LED Art with Fadecandy

Created by Micah Elizabeth Scott

https://learn.adafruit.com/led-art-with-fadecandy

Last updated on 2022-12-01 02:05:00 PM EST

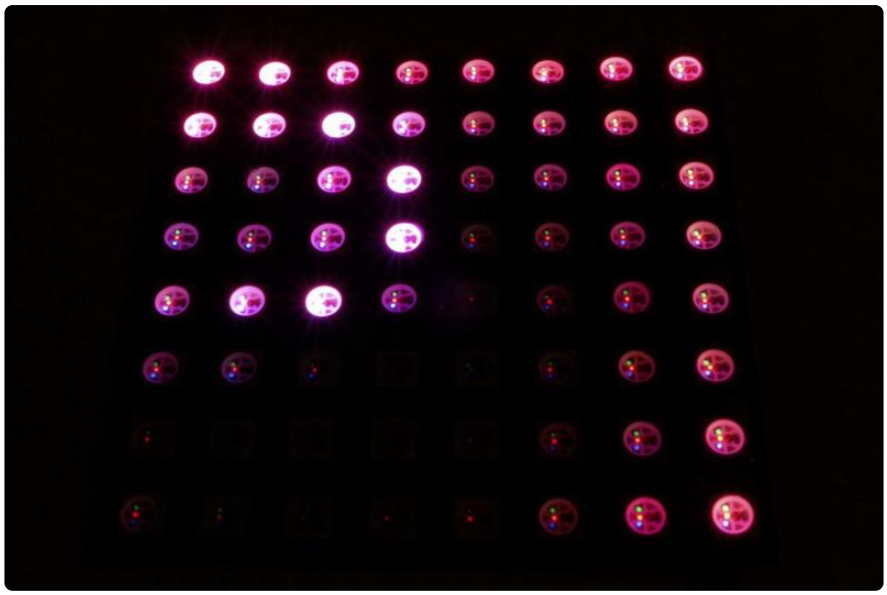
Table of Contents

Introduction

This collaboration was between Adafruit & Micah from Scanlime, Fadecandy, a NeoPixel driver with built in dithering, that can be controlled over USB. For every FadeCandy sold Micah from Scanlime was paid, Micah's work can be supported on Patreon (Adafruit Patreon sponsored from 2017 to 2020). Adafruit emailed Micah multiple times for possible updates or status of the project, but has not heard back since 2018. The software is known incompatible with 64-bit MacOS systems, possibly others.

Individually addressable LEDs are everywhere. In the quest for a smaller, cheaper, smarter LED, the latest and hottest technology is the NeoPixel (), AdaFruit's term for the WS2812. This is an amazing little chip that integrates red, green, and blue LEDs with a controller chip into a single package.

NeoPixels are inexpensive, bright, easy to use, and you can use them to build projects of almost any size. They show up in costumes, sculptures, vehicles, and signs. The saturated colors and frenetic blinking of these lights can be hard to escape at parties and festivals.



These LEDs are capable of so much more. I believe that LED lighting can be nuanced, with wide ranges of brightness from blindingly intense to barely visible, saturated hues to subtle off-white. Getting this range of expression from LEDs can be super tricky, especially when the only tool you have at your disposal is the Arduino IDE.

Fadecandy () is a project that tries to solve this problem, by making LED art both easier to build and more expressive. At the core of the project is the Fadecandy Controller, a tiny board that lets you control up to 512 NeoPixels (as 8 strands of 64) from any computer or embedded Linux board with USB. It includes unique color dithering and interpolation algorithms to get the most out of each pixel.

Fadecandy works with any WS2811 or WS2812 LEDs, and you can program your LEDs using many different environments. This tutorial covers one particular LED module and one particular programming environment. We'll be programming an 8x8 NeoMatrix (http://adafru.it/1487) in the Processing () language.

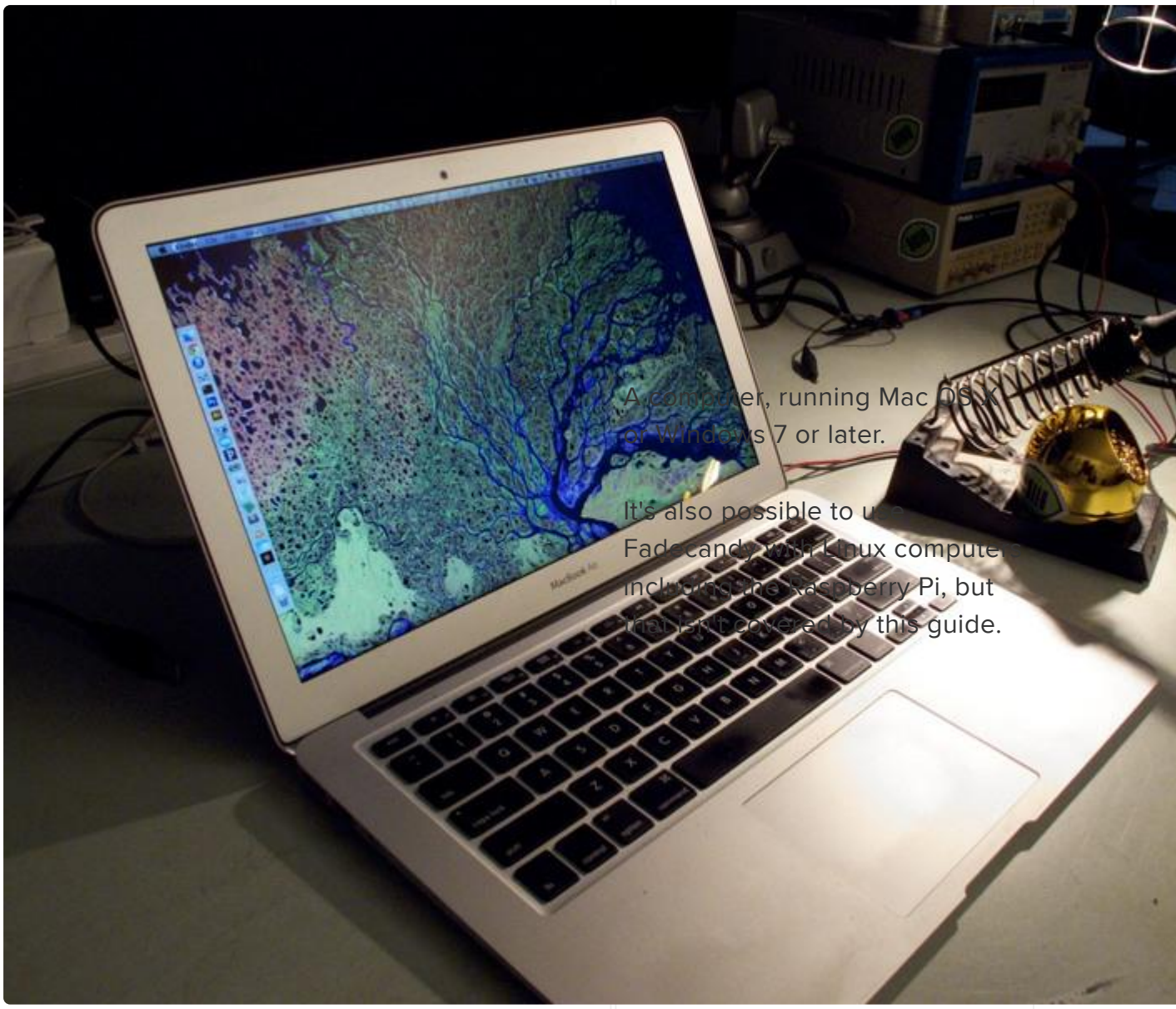
You'll be doing some programming and some basic electronics assembly. If you aren't familiar with programming or electronics yet, don't worry- there are plenty of tutorials on learn.adafruit.com () to help you if you get stuck, and you might even find someone near you who would love to help you get started.

Tools and parts

Basic electronics tools

- Soldering iron
- Solder
- Wire
- Cutting tools
- Multimeter
- Tape

This might be a good excuse to make friends with someone who has a soldering iron, but if you want to buy a set of tools Ladyada's Electronics Toolkit will have everything you need.



A computer, running Mac OS X or Windows 7 or later.

It's also possible to use Fadecandy with Linux computers including the Raspberry Pi, but that is not covered by this guide.

A Fadecandy Controller board.

One board can control up to 512 pixels. We'll only be using one of the eight channels on the board.

A USB cable.

If you have a Revision A board with the green PCB, you'll need a Micro USB cable.

If you have the Revision B board from Adafruit with the blue PCB, use a Mini USB cable.

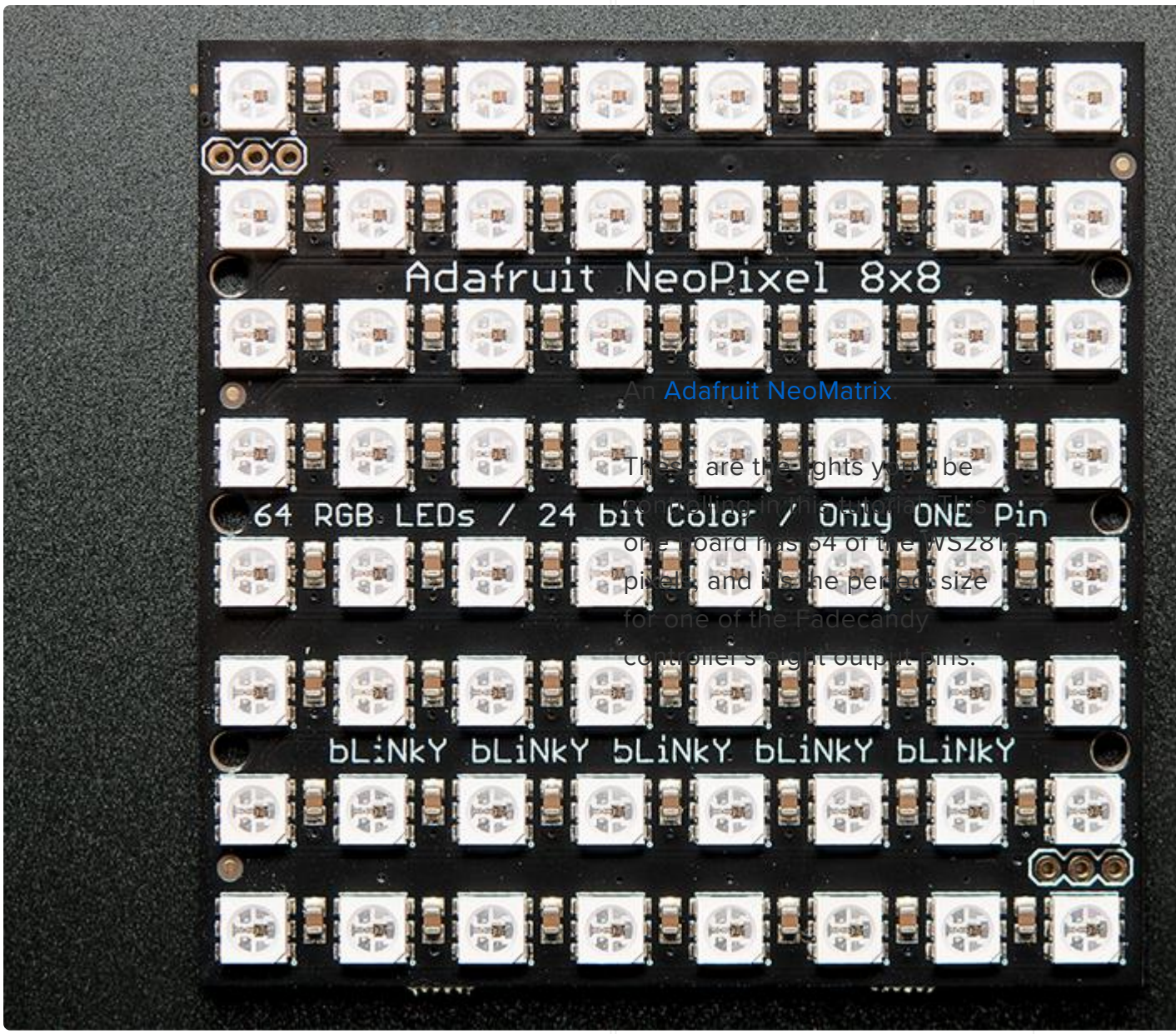
A Beefy 5V power supply.

