3D Printed LED Fire Horns
Created by Ruiz Brothers

Last updated on 2021-08-30 04:18:51 PM EDT
## Guide Contents

**Guide Contents**

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview</td>
<td>2</td>
</tr>
<tr>
<td>3D Printing + Cosplay</td>
<td>3</td>
</tr>
<tr>
<td>Prerequisite Guides</td>
<td>3</td>
</tr>
<tr>
<td>Parts</td>
<td>3</td>
</tr>
<tr>
<td>Tools &amp; Supplies</td>
<td>4</td>
</tr>
<tr>
<td>3D Printing</td>
<td>5</td>
</tr>
<tr>
<td>Ninjaflex Filament</td>
<td>5</td>
</tr>
<tr>
<td>Lorn Horns</td>
<td>5</td>
</tr>
<tr>
<td>Slicer Settings</td>
<td>5</td>
</tr>
<tr>
<td>Support Removal</td>
<td>5</td>
</tr>
<tr>
<td>Circuit Diagram</td>
<td>7</td>
</tr>
<tr>
<td>GEMMA + 2x NeoPixel 12 Ring</td>
<td>7</td>
</tr>
<tr>
<td>TRINKET + NeoPixel Strip</td>
<td>8</td>
</tr>
<tr>
<td>Slide Switch Adapter</td>
<td>8</td>
</tr>
<tr>
<td>Assembly</td>
<td>9</td>
</tr>
<tr>
<td>Arduino Code</td>
<td>12</td>
</tr>
<tr>
<td>CircuitPython Code</td>
<td>15</td>
</tr>
</tbody>
</table>

© Adafruit Industries

https://learn.adafruit.com/3d-printed-led-fire-horns
Overview

3D Printing + Cosplay

A perfect combination for making great things! These fire horns are 3d printed in Ninjaflex for a flexible wearable that won't poke your eyes out, giving you freedom to headbang to your hearts desire.

As a hollowed out shell, you can easily fit a strip or ring of LEDs inside these devilish horns for making an epic LED costume. We have 3 different types of horns, so you can pick the one that matches your style.

Prerequisite Guides

- [Introducing Gemma M0 guide](https://adafru.it/yeq) or [Introducing GEMMA guide](https://adafru.it/cHH)
- [Introducing Trinket M0 guide](https://adafru.it/Agg) or [Introducing Trinket guide](https://adafru.it/dpf)
- [NeoPixel Uberguide](https://adafru.it/dhw)

This guide was written for the Gemma v2 board, but can also be done with the Gemma M0, Trinket M0 or Trinket Mini. We recommend the Gemma M0 as it is easier to use and is more compatible with modern computers!

Parts

- [Gemma M0](https://adafru.it/ytb), [Trinket M0](https://adafru.it/zya), [Trinket Mini](https://adafru.it/egk) or [Gemma v2](http://adafru.it/1222) - We recommend using the Gemma M0.
- [NeoPixel Strip](https://adafru.it/BpR)/[Ring](https://adafru.it/BpS)
- [Slide Switch](https://adafru.it/805)
Tools & Supplies

- [Solder Iron](http://adafruit.it/1204) + [Solder](http://adafruit.it/734)
- [Wire Wrap](https://adafruit.it/diJ)
- [NinjaFlex Filament](https://adafruit.it/dtp)
- [3D Printer](https://adafruit.it/tkf)
- [Heat shrink tubing](https://adafruit.it/diK)
3D Printing

Ninjaflex Filament
Optimized for printing in Ninjaflex TPE material, each horn is shelled out and prints best with support material.

Lorn Horns
These jumbo horns have optimized support structures included in the STLs.

Slicer Settings
For the best quality when printing with ninjaflex, we recommend the following slice settings:

- Retraction: Off
- Speeds: 45/50
- Extruder Temp: 230c
- Infill 10%
- Support: On
- No Heated Bed

Support Removal
Use a pair of sharp scissors to cut away and remove support material. You can use a pair of flat pliers to grab onto the support material inside the horns.

Remember to cut the support material before pulling it out with pliers, you don't want to tear or
rip the horn!
Circuit Diagram

This diagram uses the Gemma v2 but you can also use the Gemma M0 with the exact same wiring configuration.

GEMMA + 2x NeoPixel 12 Ring
In this circuit diagram, two 12 NeoPixel Rings are wired to the GEMMA. The first NeoPixel Ring has the IN pin wired to the D0 pin on the GEMMA. The second NeoPixel ring IN pin is chained to the first NeoPixel ring OUT. Both rings will share PWR and GND pins.

This diagram uses the Trinket Mini but you can also use the Trinket M0 with the exact same wiring configuration.
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 positive+ and negative- terminals are exposed.

The NeoPixel Strip IN pin is wired to D0 on Trinket. +5V pin on NeoPixel Strip is wired to 3V/5V pin on Trinket. The GND pin on the strip is wired to GND pin on Trinket.

Note that the pin orders may be different than the diagram, so check the strip markings!!!
Assembly

Attach horns to a pair of goggles by bending the metal ends off of the straps. Put the metal piece a side and detach the strap from one side of the goggles. Now we can guide the strap through the tabs on the horns.

Measure, cut and insert the NeoPixel Strip into each horn. Position the strip so that the LEDs are facing towards the front of the horns.
Use a piece of double sided foam tape to attach the Trinket or Gemma to the inside of the horn.

Use another piece of double sided foam tape to secure the battery to another side of the horn.
Position the slide switch by the goggle strap or cut a hole on the side of the horn for the switch.

Test fit your new LED Fire horns. You can easily adjust the position of the horns by .
Arduino Code

Make sure to download the NeoPixel (https://adafruit.it/aZU) library. Below is fire code that will change the color of the NeoPixel strip/ring - 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 GEMMA/Trinket you need to download the special Adafruit version of the Arduino IDE from the Introduction to GEMMA guide. (https://adafruit.it/dgH)

Code developed by Phillip Burgress (https://adafruit.it/iPc).

```c
// 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,  0,  0,  0,  0,  0,  0,  0,  0,
    1,  1,  1,  1,  1,  1,  1,  1,  1,  2,  2,  2,  2,  2,  2,  2,  2,  2,
    2,  3,  3,  3,  3,  3,  3,  3,  4,  4,  4,  4,  4,  5,  5,  5,  5,  5,
    5,  6,  6,  6,  7,  7,  7,  8,  8,  8,  8,  8,  9,  9,  9, 10, 10, 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.
```
```c
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 = 233 + 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].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();
GEMMA M0 boards can run CircuitPython — a different approach to programming compared to Arduino sketches. In fact, CircuitPython comes factory pre-loaded on GEMMA M0. If you’ve overwritten it with an Arduino sketch, or just want to learn the basics of setting up and using CircuitPython, this is explained in the Adafruit GEMMA M0 guide (https://adafruit.it/z1B).

Below is CircuitPython code that works similarly (though not the same) as the Arduino sketch shown on a prior page. To use this, plug the GEMMA M0 into USB…it should show up on your computer as a small flash drive...then edit the file “code.py” with your text editor of choice. Select and copy the code below and paste it into that file, entirely replacing its contents (don’t mix it in with lingering bits of old code). When you save the file, the code should start running almost immediately (if not, see notes at the bottom of this page).

If GEMMA M0 doesn’t show up as a drive, follow the GEMMA M0 guide link above to prepare the board for CircuitPython.

Temporarily unable to load content:

This code requires the neopixel.py library. A factory-fresh board will have this already installed. If you’ve just reloaded the board with CircuitPython, create the “lib” directory and then download neopixel.py from Github (https://adafruit.it/yew).