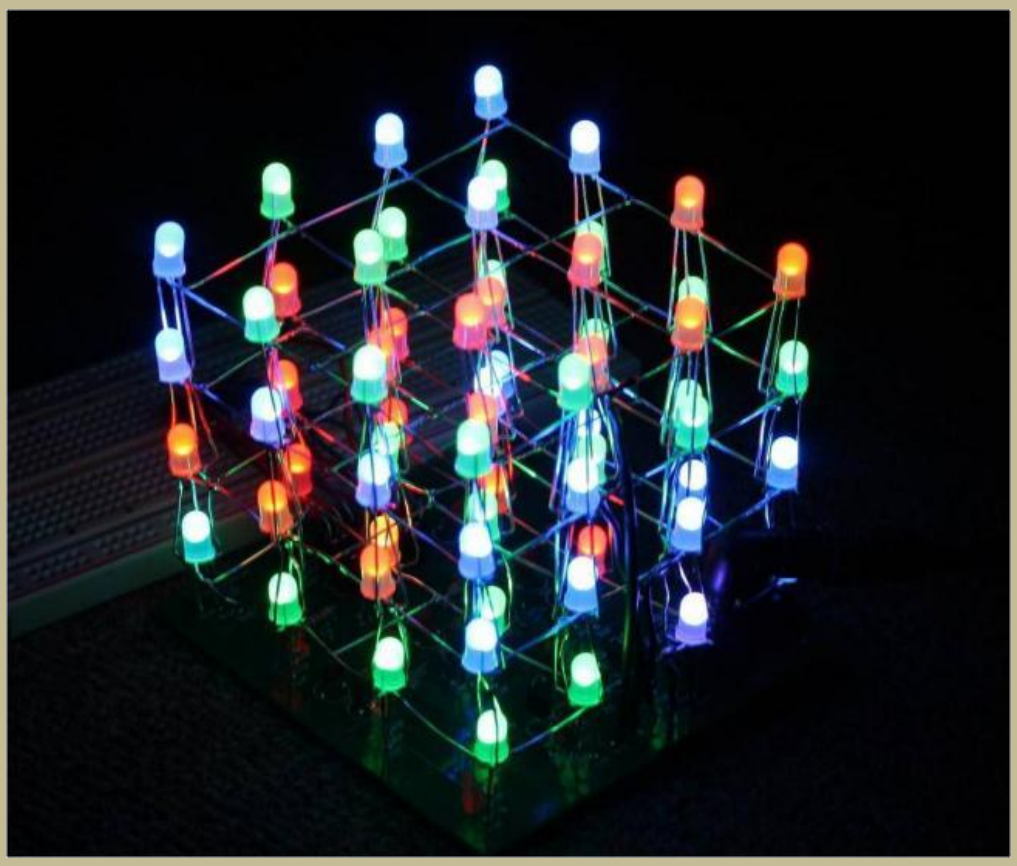
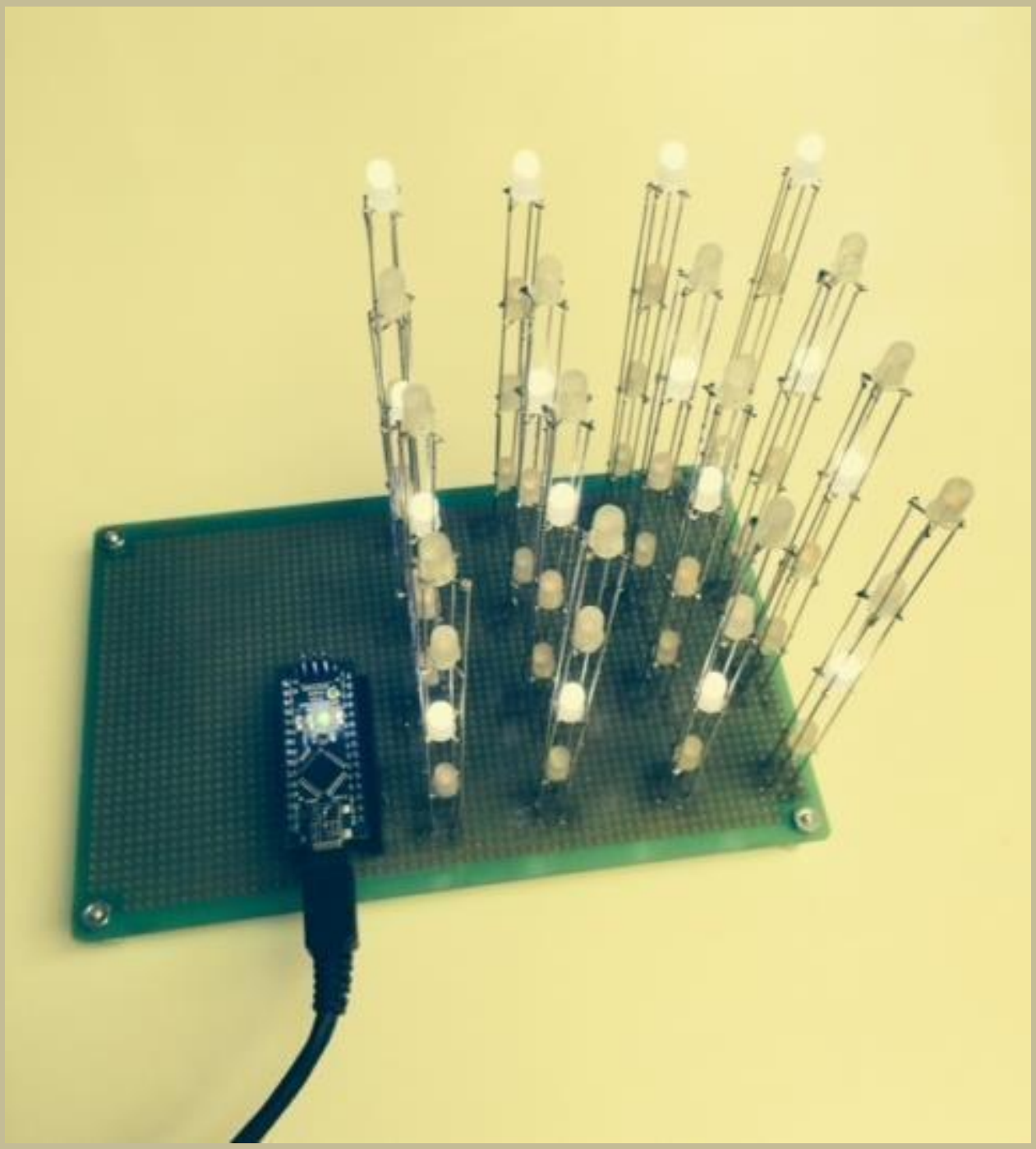
RGB LED Cube: 2218847



Time Required: 10 hours depending on experience
Experience Level: Advanced
Required tools and parts: Soldering iron, Wire cutters, Stranded or solid hookup wire

Charliecube



What is it?

A 3-dimensional lattice of LEDs assembled together to display all sorts of colorful patterns and information. Think of it as a low-resolution 3D monitor, which also doubles as a structural art piece.

How is it built?

This DIY project might be the simplest design of tri-color LED cubes out there. Without using complicated components such as shift registers, decade counters, reference shifting transistors or any other components to control LEDs, this project focuses on a much simpler, cost effective design. Other LED cubes with extra components can cost upwards of $150, leaving you with an empty wallet and a schematic-induced headache. This LED cube utilizes Charlieplexing and can operate using only 16 digital pins of an Arduino controller with no additional hardware besides the RGB LEDs themselves.

What is an RGB LED?

A 5mm tri-color LED with separate red, green, and blue chips inside. Each consists of having 4 separate color pins – Cathode being the longest pin. Use this one LED to show 3 different colors or to pulse width modulate all three and get a swirl of colors. Simply apply a low voltage, low current signal to any of the color pins and connect a return, or ground, path to the cathode pin. Any RGB that uses a diffused lens blends colors more effectively instead of appearing as 3 distinct LEDs.

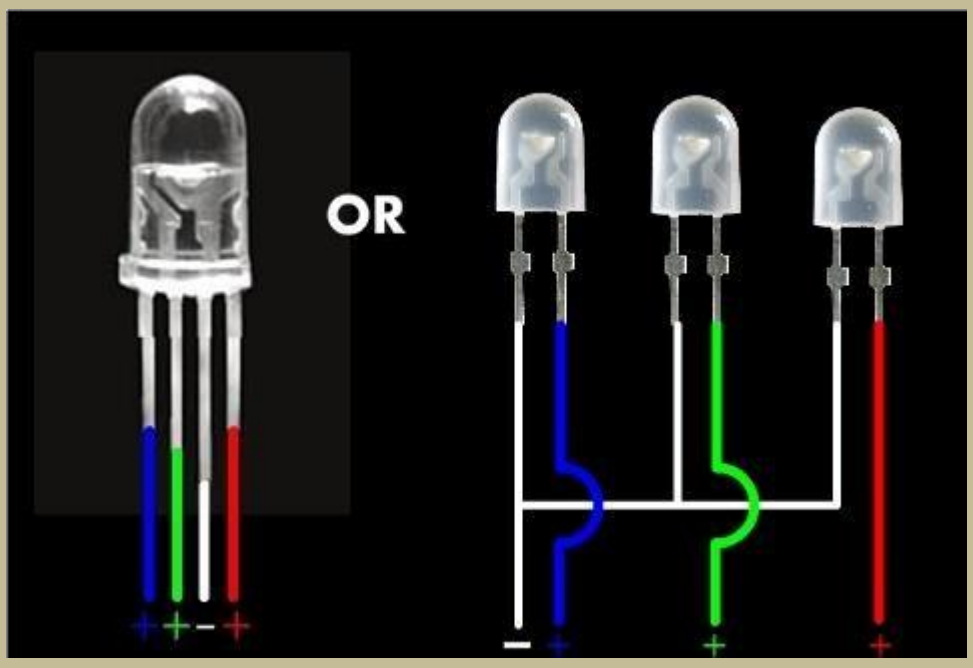


Figure 1: LED individual equivalent of RGB LED

What is charlieplexing?

It is driving many LEDs with only a few pins of microcontroller. This can be useful when needing to display a fancy dot matrix without many other components.

