if s & 0x80:                # square wave
                s1 = 256                # 100% saturation
                s = 0
            else:                       # 0% saturation
                s1 = 1
                s = 255

        # Value (brightness) = 1-256 for similar reasons
        v = wave[wave_v][value_pixel]

        # value (brightness) = 1-256 for similar reasons
        if wave[wave_v][wave_type]:     # triangle wave?
            if v & 0x80:                # downslope
                v1 = 64 - ((v & 0x7F) << 1)
            else:                       # upslope
                v1 = 1 + (v << 1)
        else:
            if v & 0x80:                # square wave; on/off
```

```
                v1 = 256
            else:
                v1 = 1

        # gamma rgb values
        gr = ((((r * s1) >> 8) + s) * v1) >> 8
        gg = ((((g * s1) >> 8) + s) * v1) >> 8
        gb = ((((b * s1) >> 8) + s) * v1) >> 8

        # gamma rgb indices range check
        if -256 < gr < 256:
            r = gamma[gr]

        if -256 < gg < 256:
            g = gamma[gg]

        if -256 < gb < 256:
            b = gamma[gb]

        pixels[i] = (r, g, b)

        # update wave values along length of strip (values may wrap, is OK!)
        for w in range(3):
            wave[w][value_pixel] += wave[w][inc_pixel]

    pixels.show()
```

Copy the above "Side Animation" code, and then paste it into a new text document in your favorite text/coding editor. Then, save it to your Trinket M0's **CIRCUITPY** drive as **code.py**

Front Animation

```
# SPDX-FileCopyrightText: 2014 Phil Burgess for Adafruit Industries
#
# SPDX-License-Identifier: MIT

# Fiery demon horns (rawr!) for Adafruit Trinket/Gemma.
# Adafruit invests time and resources providing this open source code,
# please support Adafruit and open-source hardware by purchasing
# products from Adafruit!

import board
import neopixel
from analogio import AnalogIn
# pylint: disable=global-statement

try:
    import urandom as random
except ImportError:
    import random

# /\  ->   Fire-like effect is the sum_total of multiple triangle
# ____/  \____  waves in motion, with a 'warm' color map applied.
n_horns = 1             # number of horns
led_pin = board.D0      # which pin your pixels are connected to
n_leds = 30             # number of LEDs per horn
frames_per_second = 50  # animation frames per second
brightness = 0          # current wave height
fade = 0                # Decreases brightness as wave moves
pixels = neopixel.NeoPixel(led_pin, n_leds, brightness=1, auto_write=False)
offset = 0

# Coordinate space for waves is 16x the pixel spacing,
```

```
# allowing fixed-point math to be used instead of floats.
lower = 0       # lower bound of wave
upper = 1       # upper bound of wave
mid = 2         # midpoint (peak) ((lower+upper)/2)
vlower = 3      # velocity of lower bound
vupper = 4      # velocity of upper bound
intensity = 5   # brightness at peak

y = 0
brightness = 0
count = 0

# initialize 3D list
wave = [[0] * 6] * 6, [[0] * 6] * 6, [[0] * 6] * 6, [[0] * 6] * 6, [[0] * 6] * 6,
[[0] * 6] * 6

# Number of simultaneous waves (per horn)
n_waves = len(wave)

# Gamma-correction table
gamma = [
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1,
    1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2,
    2, 3, 3, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 5, 5, 5,
    5, 6, 6, 6, 6, 7, 7, 7, 7, 8, 8, 8, 9, 9, 9, 10,
    10, 10, 11, 11, 11, 12, 12, 13, 13, 13, 14, 14, 15, 15, 16, 16,
    17, 17, 18, 18, 19, 19, 20, 20, 21, 21, 22, 22, 23, 24, 24, 25,
    25, 26, 27, 27, 28, 29, 29, 30, 31, 32, 32, 33, 34, 35, 35, 36,
    37, 38, 39, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 50,
    51, 52, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 66, 67, 68,
    69, 70, 72, 73, 74, 75, 77, 78, 79, 81, 82, 83, 85, 86, 87, 89,
    90, 92, 93, 95, 96, 98, 99, 101, 102, 104, 105, 107, 109, 110,
    112, 114, 115, 117, 119, 120, 122, 124, 126, 127, 129, 131, 133,
    135, 137, 138, 140, 142, 144, 146, 148, 150, 152, 154, 156, 158,
    160, 162, 164, 167, 169, 171, 173, 175, 177, 180, 182, 184, 186,
    189, 191, 193, 196, 198, 200, 203, 205, 208, 210, 213, 215, 218,
    220, 223, 225, 228, 231, 233, 236, 239, 241, 244, 247, 249, 252,
    255
]