5V 10A power brick

5V 4A supply

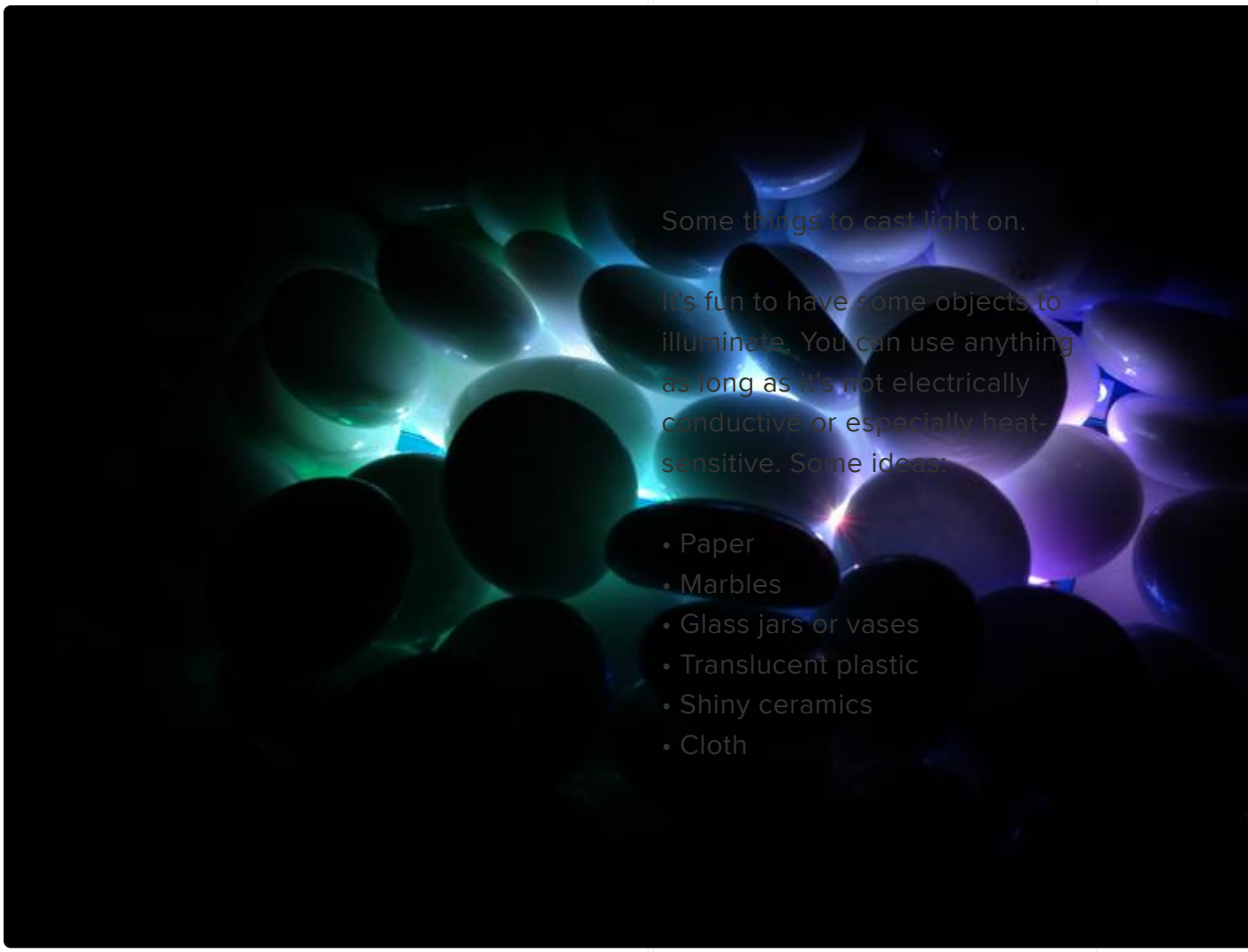
A matching Barrel Jack.

This will give you a way to connect your power supply to the LEDs. There are many options, but Adafruit sells a basic jack that will work with the power supply we're using.



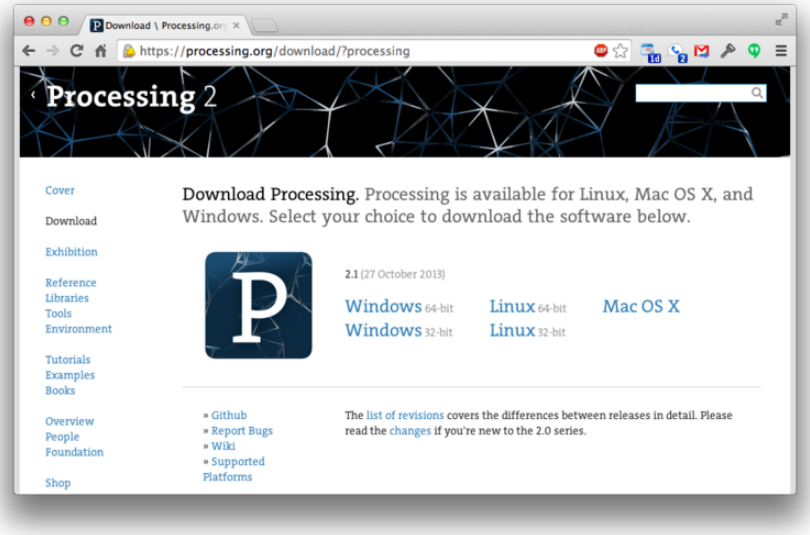
An Adafruit NeoMatrix

These are the lights you'll be controlling in this tutorial. This one board has 64 of the WS2812 pixels, and it's the perfect size for one of the Fadecandy controller's eight output pins.



Install some software

Processing is a programming language that's especially convenient for creating multimedia art. If you don't already have it installed, head to the Processing.org web site () and download it.



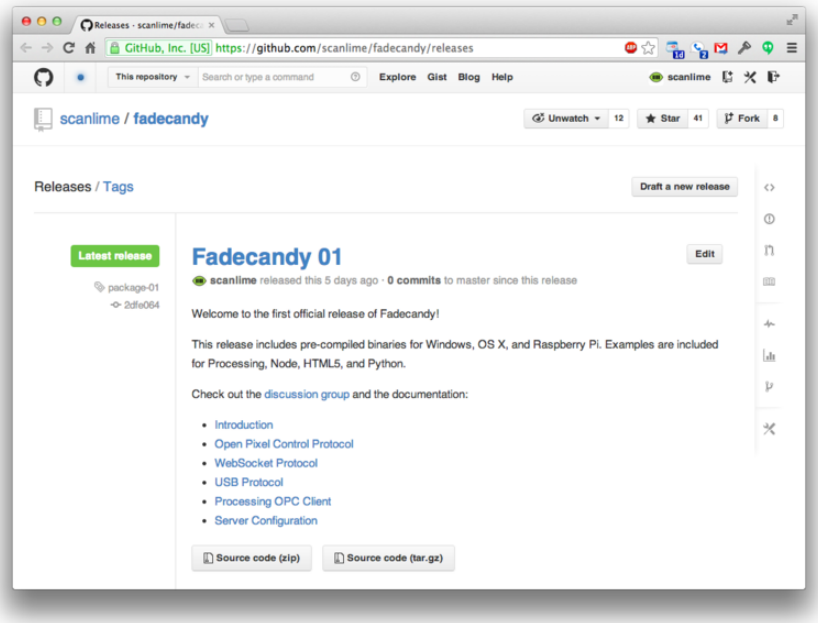
On Mac OS X, the .zip file you download will have a Processing application inside it. You can drag that to your Applications folder or anywhere else you like. On Windows, the .zip file will contain a folder. You can drag that folder anywhere you like.

If you get stuck or you'd like more information about getting started with Processing, take a look at the Getting Started tutorial for Processing ().

When you're finished, you should be able to run Processing and get a window like this one:

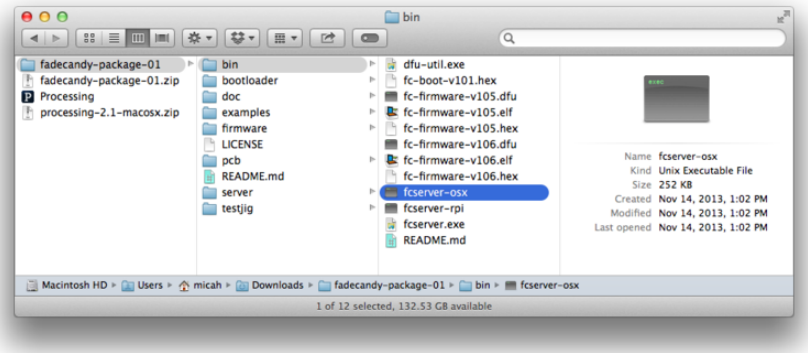
Great! Now for the next thing we'll need: The Fadecandy software. You can download this from the releases page on GitHub (). The link you want is the "Source code (zip)". This package contains both the source code and the binaries you'll need for this tutorial.

The specific release we're using here is Fadecandy 01 ().



This will be another .zip file. When you extract it, you'll see a lot of things. This package contains everything related to the Fadecandy project, including things like the source code for the board's firmware, and CAD files for the circuit board layout. You can ignore most of its contents for now. There are two things we'll need from this download:

- The Fadecandy Server (fcserver) program, in the bin directory
- Processing examples, in the examples directory

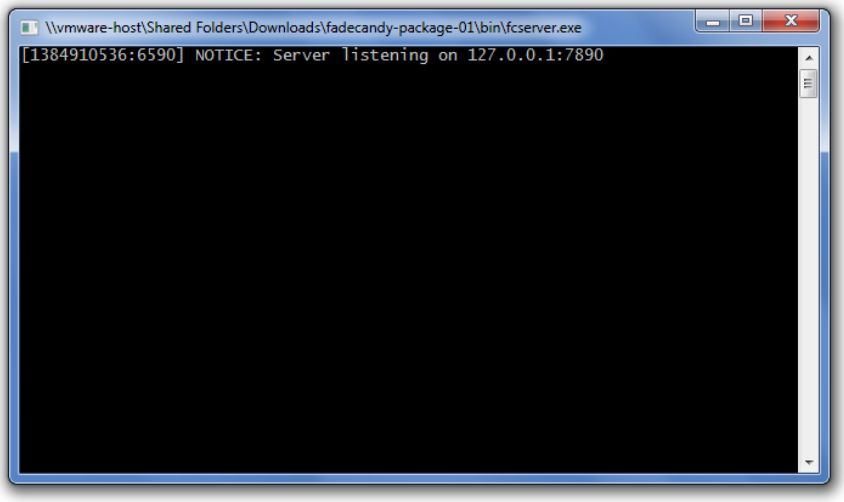


First, run the fcserver program. This program runs in the background and connects your Processing sketches with the Fadecandy Controller board. It also has a simple web interface you can use to test your LEDs.

- On Mac OS, double-click the fcserver-osx file in bin
- On Windows, double-click fcserver.exe in bin

This program will run in a Terminal window. If all is well, you should get a message like "Server listening on 127.0.0.1:7890". This tells you it's ready to accept connections

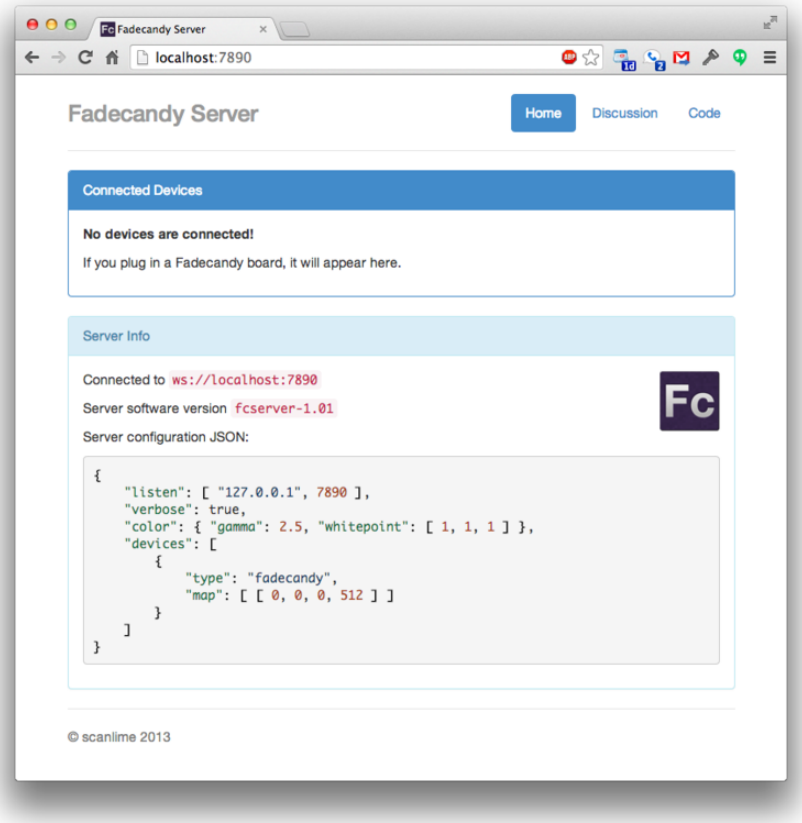
from other programs on your computer.



What is fcserver listening for? Three things, actually:

- Programs using the Open Pixel Control () protocol to talk to your LEDs
- Browser-based programs using WebSockets () to talk to your LEDs
- Web browsers requesting an HTML-based user interface

Let's try out the browser interface. Now that fcserver is running, you can navigate your web browser to http://localhost:7890 (). You should see a page like this one:



If you plug in a Fadecandy Controller board, it should show up in the Connected Devices section. No need to reload the web page.