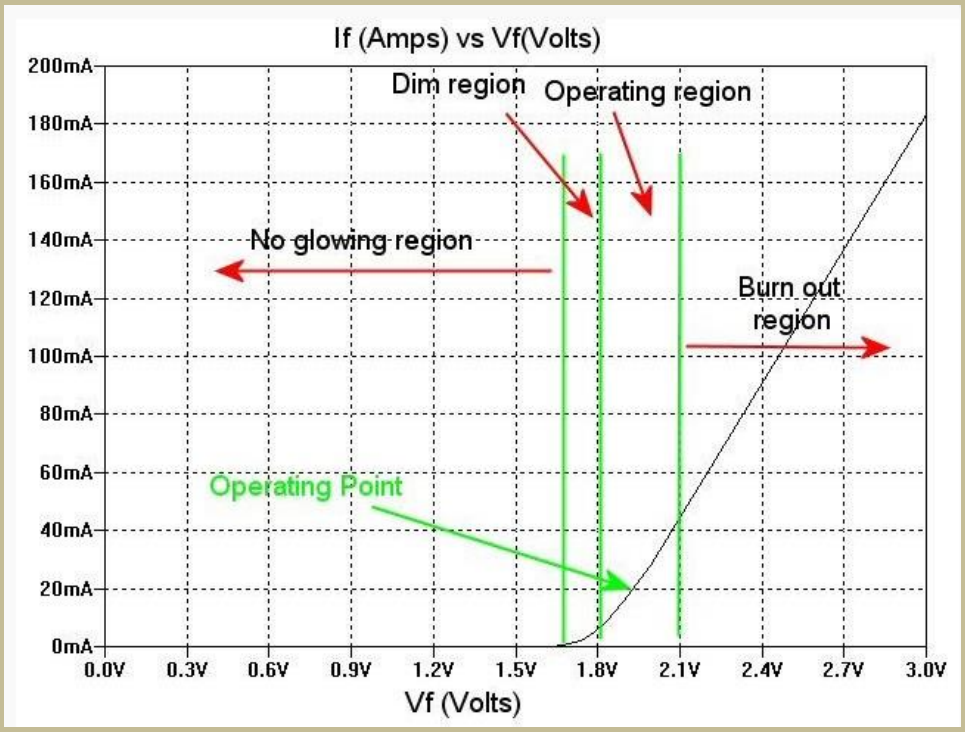


Figure 2: Foreword Current vs Foreword Voltage of LED

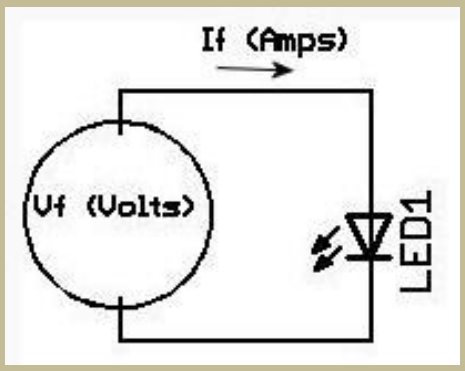


Figure 3: LED Operation

First it is important to understand how LEDs work. All LEDs have an operating point. When an amount of voltage flows across the LED a current will flow through the LED as well. These two entities are inseparable and are at the root of all electronics. Current needs to reach this operating point, usually in milliamps, for the LED to turn on. This is achieved with the use of resistors. Without reaching the operating point, the LED will remain off. There is also something interesting to note about LEDs -- in an LED's case, the relationship between voltage and current is exponential, meaning that little voltage applied will generate greater changes in current. Be careful when playing with voltage, because too much might damage the LED and you!

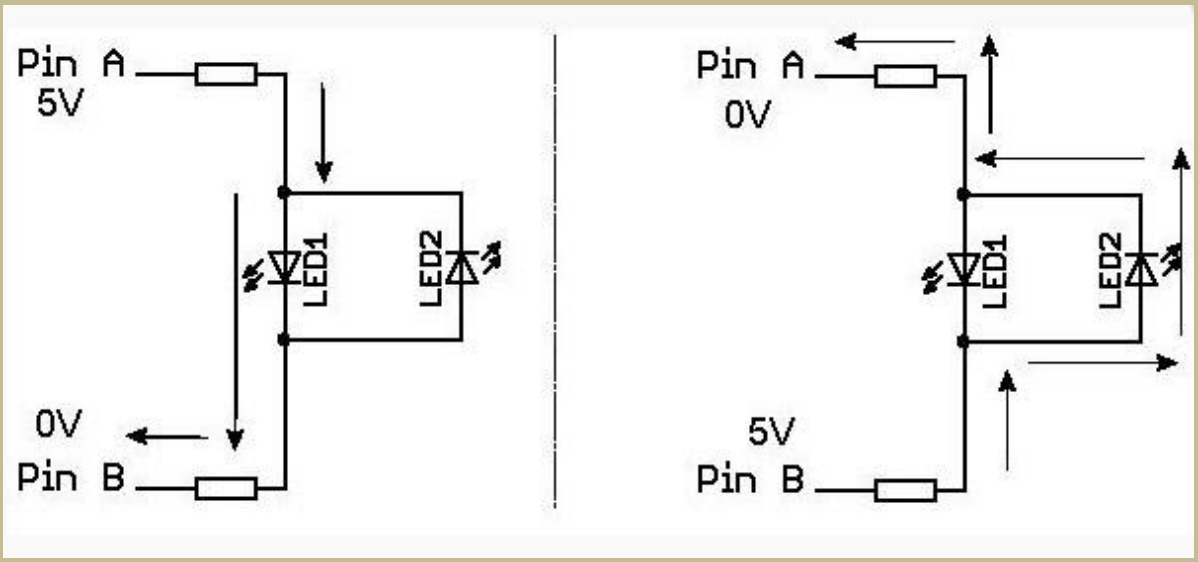


Figure 4: Example 1

Now, let's get into the basics of charlieplexing. Our Arduino has several digital pins used to output 5V. When connecting pins together, by setting one to 5V and another to 0V, we can send a current in a very specific direction to do really creative things. In our case we use current to drive LEDs. In the above example, when Pin A is ON and Pin B is OFF, only LED 1 lights up. LED 2 stays off because it is in reverse-bias. If you don't know what that means, reference this article. If we flip the pins where Pin B is ON and Pin A is OFF, we see the counter effect: LED1 is off and LED 2 lights up. This is a controlled way to light individual LEDs

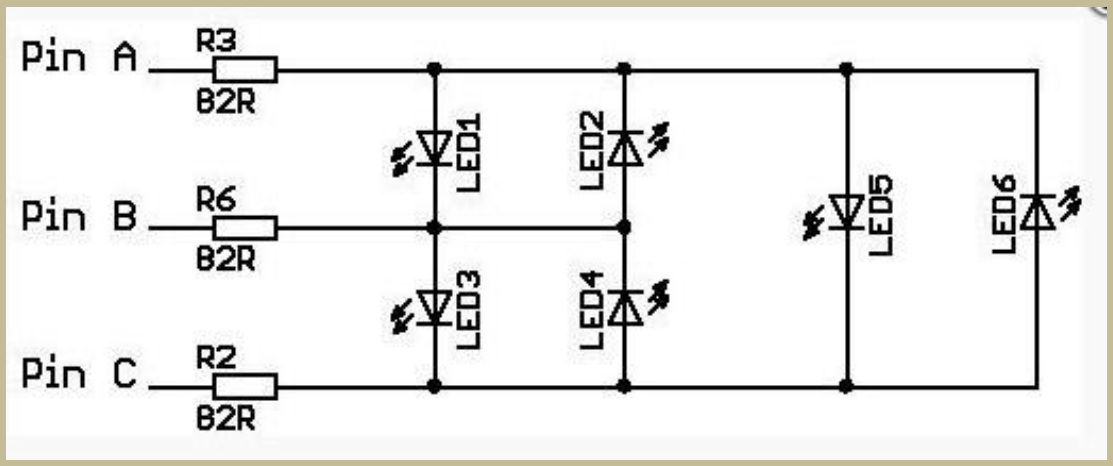


Figure 5: Example 2

In a more complex scenario, we'll take a look at the figure above. Take a look at the 3 pin combinations: A-B, A-C, B-C. Each has a pair of LEDs going in either direction. If we turn Pin A ON, Pin B OFF and disconnect Pin C, we see LED 1 turn on and everything else is off. If we turn Pin A OFF and Pin B ON, while keeping Pin C disconnected, only LED 2 turns on. If we turn Pin A ON and turn Pin C OFF with Pin B disconnected, LED 5 lights up. We can continue this pattern until each LED activates, but there is something interesting to note. When you, for example, turn Pin A ON, Pin C OFF, and disconnect Pin B, current travels through LED 5 as well as LED 1 and LED 3 in series. The reason only LED 5 turns on and

not LED 1 and LED 3, is due to not enough voltage passing across LED 1 and LED 3. In fact, this path gets as much voltage total as LED 5, but it needs to equally split the voltage across LED 1 and 3, resulting in too little current passing through each, so neither lights up. We can see this when Pin A is ON, Pin B is OFF, and Pin C is disconnected also. Only LED 1 lights up, even though there is valid path for current to travel through LED 5 and LED 4. Yet neither can get the sufficient current it needs.

With N pins on a microcontroller, you can control N * (N-1) LEDs. For 3 pins this means 3 * (3-1) = 3 * 2 = 6 LEDs. In our case we have 64 RGB LEDs which means 64 * 3 = 192 individual LEDs. We would need at least 15 pins to drive all of these. We'll use 16 pins because it is a dividend of 64, which makes things a lot easier.

Now we must clarify what is meant by turning Pins ON, OFF, and disconnected. For our microcontroller, we can program pins to be in one of three states: HIGH, LOW, or INPUT. High and low are two output modes that can either supply 5V or 0V. Use these to construct you signal paths, where current will flow from the larger to the smaller voltage. Input is a high impedance mode, which can be used to block incoming current into a pin. Imagine as if there is a very large resistor placed in front of a pin. No current will enter this pin and the pin won't act as an output. This is useful when attaching wires to a pin but you would like to "disconnect" the pin by not allowing any current to flow into it.

Lastly, we must mention a phenomenon that occurs when supplying just the right amount voltage to an LED. Notice in the LED operating chart that there is a dim region. A dim region represents an amount of voltage that when supplied, the LED will faintly light up but it won't glow like in typical application. It is important to know that the dim region and operating point will vary from each color of LED, because each color will have a different operating region. Red will need the least amount of voltage to glow, green the second least and blue will need the most amount of voltage. Each color in respect to red, green, and blue has a higher dim region and operating point than the last. You might find that red LEDs will glow dimly when blue is illuminated. This can be fixed if the forward voltage of any different colored LED is roughly the same as the operating current. This would require more pins than what is available because the blue/green charlieplex matrix would need to separate from the red one.

Parts List

Quantity	Item	SKU
64	RGB Common Cathode LED	2125181
1	PCB Board	206587
1	Arduino Nano (Rev. 3)	2121148
4	Standoffs	393474
4	Hex Nuts	40943
2	Female Header 16-Pin	308592

Assembling a Column

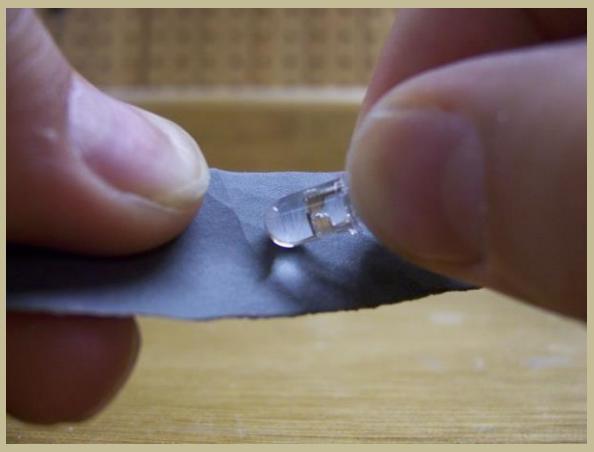


Figure 6: Diffusing the LED

The LEDs come in a clear lense, but need to be diffused to widen the viewing angle. Begin by hand sanding the entire area of each LEDs plastic dome with 200 grit sandpaper. It is important to consider the corsness of the sandpaper, as too fine will take a long time to complete and too course will shave much of the pastic off. 200 grit is a recommended approach.

