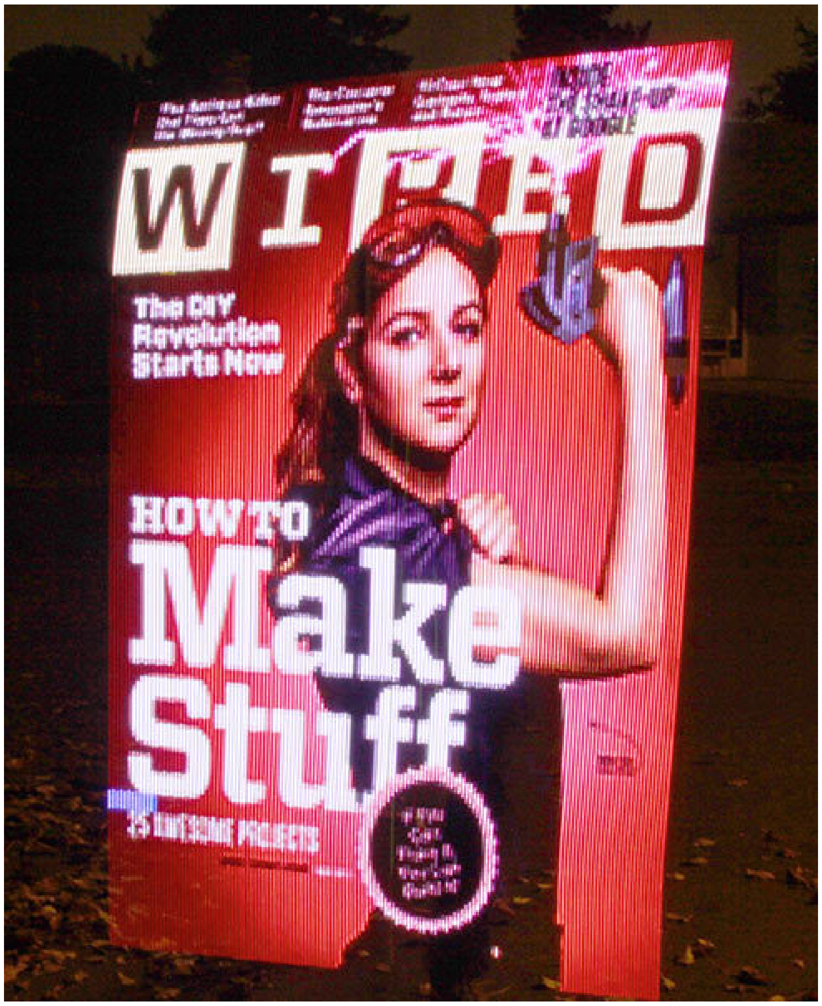
adafruit learning system

NeoPixel Painter

Created by Phillip Burgess



Last updated on 2013-12-10 03:00:37 PM EST

Guide Contents

Overview

Light painting is an artistic medium combining light, motion and long-exposure photography. For as long as a camera's shutter is open, a single point of light in motion will create a continuous streak in the final photograph.

Digital technology takes light painting to the next level...*dozens* of point lights, with color and brightness individually under computer control, weave a swath of awesome across the completed frame.

Adafruit's NeoPixel strips, combined with the Arduino microcontroller and a supporting cast of parts, make highly refined digital light painting achievable!

Things You'll Need:

- **Arduino Uno (http://adafru.it/50)** microcontroller (NOT Mega or Leonardo, see below)
- **Adafruit Data Logging Shield (http://adafru.it/1141)** (or other shield/breakout with SD or microSD card slot)
- **NeoPixel (http://adafru.it/cVO) strip (http://adafru.it/1506), sticks (http://adafru.it/1426) or pixels (http://adafru.it/1312)** (up to 170 pixels maximum — a 1 meter 144 LED strip (http://adafru.it/1506) works great!)
- **UBEC DC-to-DC converter (http://adafru.it/1385)**
- **8xAA battery holder (http://adafru.it/449)** and AA cells (NiMH rechargeable recommended)
- **SD card (http://adafru.it/102)** (or microSD with adapter), FAT-formatted
- **Camera** with a **long-exposure** mode, plus a **tripod**
- **Imaging editing software** that can output **24-bit BMP** files (e.g. Photoshop, GIMP, Pixelmator)
- **Wire:** 20 to 22 gauge or thereabouts, stranded
- **Soldering iron** and related paraphernalia
- **Optional:** JST connectors, power blocks, etc.
- **Support frame.** 3/4" square pine molding works just dandy, or you can get all fancy using

aluminum extrusion if you like.

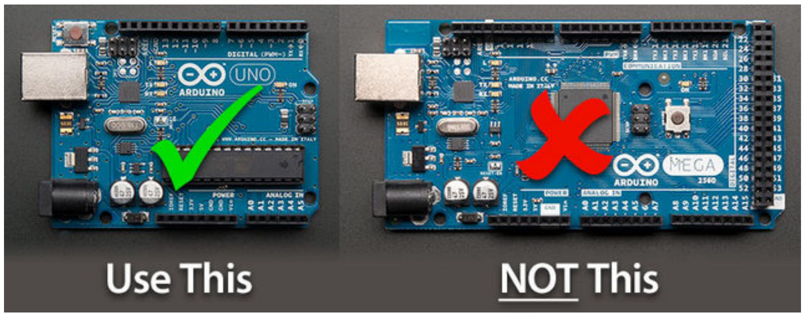
Some additional parts and tools are needed, depending how you put this together. Read through to see how we built ours. **These parts are not available from Adafruit.** You might choose different materials or assembly techniques depending on your particular skill set and items on-hand. *Improvise! Adapt! Overcome!*

Tailored to the Arduino Uno

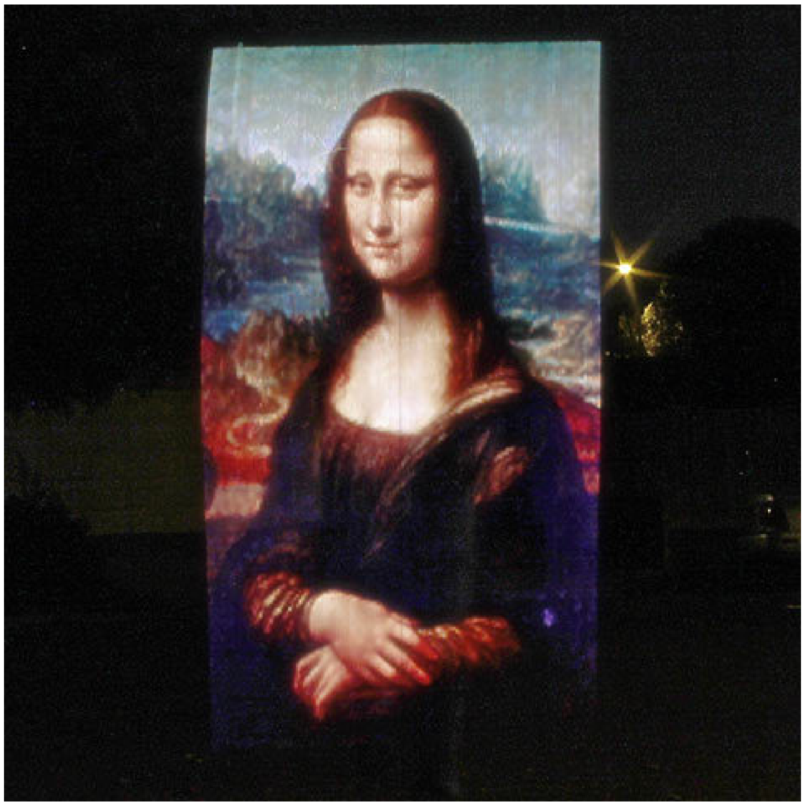
There's a frequent misconception with some projects that using an Arduino Mega will make everything just *that much better.* Some projects *can* benefit, but **this isn't one of them.** The code achieves 100% performance on the Uno; it will not run any faster on the Mega. Much slower, in fact, due to its need to "bitbang" the SPI interface to the SD shield.

The Arduino Leonardo is also not suggested for this project, for similar reasons.

Savvy users could adapt the project for other boards or SD adapters provided there's a fast SPI connection between the two. Also, some AVR-specific timer registers are used, so the code won't compile as-is on other architectures. For most of us, **the Arduino Uno and Adafruit Data Logging Shield offer the most trouble-free approach.**



How about Raspberry Pi then?
We have a different project for that! (http://adafru.it/aPk) It uses another type of LED pixels, better suited to the Raspberry Pi.
<a href="http://learn.adafruit.com/light-painting-with-raspberry-pi">We have a different project for that!</a><span class="pdf-short-link"> (http://adafru.it/aPk)</span> It uses another type of LED pixels, better suited to the Raspberry Pi.



Step 1: Slay Jabberwock

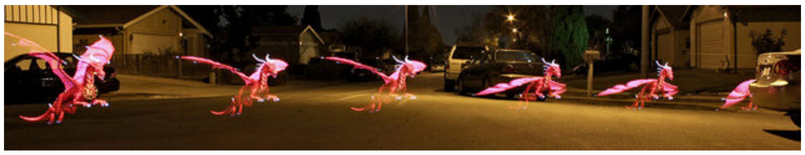
Kidding, there is no Jabberwock!