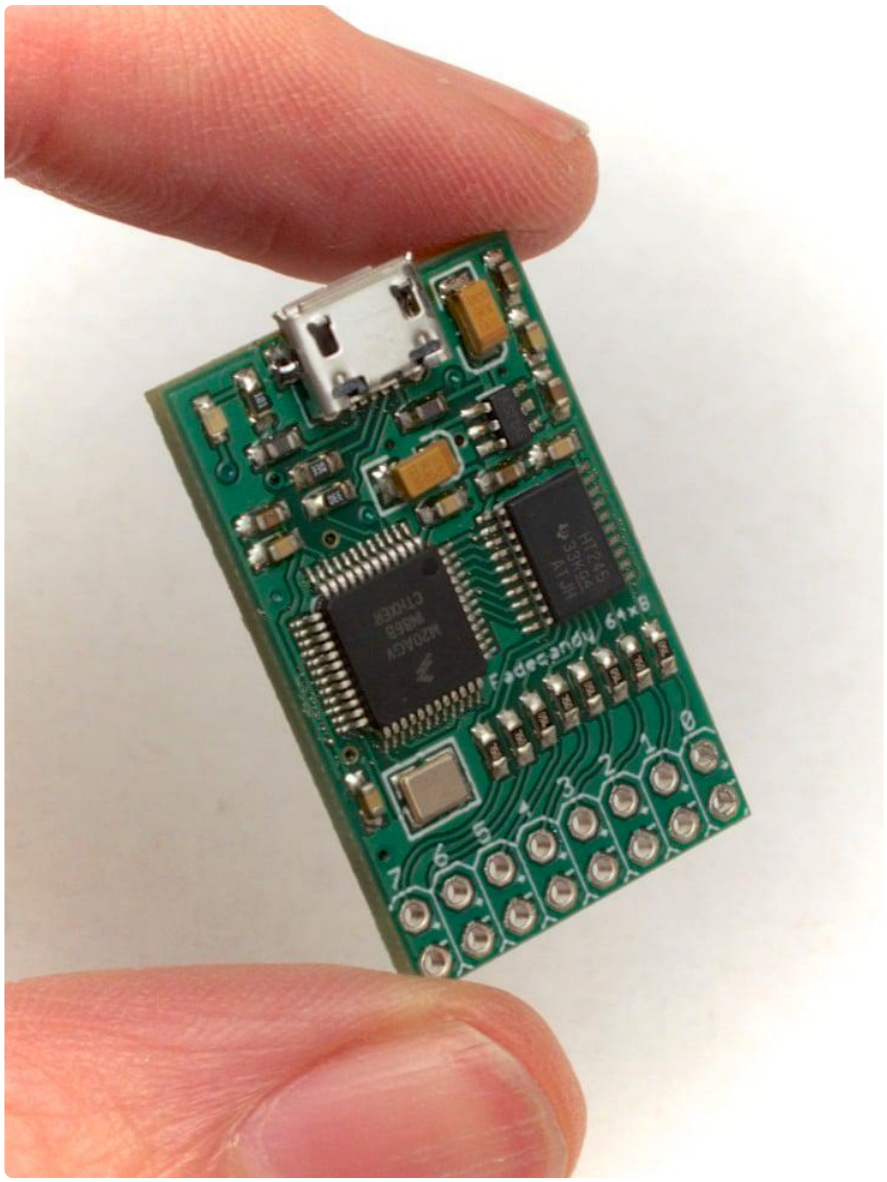
The bottom section shows you how the server is configured. This isn't really important unless you have special requirements or you're using multiple Fadecandy Controller boards, but you can read more about this in the Fadecandy server configuration () documentation.

That's all the software you need for this tutorial. Now for some wiring!

Wire your LEDs

Wiring up your Fadecandy Controller will require a little soldering, since LED strips and arrays come in all different shapes and sizes. The Fadecandy Controller board is small enough you can incorporate it directly into your art projects. Every project will need at least one USB connector and at least one power connector.

First, let's get acquainted with the Fadecandy Controller.



This is the component side of the Fadecandy Controller. At the top, you'll see a Micro USB connector. This is how Fadecandy gets data from your computer, and how it powers itself. The LEDs themselves draw so much power that they need a separate power brick, but the controller board requires very little power.

At the bottom, there are eight outputs. Each output can drive a chain of up to 64 LEDs. In this guide, we're running an 8x8 matrix with exactly 64 LEDs. It could connect to any of the eight outputs, but by convention we'll start with the first one (labeled with a Zero).

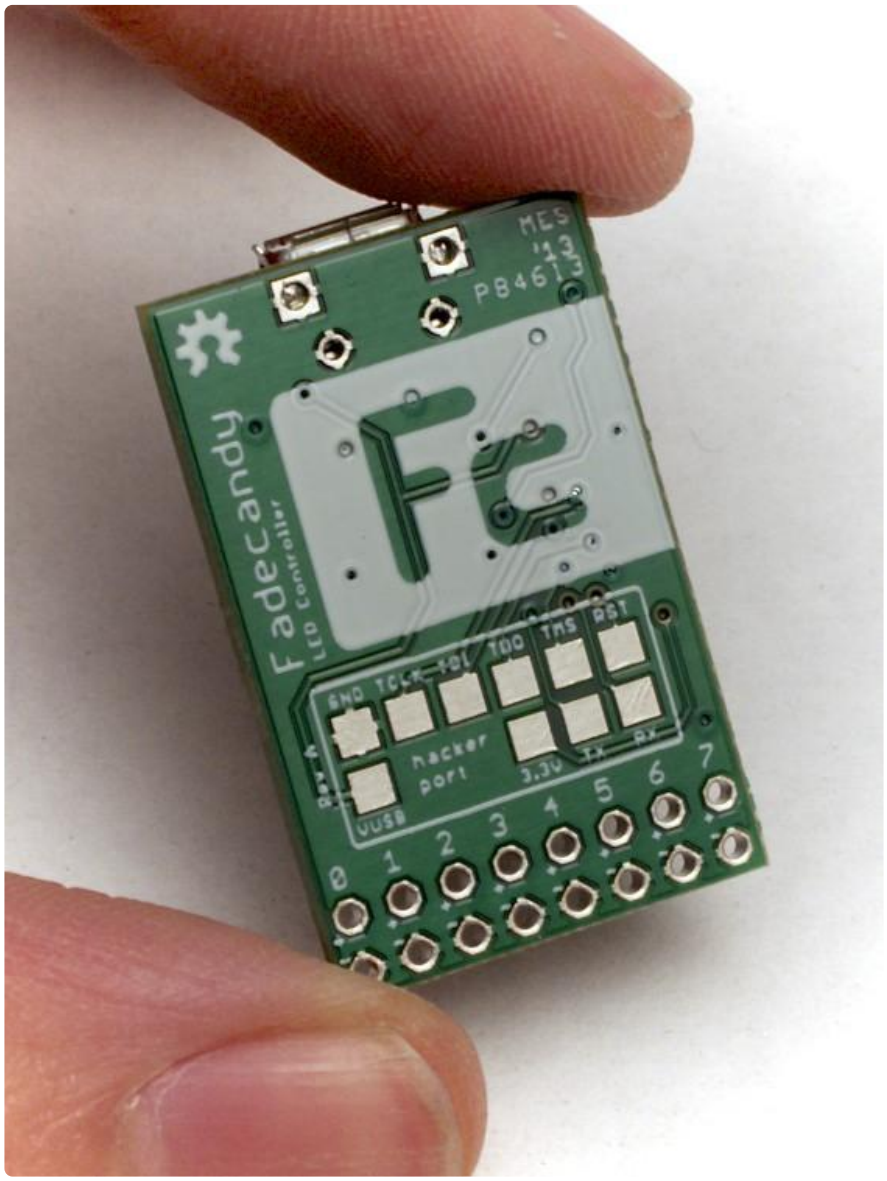
At the top-left there's an LED. You can control it in software, but by default it will blink any time the Fadecandy board receives data over USB.

The largest chip, on the left, is the brains of the Fadecandy Controller. It's a 32-bit microcontroller running at 50 MHz. This chip needs to simultaneously receive data over USB, output it to all of your LEDs, and run the dithering and interpolation

algorithms. The Fadecandy Controller ships with firmware built-in that you don't need to modify.

The second largest chip, on the right, is an electrical buffer that drives the eight outputs with a strong 5-volt signal. This helps Fadecandy run reliably even in hostile environments and with longer wires.

The tiny 6-pin chip right above that is a power supply boost chip that gives the electrical buffer a stable 5-volt power supply even if the USB power isn't so great, as is often the case when dealing with long cables and hostile environments.

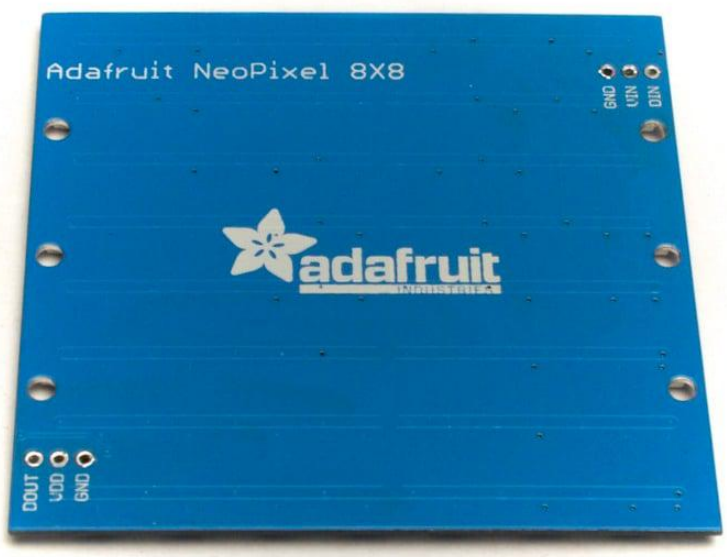


This is the flat side of the board, with the large "Fc" logo. The "hacker port" below is used during manufacturing. If you're interested in doing really strange things with your Fadecandy board it may be handy, but most people can totally ignore these pins.

There are many ways to incorporate the Fadecandy controller into your project. The

board is small enough you can use zip-ties and heat shrink tubing to incorporate it into your project's wiring harness, or you can mount it with double-sided foam tape, or you could design a bracket that it snaps into.

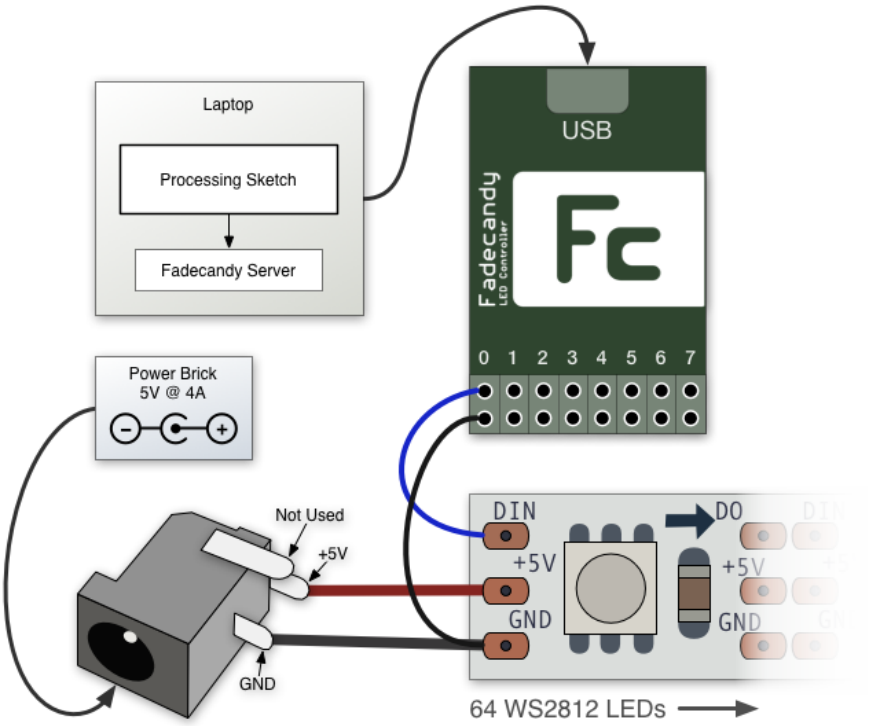
(The board dimensions are 0.8 x 1.25 inches.)



This is the NeoMatrix. It has two groups of three holes, labeled "DOUT / VDD / GND" on one side, and "GND / VIN / DIN" on the other side.

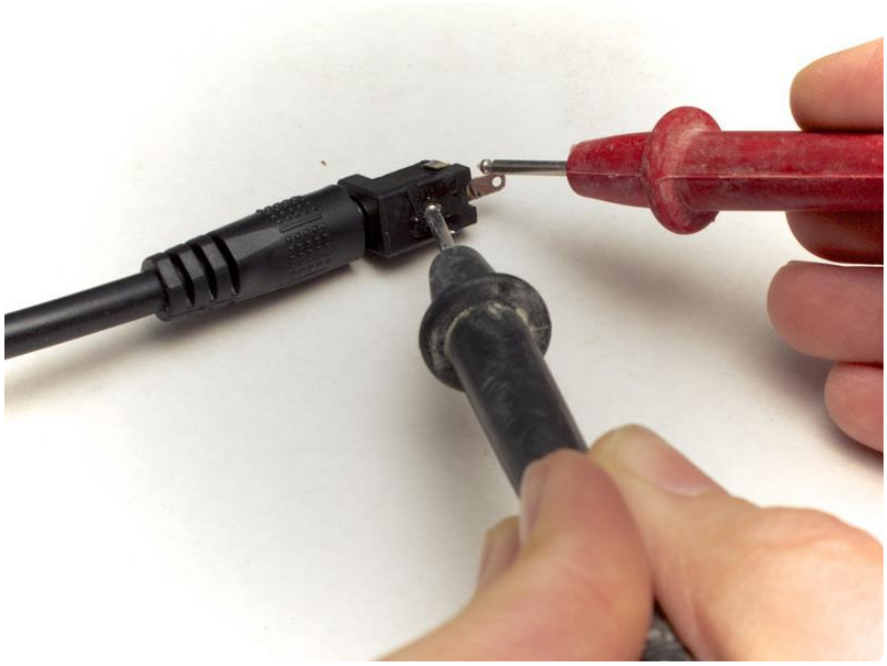
- GND: Ground. This completes the electrical circuit for power and for data. All of the GND pins are the same on this board.
- VDD: 5 volt power. This is where we power the LEDs. All of the VDD pins on this board are the same.
- DIN: Data input. This is the 5 volt data signal coming from the Fadecandy controller.
- DOUT: Data output. We don't use this pin, since the NeoMatrix already has all 64 LEDs supported on a single Fadecandy controller channel.

To make the NeoMatrix go, we just need to wire it for data and for power. The thing we're building is basically the same as the diagram you find in the Fadecandy README document ():



So, let's start with power. We're using a barrel jack (http://adafru.it/373) that matches the plug on our power brick (http://adafru.it/658). These jacks have three pins. The one on the side isn't used here, it's just for detecting when a plug has been inserted. The other two pins connect to the outside casing of the barrel plug, and the center pin. In the most common arrangement, this outside shell is negative (ground) and the inner pin is positive.