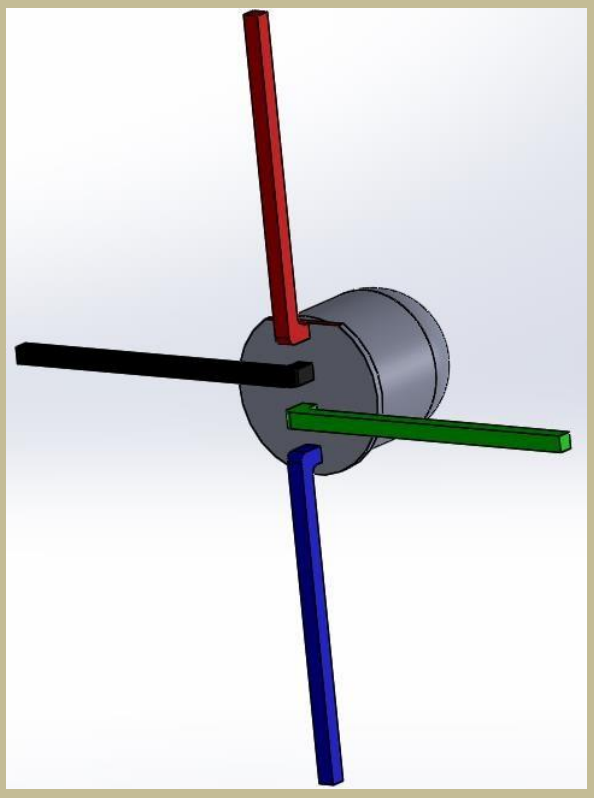


Figure 7: Bending the legs

Once all the LEDs are diffused, take all four of the leads and bend them out in 90° to each other, according to the following diagram (Fig. 7) The black pin represents the cathode. Also, note that the flat spot on the base of the LED is nearest to the red pin and you should hold each LED in the same orientation with the flat spot pointing up, like in the diagram. Be sure to bend the leads the same way for each LED so it is easier to put together later.

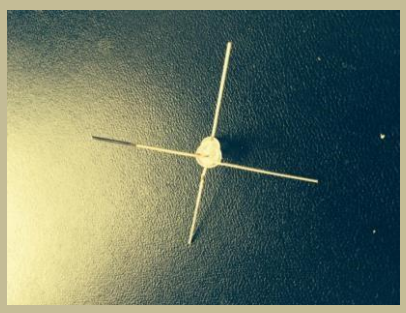


Figure 6a: Marking the ground pin

Use a black felt marker to mark the ground pin.

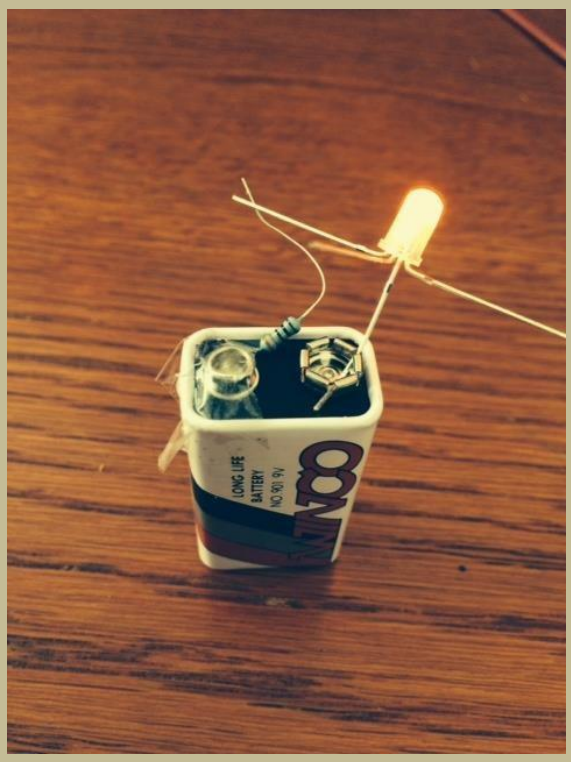


Figure 8: Testing the LED

Include a testing procedure to check if all the colors work in the LEDs. This is easier when the leads are spread apart. Use a 9V battery in series with a 330Ω resistor, by taping one end of the resistor to the positive(+) terminal and keeping the ground leg of the LED on the negative(-) terminal. Gently touch each leg with the resistor while keeping the ground leg of the LED touching the (-).

Red should appear brightest and blue the dimmest.

It is possible to use a larger resistor than 330, but not lower. Using too low of a resistance value may blow one or more colors inside the LED. Using to high of a resistor might make certain colors hard to see.

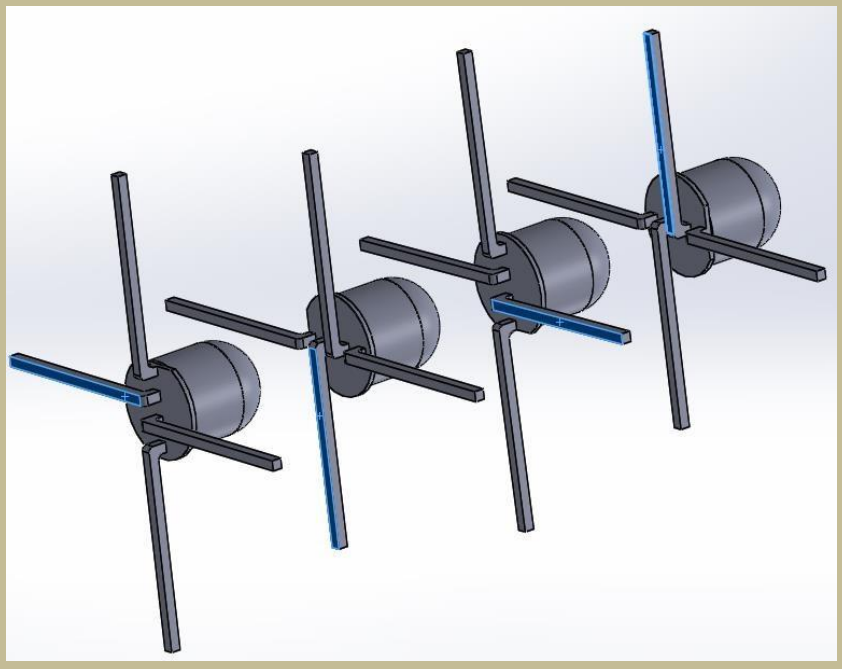


Figure 9: Lining up the LEDs

In each column, the LEDs will be all rotated 90° from each other. So keep track of which way the leads are rotated. If you have lost track of which rotation you are on, reference the marked leg of the LED. The blue leg in the image represents ground. Notice each LED's flat spot is turned 90 degrees relative to its neighbor.

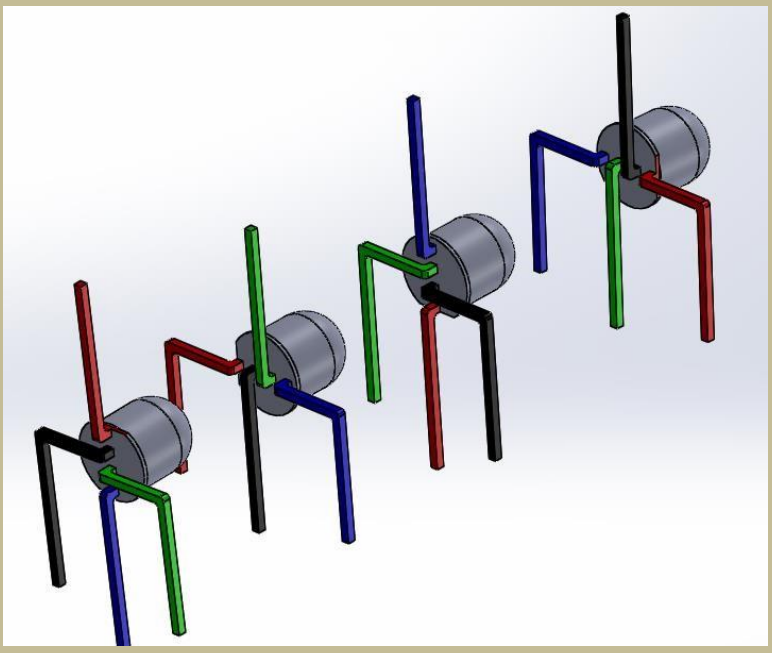


Figure 10: Bending the legs for alignment

The legs are bent to look like a pitchfork. The PCB board is used as the jig where the LEDs are placed to create the straightest columns possible. The bent legs are spaced an even 3 holes apart from each other, where each LED is spaced 12 holes away from each other. Use pliers to grip the legs up to the base and bend the legs. It just so happened that the pliers used for this demonstration were 3 holes apart. The spacing isn't so important. These distances are recommended, but can be changed to meet your own wants. In addition, you can use 10 pieces of masking tape level the bulbs with the board. Refer to this figure to see how each LED is aligned in the column. Check out this video to see how

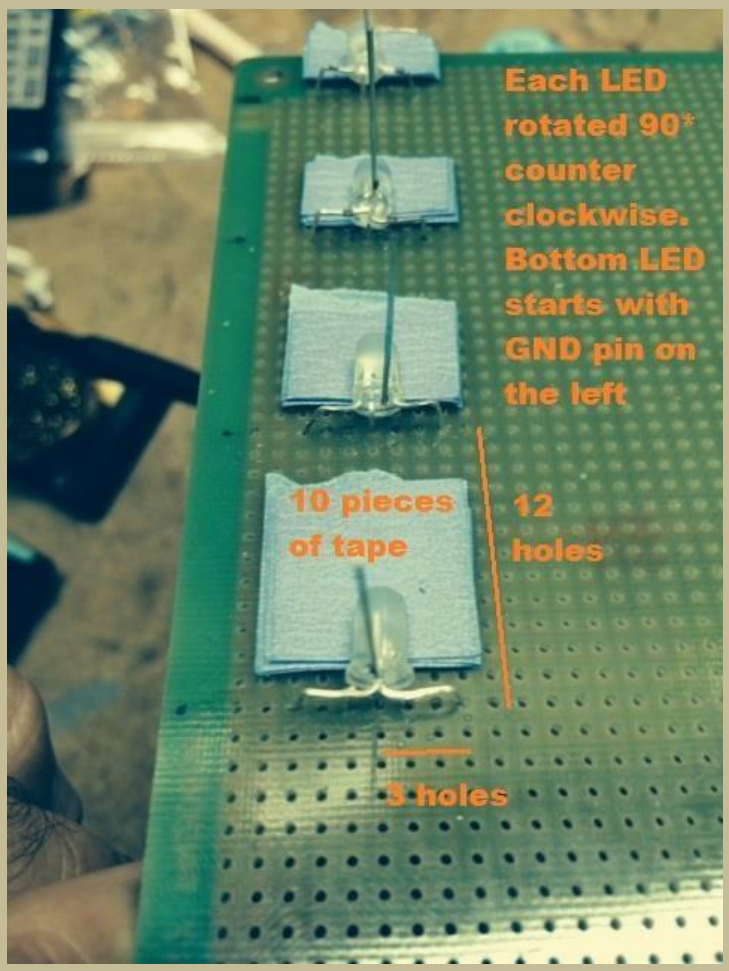


Figure 11: Placing LEDs in the jig

Caution: Do not let the iron touch the LED for more than a couple of seconds. Heat can damage LEDs.