However, we wanted to get your attention because this is a challenging project. It is really important to follow a couple of rules:

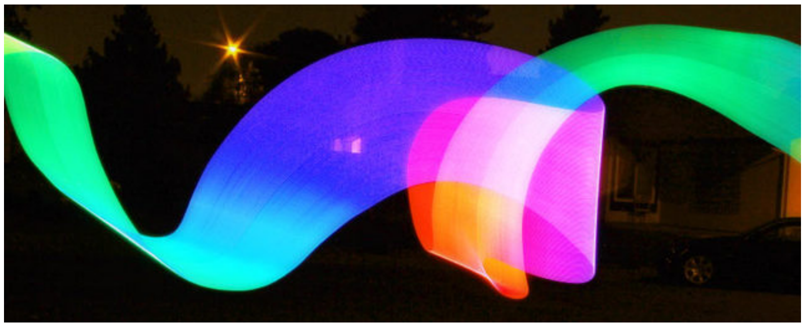
1. Have some Arduino/crafting/hacking experience before this project - this is a great project for someone who is comfortable with wiring, soldering, heat shrink, image conversion, etc. It's not a good first project, there's a lot of expensive components that need careful handywork
2. Read through the entire guide first to get a feel for the parts, tools and skills required. Be

realistic: **decide if this is really for you.** We don’t offer every component needed, and some steps require your own ingenuity to complete. Young makers should read through with a parent to help decide.
3. If you do choose to tackle this, work in the proscribed order, testing subassemblies at each step. **Do not proceed until each test is confirmed working.** If you skip a step, or assume "it'll just work later", troubleshooting is then incredibly difficult — you’ll probably have to tear it back down to testable subassemblies anyway. That’s no fun. So get everything working step-by-step first!

That said, this is a fun project that will take a weekend or two to complete and it's by far the easiest and least-expensive way to make a high-resolution light painting rig. For hackers, its very easy to customize if you want to add sensors or other technologies!



Download Software



Several software components are required to make the system work. Although we'll be building and testing one subsection at a time, let's get all of the software downloads out of the way now.

If running an older version of the Arduino IDE software (prior to 1.0.4, or anything in the 0023 or earlier series), this would be a good time to upgrade (http://adafru.it/aHs). If you don't know what an "Arduino IDE" is, this project may be too much to tackle right now, consider starting with the Learn Arduino (http://adafru.it/c51) tutorials. (Come back in a few months once you've flexed your 'duino-muscles!)

Two libraries are required. First is the Adafruit NeoPixel library (http://adafru.it/aZU), if you don't already have it installed from prior projects:

Click to download NeoPixel library
http://adafru.it/cDj

The SdFat library (http://adafru.it/cYc) is for reading and writing SD cards. The Arduino IDE already includes a library for SD cards (built atop an earlier version of SdFat), but this latest release provides much better performance and some extra features that we need:

Click to download SdFat library
http://adafru.it/cY3

And finally, the Arduino sketch that does the light painting:

Click to download NeoPixel Painter sketch http://adafru.it/cY5

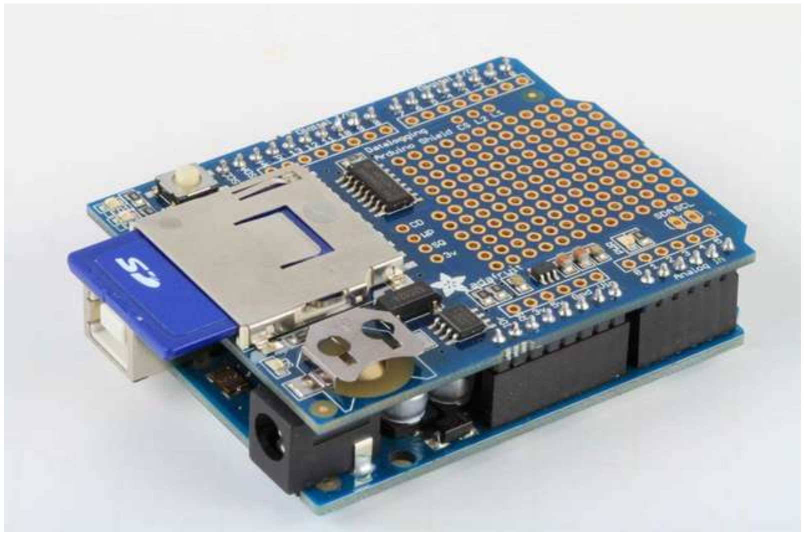
Installing Arduino libraries is a frequent stumbling block. If this is your first time, or simply needing a refresher, please read the All About Arduino Libraries (http://adafru.it/aYM) tutorial.

The sketch and both libraries will need to be uncompressed, renamed and relocated, and the Arduino IDE restarted so the new pieces register. However…

DO NOT LOAD THE NEOPIXEL PAINTER SKETCH YET!

One must resist the temptation to get into the light-painting code right away. It is **vitally important** that we first **test each piece of the system in isolation**, as there are multiple potential points of failure and diagnosing a fully-assembled rig is a massive headache (don't ask us how we know! ;-)

Prepare SD Shield & Card

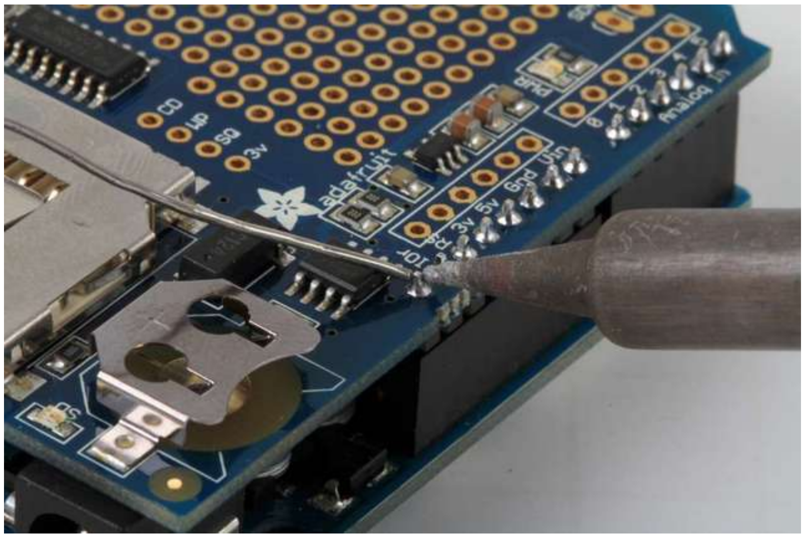


We chose the Adafruit Assembled Data Logging Shield (http://adafru.it/1141) for this project. It's the most affordable and trouble-free way to add an SD card reader to an Arduino, and the combined board stack is slim enough to fit inside a popular mint tin.

There are other shields with SD (or microSD) card slots in addition to other features. Most can work just fine as a card reader for this project. Some may have a display or buttons, but our example software doesn't support these (nor will thicker shields fit inside the mint tin).

With a different enclosure and some custom code, you might be able to add a file selection user interface…or you might not. Memory is exceedingly tight, and we can't really say whether adding a UI is even practical…it's something we've not explored yet. For now it's very basic.

Breakout boards for SD/microSD could also work, but we recommend using the shield…the prototyping area provides a solid point of contact for wires and parts we'll be adding later.



Solder the included male pin headers to the shield following the directions in the Adafruit Data Logger Shield guide (http://adafru.it/cWw). Stacking headers are not recommended for this project; they won't fit inside the mint tin.

Notice how the solder joints in the photo above are shiny, smooth and concave. The Adafruit Guide to Excellent Soldering (http://adafru.it/c6b) illustrates proper soldering technique. "Cold" solder joints (cloudy and balled up on the surface, not flowing smoothly between pin and board) will cause the shield to work unreliably or not at all.

Installing the battery on the shield is optional. This project doesn't require use of the realtime clock, so it's okay to leave it out. If you're planning to dismantle and use the shield in other future projects, or have an idea how you might use the clock in your light painting, go ahead and install the battery. It'll last for years.

Plug the shield into an Arduino Uno, then connect a USB cable between the board and your computer. The green PWR led on the shield should light up. If it does not (and especially if the computer complains about a USB device drawing too much power), there's probably a solder bridge between pins. Unplug USB, remove the shield and look it over for any soldering mistakes.

Card Tricks

It's a good idea to **designate a "scratch" card (or several) for this project.** Do not use an important card with irreplaceable family photos on it! Our software reads and writes to the card, and perhaps there are bugs we haven't spotted…there's a small chance here of data being overwritten.

This task does not require massive storage. It's a great opportunity to recycle those "tiny" old SD cards cluttering a drawer. You don't need to buy anything exotic or high-end.

Plug the card into a USB SD card reader (or SD slot if your computer has one) and format it as a

FAT filesystem. This has caused some trouble in the past, but more recent releases of both Windows and Mac OS X produce a card that the Arduino can read. If you encounter issues in the next steps, try formatting the card in a digital camera, or download the SD Association's official SD card formatter application (http://adafru.it/cfL).

Toss a few small files or folders on the card for testing the next step, then eject the card.

Insert the card into the SD socket on the shield, then connect a USB cable between the Arduino and computer.