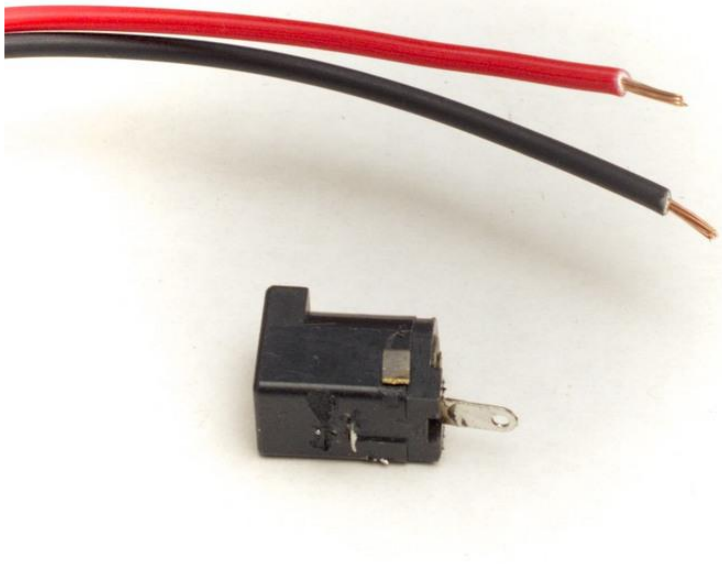
It's really important to make sure your power supply voltage and polarity are correct. If either is incorrect, you can damage your LEDs. If you have a digital multimeter handy, use its DC Volts setting to test the voltage on the barrel jack before you start soldering.



Following the polarity on the diagram above, we'll put the black probe on the pin that attaches to the outside casing of the plug, and the red probe goes on the center pin. If all is well, you should see a positive number close to 5V.

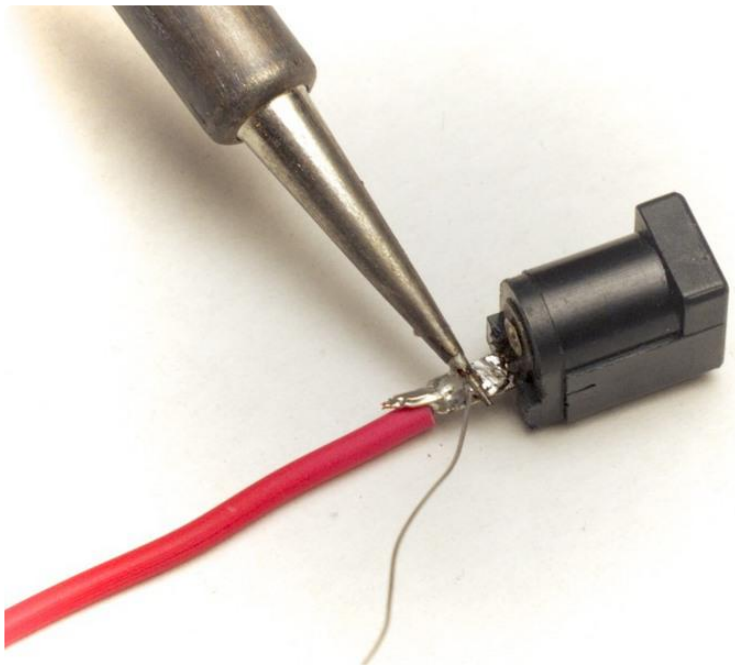
The third pin isn't used, and you might find it convenient to cut that pin off.

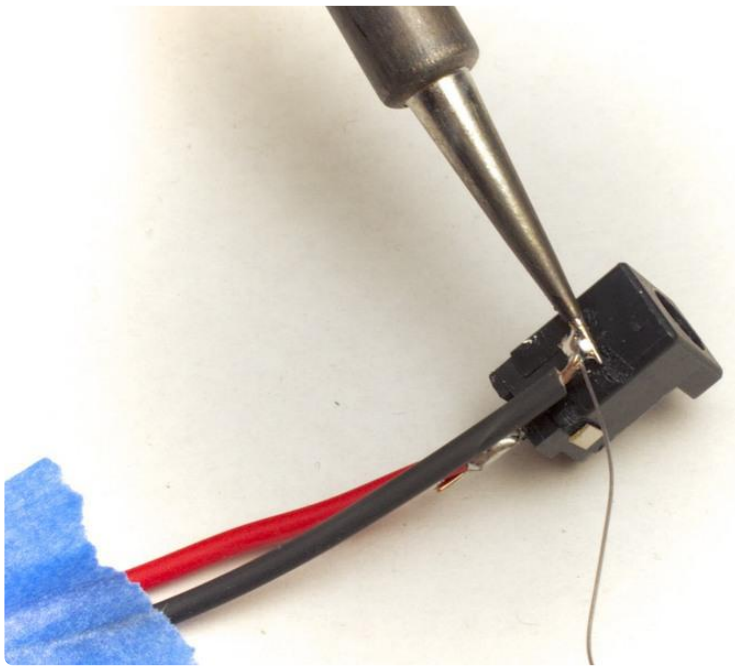
Now we'll get ready to attach wires to the barrel jack.

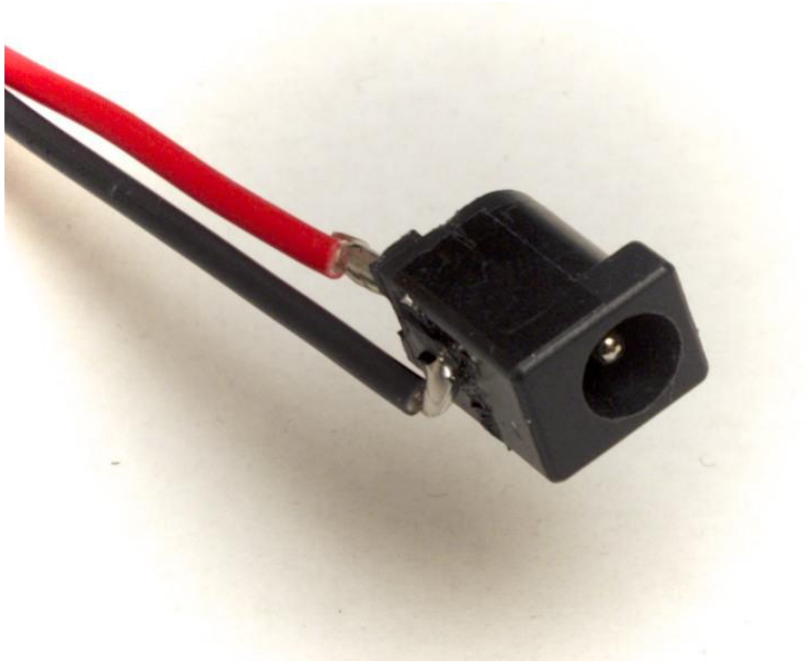


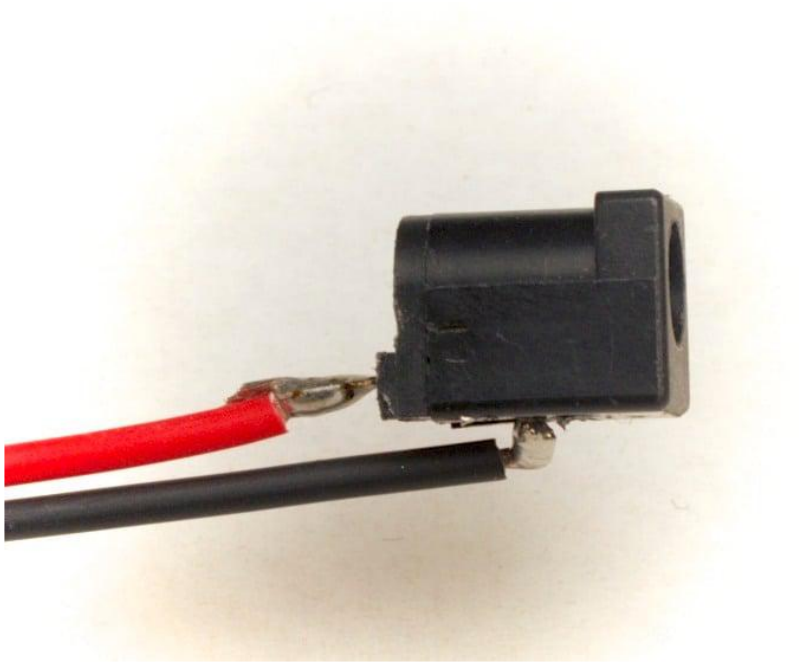
If you have multiple colors of wire, the tradition is to use black or green for ground and red or another bright color for positive. I used red and black wire. This wire should be relatively thick, since it has to carry several amps of power. I used 20 gauge stranded wire.

Prepare your two pieces of wire by stripping off about a quarter inch of insulation. You'll be putting each wire through the hole in your barrel jack's terminal, wrapping it around, and soldering it.



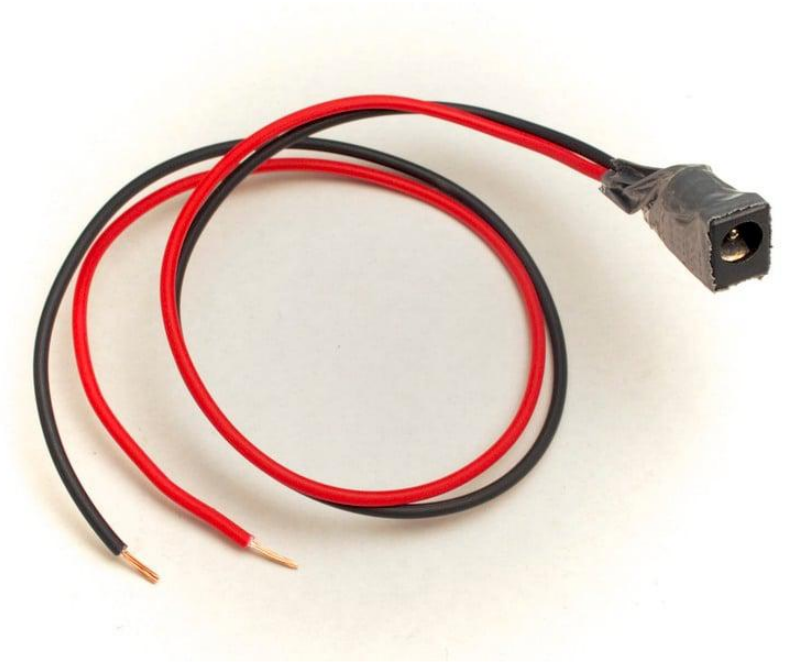




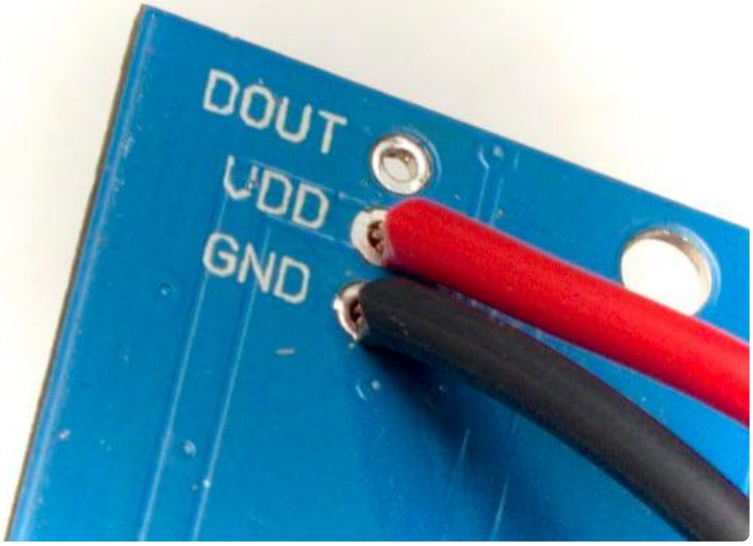


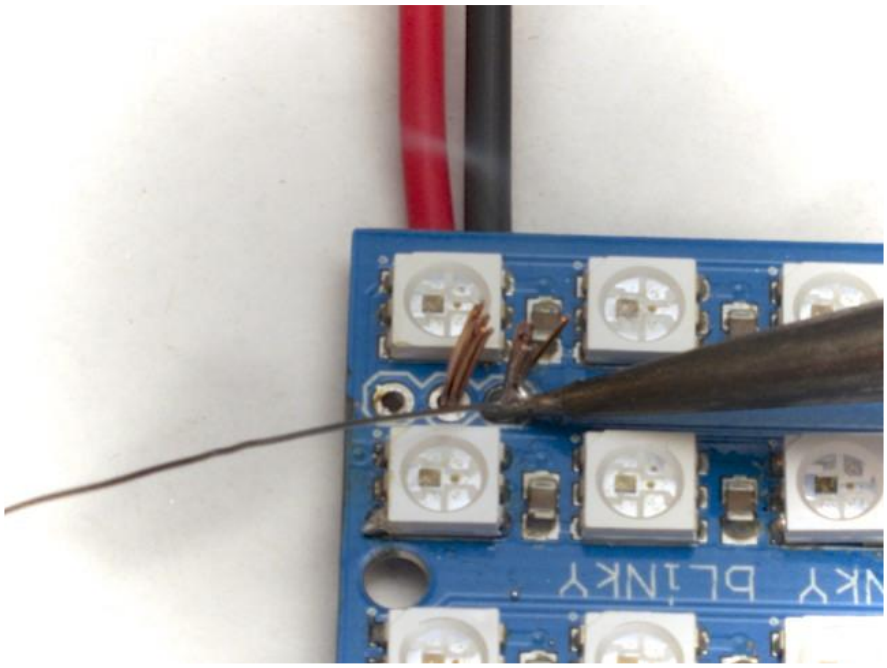
The barrel jack is almost ready to use! You'll want to insulate it, though, so you don't have to worry about metal objects bridging the two power terminals. It's especially important to watch out for short circuits when you're using such a large power supply!