def random_wave(he, wi):
    wave[he][wi][upper] = -1                                    # Always start below
head of strip
    wave[he][wi][lower] = -16 * (3 + random.randint(0,4))     # Lower end starts
~3-7 pixels back
    wave[he][wi][mid] = (wave[he][wi][lower]+ wave[he][wi][upper]) / 2
    wave[he][wi][vlower] = 3 + random.randint(0,4)            #  Lower end moves at
~1/8 to 1/pixels
    wave[he][wi][vupper] = wave[he][wi][vlower]+ random.randint(0,4) # Upper end
moves a bit faster
    wave[he][wi][intensity] = 300 + random.randint(0,600)

def setup():
    global fade

    # Random number generator is seeded from an unused 'floating'
    # analog input - this helps ensure the random color choices
    # aren't always the same order.
    pin = AnalogIn(board.A0)
    random.seed(pin.value)
    pin.deinit()

    for he in range(n_horns):
        for wi in range(n_waves):
            random_wave(he, wi)

    fade = 233 + n_leds / 2
```

```
    if fade > 233:
        fade = 233

setup()

while True:

    h = w = i = r = g = b = 0
    x = 0

    for h in range(n_horns):                # For each horn...
        x = 7
        sum_total = 0
        for i in range(n_leds):             # For each LED along horn...
            x += 16
            for w in range(n_waves):        # For each wave of horn...
                if (x < wave[h][w][lower]) or (x > wave[h][w][upper]):
                    continue                # Out of range
                if x <= wave[h][w][mid]:    # Lower half of wave (ramping up peak
brightness)
                    sum_top = wave[h][w][intensity] * (x - wave[h][w][lower])
                    sum_bottom = (wave[h][w][mid] - wave[h][w][lower])
                    sum_total += sum_top /  sum_bottom
                else:                       # Upper half of wave (ramping down from
peak)
                    sum_top = wave[h][w][intensity] * (wave[h][w][upper] - x)
                    sum_bottom = (wave[h][w][upper] - wave[h][w][mid])
                    sum_total += sum_top / sum_bottom

            sum_total = int(sum_total)          # convert from decimal to whole
number

            # Now the magnitude (sum_total) is remapped to color for the LEDs.
            # A blackbody palette is used - fades white-yellow-red-black.
            if sum_total < 255:                 # 0-254 = black to red-1
                r = gamma[sum_total]
                g = b = 0
            elif sum_total < 510:               # 255-509 = red to yellow-1
                r = 255
                g = gamma[sum_total - 255]
                b = 0
            elif sum_total < 765:               # 510-764 = yellow to white-1
                r = g = 255
                b = gamma[sum_total - 510]
            else:                               # 765+ = white
                r = g = b = 255
            pixels[i] = (r, g, b)

    for w in range(n_waves):                    # Update wave positions for each
horn
        wave[h][w][lower] += wave[h][w][vlower] # Advance lower position
        if wave[h][w][lower] >= (n_leds * 16):  # Off end of strip?
            random_wave(h, w)                   # Yes, 'reboot' wave
        else:                                   # No, adjust other values...
            wave[h][w][upper] += wave[h][w][vupper]
            wave[h][w][mid] = (wave[h][w][lower] + wave[h][w][upper]) / 2
            wave[h][w][intensity] = (wave[h][w][intensity] * fade) / 256 # Dimmer