Launch the Arduino IDE. From the **File** menu, select: **File→Examples→SD→CardInfo**

This example is part of the standard Arduino SD library; we'll get into SdFatlib later.

You will need to make a small change to this code. Locate this line:

```
const int chipSelect = 4;
```

and change it to:

```
const int chipSelect = 10;
```

From the **Tools** menu, select **Tools→Board→Arduino Uno**

Also from the **Tools** menu, select the **Serial Port** corresponding to the Arduino USB connection, then click the **Upload** button (the circle with a right-facing arrow). In a moment, it should report "Done uploading," and the red LED on the SD shield will light briefly.

Now open the **Serial Monitor** (the magnifying glass icon at the top-right of the Arduino editor window) and set the baud rate to **9600**.

If this succeeds, you'll see a listing of all the files and folders on the card (if any — but at the very least, it will report the card type and size).

If this reports an error, it's usually one of three problems:

- The chipSelect pin number is not set correctly in the software (should be 10).
- The SD card format is not strictly following the specification. If this was formatted in your computer, try formatting the card in a digital camera instead, or use the aforementioned SD formatter application.
- A soldering problem with the shield. If you post a clear photo in the Adafruit Customer Support Forums (http://adafru.it/cer), we'll look it over for any gremlins.

Do not proceed until you have the CardInfo sketch working, displaying the contents of the SD card. You need to have the SD card fully working before you can store any images on it!

Test NeoPixel Strip

Let’s confirm the NeoPixel strip is **100% working** before moving on.

Unplug the Arduino and remove the Data Logging Shield. We’ll return to that later, but for now we need access to the Arduino headers.

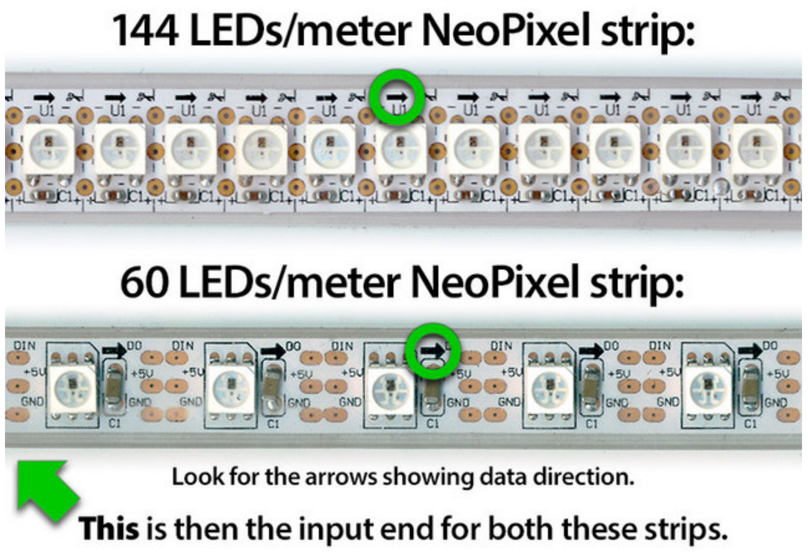
The NeoPixel library should already be installed from a prior step.

Know Your NeoPixels

IMPORTANT: pre-installed wires (if any) on the end of a NeoPixel strip don’t necessarily indicate the “input” end.

Look closely at the face of the strip for small arrows showing the data direction. It’s a little easier with 60 LEDs/meter strip, which adds “DIN” or “DI” (data in) and “DO” (data out) labels next to solder pads. The Arduino will connect to the “in” end, where the arrows are originating.

144 LEDs/meter strips always arrive in 1-meter lengths with wires pre-installed at both ends. 60 LEDs/meter strip is manufactured in longer 4-meter reels (also with wires at both ends), which are then cut when a smaller order is placed. So you might receive a strip with no wires, or there’s a 50/50 chance that the wires are really the output end. So you may have to solder on your own.


144 LEDs/meter NeoPixel strip:
60 LEDs/meter NeoPixel strip:
DIN
DO
+5V
GND
Look for the arrows showing data direction.
This is then the input end for both these strips.

These photos show white NeoPixel strip — it’s easier to read in photographs — but you can use either the black- or white-backed type.

Running the Test

DO NOT use the NeoPixel "strandtest" example - it lights up all the LEDs which draws tons of power and could make the Arduino crash due to a 'brown out'. Instead, copy and paste the following code into a new Arduino sketch:

```
// Simple NeoPixel test.  Lights just a few pixels at a time so a
// 1m strip can safely be powered from Arduino 5V pin.  Arduino
// may nonetheless hiccup when LEDs are first connected and not
// accept code.  So upload code first, unplug USB, connect pixels
// to GND FIRST, then +5V and digital pin 6, then re-plug USB.
// A working strip will show a few pixels moving down the line,
// cycling between red, green and blue.  If you get no response,
// might be connected to wrong end of strip (the end wires, if
// any, are no indication -- look instead for the data direction
// arrows printed on the strip).

#include <Adafruit_NeoPixel.h>

#define PIN      6
#define N_LEDS 144

Adafruit_NeoPixel strip = Adafruit_NeoPixel(N_LEDS, PIN, NEO_GRB + NEO_KHZ800);

void setup() {
  strip.begin();
}

void loop() {
  chase(strip.Color(255, 0, 0)); // Red
  chase(strip.Color(0, 255, 0)); // Green
  chase(strip.Color(0, 0, 255)); // Blue
}

static void chase(uint32_t c) {
  for(uint16_t i=0; i<strip.numPixels()+4; i++) {
      strip.setPixelColor(i  , c); // Draw new pixel
      strip.setPixelColor(i-4, 0); // Erase pixel a few steps back
      strip.show();
      delay(25);
  }
}
```

Edit the N_LEDS value to reflect the actual length of your NeoPixel strip.

Upload this code to the Arduino Uno, disconnect the USB cable, then we'll connect the strip.

A wiring diagram is not provided…this is on purpose! Different types of LED strips have their connections in a different order, and the manufacturer sometimes makes changes to the design. Impulsively following a picture (rather than reading a description) increases the chance of an improper hookup and a damaged strip!

Make the following three connections:

- **GND** from Arduino to **GND** or **–** on strip (always connect GND first)
- **5V** from Arduino to **+5V** or **+** on strip
- **Pin 6** from Arduino to **DIN** (or **unmarked** input) on strip

Lay the strip out flat so you can see the entire thing, then re-connect the USB cable. You should see a few LEDs chasing down the length of the strip, cycling between red, green and blue. Watch carefully, noting any skipped or off-color pixels.

Nothing lights up!

1. If the computer reports a USB device is drawing too much power, unplug the Arduino immediately.
2. If there are extra wires at either end of the strip, make sure the tips are not touching each other or anything conductive.
3. Confirm the three connections between the strip and Arduino: GND, +5V and pin 6.
4. If you soldered wires on, make sure there's no cold joints or solder bridges between adjacent pads.
5. Make sure you're connected to the INPUT end of the strip.
6. Check the USB cable is properly seated between the Arduino and computer or powered USB hub.

If you have a multimeter, check the voltage across +5V and GND at the OUTPUT end of the strip. It should be around 5 Volts.

<ol> <li>If the computer reports a USB device is drawing too much power, unplug the Arduino immediately.</li> <li>If there are extra wires at either end of the strip, make sure the tips are not touching each other or anything conductive.</li> <li>Confirm the three connections between the strip and Arduino: GND, +5V and pin 6.<br> </li> <li>If you soldered wires on, make sure there's no cold joints or solder bridges between adjacent pads.</li> <li>Make sure you're connected to the INPUT end of the strip.<br> </li> <li>Check the USB cable is properly seated between the Arduino and computer or powered USB hub.<br> </li> </ol>If you have a multimeter, check the voltage across +5V and GND at the OUTPUT end of the strip. It should be around 5 Volts.<br>

The lights cut out part way down the strip.

Confirm the value of N_LEDS in the code matches the actual NeoPixel strip length.

Confirm the value of <span class="editor-monospace">N_LEDS</span> in the code matches the actual NeoPixel strip length.

One or more pixels won't light up, or show the wrong color.

Possibly defective pixel(s). Read on…

Possibly defective pixel(s). Read on…

If you encounter any of these problems (or others), search the Adafruit Customer Support Forums (http://adafru.it/cer) for similar issues and their resolutions. If your situation is not addressed, make a new post with a description of the problem and (if possible) a clear photo showing the wiring between Arduino and NeoPixels. We'll help troubleshoot the problem or have a replacement sent if needed.

Can I use more (or fewer) than 144 pixels?

Certainly! But there's an upper limit of **170 pixels.** This is the maximum that can be recorded in one SD card block (512 bytes). Going beyond this would require reading two (or more) blocks and would halve the speed of the entire system…you'd have to move at a snail's pace when taking photos.

You're welcome to hack it up and try, but the code will need significant changes, and there's no guarantee the Arduino even has enough free RAM to handle this.