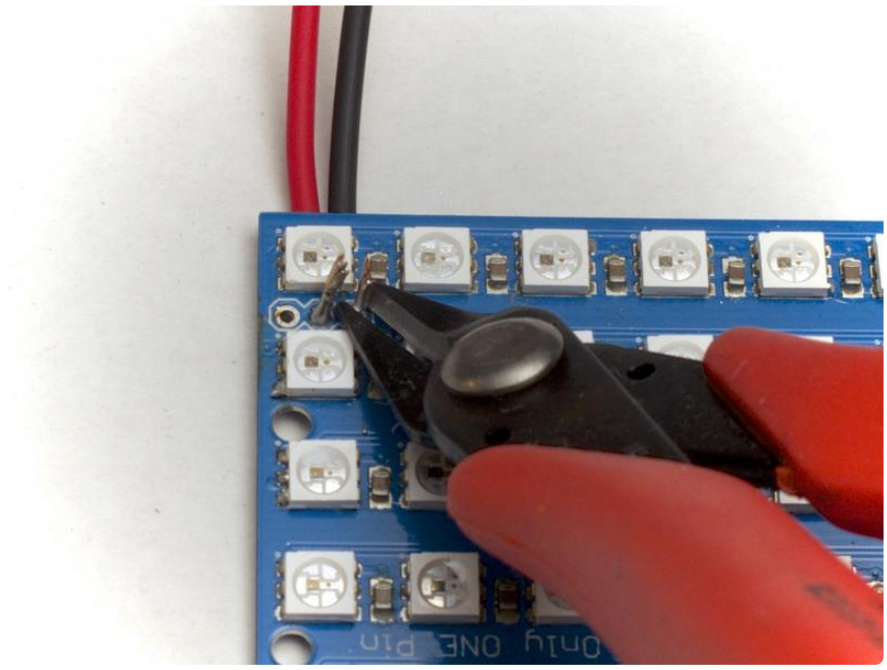
If you have heat shrink tubing or electrical tape, this would be a good time to use it. But duct tape works fine too.

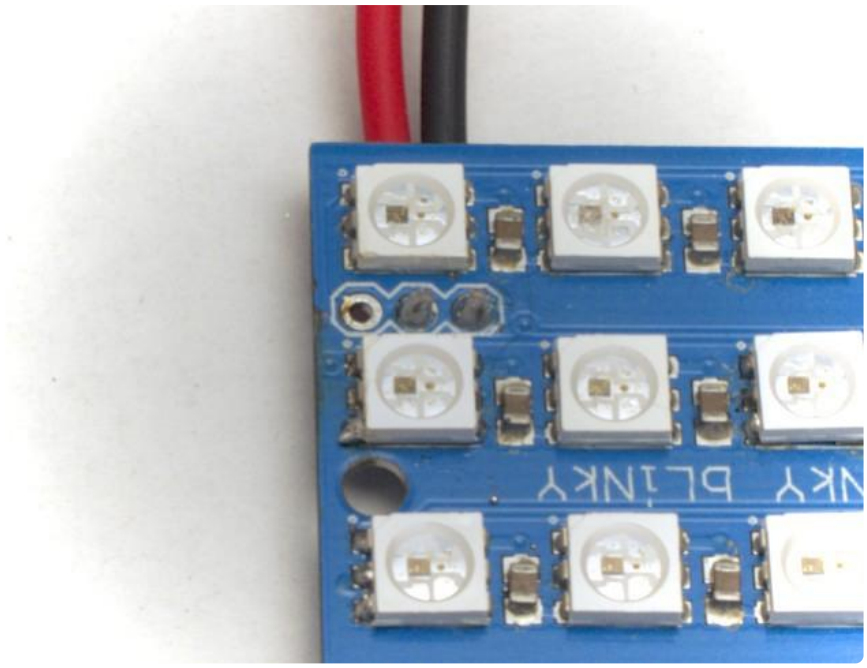


These wires will go to the VDD and GND pins on the NeoMatrix. Since we'll need a GND pin for the Fadecandy board too, it's convenient to attach these wires to the side of the NeoMatrix with DOUT instead of DIN.



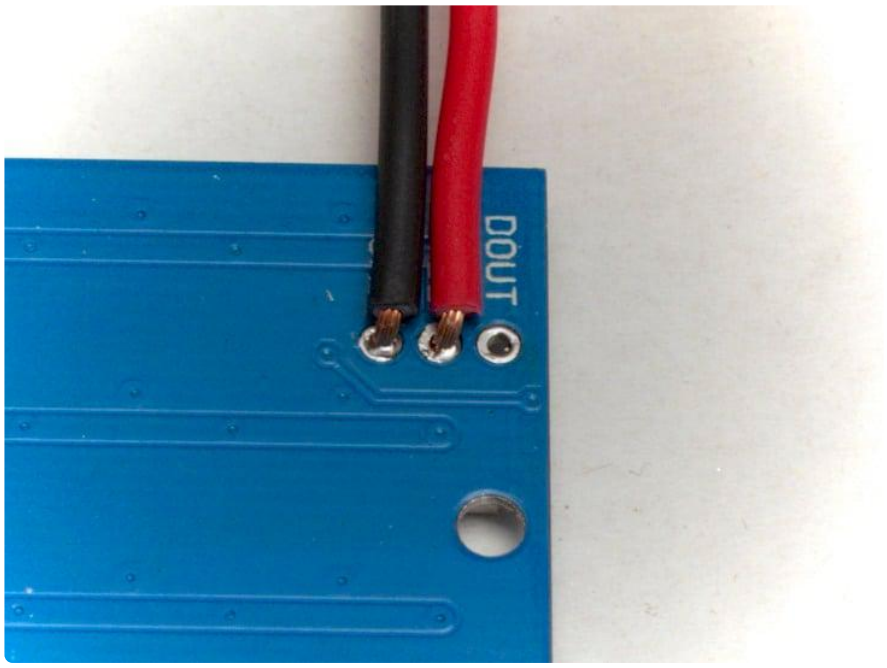

BLINKY

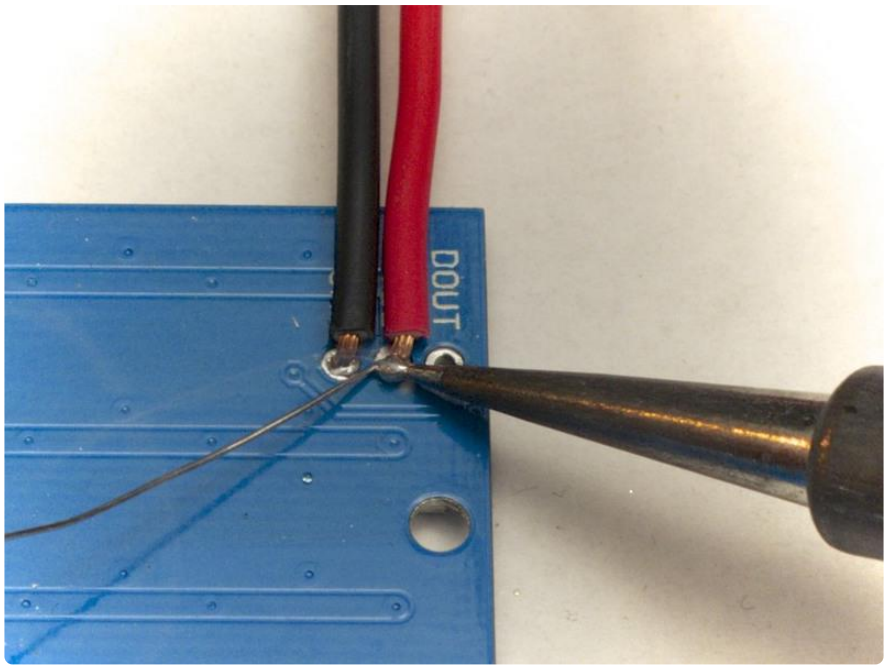

Only ONE pin

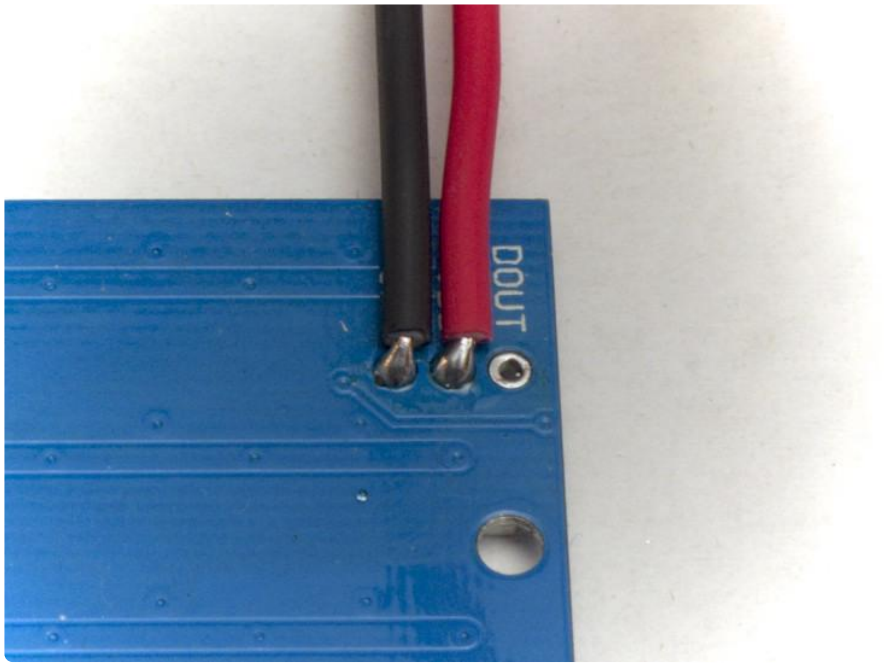


I also like to put a little solder on the other side, where the wires come out. This adds a little mechanical strength, making the wires less likely to break.

It helps to keep these wires flat against the board when you do this, since you'll want the NeoMatrix to lay flat when you're using it.





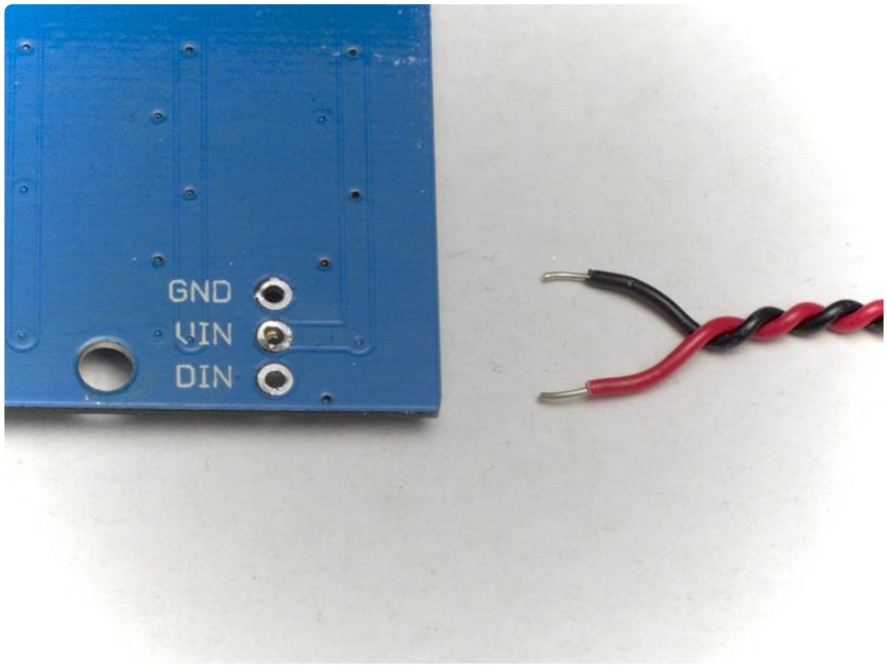


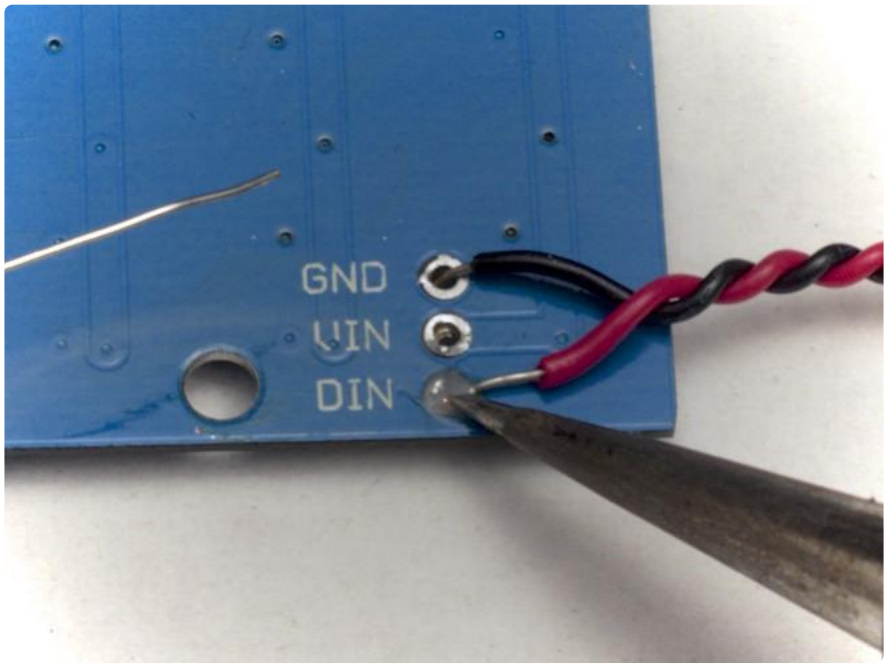
Alright! Now we can power our LEDs. Next, we need to get data from the Fadecandy Controller to the NeoMatrix. Just like power, we need two wires for data. The positive wire (DIN) carries video data, and the negative wire (GND) completes the circuit.