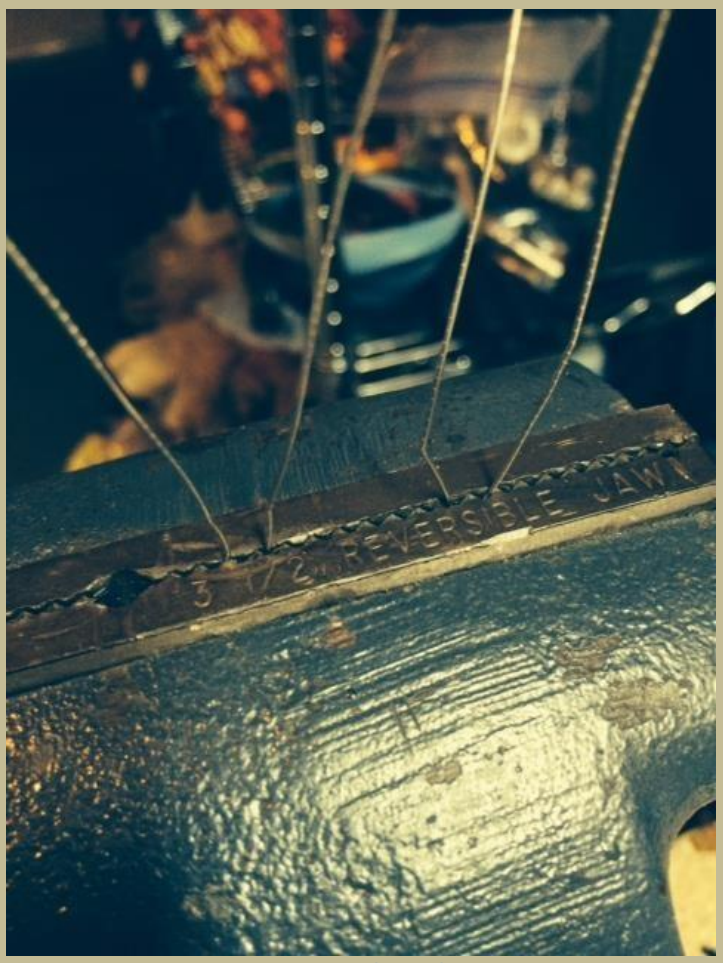
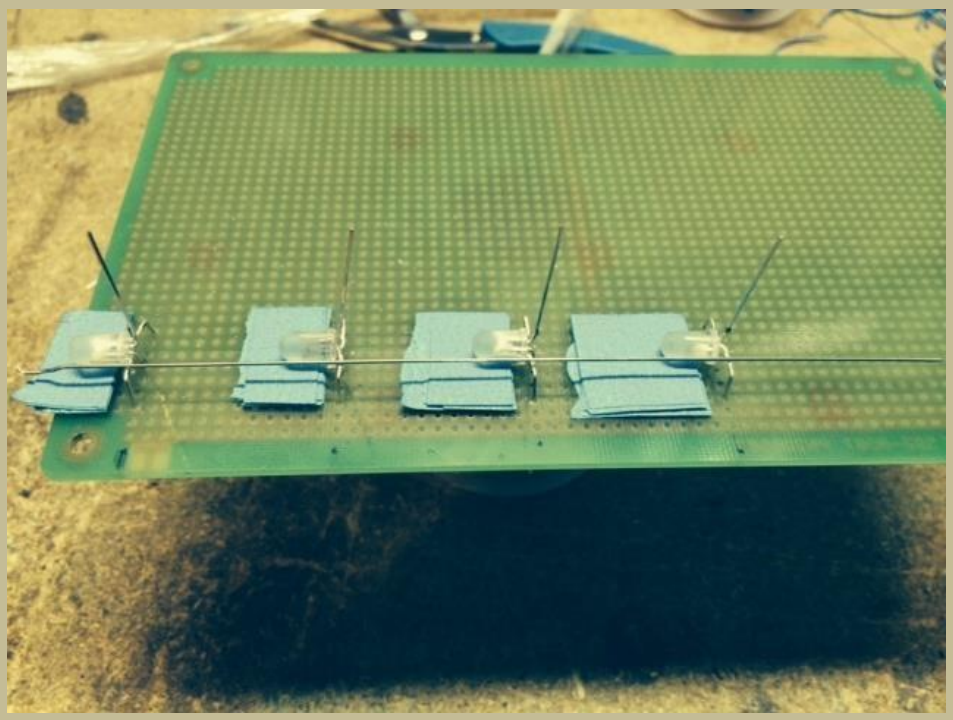


Figure 12: Wire tightened in a vice

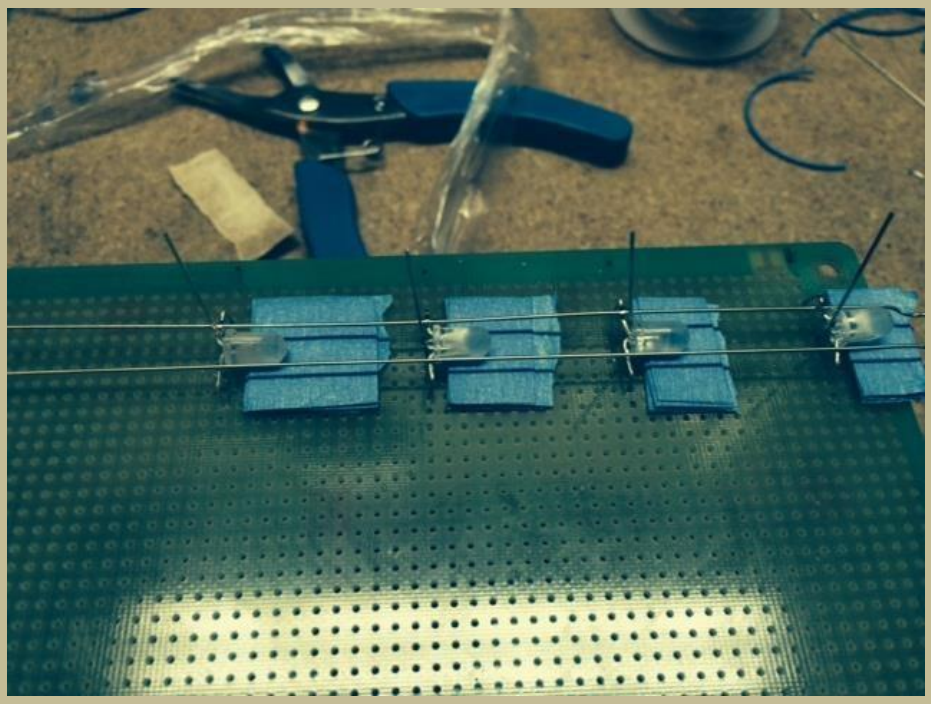
Make straight wire!

Cut some wire about 4.5” to 5” long and strip the insulation. You can pull on the wire with pliers in each hand or use a vice grip and one set of pliers to straighten the wire. Watch the video to see how…



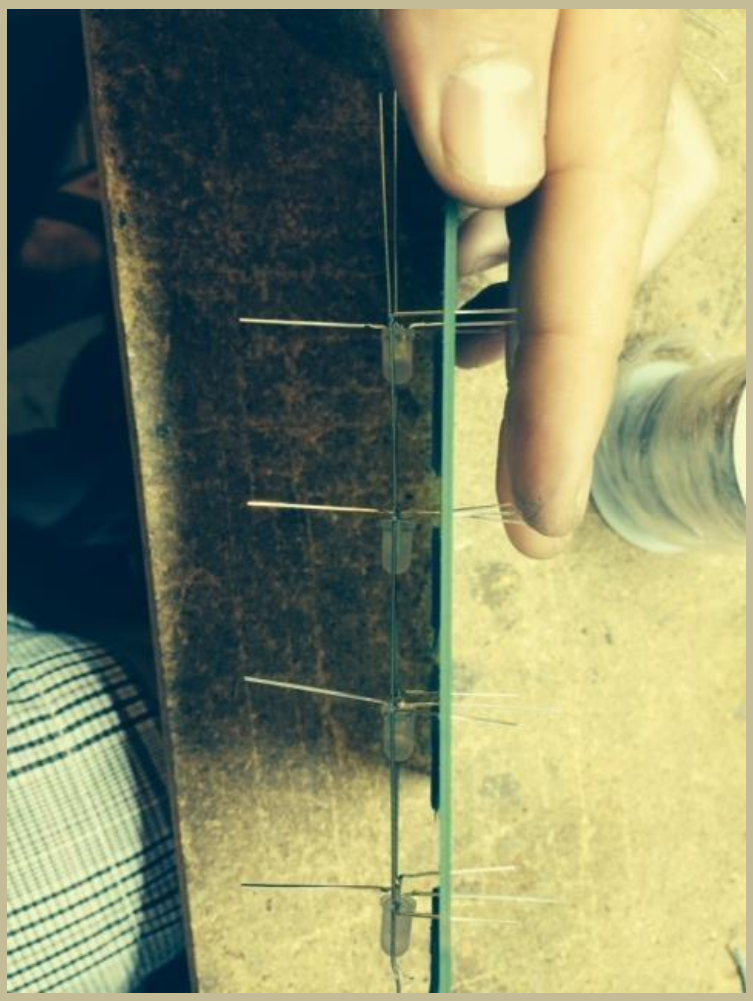
While keeping the LEDs in place, lay a straight wire across one side of the column. Solder it to the legs of the LEDs and repeat for the other side.

Figure 13: Laying the first support



Solder another piece of straight wire to the opposite side of the column.

Figure 14: Laying the second support



Flip the board to its back side and push the column out of the jig. Use two hands when pushing to keep it from bending.

Figure 15: Taking the column out of the jig

Figure 16: Laying the third support

Lay the column on one side. Use a free hand to lay a straight piece of wire across the middle legs of the column. Solder the wire into place.

Figure 17: Laying the third support

Solder a piece of wire to the last side of the column. Repeat for the other 15 columns.