    pixels.show()
```

Copy the above "Front Animation" code, and then paste it into a new text document in your favorite text/coding editor. Then, save it to your Trinket M0's **CIRCUITPY** drive as **code.py**

Assembly

Clean Up The Print

Use a pair of flush diagonal cutter to clean up the deposits left over from printing

Masking Tape

Apply masking tape to the front of the helmet. Use a flat edge to get the side walls.

Cover it Up

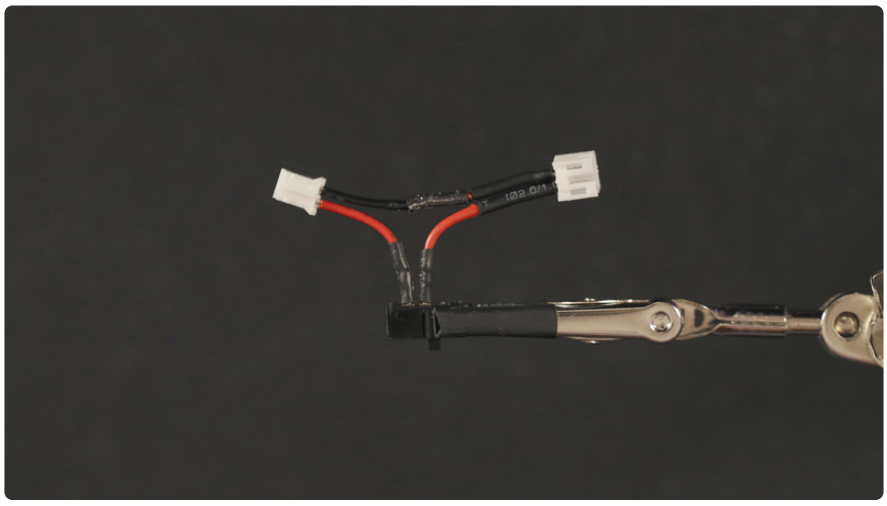
Make sure to cover up the front visor area. One layer of masking tape is suffice.

Spray Paint

A coat of glossy golden spray paint will add shiny separation to the transperanet purple front.

Peel it

Wait about 45 mintues for the paint to dry and remove the masking tape. You can use tweezers to help get the stuck edges.

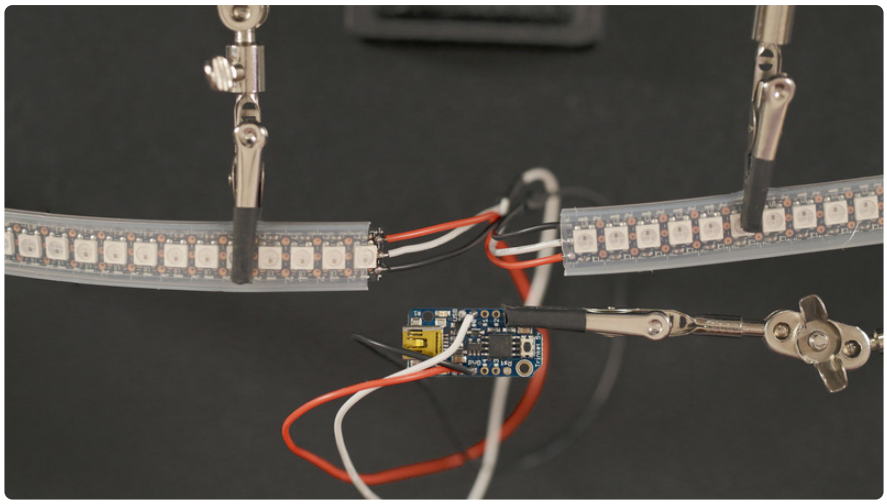


Slide Switch

Build an off and on switch by shortening and soldering a JST extension cable to a slide switch as shown. Don't forget to add the heat shrink tube before soldering!

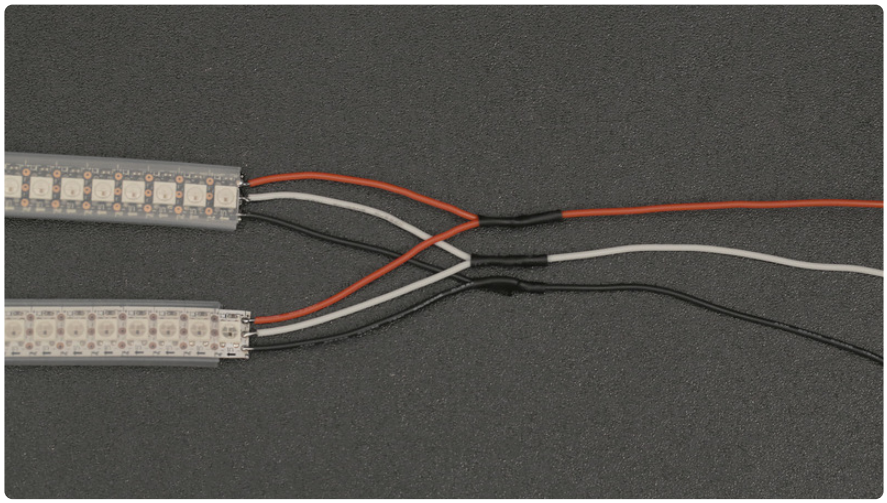
Measure and Cut NeoPixel Strips