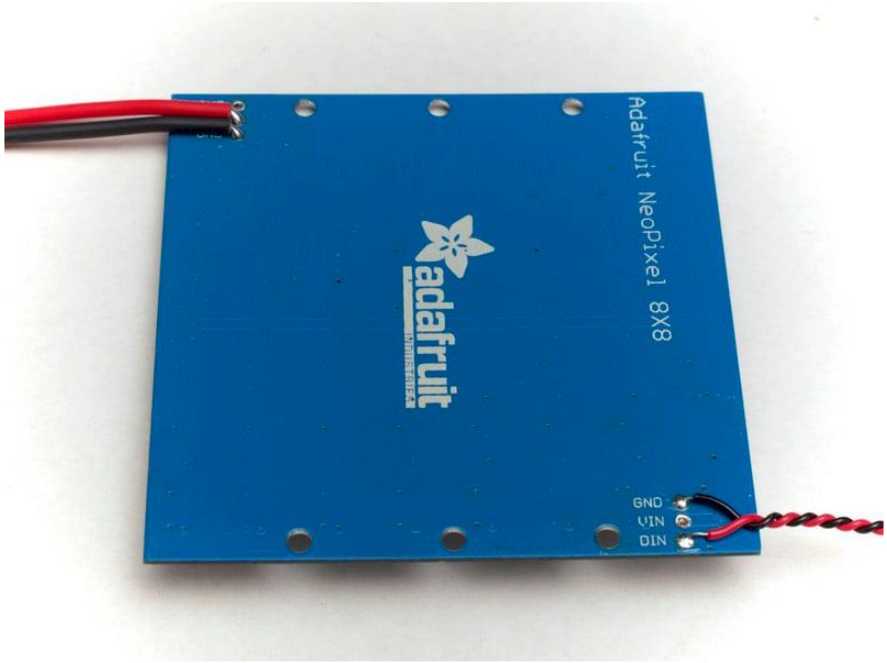
The recommended way of wiring the Fadecandy Controller's ground (GND) wire is to run separate wires from the Fadecandy Controller to your LED strips, and to keep its ground wire paired with its data wire. Keeping your data wires and power wires separate is good practice for creating reliable projects. At this small of a scale it isn't a big deal, but this will help a lot with reliability on larger projects.

For the signal wires, I used slightly thinner wires, 26 gauge. It's useful to use two

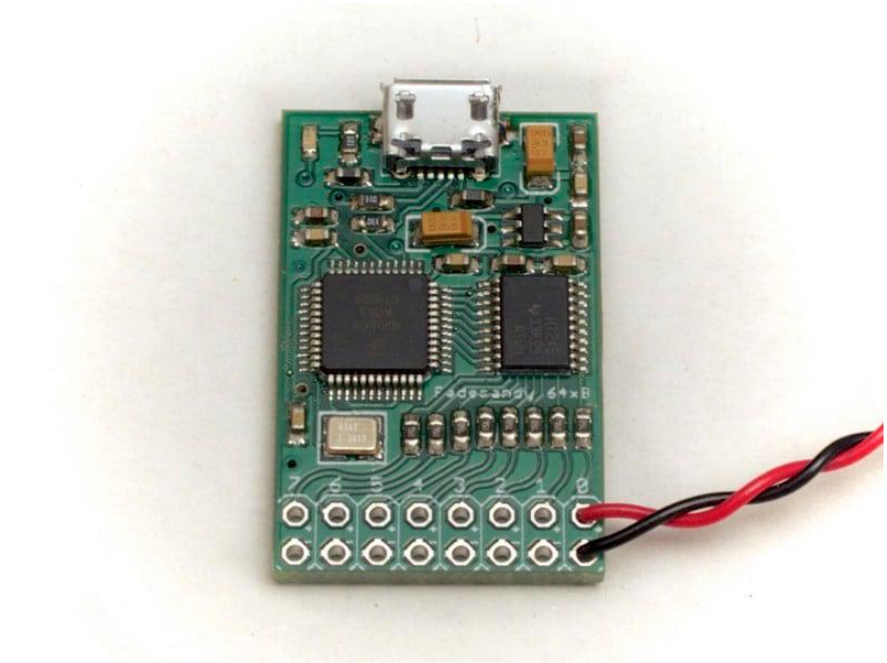
different colors, so you can tell the wires apart. I used red and black again. If you like to, you can twist the wires to keep them tidy, but this isn't required.

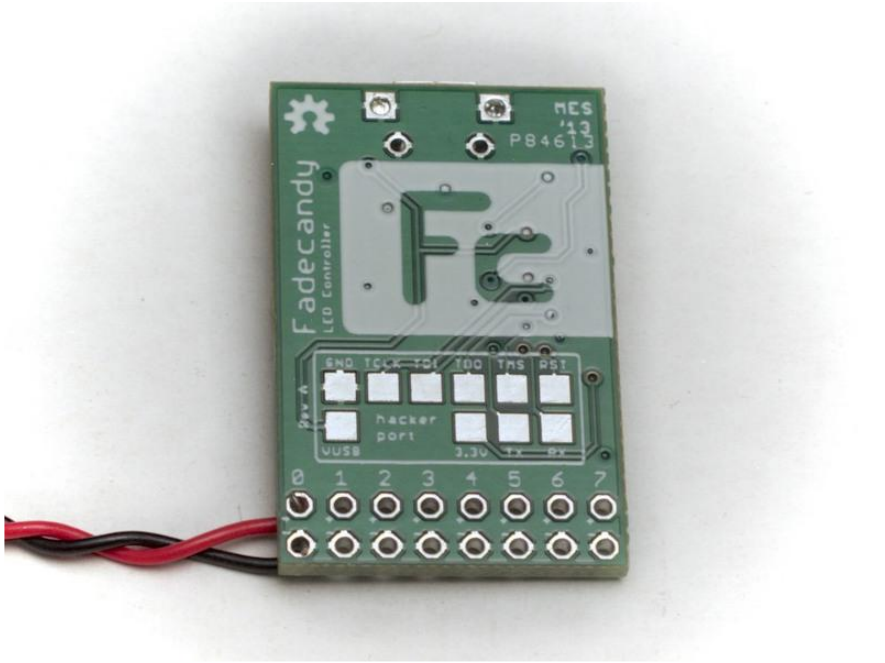


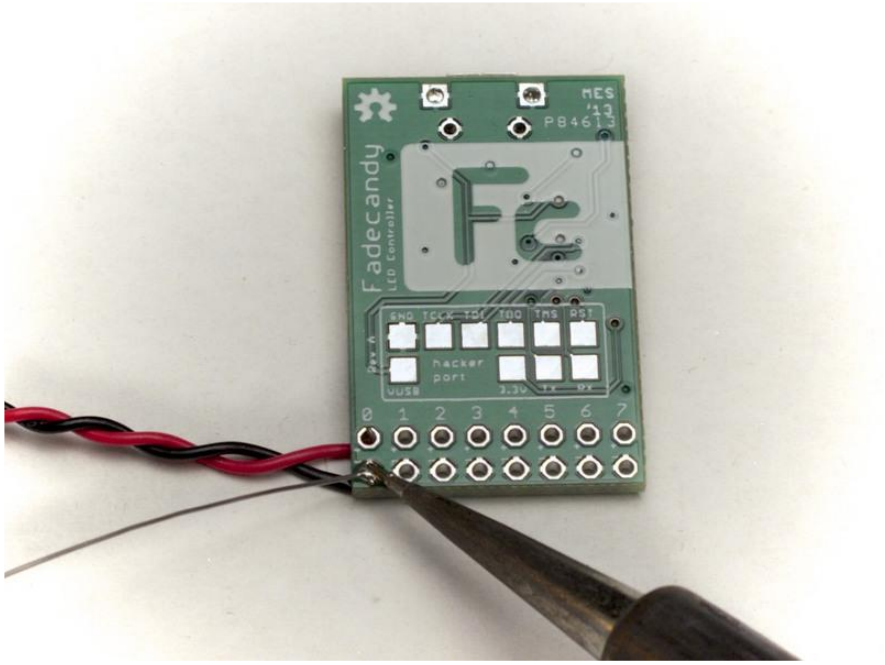


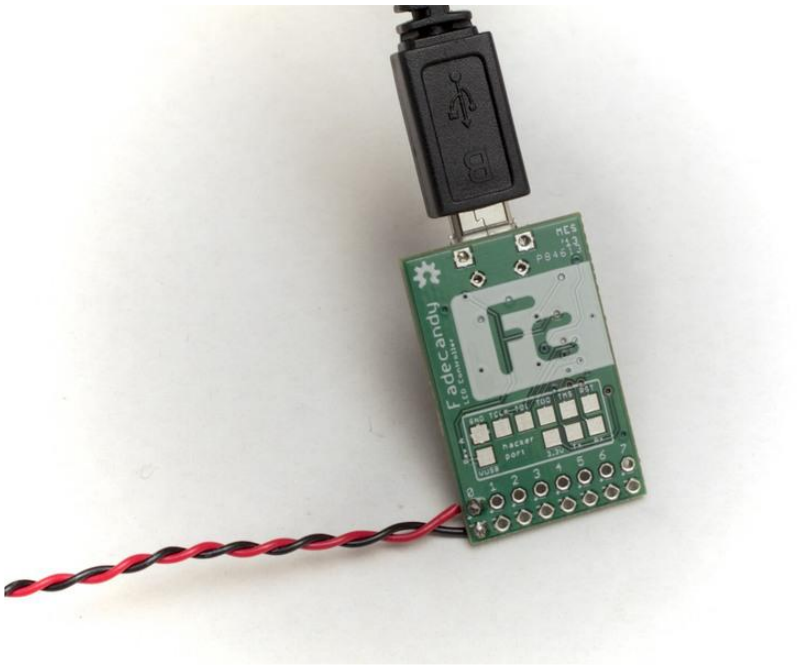


The other end of this wire pair connects to channel Zero on the Fadecandy Controller. The GND wire needs to connect to the - terminal on the Fadecandy, and the DIN wire connects to +. There are tiny + and - labels on the board, or you can remember that the pins nearest to the edge are all -.

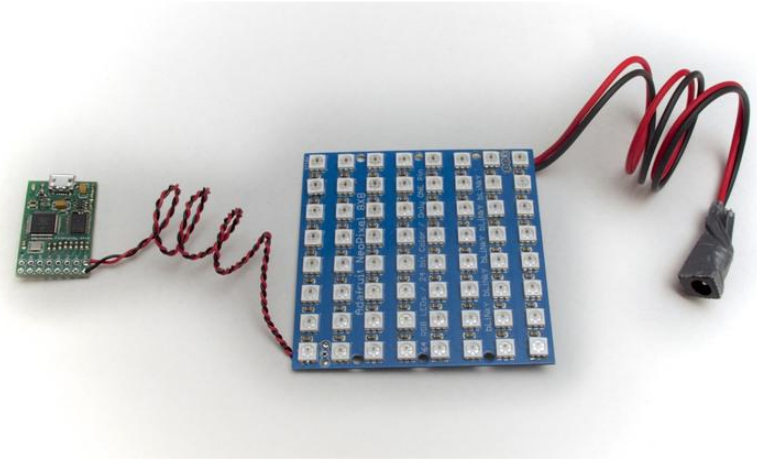



MES
'13
P84613
Fadecandy
LED Controller
GND TCK TDI TDO TMS RST
hacker
port
VUSB
3.3V
0 1 2 3 4 5 6 7


MES
'13
P84613
Fadecandy
LED Controller
GND TCK TDI TDO TMS RST
hacker
port
VUSB
3.3V
0 1 2 3 4 5 6 7



Congrats! Your wiring is all done, and now you have a NeoMatrix that's ready to attach to your computer.



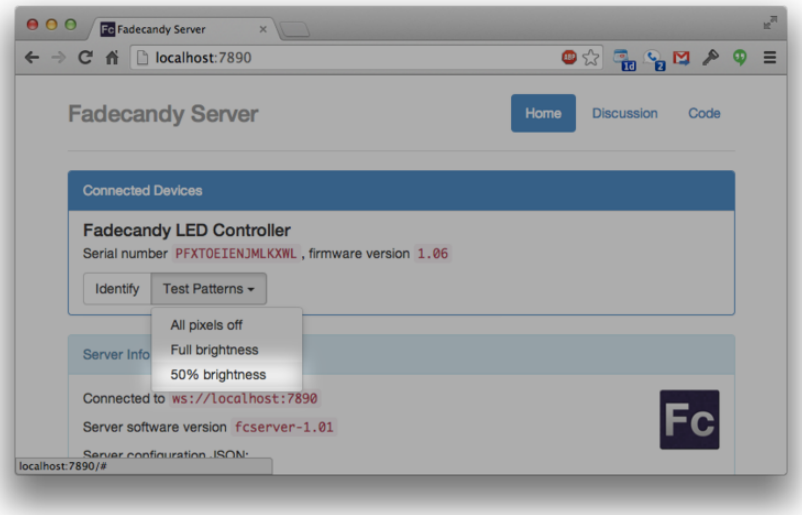
Try some examples

Now your hardware is ready! Let's try it out.

- If you don't still have fcserver running from earlier, start it.
- Plug power into your LEDs
- Plug the Fadecandy Controller into your computer with a Micro USB cable
- Open your web browser to http://localhost:7890 ()

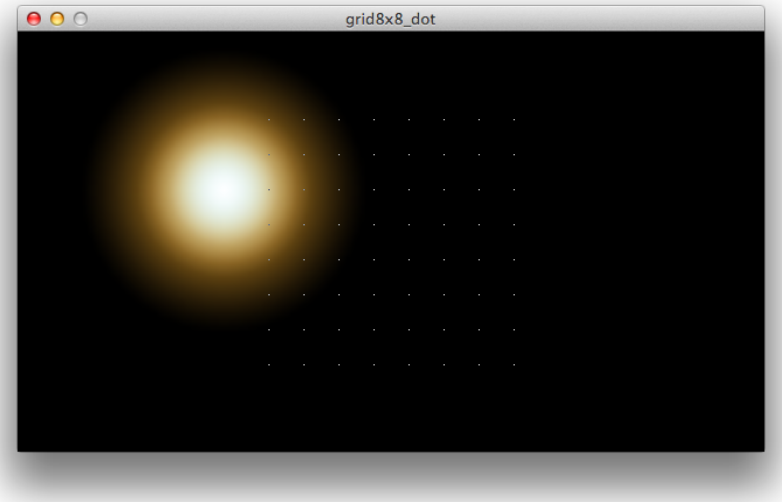
You should see your Fadecandy Controller listed under Connected Devices. If you're on a Mac, this should happen nearly instantly. On Windows it may take up to 30 seconds or so the first time you plug in your Fadecandy, since Windows will need to automatically install drivers.