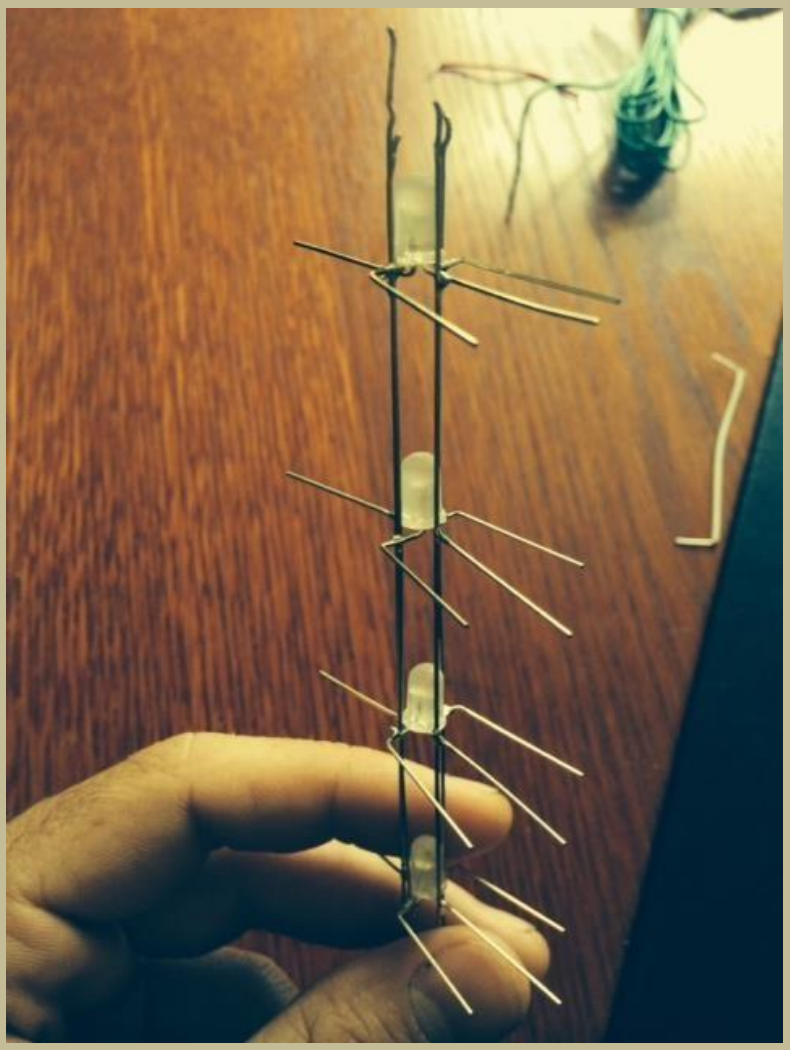


Figure 18: One column with all the trimmings in place

Here's how the column should look like after soldering all 4 pieces of straight wire to the legs. Do not cut the legs or any excess wire at this point. Each column will need to be tested to ensure proper functionality after soldering.

While in the photo it looks like only 2 straight wires were soldered, really there are 4.

Testing the Columns

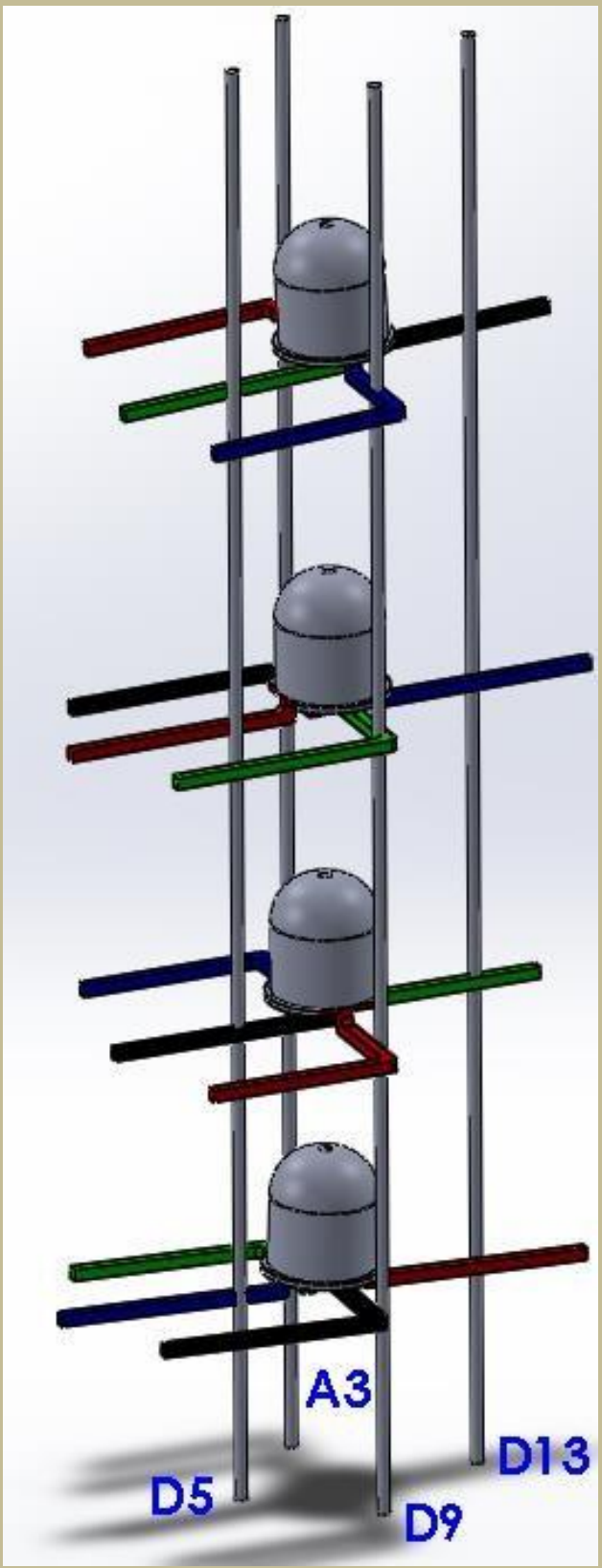


Figure 19: Hookup diagram

Test the columns to make sure LEDs weren't damaged in the soldering process. Included is a test sketch and a hookup diagram. Here's a video to show how the sketch works…

Wire the legs of the column to the corresponding pins of the Arduino. Assemble each column into breadboard one at a time, run the legs to the Arduino via wires, then run the sketch by copying or opening it up in the Arduino IDE.

Testing:

Here's some code to test all the LEDs in a column after soldering. It will start at the bottom-most LED and cycle through possible colors moving up to the next LED. Make sure to align the column specified by the image to test brightness and LED orientation.

Tip: Wear sunglasses when running software! While the code will not damage the LEDs, staring at the LEDs for an extended amount of time will damage your sight.

Disclaimer on Light Bleeding:

When testing an individual column, don't worry about other LEDs bleeding colors.

Also note that there is no actual ground, but just a potential difference, or voltage drop, across the different pins. Each pin is 'on' but when tested with a multimeter notice how each pin combination is set to a different voltage resulting in different amounts of current moving through the pins at all times. "Ground" is set to a lower potential than all the rest, so that's why we see some light bleeding because the voltage difference across the same pin combination for a different LED might be enough for it to dimly turn on. There is no real ground, but only higher and lower potentials across different pins resulting in different colors turning on at different intensities.

An analogy: Imagine as if there are 4 different waterfalls rushing water at different heights. The highest waterfall will achieve the fastest water falling, assuming a common source. The fastest water is the strongest and will therefore turn the water wheel located at the bottom. The other weaker waterfalls will still fall on their water wheels, but will offer a lesser strength to turn them, resulting in slow moving wheels with respect to the fastest one. They are still turning, but slowly. The same case is in our LED project except water speed is current, height is voltage, and the wheel is the color that turns on. A higher potential, or voltage, will look the brightest.

For example:

Pin combination 1-2 (Red): Voltage = 0.8V

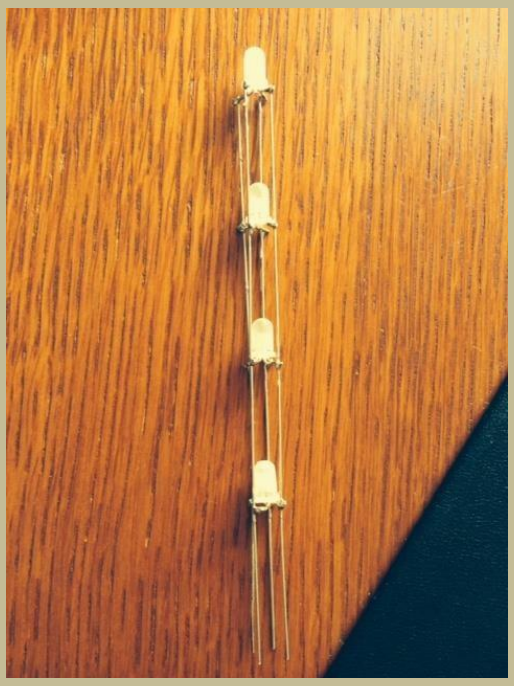
Pin combination 1-3 (Blue): Voltage = 1.2V

Pin combination 1-4 (Green): Voltage = 2.2V

Combination 1-4 receives the most amount of current passing through it, because it has the highest potential difference between the pins (2.2V), resulting in the color green to turn on!

Other current may leak into another LED because the potential difference is high enough to pass some current to notice a visual effect. Other LEDs in the column get this same current, but not to the same level, that is why one LED is the brightest, one is dim, and the others can hardly be seen. Not enough current passes to these LEDs to notice the same visual intensity.

Building the Whole Cube!



Trim the excess legs off of the columns to get a clean look.

Figure 20: Trimming excess wire

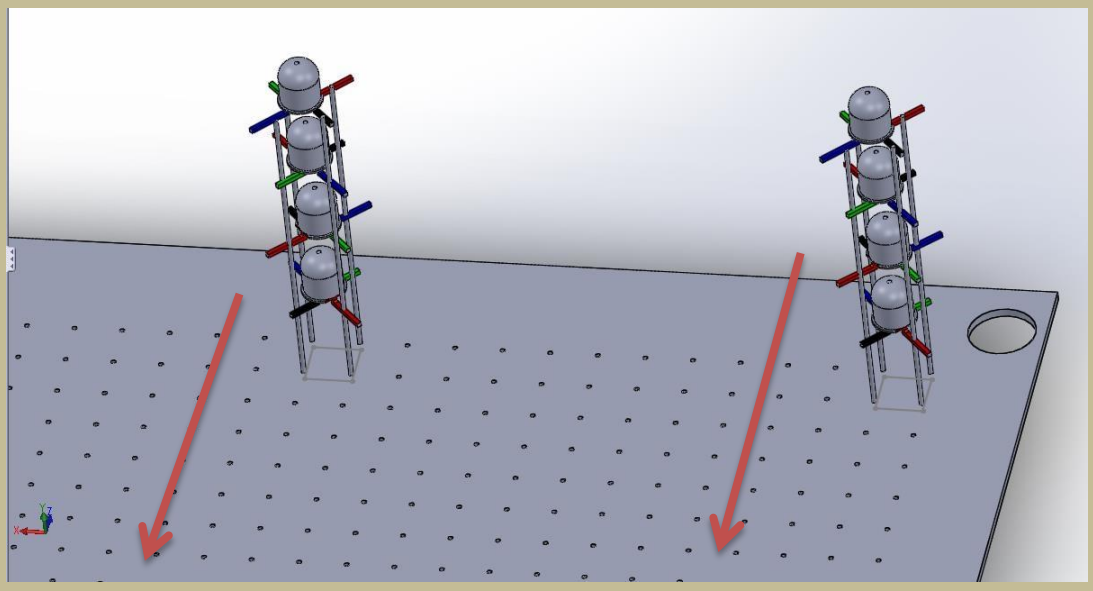


Figure 21: Direction is important when placing columns

Time to strategically fit the columns into the PCB! It is very important to keep them all aligned in the same direction, so to make wiring as easy as possible. Begin by fitting one column into a corner of the board. Confirm the proper height you would like the column to stand and solder the legs into place.