<b>Certainly!</b> But there’s an upper limit of <b>170 pixels.</b> This is the maximum that can be recorded in one SD card block (512 bytes). Going beyond this would require reading two (or more) blocks and would halve the speed of the entire system…you’d have to move at a snail’s pace when taking photos.<br><br>You’re welcome to hack it up and try, but the code will need significant changes, and there’s no guarantee the Arduino even has enough free RAM to handle this.<br>

Do not proceed until you have a fully tested and working NeoPixel strip. Once the strip is glued to the painting wand, it will be really really hard to fix any wiring problems

Test Power

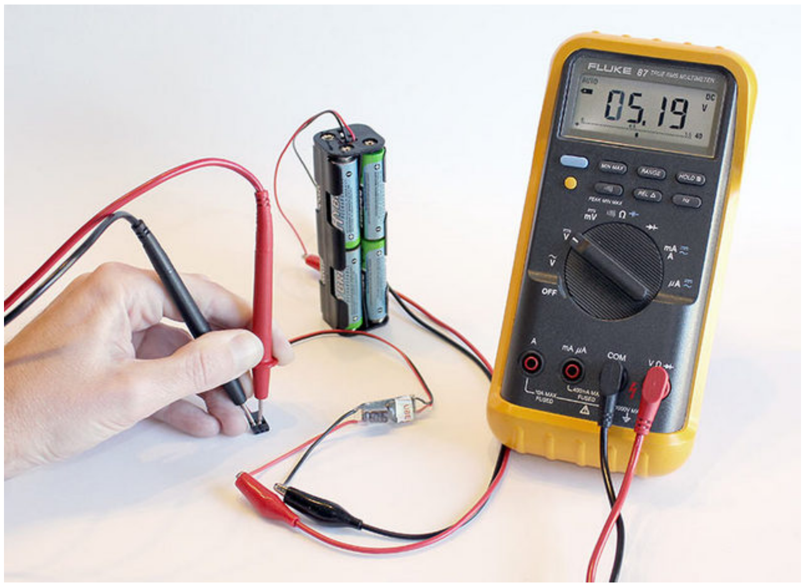
Next up, let’s test the battery and UBEC (http://adafru.it/1385) (voltage converter).

Our rig is powered off 8 AA cells. We think this is the best choice: compact, lightweight, and the cells are fairly inexpensive (even adding a charger) and can be used in other projects and toys. But if you already have a 6 to 23 Volt battery pack from some other hobby (such as RC cars), this can be used just as well.

With NiMH rechargeables, the 8-cell pack provides about 9.6 Volts, while single-use alkalines give about 12V. Either way, the UBEC converts this to 5 Volts DC. The rechargeables are strongly recommended…not just for their reusability, but that they can deliver more current. With alkaline cells, the NeoPixels may go dim or pink on particularly bright images as the battery can’t push enough current.

Connect the battery holder (red+, black–) to the short, thick wires on the UBEC using alligator clips (if you don’t have clips, the wires can be soldered temporarily). Charge the cells and load up the battery holder. If you have a multimeter, test the output voltage (the long, thin wires from the UBEC). It should be around 5 Volts, give or take a little…possibly a little higher, since the circuit isn’t under load yet.

If you don’t have a multimeter, a basic pass/fail test can be done with an LED and resistor (150 to 500 Ohm) in series. You may need to get creative with whatever combination of jumper wires, breadboard and/or alligator clips you have on-hand. Remember that LEDs have a specific polarity — the longer leg should connect to the red (+) wire.



The UBEC can deliver about 3 Amps of continuous current, or a little more for very short intervals. A single NeoPixel can draw up to 60 milliamps…that’s not much, but with many pixels, it quickly adds up. 144 NeoPixels (1 meter of our high-density strip) can potentially demand over *8 Amps!* This far exceeds the UBEC’s capabilities, so the software limits the brightness to a reasonable power limit.

There’s a setting in the code for using a more powerful UBEC for extra brightness (RC hobby shops may have these). But really, the 3A UBEC is still plenty bright for most things, you're not missing out.

I’m reading 0V out from the UBEC (or the LED test isn’t working)

- Remove the cells immediately. Check for shorts on both the input and output wires.
- Are the alligator clips or bare wire ends touching?
- Confirm all cells are installed in the correct orientation.
- Confirm all cells are good using a multimeter or battery tester. Even a single bum cell can interfere with the circuit.

If it’s still not working, refer to the Adafruit Customer Support Forums (http://adafru.it/cer) for assistance. If possible, post a photo clearly showing the parts and connections. We’ll arrange for a replacement if the battery holder or UBEC is defective.

<ul> <li>Remove the cells immediately. Check for shorts on both the input and output wires.<br> </li> <li>Are the alligator clips or bare wire ends touching?<br> </li> <li>Confirm all cells are installed in the correct orientation.<br> </li> <li>Confirm all cells are good using a multimeter or battery tester. Even a single bum cell can interfere with the circuit.<br> </li> </ul>If it’s still not working, refer to the <a href="http://forums.adafruit.com">Adafruit Customer Support Forums</a><span class="pdf-short-link"> (http://adafru.it/cer)</span> for assistance. If possible, post a photo clearly showing the parts and connections. We’ll arrange for a replacement if the battery holder or UBEC is defective.<br>

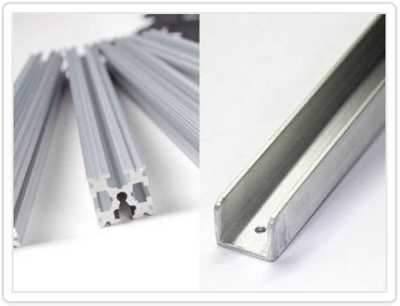
At this point, the Arduino, SD shield and card, NeoPixel strip(s) and power source are all separately tested and confirmed working. Do not proceed until this is the case. Troubleshooting a defective part in an assembled rig is difficult to impossible - especially once it's all glued in place

Build Frame

A frame of some sort holds your “light paintbrush” together, providing a base onto which components can be firmly attached.

Thus begins the *Choose Your Own Adventure* part of the project. There is no One Right Way to do this. Your access to and experience with different materials and tools might be entirely different than what’s shown here, so consider these merely guidelines, not a blueprint.

Think About Materials



There exist some really nifty metals out there — including aluminum extrusion and U-channel — but these are relatively costly and require special tools and skills to cut and join. If you’ve got the chops and the budget, go wild.

For most folks, **wood is probably a better choice.** Inexpensive, easy to acquire, easy to work with.

This **3/4" pine molding** is the perfect width for the NeoPixel strip. It typically comes in 8 foot lengths, which can be cut as needed.

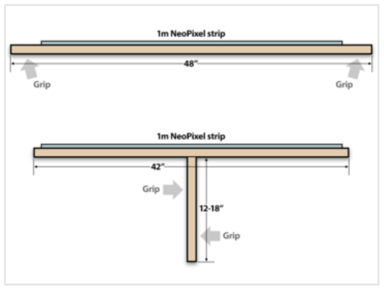
Unfortunately, the dowels in the “hobby wood” section of most hardware stores are typically 36 inches long…just a bit too short to support a full meter of NeoPixel strip. So you’ll probably need to buy a full 8' length of molding. Many stores will provide one or two cuts for free.

If you must buy your wood from a big home improvement store, sight down the length of the boards and try to find one that’s relatively straight. A local independent lumberyard or hardware store will usually have better quality stock in this regard.

PVC pipe might seem an alluring option, but it’s not recommended. It lacks rigidity, and the round cross-section is difficult to attach the NeoPixel strip to.

Think About Shape

How big will your paintbrush be? In what ways do you envision moving it around?

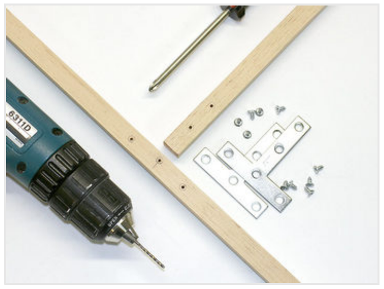


The simplest support frame is just a straight bar a bit longer than the strip itself. This provides a grip at one or both ends…it could be lifted like a barbell, or brandished like a light saber.

A "T" configuration puts the grip at the center, providing different options (such as spinning).

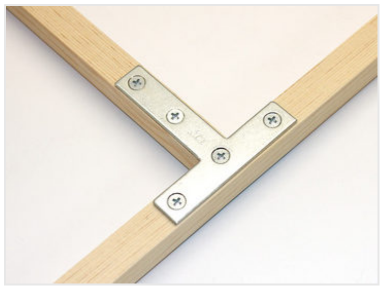
Or you could mix and match. A "T" with a longer bar for multiple grip options.

The bar should be a <u>minimum</u> of 42 inches long…that's just a little longer than the 1 meter strip with the end caps.



If you've opted for wood and a "T" configuration, the two pieces can be held together with a pair of 3" zinc tee plates and ten 3/8" wood screws.

Mark and drill some small pilot holes, and screw the frame together…



Super easy!

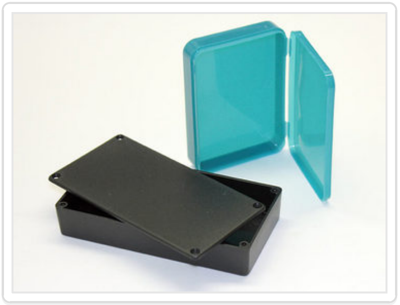
Or, to make a frame that can be dismantled, drill all the way through the wood and substitute bolts and wing nuts.