Once you see your controller, try turning on your LEDs. We'll start with 50% brightness, since full brightness is extremely bright!



At this point, you should see your LEDs quickly fade on. Hurray. Now for something more interactive:

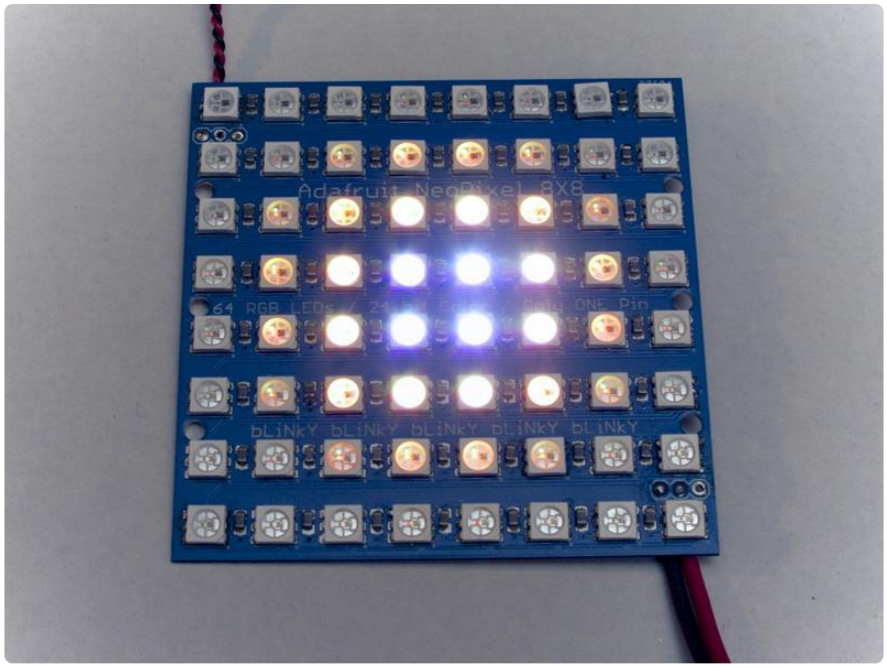
- Open Processing
- File -> Open
- Navigate to the processing folder inside the examples folder from the Fadecandy package.
- Open grid8x8_dot.pde, inside the grid8x8_dot folder.
- Press the Run button, in the top-left of the window.



The large orange glowing dot follows your mouse. Each of the tiny single-pixel dots represents one of your 64 LEDs. With Fadecandy's Processing library, it's easy to

create effects that draw to the screen and sample specific on-screen pixels for each LED.

As you move your mouse around the window, the dot follows along on your NeoMatrix.



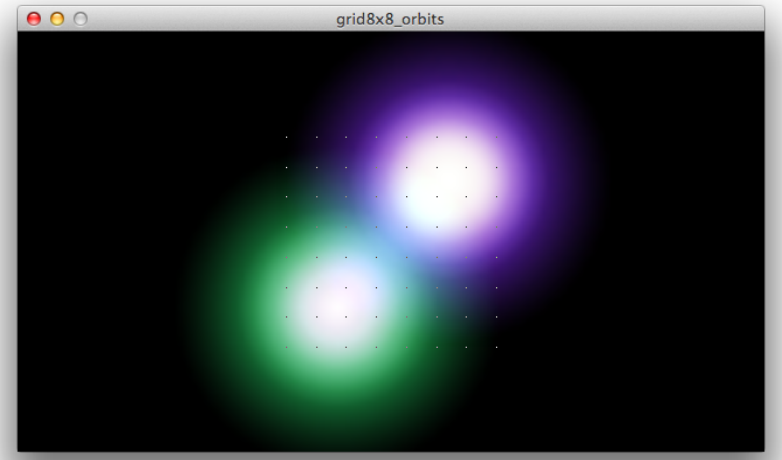
You can see the code for this example in the Processing window:

```
OPC opc;
PImage dot;

void setup()
{
  size(640, 360);

  // Load a sample image
  dot = loadImage("dot.png");

  // Connect to the local instance of fcserver
  opc = new OPC(this, "127.0.0.1", 7890);

  // Map an 8x8 grid of LEDs to the center of the window
  opc.ledGrid8x8(0, width/2, height/2, height / 12.0, 0, false);
}

void draw()
{
  background(0);

  // Draw the image, centered at the mouse location
  float dotSize = height * 0.7;
  image(dot, mouseX - dotSize/2, mouseY - dotSize/2, dotSize, dotSize);
}
```

If you need a refresher on Processing syntax, be sure to keep the reference () handy. The OPC object we're using is the Open Pixel Control client that comes with Fadecandy. There's a reference for OPC.pde () in Fadecandy's documentation.

If you've never used Processing, this may be a good time to work through some of the Processing Tutorials (). When you're ready, try modifying this example. Can you make the dot change size? Can you make it move on its own?

When you're done with grid8x8_dot, load the grid8x8_orbits example. We'll be needing it in the next section.



This example also uses images, but this time it uses two of them blended together. The vertical position of the mouse in the window changes the size of the orbiting dots, and they spin on their own at a fixed rate.

Play with light

If you're following along, at this point you have the grid8x8_orbits example running on your NeoMatrix. It looks like this:

It's a really simple example, but there's already a lot going on. There's a huge range in brightness, from pixels that are just barely on to pixels that are blindingly bright. Colors mix in the center. By themselves, the LEDs convey some of this, but there's also just an overwhelming sense of brightness that results from the light being so concentrated in each pixel.

In a sense, what you're doing now is staring directly into a light source. It's really extreme, and it's attention-grabbing, but there are a lot of other aesthetic options for a project like this. The lights could be illuminating an object, or they could be casting shadows, or they could be projecting a pattern onto a surface.

This is a great time to experiment with different materials, to get a sense for what the LED light looks like under different conditions. Maybe this will give you ideas about other materials you'd like to try, or other visual effects you want to design.

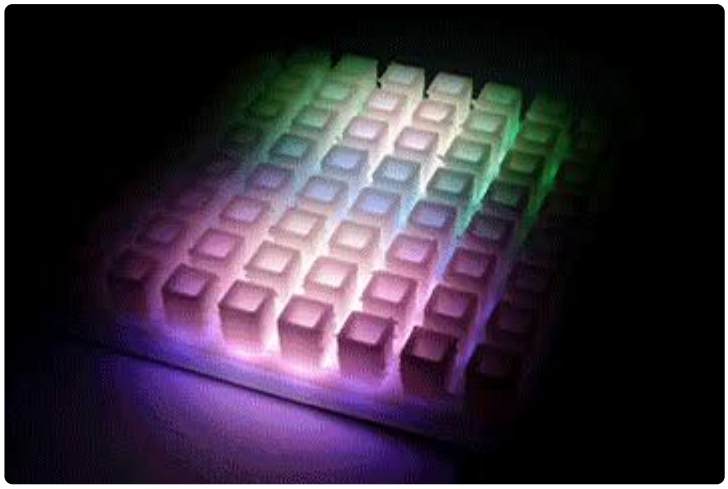
First, a simple piece of printer paper. I folded each edge down into a sort of box-lid shape, so that it's separated from the LEDs by a fraction of an inch. This gap gives the light some distance to spread out before it hits the paper.

The paper acts like a rear-projection screen. In signal processing terms, it's also acting like a low-pass filter () or blur. The hard edges of each LED are filtered out, and the result looks much more like what appears on-screen.

Next, let's take the same piece of paper, crumple it up, then flatten it. This adds a lot of interesting texture, and you can start to see how the moving light interacts with a complex 3D object.

Objects that cause interesting shadows and reflections are especially fun. If you have glass beads, marbles, or other small non-conductive objects, you can try piling them directly on top of the NeoMatrix. This is what it looks like with a pile of opaque white glass marbles:

Once you get an idea of what shapes look interesting, you can try fabricating your own. This is a shape I made with a Makerbot () 3D printer:



Write a sketch

At this point we've assembled a NeoMatrix and Fadecandy Controller, we tried some examples, and we put some materials in front of the LEDs to see how they look. Now let's try making a Processing sketch from scratch.

In this example, we'll make a simple particle system that creates expanding wavefronts in response to mouse gestures.

Here's how it looks:

(Direct link to the video ())

So that's shiny. Let's write some code!

To get the OPC.pde library () easily, we'll start by opening the template example included in Fadecandy's processing examples folder. Then we can save that example with a new name. The template starts you out with just enough code to connect to fcs erver:

```
// This is an empty Processing sketch with support for Fadecandy.

OPC opc;

void setup()
{
  size(600, 300);
  opc = new OPC(this, "127.0.0.1", 7890);

  // Set up your LED mapping here
}

void draw()
{
  background(0);

  // Draw each frame here
}
```

First, let's set this up to talk to an 8x8 pixel NeoMatrix. Change the setup() to look like this:

```
void setup()
{
  size(500, 500, P3D);

  opc = new OPC(this, "127.0.0.1", 7890);
  opc.ledGrid8x8(0, width/2, height/2, height / 16.0, 0, false);
}
```

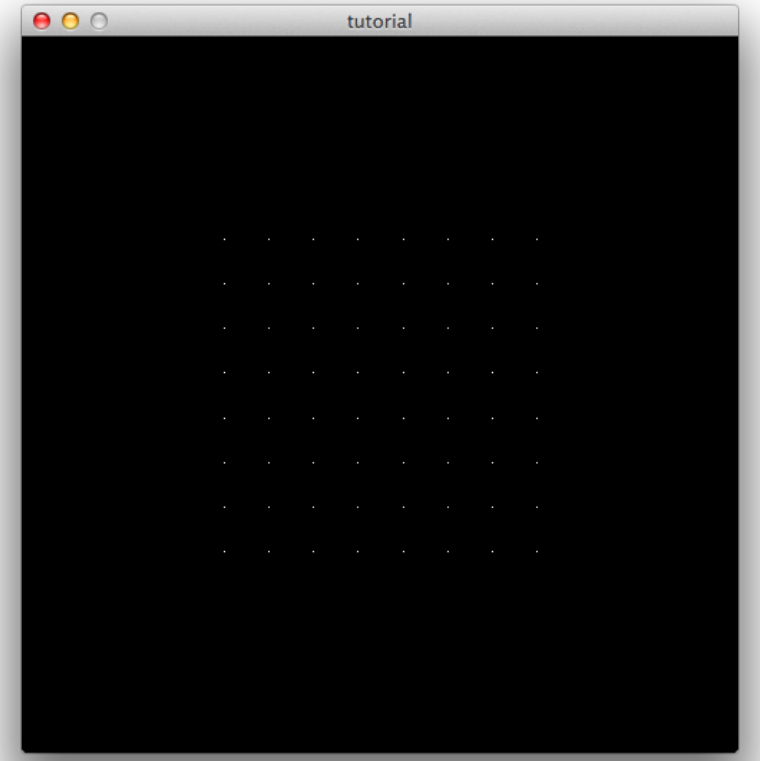
We made the window square, which is more convenient when we're working with a square NeoMatrix. And while we're at it, we'll select the P3D renderer for Processing. This makes it much faster at drawing images. Since we'll be drawing a large number of particles, this is really important!

Next, we added the ledGrid8x8() command. This is a shortcut for creating 8x8 LED matrices.

- 0 is the index number for the first LED in the matrix, since we connected this matrix to output 0 on the Fadecandy board. If we were using output 1, we would start the matrix at LED number 64.
- width/2 and height/2 are the center of the matrix. We place it in the center of the window.
- height/16 is the spacing between LEDs. 16 is double the size of our matrix, so this will make the matrix take up about half the height of the window.
- 0 is the rotation angle. We aren't rotating the matrix away from its default position, so the first LED will be in the top-left corner.
- false here states that the matrix does not zig-zag. Every row of LEDs points the same direction.

These parameters, as well as the other LED mapping functions available, are all documented in the OPC.pde reference ().

You can try running the sketch. At this point, our window should be blank except for the LED matrix:

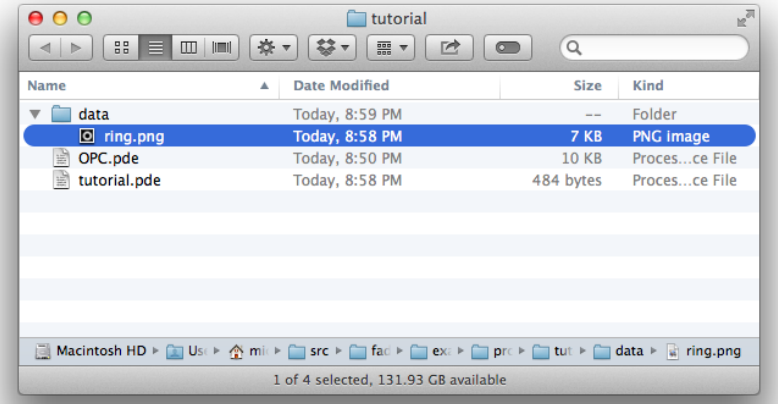


Now let's add an image. Even though LEDs like this are more about creating interesting light patterns than displaying video, images can be super useful as building-blocks for these light patterns. It may be helpful to think about images as if they're textures or color look-up tables.

