Steven Cheng sc618@duke.edu

Document on the LED Cube

For the summary, go to the other file named "LED Cube Project".

This document details the process of creating the 4x4x4 LED Cube and the programming that allows it to create different light patterns. The document will go the concept of Charlieplexing, soldering components, 3D modeling of the box, and the general code infrastructure. Detailed comments for the code can be found in the code files themselves.

Background

Creating the cube requires 64 LEDs, as each edge consists of 4 LEDs. The LEDs assembled in this cube are RGB LEDs, which can display a spectrum of colors utilizing different shades of red, green, and blue. Each RGB LED consist of 4 separate color pins: red, green, blue, and the anode, which is connected to the voltage supply to light up the LED.

To allow these LEDs to light up at different times, a concept called Charlieplexing is utilized. Charlieplexing is a technique for driving multiplexed display in which relatively few pins on a microcontroller (in our case an Arduino Uno) are used. In our case, Charlieplexing enables us to control 64 LEDs with only 16 inputs, and the details of this capability will be explored further in the document.

This cube is a variant of the a 4x4x4 tri-color LED Cube designed and created by Asher Glick and Kevin Baker, who were one of the first ones to create LED cubes. The instructions for their cube can be found at: https://aglick.com/charliecube.html

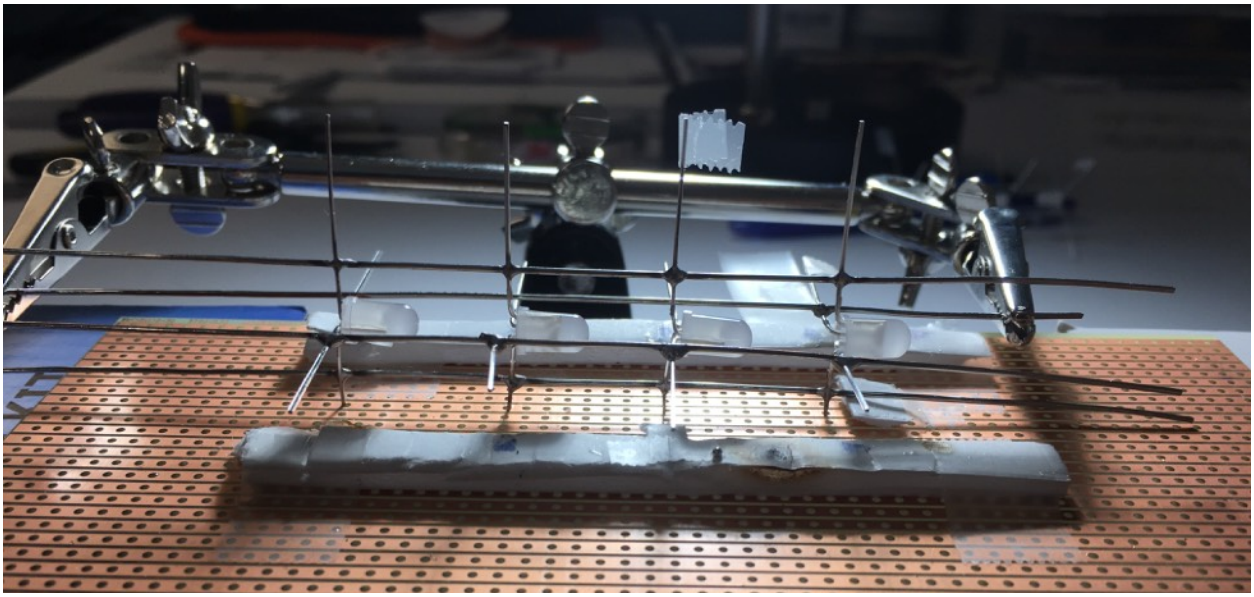
Building a Charlieplexed 64 RGB LED cube can be broken down into several groups:

1) Assembling an LED column
2) Assembling the cube
3) Creation of the box (3D modeling)
4) Programming

Materials List

- 64 RGB LEDS, diffused, ours were common anode
- 1 Large PC Grid Proto Board (16" x 10")
- Soldering equipment (soldering iron and soldering wire)
- 40 feet of solid core 20 gauge wire (or similar size) for supporting the columns of LEDs
- 1 Arduino Uno
- Wires
- Breadboard
- Black Enamel Paint
- Access to 3D Printer and printing supplies