Think About Stray Light

There are a few LEDs on the Arduino board and shield that could produce undesirable light streaks in your paintings, so consider some kind of enclosure.

We used an Altoids-sized mint tin (http://adafru.it/97) for ours…it's almost exactly the same size as the Arduino and shield.

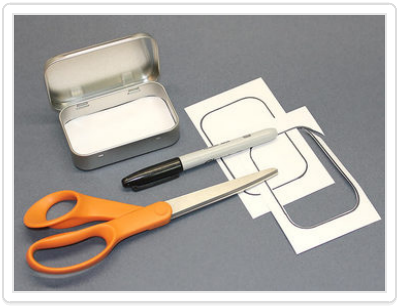
If you prefer plastic, there's a nice weatherproof box in the shop that is easier to work with (http://adafru.it/903) - you can drill and cut it with basic hand-tools.



Some sections of the enclosure need to be cut away. Metal can be difficult in this regard, and there's the risk of cuts from sharp edges.

If you prefer plastic, there's a nice weatherproof box in the shop that is easier to work with (http://adafru.it/903) - you can drill and cut it with basic hand-tools.

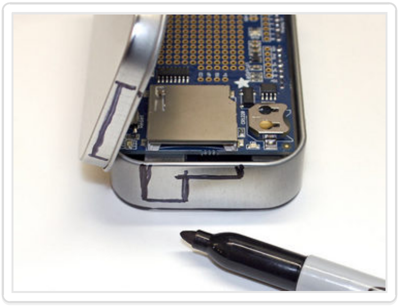
As an alternative, you might track down a plastic box that's big enough for the Arduino and shield. Even a small paper box (like playing cards come in) may suffice. Look around you!



If you do use a metal tin, devise a scheme to insulate the electronics inside. This needs to be done on the inside top and bottom.

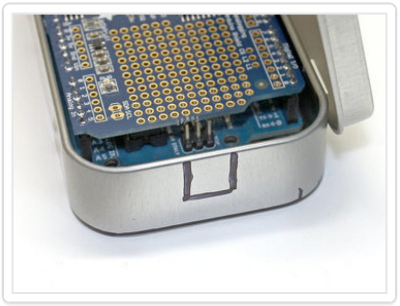
Here the footprint of the tin is traced onto index cards and cut out to produce a liner. Later this will be glued or taped in place.

Vinyl contact paper is another option, if you have some in your craft stash.



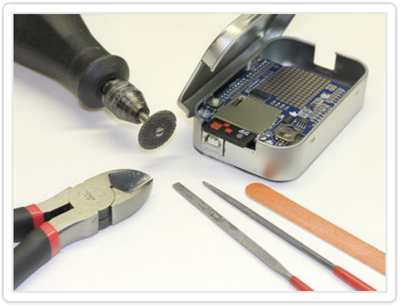
Cutouts must be made for the USB port and SD card socket. Using the Arduino and shield for size and position reference, outline the planned cuts with a permanent marker.

Notice with this tin that the SD slot needs cutouts both in the base and the hinged lid.



Add a single cutout for wires at the opposite end of the enclosure as well.

Also, mark two spots on the bottom where screw holes can be drilled. These will hold the enclosure to the frame.

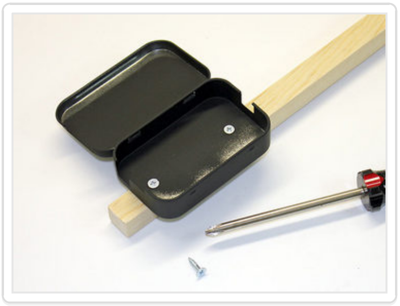


How you cut the tin will depend on your tools on hand. Dremel? Metal snips? Files?

Watch out for sharp edges, and wear eye protection when using tools!

After cutting, metal edges can be smoothed with sandpaper or an emery board (nail file).

Wash out the case thoroughly after doing this, and allow it time to dry. Any lingering metal shavings or dust will wreak havoc with the electronics!



The enclosure is mounted near one end of the bar in order to **minimize the wire lengths between the Arduino and NeoPixel strip**. Attach it to the face of the wood that's opposite where the LED strip will go. Later, we'll position the batteries to provide a counterweight.

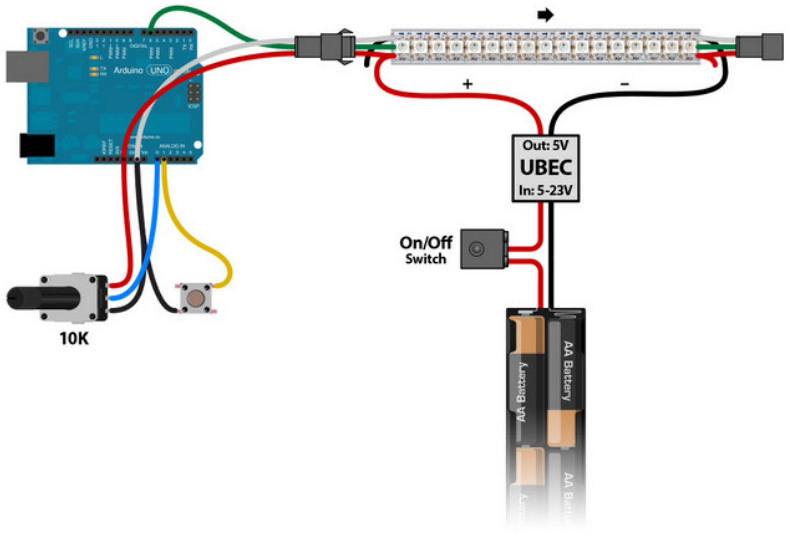
The tin shown here was painted matte black to reduce reflected light from the environment. Not a crucial step, but go ahead if you already have some paint on hand.

The insulating layers that were cut earlier (if needed) can be installed in the case now, using glue or double-stick tape.

Plan Wiring

20 to 22 gauge stranded wire is recommended for this project, as its more flexible than solid-core wire. Color-coding the wires by function (for example: red for +, black for –, green for signal) is optional but helps prevent mishaps.

Here's a simplified view of the planned wiring. The actual setup will include the Data Logging Shield, using the prototyping area for some of the connections, and perhaps a Perma-Proto board for the controls.

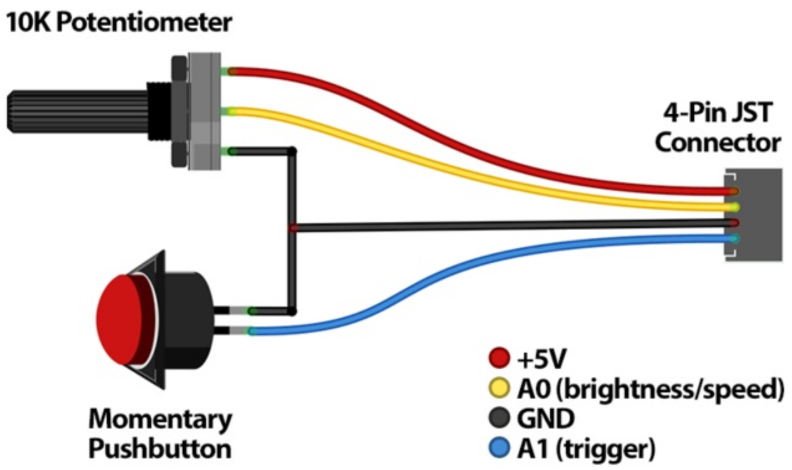


You might notice a couple of unusual things here:

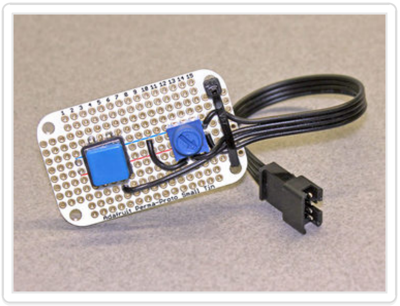
- + and – are connected to opposite ends of the NeoPixel strip (using the extra end wires). This is a little trick to ensure more uniform color and brightness along its length. When a long strip is powered from one end, the furthest pixels may be dim or tinted red.
- The Arduino draws power off the NeoPixel strip; there's no split in the UBEC output. This saves some wiring, since the Arduino needs to connect to the input end of the strip anyway.
- There's a 3-pin JST connector between the Arduino and NeoPixel strip. We don't currently offer these for sale…you'll either need to find one from a parts supplier such as Digikey or Mouser, or — once you've tested the NeoPixel strip and are 100% confident in its operation — you can cut the plug off the output end and use that for the Arduino side of the connection (you may need to splice in a little extra length to the wires).

Connected to the Arduino is a minimal "user interface" of one dial and one button. The dial sets the image brightness at startup, and then the painting speed after that. The button triggers playback of the image (repeatedly if held down).

These components could be installed directly on the prototyping area of the shield, provided cutouts are made in the enclosure. To offer more flexibility in how the paintbrush is used, we instead mounted these on a Perma-Proto board and ran wires. A 4-pin JST connector (a type we do carry (http://adafru.it/578)) lets us swap out the control board for specialized variants.



5 Volts connects to one outer leg of a 10K potentiometer. Ground connects to the opposite outer leg, and one side of the trigger button. The center tap of the potentiometer connects to Arduino analog pin #0, while the opposite pin of the trigger button connects to analog pin #1. The JST connector is optional, if you'd prefer just to hard-wire everything in.