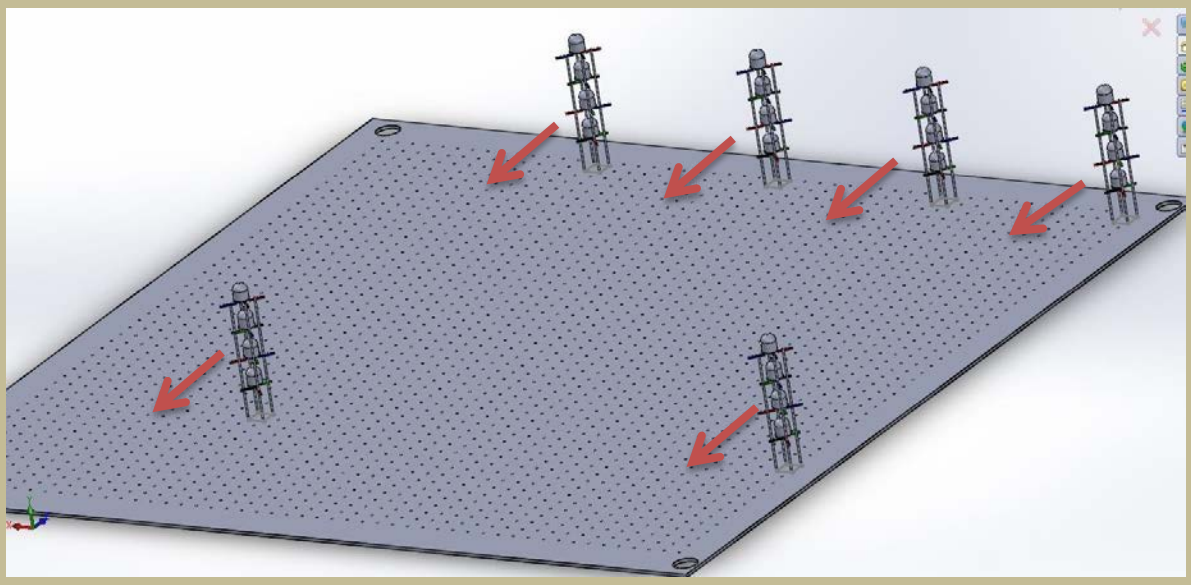
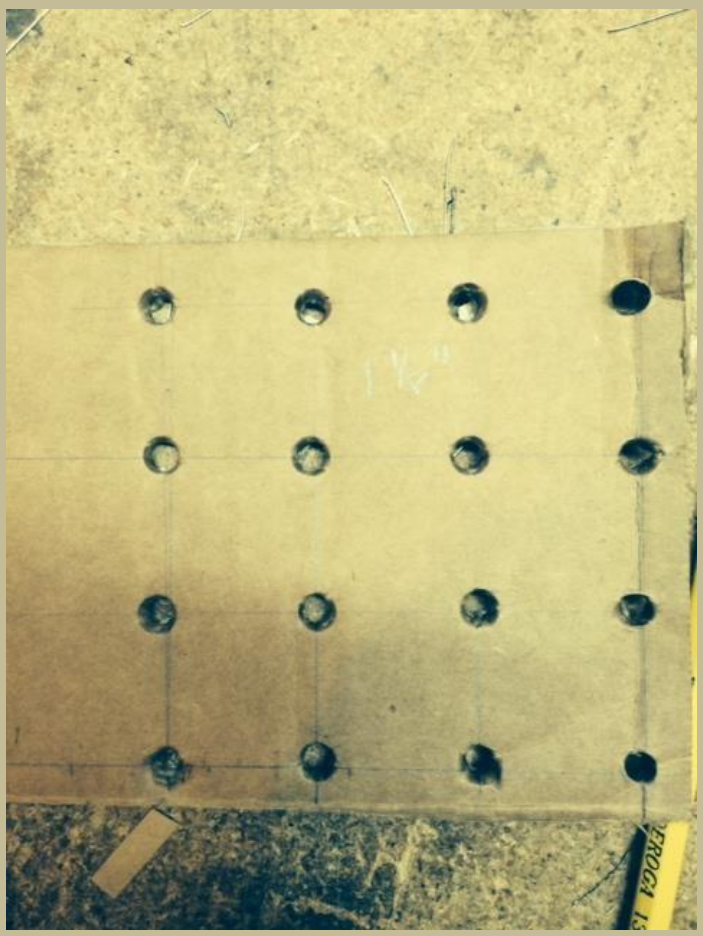


Figure 22: Corners all go in the same direction

Start with the corners and then place the columns row by row. In this example, each column is spaced out 12 holes away from each other. Orient the flat spots of the bottom most LED in the same direction.



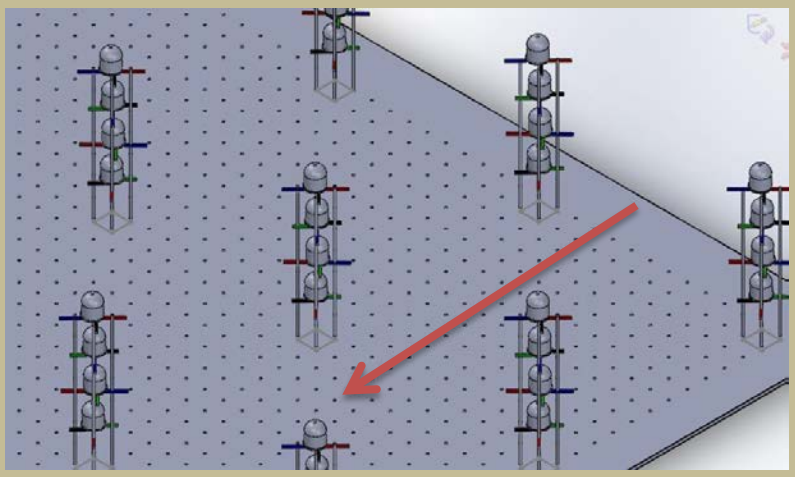
After the corners have been filled it becomes easier to solder the rest of the columns with the help from a jig. This one is made from cardboard, spacing holes out 1.125” apart. The holes are big enough for the tops of the LEDs to fit through and touch through the other side.

The idea is to fit the columns into the PCB first, flip over the board with all of the columns in, and solder them all at once from the other side. Before placing the other columns in, focus on making the PCB sit level with the surface .The corner columns should be soldered in first to level the board for the other columns to sit. Begin by soldering one leg of each column to board at the height you want. Once the board is flipped over, loosen some solder and hand adjust each column to fit into the holes. Once you are happy with the position of each column, solder the legs into place.

Figure 23: Making a cardboard jig

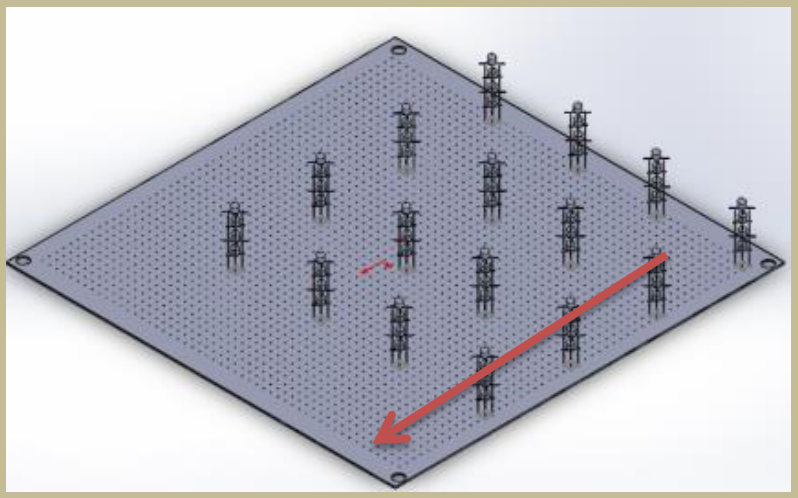
Here's what the board should look like when all of the columns are at their proper height and fit into their holes.

Figure 24: Soldering all the columns at once



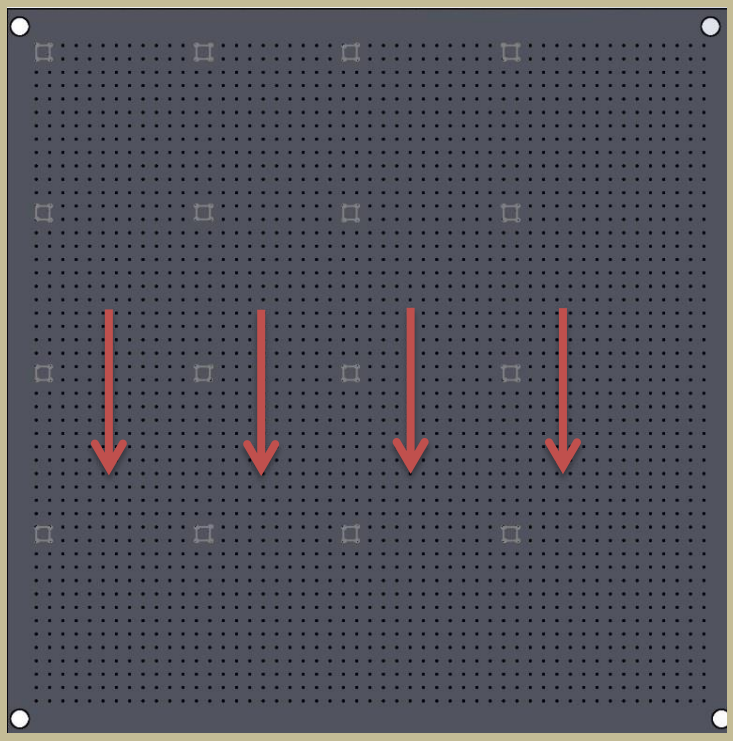
All of the columns should face the same direction. This would be a good point to check to see if in fact they are. Use the "flat spot" of the bottom-most LED to check each column.

Figure 25: Close-up of columns facing same direction



Finished looking product, without the Arduino. Note the columns in the figure aren't to scale with the board.

Figure 26: All 16 columns



Flip the board over to attach the wires. The easiest way to wire the board is in four groups of four.

Red arrows indicate where the columns were pointing on the top side.

Figure 27: Flip the board and mark the front

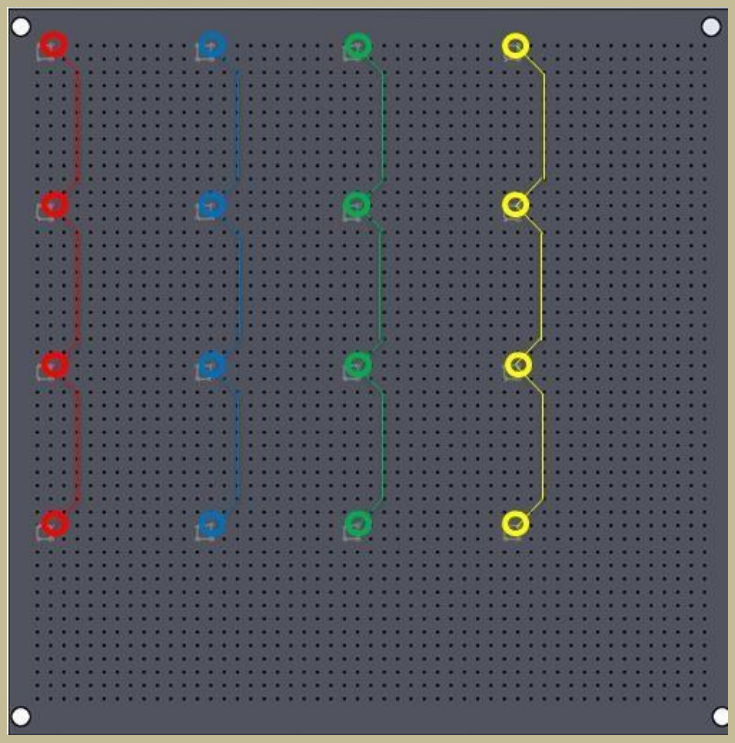


Figure 28: Easy, first group

The first and second groups are fairly simple.