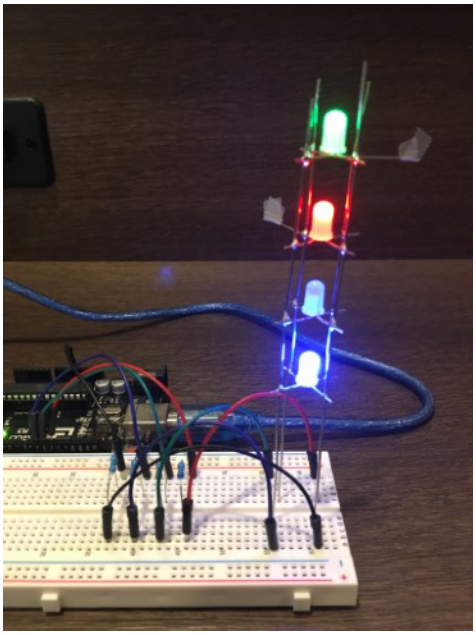
Step 1: Construction of Scaffold for columns



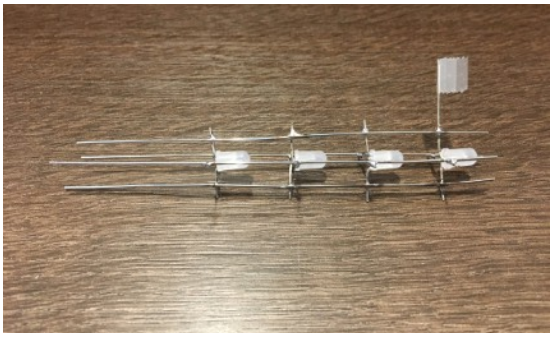
Scaffold for building LED columns

In this project, 16 columns each with 4 LEDs must be created. Building a column requires the LEDs to be in a straight line, with the LEDs evenly spaced all facing the right direction. There are many ways to accomplish this: I created a cardboard scaffold with grooves spaced out evenly for the 4 LEDs, then had stabilizers that held the wires in place. This allowed me to easily solder the wire supports to the LEDs to create the cube. In each column, each LED is rotated 90° from each other, which is crucial for multiplexing and controlling the lights when the cube is assembled.

After creating each column, it is imperative to check whether the LEDs in the column are responding correctly to the correct inputs. Hence, I created a simple code (code on Github) to check each LED's blue, red, and green lights to see if they were lighting up.