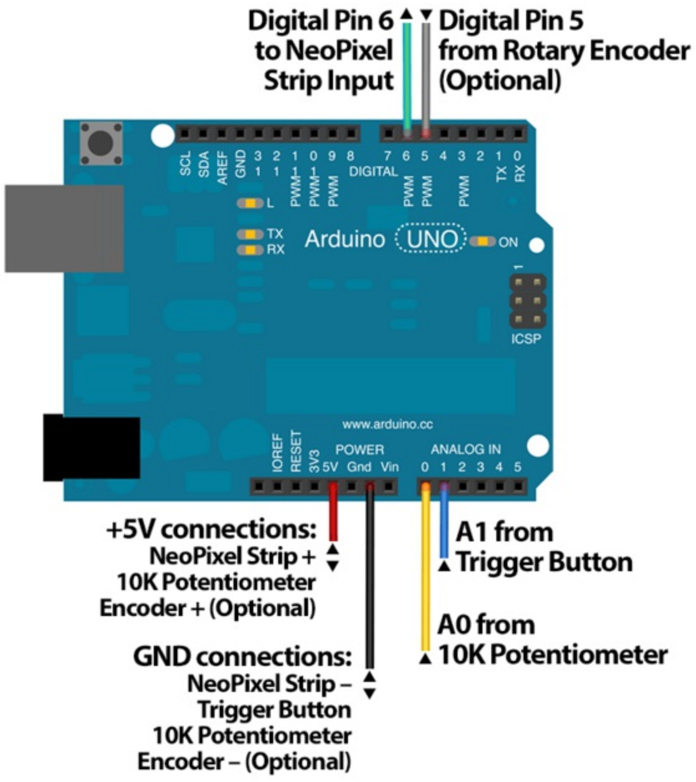


Here's the dial and trigger on a "Small Tin" Perma-Proto board. The short wires require this to be mounted within a few inches of the Arduino.

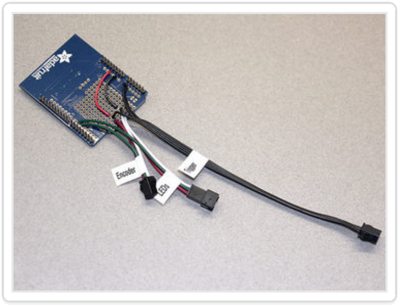
Here's a different one with a long (about 4 feet) ribbon cable attached. The paintbrush was strapped to the rear rack of a bicycle, while the controls are accessible from the handlebar. Packing tape on the underside prevents contact between the electrical components and any metal bike parts.

Here's a clearer diagram of where wires will go on the Arduino. **The actual connections will be made on the SD shield**; this is just a pictorial reference of the circuit.

The rotary encoder mentioned here is a complex and pricey option. Not for everyone, its explained on the last page of the guide. You can leave this out, or add a vestigial 3-pin JST connector to add this feature later.



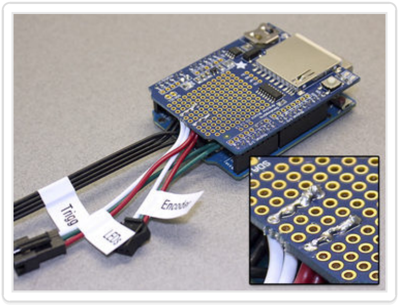
Digital Pin 6 to NeoPixel Strip Input
Digital Pin 5 from Rotary Encoder (Optional)
+5V connections: NeoPixel Strip + 10K Potentiometer Encoder + (Optional)
GND connections: NeoPixel Strip – Trigger Button 10K Potentiometer Encoder – (Optional)
A1 from Trigger Button
A0 from 10K Potentiometer
Arduino UNO



Here’s a Data Logging Shield with wires attached.

To better fit the narrow clearance under the lid of the mint tin, the wires were routed on the underside of the shield rather than the top.

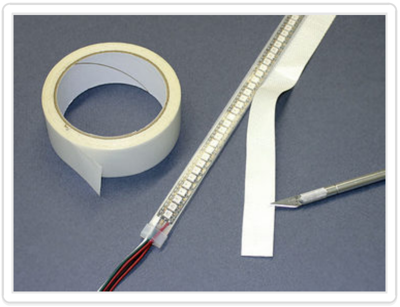
There’s a 3-pin JST connector for the LEDs, a 4-pin JST connector for the control board, and an optional second 3-pin JST for a rotary encoder.



+5V and GND need to connect to multiple points. Several adjacent pads in the prototyping area provide makeshift power rails.

Install Electronics

Attaching the LED strip to the frame can be a challenge. There are very few things in this world that will adhere to the silicone weatherproof sleeve that protects the NeoPixel strip.



Double-stick carpet tape provides a reasonable hold. This does not provide a strong or permanent bond to the silicone sleeve; it's more like a Post-It Note.

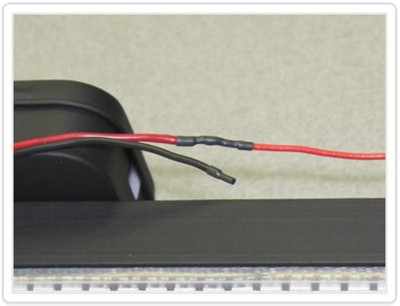
The tape is a lot wider than the strip, and has a peel-away backing. Cut a piece that's half as long as the strip, align one edge while pressing the tape to the strip, then trim away the extra with an X-Acto knife. The trimmed section can then be similarly applied to the second half of the strip and the remaining extra width trimmed and discarded.

To apply this to the frame: peel away a few inches of the tape backing from the center of the strip, leaving the "tails" hanging off to one side. Center the strip on the frame and press into place. Then work from the center outward a few inches at a time, pulling at the tails and pressing the strip into place.

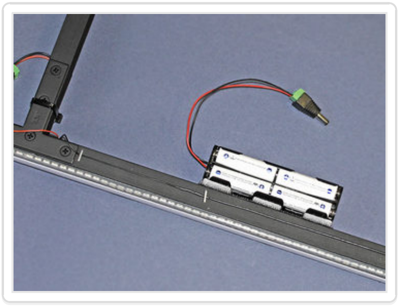
For a more permanent installation, the only glue we've found that sticks to the waterproof covering is **Permatex 66B Clear RTV Silicone Adhesive** (even other silicones refused to work).

If you go this route, lie the LED strip face down, run a bead or a series of dots along the back of the strip, then place the frame on top of this and set some weight on it for a few hours (a couple heavy books should suffice). Do not use anything like A-clamps…these squeeze very tightly and could damage the NeoPixel strip!

A staple gun can come in handy for securing wires in a few places, but be extremely careful with your aim! Just a slight misalignment could punch a conductive metal staple right through a signal- or current-carrying wire.

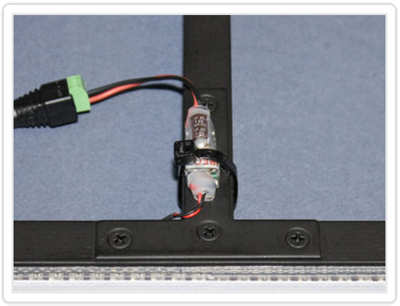


When joining the UBEC output to the NeoPixel power wires, remember that just one wire at each end is used (doesn't matter which end is + or –). Clip off the exposed tip of the unused wire or cover it with heat-shrink insulation to avoid mishaps.



To keep the data wire short, the Arduino was mounted near one end of the bar. The battery pack can be mounted on the opposing side, a bit closer to the center so the two sides are balanced.

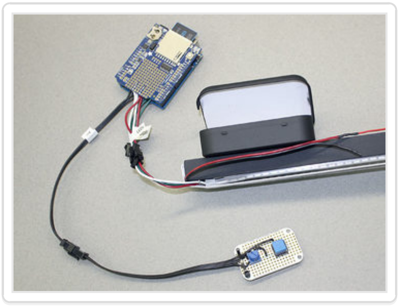
Self-stick hook-and-loop (Velcro) fasteners were used so the pack can be removed for easier battery access.



The UBEC can be held down somewhere with a cable tie, hot glue, double-stick foam tape or Sugru (http://adafru.it/436). Whatever works for your situation.

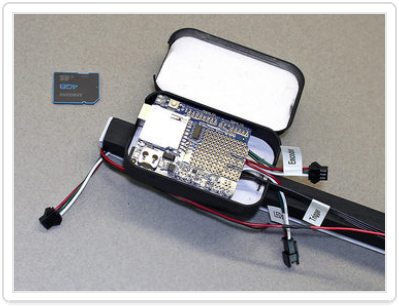
In the photos above we're using female (http://adafru.it/368) and male (http://adafru.it/369) power adapters. An in-line power switch (http://adafru.it/1125) can then be added without soldering, held in place with cable ties, Sugru, etc.

For a slimmer profile, our tactile on/off switch (http://adafru.it/1092) can be soldered between the + terminals on the battery pack and UBEC.

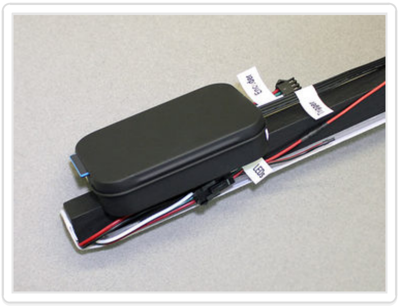


Once all the connections are made, you may want to do a last preflight check of all the parts before final assembly.