In this case, we're creating a particle system that uses many copies of the same image to create a moving field of light. The light will look like expanding wavefronts, and each particle is a ring of light that we can tint however we like.

I made a simple ring texture in Photoshop. You can make your own, or save a copy of mine. If you do, make sure to save the image as a PNG file.

The texture needs to go in a data folder inside your Processing sketch's folder. Save it as ring.png inside this folder.



Now let's try out this texture! The sketch has a few new parts:

- colorMode ()() switches to the Hue, Saturation, Brightness color space.
- loadImage ()() loads ring.png into a PImage variable.
- drawRing() draws our texture with a particular hue, intensity, and location
- Temporary code in draw() puts a single ring at the mouse cursor location

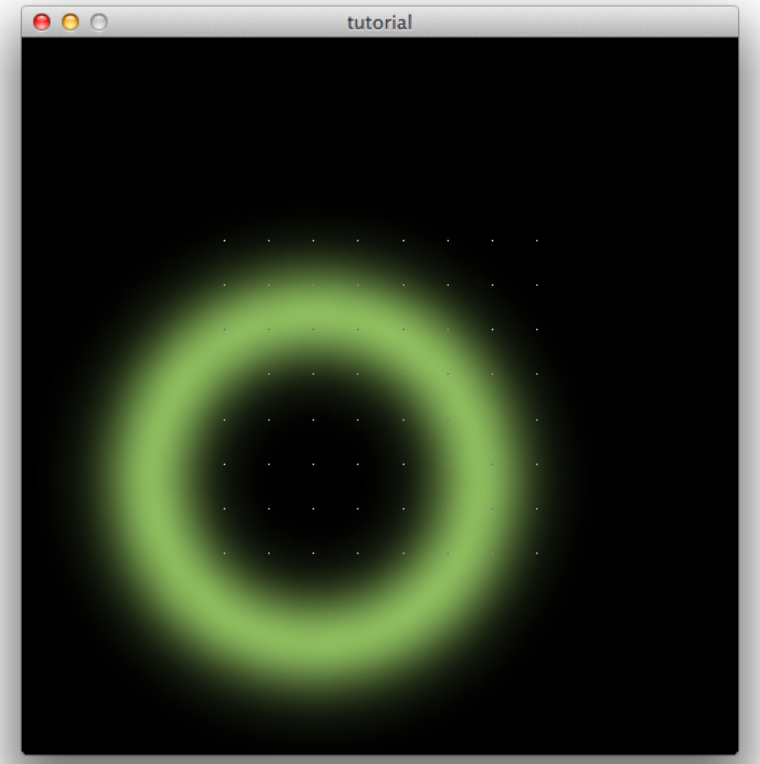
Try it out. Move the mouse around, and get a sense for how you can use textures to control the shape and the behavior of the LEDs.

```
OPC opc;
PImage texture;

void setup()
{
  size(500, 500, P3D);
  colorMode(HSB, 100);
  texture = loadImage("ring.png");

  opc = new OPC(this, "127.0.0.1", 7890);
  opc.ledGrid8x8(0, width/2, height/2, height / 16.0, 0, false);
}

void drawRing(float x, float y, float hue, float intensity, float size) {
  blendMode(ADD);
  tint(hue, 50, intensity);
  image(texture, x - size/2, y - size/2, size, size);
}

void draw()
{
  background(0);
  drawRing(mouseX, mouseY, 25, 80, 400);
}
```

You might also notice a little bit of flicker at the edges, where the LEDs that are just barely on rapidly flick between on and off. This is a side-effect of Fadecandy's temporal dithering algorithm. This is how it can simulate brightness levels that are dimmer than the dimmest level the WS2811 controller supports in hardware. The

flicker can be distracting when you're looking very closely at the LEDs, but if you're using a diffuser or you're using the LEDs as a light source for your art, this flicker is almost never noticeable.

If you do find the flicker bothersome, you can change the fcserver color management configuration. The documentation () describes how you can reconfigure the color correction curves to have a linear section at the dark end, which completely eliminates this flicker at the cost of reduced accuracy in rendering very dark colors.



Now let's make a particle system! I added a Ring class to manage each particle. The individual particles have a lifespan during which they expand and fade. Particles randomly respawn at the mouse cursor location. The mouse cursor location itself is smoothed so that we don't see such discrete bursts of particles every time the mouse moves by a pixel.

I decided to use mouse speed and direction as inputs. Mouse speed controls the intensity that new particles spawn with, and mouse direction controls their hue. As you move your mouse or swipe your touchpad, the speed and angle of your gesture creates a unique pattern of light. The expanding wavefronts from each gesture interact with each other, forming new patterns of light and color.

The Ring class has a respawn() function which computes a new particle's intensity and

hue based on the speed and angle of mouse motion. As parameters, it takes the X and Y coordinates for a previous and a new mouse position.

```
class Ring
{
  float x, y, size, intensity, hue;

  void respawn(float x1, float y1, float x2, float y2)
  {
    // Start at the newer mouse position
    x = x2;
    y = y2;

    // Intensity is just the distance between mouse points
    intensity = dist(x1, y1, x2, y2);

    // Hue is the angle of mouse movement, scaled from -PI..PI to 0..100
    hue = map(atan2(y2 - y1, x2 - x1), -PI, PI, 0, 100);

    // Default size is based on the screen size
    size = height * 0.1;
  }

  void draw()
  {
    // Particles fade each frame
    intensity *= 0.95;

    // They grow at a rate based on their intensity
    size += height * intensity * 0.01;

    // If the particle is still alive, draw it
    if (intensity &gt;= 1) {
      blendMode(ADD);
      tint(hue, 50, intensity);
      image(texture, x - size/2, y - size/2, size, size);
    }
  }
};
```

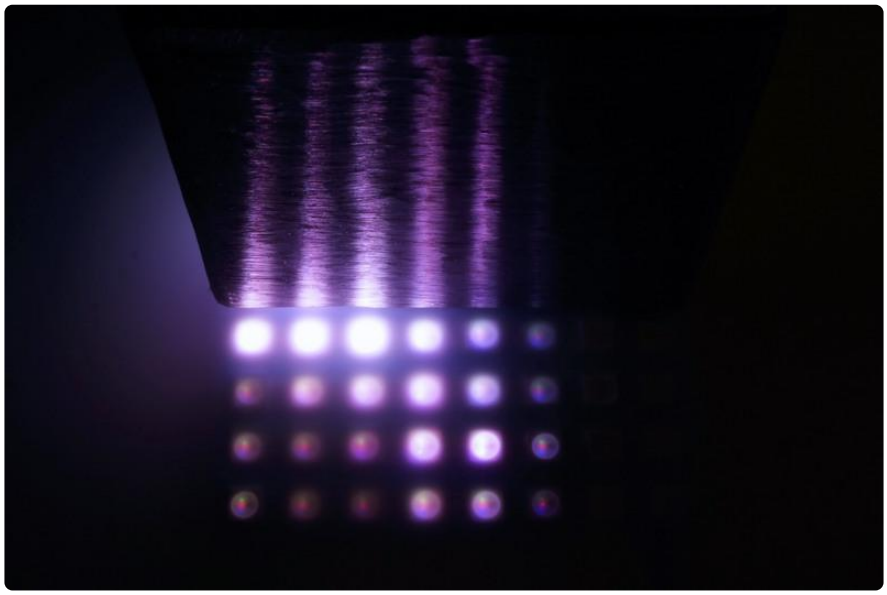
Now we need to define the rest of the program to drive a system of these Ring particles. Aside from the Ring class, this is what the rest of our program looks like:

```
OPC opc;
PImage texture;
Ring rings[];
float smoothX, smoothY;
boolean f = false;

void setup()
{
  size(500, 500, P3D);
  colorMode(HSB, 100);
  texture = loadImage("ring.png");

  opc = new OPC(this, "127.0.0.1", 7890);
  opc.ledGrid8x8(0, width/2, height/2, height / 16.0, 0, false);

  // We can have up to 100 rings. They all start out invisible.
  rings = new Ring[100];
  for (int i = 0; i &lt; rings.length; i++) {
    rings[i] = new Ring();
  }
}
```

```
void draw()
{
  background(0);

  // Smooth out the mouse location. The smoothX and smoothY variables
  // move toward the mouse without changing abruptly.
  float prevX = smoothX;
  float prevY = smoothY;
  smoothX += (mouseX - smoothX) * 0.1;
  smoothY += (mouseY - smoothY) * 0.1;

  // At every frame, randomly respawn one ring
  rings[int(random(rings.length))].respawn(prevX, prevY, smoothX, smoothY);

  // Give each ring a chance to redraw and update
  for (int i = 0; i &lt; rings.length; i++) {
    rings[i].draw();
  }
}
```

And that's it! You can see the complete source code for this example () in GitHub.

Keep trying new things



If you made it this far, you have all the tools necessary to start exploring the creative possibilities of programmable LED lighting.

There's more documentation on the Fadecandy project available in its GitHub repository ():

- Introduction ()
- Processing OPC client ()
- Server configuration ()
- Open Pixel Control protocol ()
- WebSocket protocol ()

Where else is there to go from here?

- Try more examples.
- Modify examples designed for other LED configurations so they work with yours.
- Design interesting shapes and find new materials for the light to interact with.
- Try using more lights. One Fadecandy board can control up to 512 pixels, and you can connect many Fadecandy boards to one computer using USB hubs.
- Make your art portable with a Raspberry Pi or other single-board computer.
- And of course, writing more visual effects.

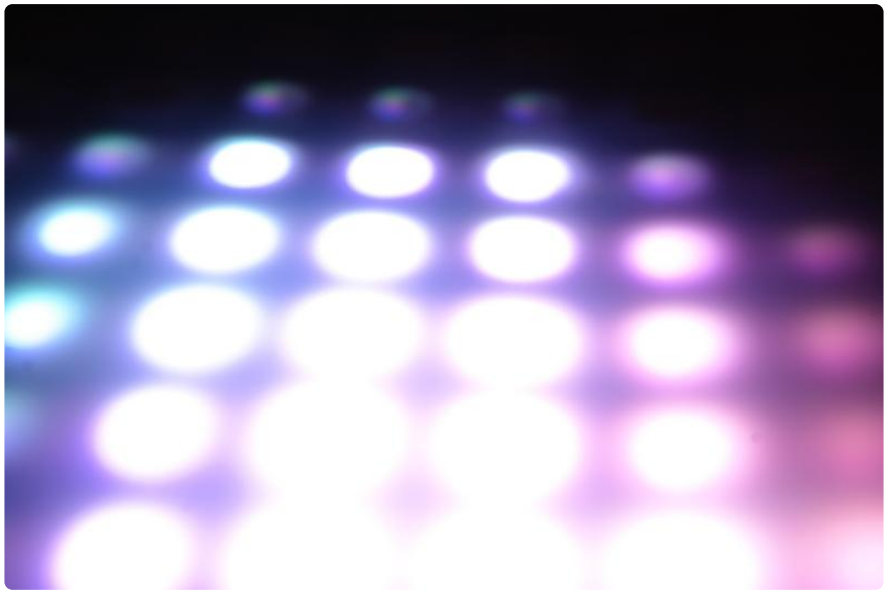
There are also many different strategies for writing effects. This example covered using Processing and mapping each LED to a pixel on the screen, but there are plenty of other options. The Fadecandy project already has examples for browser-based effects (), Node.js (), and Python (), and it's easy to add support for new languages.

What about pixel mapping? It works for some kinds of art, but other kinds of art may be easier with a 3-dimensional mapping or something custom. Work is in progress to make more kinds of LED mappings easy, and to support automatic mapping using computer vision.

If you're interested in contributing to the Fadecandy project, check out these resources:

- Discussion group ()
- GitHub page ()
- Introduction blog post ()

Thanks for participating!

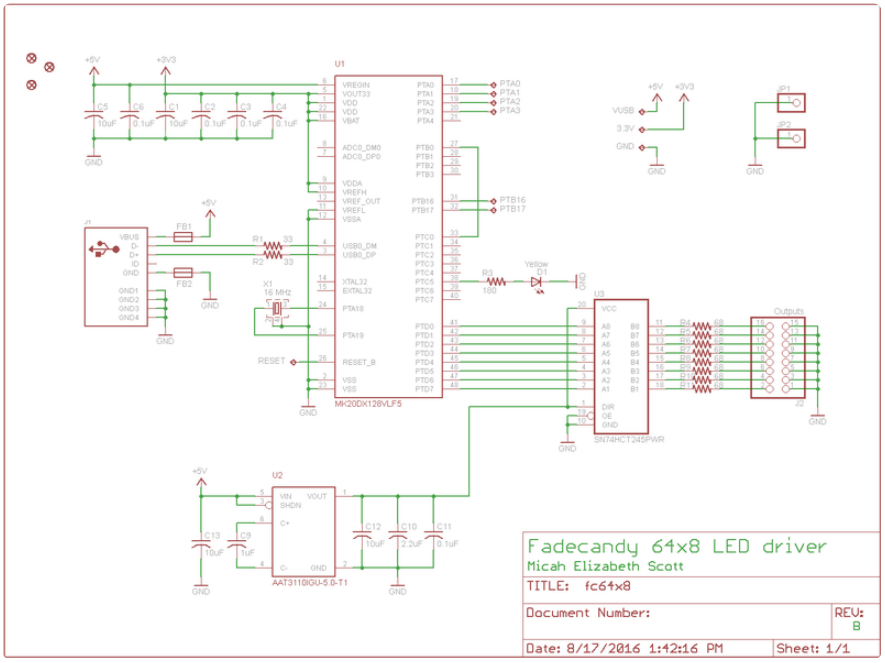


Downloads

Datasheets & Files

- Github Repository for FadeCandy ()
- Fritzing object in Adafruit Fritzing library ()

Schematic



Fabrication Print