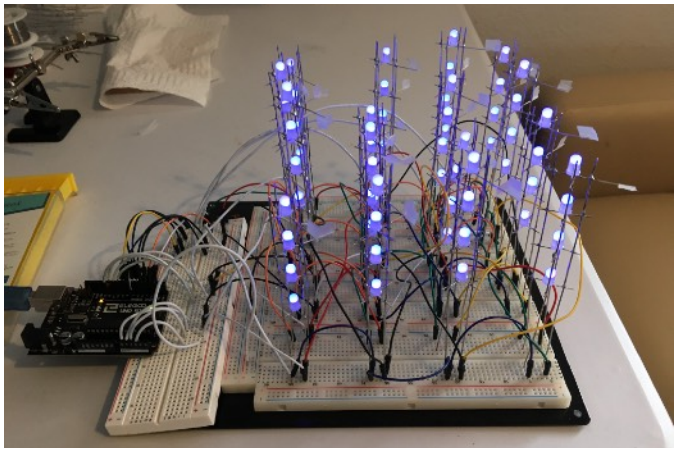


Checking LED column

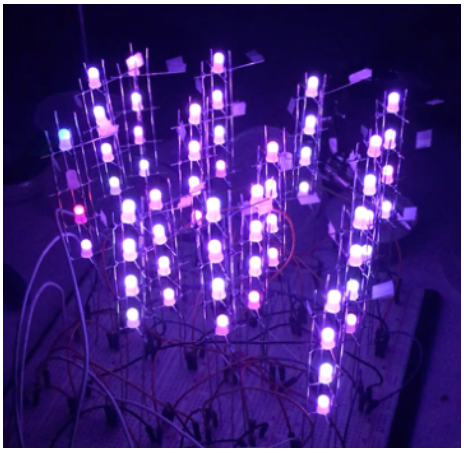


One LED column

Step 2: Assembly of the Cube



Testing all columns together

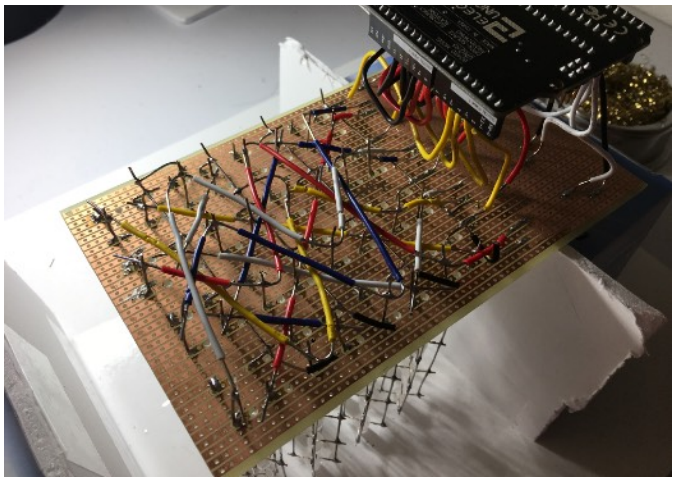


Testing all columns together in the dark

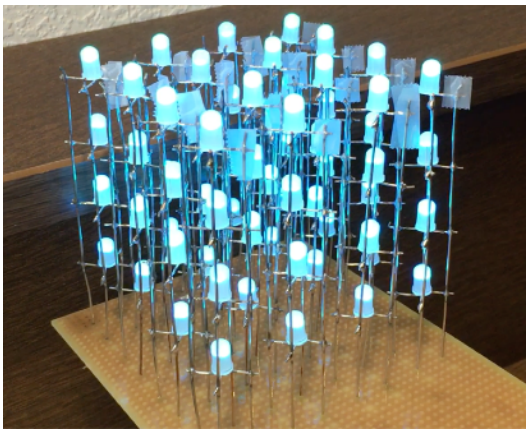
Once all 16 columns have been created, it is time to check if all 16 columns can work together before soldering them permanently to the PCB board. This can be achieved by creating a prototype of what the cube should look like on a breadboard.

The wiring for the cube utilizes a concept called Charlieplexing, which is a technique that allows a small number of microcontroller pins (Arduino in this case) to drive a large amount of LEDs. In fact, with N pins of a microcontroller you can potentially drive N * (N - 1) LEDs. In this case, I used 16 pins on the Arduino, which means the max amount of LEDs I can control is 240. For each column of LEDs, there are four wire supports; in total, there are 64 wires that must be connected. Simply explained, the layering is separated into four groups, and the engineering of it is best explained at https://aglick.com/charliecube.html.

After testing the cube, all the components are soldered onto a PCB board. Then, the wires are connected to 16 inputs on the Arduino Uno. The Arduino Uno is then connected to the computer to run LED tests.