Load up the SD card with an image (as explained in the "Preparing Images" section), set the dial to a middle position, power it up and see if it works. You should see the red "SD" led flicker for a few seconds while the software initializes, then tapping the button should produce a light show on the NeoPixel strip.



If everything checks out, turn off the power and install the Arduino and shield inside the mint tin. Double-stick foam tape works well, or this might be another Sugru moment.



Close it up and you're good to go!

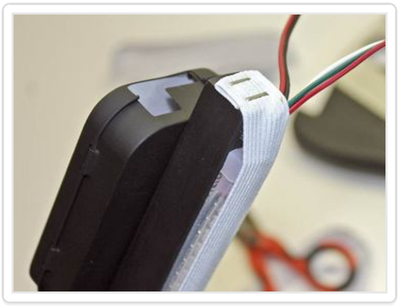
Finishing Touches

A little bit of light diffusion in front of the NeoPixel strip makes it photograph better. This blends adjacent pixels, helping remove the dark striations between the tiny focused points of light.

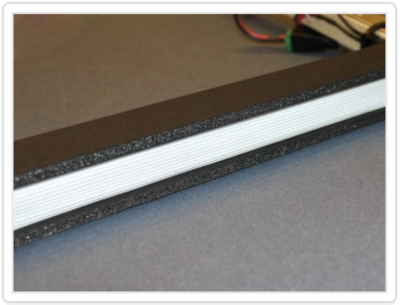
Hardware stores were scoured in search of just the right diffuser material that could be bought ready-made or didn't require too much extra work or precise cutting. Acrylic? Vinyl? Something higher tech and laser-cut? As it turns out, the easiest solution wasn't in the hardware store at all, but over at the craft store…

3/4 inch white elastic! It's already the right width, and it's super easy to cut and attach.

You don't even need a whole roll like this. It'll be stretched out, so a piece about 2/3 the length of the bar will suffice.

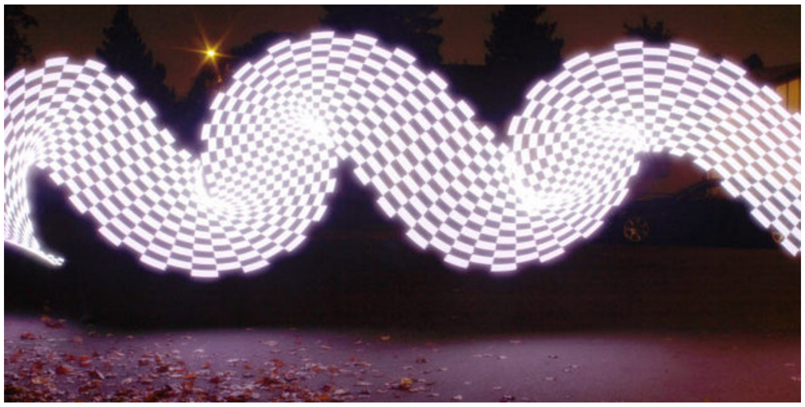


The ends of the elastic are folded over to prevent fraying, then stapled to the ends of the bar.

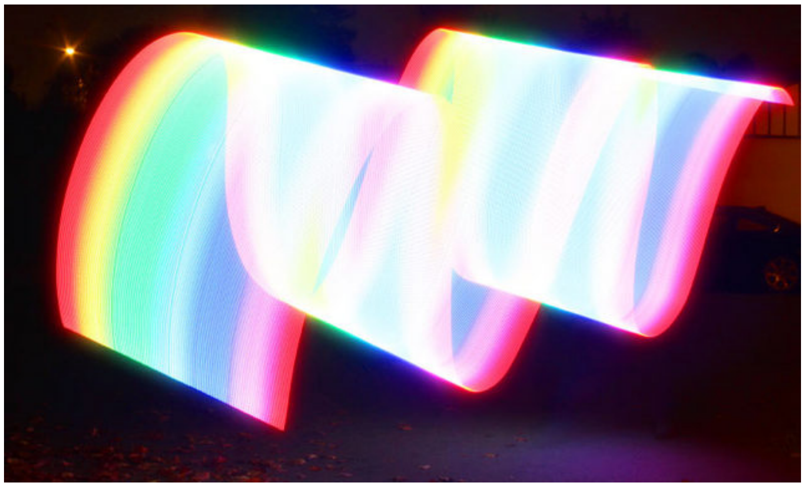


Optional: for better control over the light direction, create a channel to prevent stray light out the sides.

A few 1" strips were cut from black foam core board (mat board or illustration board can work as well) then glued to the sides of the frame to create a shallow channel. It ain't rocket surgery!



Preparing Images



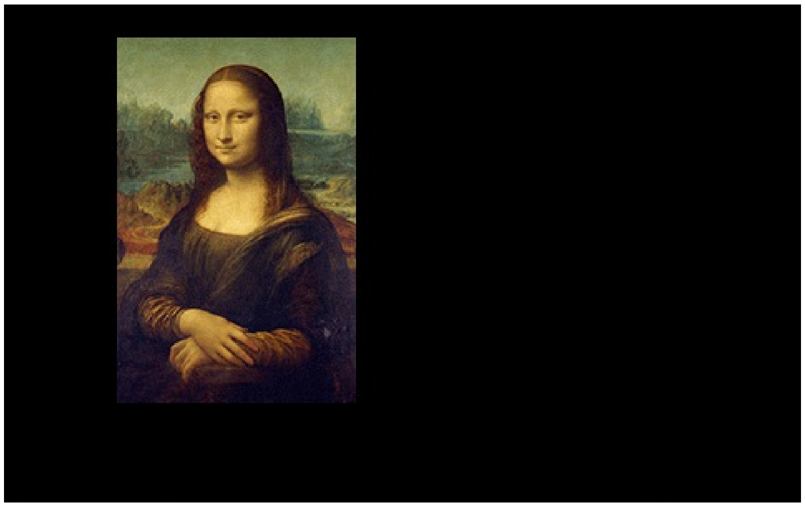
Images will require some resizing and conversion in preparation for the NeoPixel Painter code. The software can only read one specific format — **24-bit BMP** — because the Arduino lacks the sauce to decode complex formats like JPEG or PNG.

The BMP image format is sometimes called “Microsoft Bitmap” or “Windows Bitmap,” but there’s really nothing Windows-specific about it; plenty of software on Mac and Linux handles the format just as well, typically an option in a “Save As…” or “Export…” dialog box. Software like The GIMP, Pixelmator or Photoshop can export this format.

Be sure to select **24-bit BMP**. Not 1- or 8- or 32-bit. The Arduino code only handles the 24-bit variety. Its OK if your image is not 24-bit color before saving, or monochrome. Its just that the file format is much easier to handle if its saved this way.

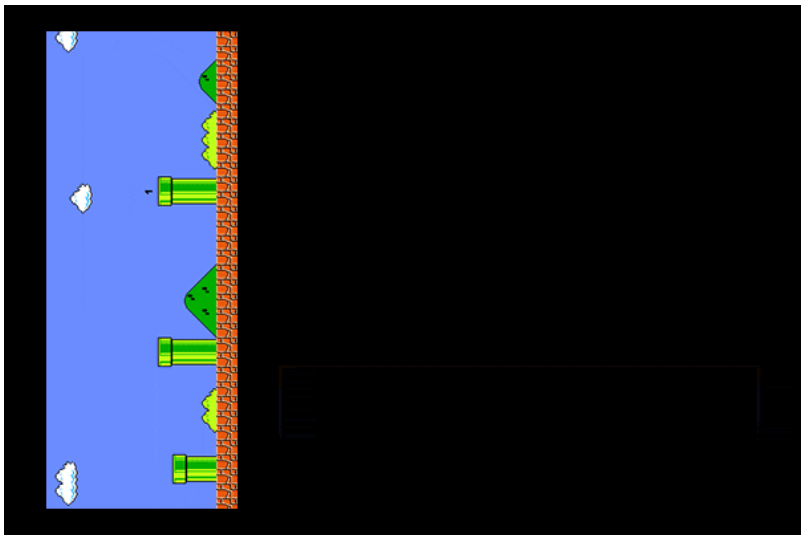
If painting an image **vertically** (that is, holding the strip horizontally and lifting it upward like a barbell), the image should be resized so its **width matches the NeoPixel strip length** (typically 144 pixels, unless you’ve made a custom variant). Images narrower than this will be centered in the strip, while wider images will be cropped; no scaling is performed.

Vertical images are painted **bottom to top**. This might seem odd, but is on purpose: the ground provides a consistent point of reference for starting. If you try painting from the top down, you may bump into the ground before the image is finished.



If painting an image **horizontally** (holding the strip vertically, moving left to right across the camera’s field of view), the image should be resized so its **height matches the strip length** (typically 144 pixels) and then **rotated 90 degrees counter-clockwise** (so the top of the image is now at the left side) before saving.

This extra step is necessary to reduce the amount of processing done on the Arduino; it would otherwise take minutes (instead of seconds) to decode a horizontal image at startup.