Zoom in to see which pins are connected.

Wire 1

Wire 2

Wire 3

Wire 4

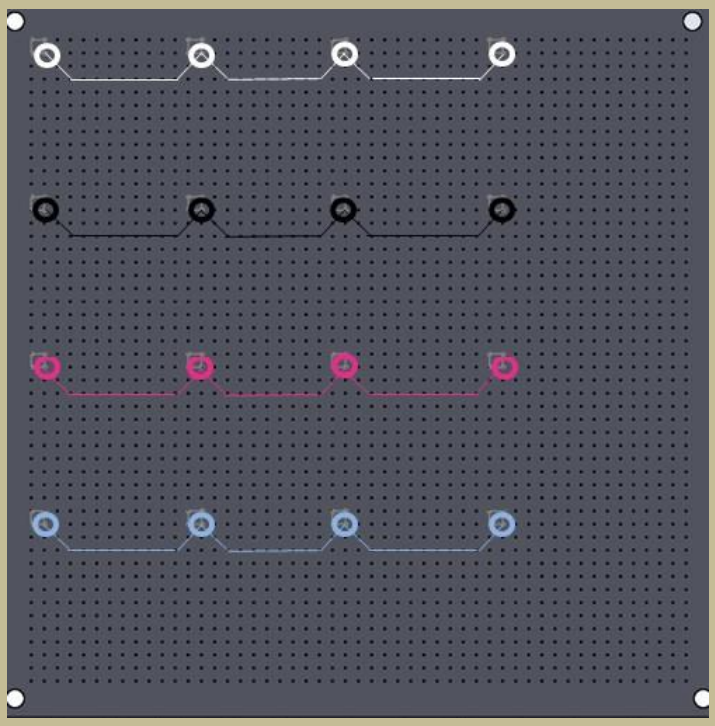


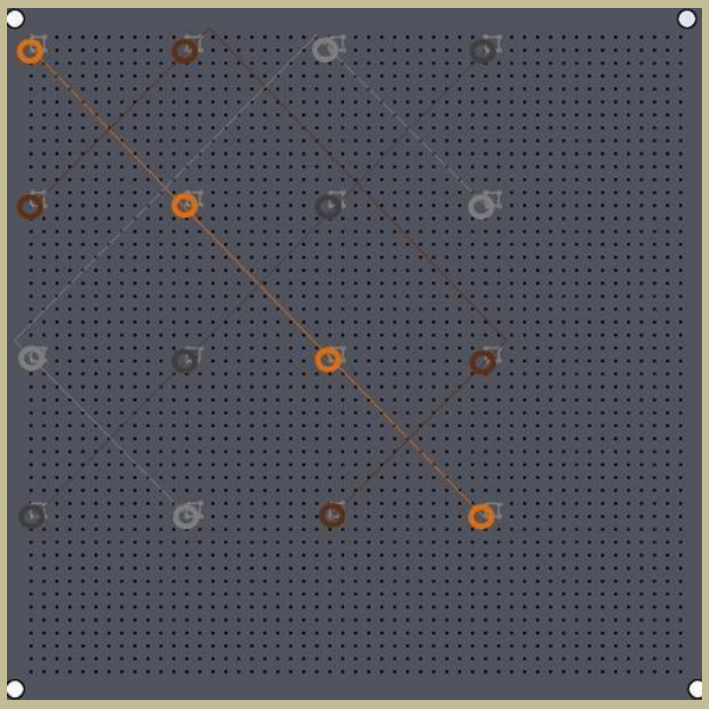
Figure 29: Easy, second group

Wire 5

Wire 6

Wire 7

Wire 8



The third group is kind tricky and the fourth group is the hardest in my opinion.

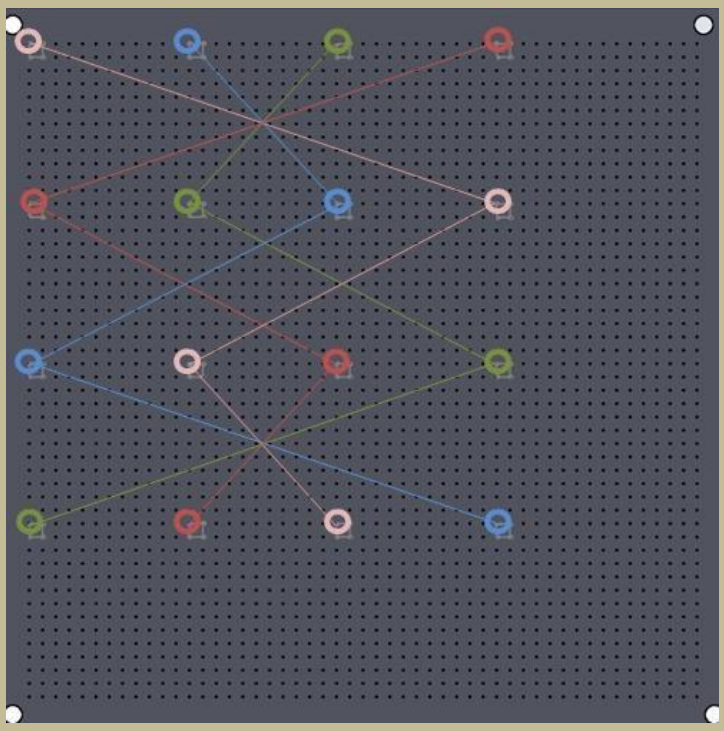
Wire 9

Wire 10

Wire 11

Wire 12

Figure 30: Intermediate, third group



Wire 13

Wire 14

Wire 15

Wire 16

Figure 31: Difficult, fourth group

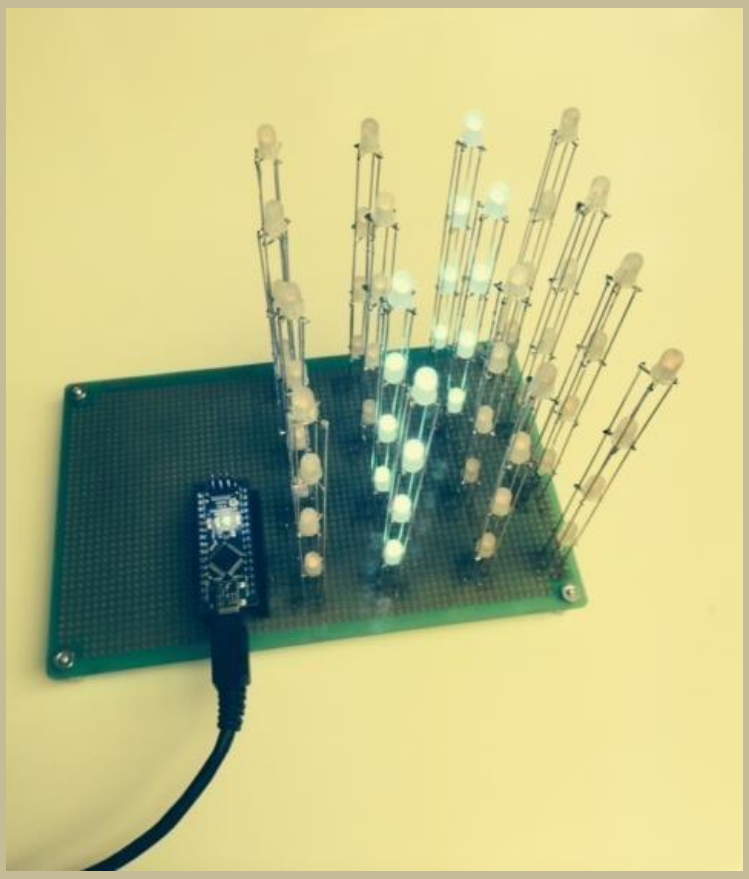


Figure 32: Finished product. Just needs code!

Connect those 16 wires to the pins on the Arduino. A table for wiring is included below (Table 1).

The Arduino is fitted into headers so not to permanently fix it into the PCB. Solder the wires to the respective pins of the headers.

Screw the standoffs to the board.

Use a mini USB cable for power.

Table 1: Wiring reference chart

Wire umber		Arduino Pin	AVR Pin
1		Digital 2	Port D – Pin 2 [PD2]
2		Digital 3	Port D – Pin 3 [PD3]
3		Digital 4	Port D – Pin 4 [PD4]
4		Digital 5	Port D – Pin 5 [PD5]
5		Digital 6	Port D – Pin 6 [PD6]
6		Digital 7	Port D – Pin 7 [PD7]
7		Digital 8	Port B – Pin 0 [PB0]
8		Digital 9	Port B – Pin 1 [PB1]
9		Digital 9	Port B – Pin 2 [PB2]
10		Digital 11	Port B – Pin 3 [PB3]

11		Digital 12	Port B – Pin 4 [PB4]
12		Digital 13	Port B – Pin 5 [PB5]
13		Analog 0 (Digital 14)	Port C – Pin 0 [PC0]
14		Analog 1 (Digital 15)	Port C – Pin 1 [PC1]
15		Analog 2 (Digital 16)	Port C – Pin 2 [PC2]
16		Analog 3 (Digital 17)	Port C – Pin 3 [PC3]

Testing the Cube

Once the cube is all wired up to the Arduino and you would like to make sure the LEDs all still light up, run this simple test. Just upload this piece of code to the Arduino and watch different colors cycle through the columns. You should see all the different columns light up in order.

How is it programmed?

The libraries developed by Asher Glick for the charliecube provide you with a nice set of functions that you can use to draw animations.

Click on the hyperlink provided, and download the 5 files provided. Make sure they all sit in the same directory/folder before compiling. You might need to manually include the header files (.h) to the IDE, if compiling doesn't work right away.

In addition to Asher's code, Jules Meredith has uploaded more patterns based on Asher's build and code.

drawLed()	drawBoxOutline()	drawHollowBox()	flushBuffer()
drawBox()	drawBoxWalls()	drawLine()	clearBuffer()

drawLed()

This function turns on LEDs at a specified position. Depending on which color this function turns on different colors of the LED.