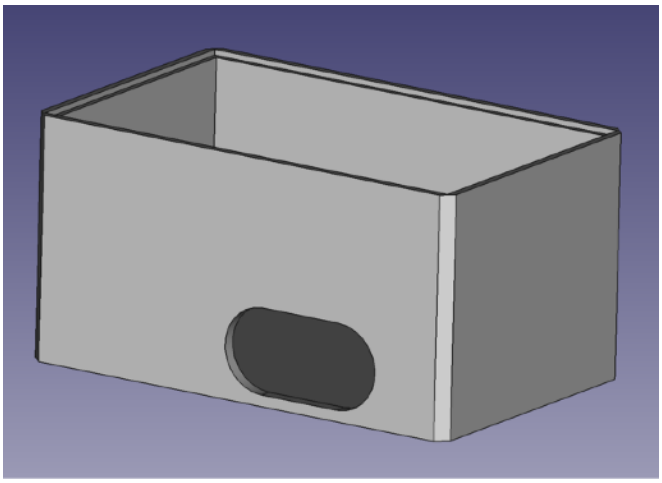


Underside of PCB Board after Soldering



Testing the cube

Step 3: Creation of the Box



3D Modeling of the Box

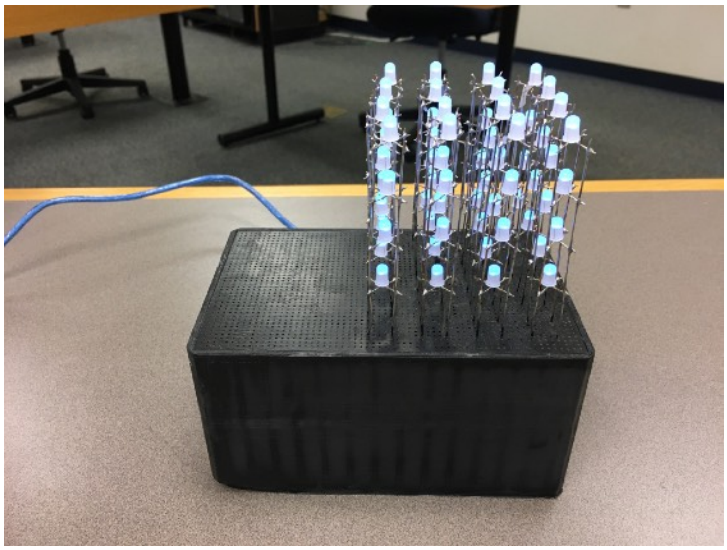
3D-printed and painted Box

The box for the RGB LED was 3D printed using Duke's 3D printers at the CoLab. The box was created in Inventor AutoCAD, and consists of a rectangular shape with an indent surrounding the top side to allow the PCB board to sit in. The hole on the side provides space to connect the Arduino Uno to an outside voltage source or computer.

After the box is printed, it is covered with black paint so that reflection from the light on the surfaces will be minimal and ensures that the light from the LEDs is not scattered.

Finally, the PCB board with the columns of LEDs are placed on top of the box. The construction is complete! Now, it is time to work on the program.

Final Product!



Step 4: Programming Light Patterns

The main code of the LED cube is in the file named “CubeProject.ino”. Detailed comments are shown in the code itself. The CubeProject.ino file uses a couple of support files taken from https://aglick.com/charliecube.html

The LED cube current supports four main patterns: boxFade(), movingDots(), fallingRows(), and tunnelWarp().

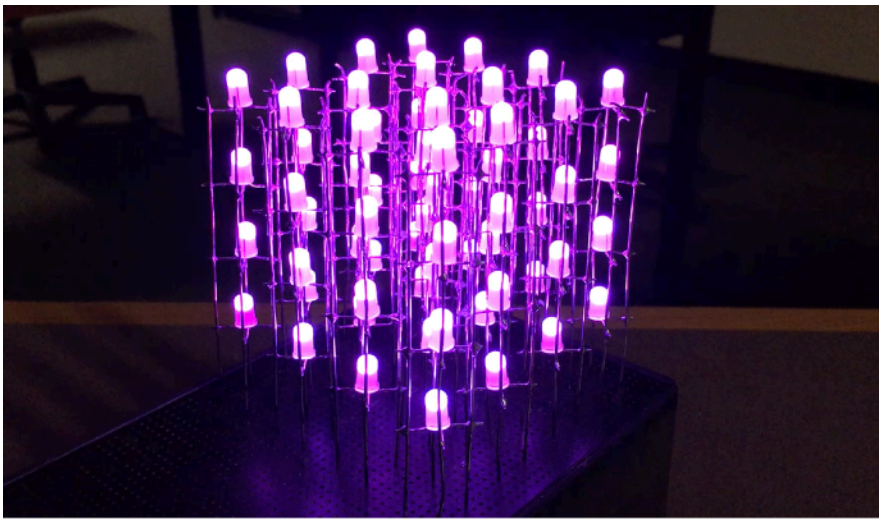
boxFade(): This animation fades through all 255 values (colors) of the LEDs, creating different mixtures of each color. This pattern utilizes a drawBox function, which essentially lights up the whole cube (all 64 LEDs) and syncs them together so that they all emit the same color. The first for loop slowly lights up the cube, while the second for loop slowly allows the LEDs to fade out. The process repeats.

movingDots(): This animation creates random points of light that move around the cube in random fashion and constantly changes color. This pattern utilizes the drawLed function, which lights up a single LED at the specified position. Similar to the boxFade, the movingDots uses a fade-in, fade-out technique to create transitions.

fallingRows(): This animation lights up each layer in descending order, similar to falling. Each iteration will change the color of the lights that light up. This pattern is achieved by creating helper functions called drawRow and diffusedRow. drawRow simply lights up a whole layer of the cube, while diffusedRow fades in each row, then fades out, creating a more fluid transition.

tunnelWarp(): This animation lights up the whole cube like the boxFade, but it changes the color of specific LEDs in the cube to create a dynamic feeling, as if the lights are travelling through a tunnel and constantly changing. This animation is achieved by calling repeatedly a helper function called drawBoxWalls, which draws the vertical walls and all four sides of a defined box.

All these patterns utilize helper functions, which are explained in depth in the code.



LED Cube in action