If the trigger is held (rather than tapped once) while painting, the image will be repeated. This can be used to create repeating patterns.

When transferring or saving a BMP file to the SD card, it must go in the **root directory** (not a sub-folder) and named **frame000.bmp**. No other file prefix or location is currently recognized, and no UI is provided for selecting alternates.

If the Arduino is refusing to load an image, connect a USB cable and open the Serial Monitor window in the Arduino IDE. The software will give some indication of its hangup; either accessing the card, locating the file or decoding the image format.

If you store multiple images to the card as frame000.bmp, frame001.bmp, frame002.bmp, etc. then each tap of the button will draw the next image in the series. The frame numbers must be contiguous, three digits, starting from zero (000). After the last image is painted the code will repeat the cycle from the start. **Converting multiple images at startup can be very time consuming!** Watch the red “SD” LED on the shield to see if it’s working.

Photography

You’ll just have to experiment with this part, there’s no one-size-fits-all setting. This requires a large, dark room, or go outdoors at night. It’s also much, *much* easier with a helper…one person can wield the paintbrush while the other tries different camera settings, sets focus, etc.

Place the camera on a tripod and set it to full manual mode, with a 3 second exposure to start. If the image is cut short, use longer exposures. **Manual focus is recommended.**

It’s not necessary to dress like a ninja for this. As long as you keep moving you won’t register in the photo.

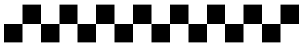
For horizontal painting, **move the brush from left to right** across the camera’s field of view. To work in the opposite direction, flip the image before saving the BMP file. If the painted image appears upside-down (or mirrored when painting vertically), just turn the brush around… no need to edit the image file.

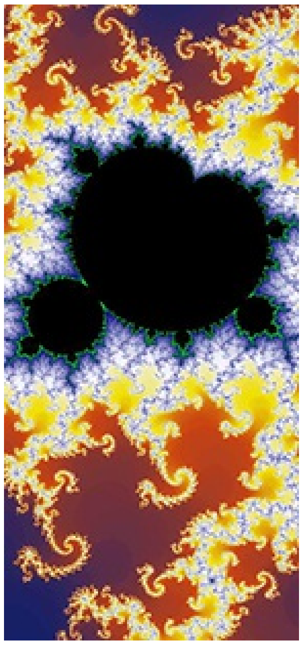
Sample Images

These are all in BMP format, 144 pixels wide, ready to go. Right-click over each image and select “Save Image As…” and rename as necessary (frame000.bmp, etc.)

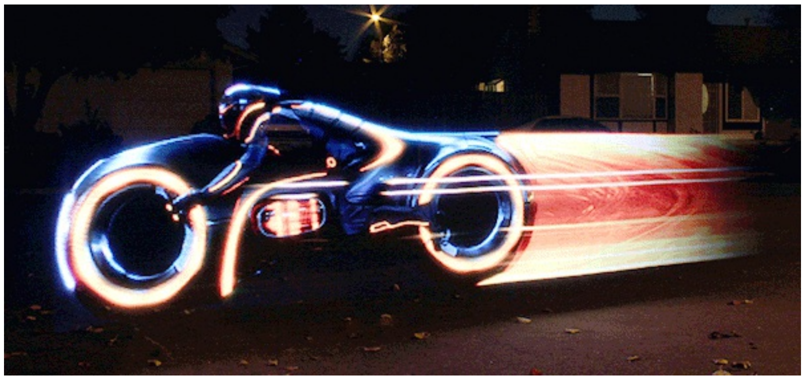
It’s easiest to load up **one image (or one animation sequence) per card**, then swap cards and reset to paint different images. The code interprets multiple files as an animation sequence, and all frames must be processed at startup to ensure uniform brightness. This can be really time-consuming when there are a lot of files!

If you end up using these, or you have other awesome light-painting photos, send us a picture! (http://adafru.it/d0X)





Optional: Positional Encoder



Creating a uniformly-proportioned digital light painting — one that appears neither stretched nor compressed — is often a matter of trial, error and luck. One must experiment with speed settings on the control board or try a different walking speed. The human element means it will change from one photo to the next or even across the frame in a single picture. For really bulletproof pictures, we'd ideally like to show 144 lines per meter of motion, to match the pixel density of the NeoPixel strip.

There's a technical solution to this, but be warned that it's a costly and fussy option. Making this work requires more improvised McGuyver tech and craft than the rest of the project. Fortunately you can build the project without it and still have a lot of fun, then add it later if you want.

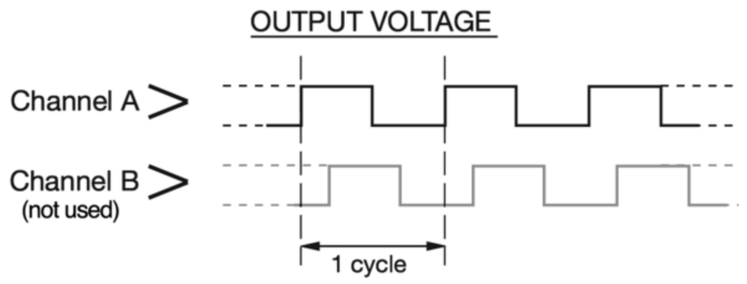
First, the light bar must be attached to some sort of wheeled motion base…bicycle, stroller, wheelchair, whatever you have access to. That's the easy part. Then one must interface a *positional encoder* — an accurate motion counter – to a wheel. This isn't like a bicycle speedometer where the speed can be extrapolated from a magnet once per turn…no, we need <u>right now</u> precision, an exact count.

The best tool for this is an optical rotary encoder, such as one of the Honeywell 600 series or Bourns EN series encoders. At the low end you *might* be lucky to find one for $35…but twice that is pretty typical. This is just for the component itself; mounting hardware is extra.

A mechanical rotary encoder (such as the one in the Adafruit shop) is <u>not</u> recommended for this. The resolution is coarse in comparison, the detents work against smooth rotation and the mechanical output requires debouncing. They're great for control panels, but accurately measuring movement really demands an optical solution.

Whatever you choose, it must have *excellent* resolution. The NeoPixels on a 144 LEDs/meter strip are spaced just 7 millimeters apart. A coarse encoder that can read only a few pulses per revolution simply won't cut it.

If you're really crafty, it *might* be possible to gut an optical "chopper wheel" from an old ball mouse (remember those?) and drive it from a larger wheel with a belt or gear system.



Most of these encoders are the "quadrature" type: they output two square waves 90 degrees out of phase, and can be used to determine both position and direction. Our application only uses one output and always advances forward; it doesn't distinguish direction. This lets us exploit a hardware feature of the Arduino microcontroller and saves a lot of code.

The encoder output must connect to digital pin #5 on the Arduino Uno. This pin can be used as an input to Timer 1, so the exact same code that normally relies on regular timer intervals now uses pulses from the wheel instead.

The encoder also requires a **+5V** and **GND** connection. **One** output is used (channel A or B), the other can be left **unconnected**.

To enable the encoder feature, un-comment this line in the code:

```
#define ENCODERSTEPS 8 // # of steps needed to advance 1 line
```

You'll probably need to change ENCODERSTEPS depending on your encoder's shaft diameter, number of steps, and factoring in any sort of gear ratio you might have between your motion base's wheels and the encoder. Estimate the number of steps covered over 7mm movement (the approximate distance between NeoPixels), then get some trial photographs to fine-tune the ENCODERSTEPS value.

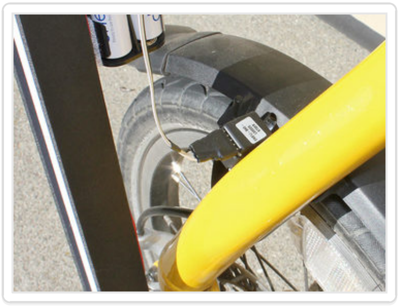
When using the encoder feature, the speed dial will have no effect. It's still used at startup for setting the image brightness, but isn't involved in timing the image playback.

Here the light bar is mounted on the rear rack of a folding bicycle, which is then pushed (not ridden). The control board has a long ribbon cable and is strapped to the handlebar.

You **do not** need to go get a folding bicycle to do this! They're pricey. Root around the house, see what you have

around or can source at a secondhand store. Razor scooter? LEGO wheels? TV cart? Fisher Price "Corn Popper" toy?

The encoder selection and code adjustment are just a small part of the challenge. Physically interfacing the encoder to the wheel is the hard part…and a problem we unfortunately can't help solve because every motion base will be differently improvised and manufactured. You may need to devise a custom bracket of some sort to hold the encoder to the wheel. Machining? 3D printing? Cable ties and LEGO Technic parts? You'll have to decipher that for your particular wheel and encoder.



By sheer dumb luck the encoder I had on hand fit precisely in an unused mounting hole on my particular bike fenders, putting the encoder shaft (by sheer dumb luck the "round" variety, with no knob alignment notch) directly in contact (by sheer dumb luck) with the tire. This was like winning the geek lottery; don't count on it happening. In all likelihood you will need to create a custom bracket or belt/gearing system of some sort.

The wavy animated effect was created by backing the bike's rear tire up against a brick so it always starts in the same location, then taking several photographs of the same image. The slight variations in the path and tilt of the bicycle create a sort of rippling effect when these images are then stacked and animated in Photoshop.