Determine how long you'd like the LED strip to be and carefully cut the strip and sheathing.



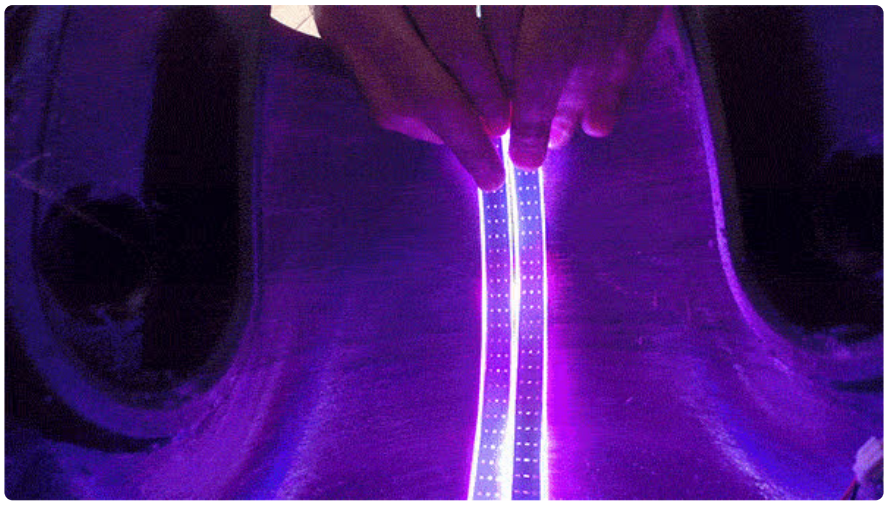
Solder Trinket

Solder the three silicone wires for power, ground and digital to the top of the trinket. Use a pair of third helping hands to secure hold the neopixel strips and trinket while soldering.



Solder Strips

Build a "Y" connection in the wiring to connect the two strips on each side of the helmet. Build another set with two on each side for a total of four strips for the front part of the helmet.

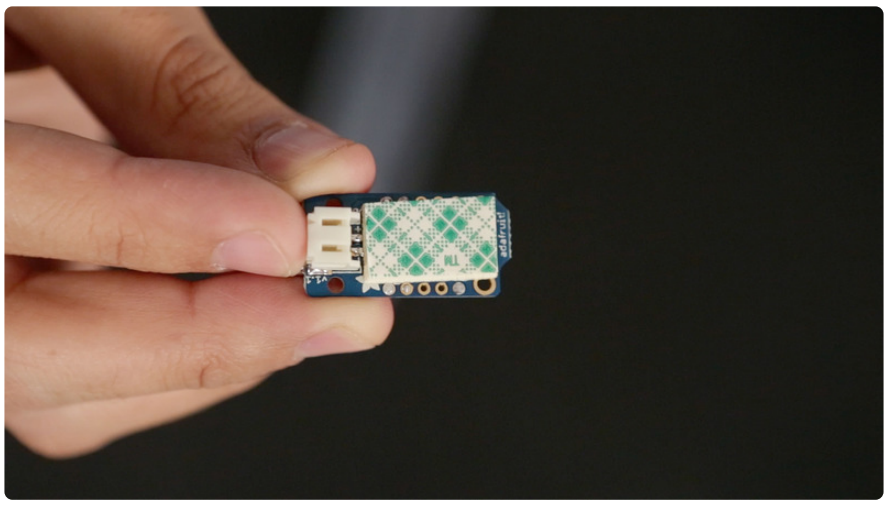


Strip Placement

Check the placement of the strips by spacing each from the center of the helmet.

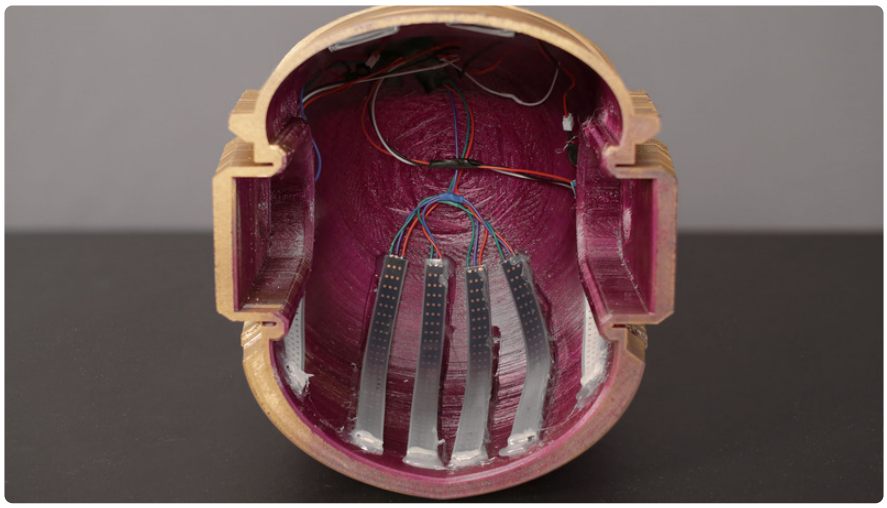
Secure LED Strips to Helmet

Once your happy with the layout of the strips, secure each one down with your favorite adhesive.



Foam Tape

Use small pieces of double sided foam tape to secure the trinkets right above the ears curves.



Wires

Arrange the wires along the top and back of the helmet to keep them out of the way.

Now your ready to code some animation into the two Trinkets!