Valid Permutations

```
drawLed(color, brightness, x-pos, y-pos, z-pos);
drawLed(color, x-pos, y-pos, z-pos);
```

Arguments

Color: what color the led should be

Integer: red. blue, green .purple, yellow, teal, white, off

Brightness: what brightness should the led be at

Integer 0 1 2 3 4 5 6 7 8 HALF=4 FULL=8

Start-x: x coordinate for the starting point

Integer 0 1 2 3

Start-y: y coordinate for the starting point

Integer 0 1 2 3

Start-z: z coordinate for the starting point

Integer 0 1 2 3

Example

```
#include "cubeplex.h"
int color = red; void
setup() {
  // initilize the cube display
initCube();

  // how many seconds until continue Pattern is set to false
animationMax = 10;
}
void loop() {
randomLed();
}
void randomLed(){
continuePattern = true;   int
animationSpeed = 100;   while
(continuePattern) {     int
xpos = random(0,4);     int
ypos = random(0,4);     int
zpos = random(0,4);

    drawLed(color,xpos,ypos,zpos);

    flushBuffer();
clearBuffer();
    delay(animationSpeed);
  }
}
```

drawBox()

This function will draw a filled in box of the specified color on the cube

Valid Permutations

```
drawBox (color, brightness, start-x , start-y, start-z, end-x, end-y,
end-z);
drawBox (color, start-x , start-y, start-z, end-x, end-y, end-z);
```

Arguments

Color: what color the led should be

Integer: red. blue, green .purple, yellow, teal, white, off

Brightness: what brightness should the led be at

Integer 0 1 2 3 4 5 6 7 8 HALF=4 FULL=8

Start-x: x coordinate for the starting point

Integer 0 1 2 3

Start-y: y coordinate for the starting point

Integer 0 1 2 3

Start-z: z coordinate for the starting point

Integer 0 1 2 3

End-x: x coordinate for the ending point

Integer 0 1 2 3

End-y: y coordinate for the ending point

Integer 0 1 2 3

End-z: z coordinate for the ending point

Integer 0 1 2 3

Example

```
#include "cubeplex.h" void
setup() {
  //initilize the cube display
initCube();

  // set the number of seconds until continuePattern is set to false
animationMax = 10;
}
void loop() {
bigBlueBox();
tinyGreenBox();
}
void bigBlueBox() {
continuePattern = true;
draw(blue,0,0,0,3,3,3);
flushBuffer();   clearBuffer();
  // do nothing while the pattern continues
while(continuePattern);
}
void tinyGreenBox() {
continuePattern = true;
drawBox(green,FULL,1,1,1,2,2,2);
flushBuffer();   clearBuffer();
  // loop until the pattern is done
while(continuePattern);
}
```

drawHollowBox()

This function will draw the walls, ceiling, and floor of a defined box

Valid Permutations

```
drawHollowBox (color, brightness, start-x , start-y, start-z, end-x,
end-y, end-z);
drawHollowBox (color, start-x , start-y, start-z, end-x, end-y,
endz);
```

Arguments

Color: what color the led should be

Integer: red. blue, green .purple, yellow, teal, white, off

Brightness: what brightness should the led be at

Integer 0 1 2 3 4 5 6 7 8 HALF=4 FULL=8

Start-x: x coordinate for the starting point

Integer 0 1 2 3

Start-y: y coordinate for the starting point

Integer 0 1 2 3

Start-z: z coordinate for the starting point

Integer 0 1 2 3

End-x: x coordinate for the ending point

Integer 0 1 2 3

End-y: y coordinate for the ending point

Integer 0 1 2 3

End-z: z coordinate for the ending point

Integer 0 1 2 3

Example

```
#include "cubeplex.h"
int color = red; void
setup() {
  // initilize the cube display
initCube();
  // set the number of seconds each animation should run for
animationMax = 10;
}
void loop() {
pulsingCube();
}
void pulsingCube() {   continuePattern
= true;   int animationSpeed = 100;
while (continuePattern) {     for (int
i = 0; i < 4; i++) {
drawHollowBox(color,0,0,0,i,i,i);
flushBuffer();       clearBuffer();
      delay(animationSpeed);
    }
    for (int i = 0; i < 4; i++) {
drawHollowBox(color,i,i,i,3,3,3);
flushBuffer();       clearBuffer();
      delay(animationSpeed);
    }
    color=nextColor(color);
  }
}
```

drawBoxOutline()

This function will draw edges of a defined box but none of the planes

Valid Permutations

```
drawBoxOutline (color, brightness, start-x , start-y, start-z, end-x,
end-y, end-z);
drawBoxOutline (color, start-x , start-y, start-z, end-x, end-y,
endz);
```

Arguments

Color: what color the led should be

Integer: red. blue, green .purple, yellow, teal, white, off

Brightness: what brightness should the led be at

Integer 0 1 2 3 4 5 6 7 8 HALF=4 FULL=8

Start-x: x coordinate for the starting point
Integer 0 1 2 3
Start-y: y coordinate for the starting point
Integer 0 1 2 3
Start-z: z coordinate for the starting point
Integer 0 1 2 3
End-x: x coordinate for the ending point
Integer 0 1 2 3
End-y: y coordinate for the ending point
Integer 0 1 2 3
End-z: z coordinate for the ending point
Integer 0 1 2 3

Example

```
#include "cubeplex.h" int
color = red;

void setup() {
  // initilize the cube display
initCube();
  // set the number of seconds each animation should run for
animationMax = 10;
}
void loop() {
pulsingCube();
}
void pulsingCube() {   continuePattern =
true;   int animationSpeed = 100;
while (continuePattern) {     for (int i
= 0; i < 4; i++) {
drawBoxOutline(color,0,0,0,i,i,i);
flushBuffer();       clearBuffer();
      delay(animationSpeed);
    }
    for (int i = 0; i < 4; i++) {
drawBoxOutline(color,i,i,i,3,3,3);
flushBuffer();       clearBuffer();
      delay(animationSpeed);
    }
    color=nextColor(color);
  }
}
```

drawBoxWalls()

This function will draw the vertical walls and all four sides of a defined box

Valid Permutations

```
drawBoxWalls (color, brightness, start-x , start-y, start-z, end-x,
end-y, end-z);
drawBoxWalls (color, start-x , start-y, start-z, end-x, end-y, end-z);
```

Arguments

Color: what color the led should be

Integer: red. blue, green .purple, yellow, teal, white, off

Brightness: what brightness should the led be at

Integer 0 1 2 3 4 5 6 7 8 HALF=4 FULL=8

Start-x: x coordinate for the starting point

Integer 0 1 2 3

Start-y: y coordinate for the starting point

Integer 0 1 2 3

Start-z: z coordinate for the starting point

Integer 0 1 2 3

End-x: x coordinate for the ending point

Integer 0 1 2 3

End-y: y coordinate for the ending point

Integer 0 1 2 3

End-z: z coordinate for the ending point

Integer 0 1 2 3

Example

```
#include "cubeplex.h" int
color = red;

void setup() {
  // initilize the cube display
initCube();
  // set the number of seconds each animation should run for
animationMax = 10;
}
void loop() {
fountian();
}
void fountian() {
continuePattern = true;   int
animationSpeed = 100;   while
(continuePattern) {     for (int z
= 0; z <= 3; z++) {
      drawBoxWalls(color,1,1,z,2,2,z);
flushBuffer();       clearBuffer();
delay(animationSpeed);
    }
    for (int z = 3; z >= 0; z--) {
drawBoxWalls(color,0,0,z,3,3,z);
flushBuffer();       clearBuffer();
delay(animationSpeed);
    }
    color=nextColor(color);
  }
}
```

drawLine()

This function will attempt to draw a line between the two points given. Due to the limited availability of pixels the best approximation is chosen for each pixel value

Valid Permutations

```
drawLine (color, brightness, start-x , start-y, start-z, end-x, end-y,
end-z);
drawLine (color, start-x , start-y, start-z, end-x, end-y, end-z);
```

Arguments

Color: what color the led should be

Integer: red. blue, green .purple, yellow, teal, white, off

Brightness: what brightness should the led be at

Integer 0 1 2 3 4 5 6 7 8 HALF=4 FULL=8

Start-x: x coordinate for the starting point

Integer 0 1 2 3

Start-y: y coordinate for the starting point

Integer 0 1 2 3

Start-z: z coordinate for the starting point

Integer 0 1 2 3

End-x: x coordinate for the ending point

Integer 0 1 2 3

End-y: y coordinate for the ending point

Integer 0 1 2 3

End-z: z coordinate for the ending point

Integer 0 1 2 3

Example

```
#include "cubeplex.h"
int color = red; void
setup() {   initCube;
  animationMax = 10;
}
void loop() {
planarSpin();
}
void planarSpin() {
continuePattern = true;   int
animationSpeed = 50;
  int spinsPerColor = 5; // a spin is actually half a
revolution   while (continuePattern) {     int x = 0;     int y
= 0;
    for (int i = 0; i < spinsPerColor; i++) {
for (int x = 0; x < 3; x++) {
drawLine(color,x,0,0,3-x,3,0);
drawLine(color,x,0,1,3-x,3,1);
drawLine(color,x,0,2,3-x,3,2);
drawLine(color,x,0,3,3-x,3,3);
flushBuffer();         clearBuffer();
        delay(animationSpeed);
      }
      for (int y = 0; y < 3; y++) {
drawLine(color,3,y,0,0,3-y,0);
drawLine(color,3,y,1,0,3-y,1);
drawLine(color,3,y,2,0,3-y,2);
drawLine(color,3,y,3,0,3-y,3);
flushBuffer();         clearBuffer();
        delay(animationSpeed);
      }
}
    color = nextColor(color);
  }
}
```

flushBuffer()

This takes the buffer frame and sets the display memory to match, because the display memory needs to be faster it is split up into two arrays instead of just one. The display frame is actually a cyclic

linked list which allows the program to just loop through and turn on the LEDs without the need to check to see if it is at the end of the loop

Valid Permutations
`flushBuffer();`
Arguments
No Arguments

`clearBuffer()`

This takes the buffer frame and sets the display memory to match, because the display memory needs to be faster it is split up into two arrays instead of just one. The display frame is actually a cyclic linked list which allows the program to just loop through and turn on the LEDs without the need to check to see if it is at the end of the loop

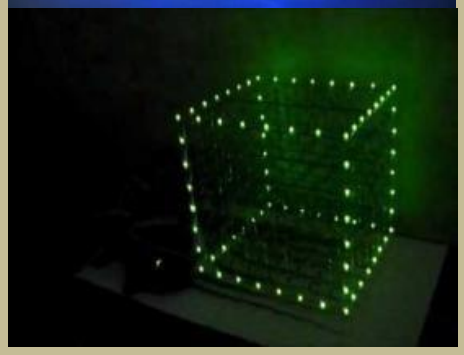
Valid Permutations
`clearBuffer();`

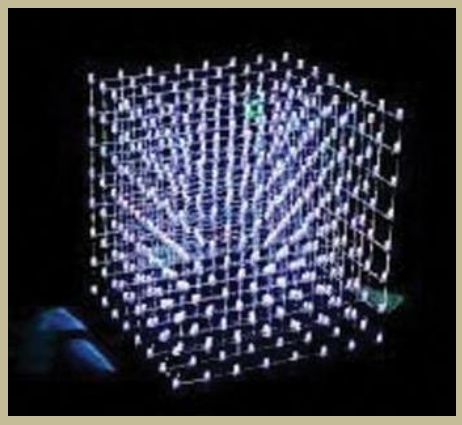
Arguments
No Arguments

You may also like –

Blue LED Cube 8x8x8: The Party Dazzler SKU: 2168940

Green 8x8x8 LED Cube SKU: 2172017



White 8x8x8 LED Cube SKU:		2169539