Lab 3c
Fun with your LED cube

ENGR 40M
Chuan-Zheng Lee
Stanford University
19 May 2017
Announcements

- Homework 6 is **not** released today. It will be released on Monday (May 22). It will be due at 11am Tuesday week (May 30).

- Homework 6 prepares you for lab 4.

- There is still a prelab 4.

- Homework 7 will be released Monday, May 29, and will be due Monday, June 5.
Overview of lab 3c

• Your task is to get your cube to respond to some sort of input.

• For most of you, this will involve adding hardware.

• You choose what you want to do, subject to minimum requirements.
Our suggestions for input

- Serial data
- Potentiometer
- Pushbutton switches (or other switches)
- Audio
- Capacitive touch sensing

You’re free (and encouraged) to propose something else, but be sure to tell your TA well in advance!
Serial data

• You’ve already used the Serial Monitor

• You can also write computer software to interact with the serial port directly

• If you only respond to serial data, we’ll expect to see some impressive software
Analog-to-digital converter

- `analogRead()` reads the voltage on the pin, scaled to be a number between 0 and 1023

```cpp
void loop() {
  int reading = analogRead(A3);
  Serial.println(reading);
}
```

Serial Monitor:
819
819

\[
\frac{4}{5} \times 1024 \approx 819
\]

- This takes longer than `digitalRead()`, but this probably won’t bother you
Potentiometer

• Essentially a variable voltage divider
Potentiometer

- Essentially a variable voltage divider

\[ R_1 \text{ and } R_2 \text{ vary with the wiper, but } R_1 + R_2 = \text{constant} \]
Potentiometer

- Connect the wiper to the analog input
- Read using `analogRead()`
Audio

- You can connect an audio output to your Arduino... almost.
Audio

• `analogRead()` the instantaneous value
• The input will be “centered” at 2.5 V, so you’ll need to process it in software
• An additional handout guides you through this
Audio—frequency response

- The ArduinoFFT3 library can process your signal to return a frequency-domain representation.

- Implements the fast Fourier transform, an algorithm which computes a close cousin of the Fourier series.

- An additional handout guides you through this.

- Video: https://youtu.be/FRXDTiOHFII
Audio–frequency response

ArduinoFFT bins, shown with frequency spectrum of chorus of "Royals" (Lorde)

- bins[0]: low frequencies (bass), near 250 Hz
- bins[1]: above 500-750Hz
- bins[2]: above 1kHz (treble)

amplitude

frequency (Hz, log scale)

10^1 10^2 10^3
Pushbutton switch

- SPST, momentary and normally open (sometimes known as *push-to-make*)
Pushbutton switch: debouncing

What we think a switch does:

What it actually does:
Pushbutton switch: debouncing

• As the switch bounces, the Arduino can register many transitions

• Dealing with this is called *debouncing*

• One strategy: Ignore transitions too soon after the last one

• *An additional handout guides you through this*
Programming a pattern

• Since you’ve *abstracted* away the time-multiplexing part, displaying a pattern consists mainly of filling in the pattern array.

• Recall triply-nested loops:

```c
for (int z = 0; z < 4; z++) {
    for (int y = 0; y < 4; y++) {
        for (int x = 0; x < 4; x++) {
            pattern[z][y][x] = (some value);
        }
    }
}
```
Raindrop pattern

• The raindrop pattern looks like rain is falling from the top of the cube to the bottom

• Videos
  • https://youtu.be/-tZJ-3NSIhY?t=52
  • https://youtu.be/DahwcDeqyA0
Raindrop pattern

Every time period (say, 150 ms):
1. Move the pattern down by one plane
2. Choose an LED at random in the top plane
Decomposition

Every time period (say, 150 ms):
1. Move the pattern down by one plane
2. Choose an LED at random in the top plane

• Decompose this into smaller steps:
  1. movePatternDown(pattern)
  2. chooseRandomLEDInTheTopPlane(pattern)

• As a principle, each function should do exactly one thing
Timing and inputs

- Update the pattern once every (say) second
- With no inputs, this will work:

```c
void loop() {
    static byte ledOn[4][4][4];
    updatePattern(ledOn);  // updates pattern
    delay(1000);
}
```
Timing and inputs

- Stop/start whenever the button is pressed
- Why won’t this work?

```cpp
void loop() {
    static byte ledOn[4][4][4];
    static bool running = false;
    if (running)
        updatePattern(ledOn);  // updates pattern
    if (digitalRead(BUTTON) == HIGH)
        running = !running;
    delay(1000);
}
```
Timing and inputs

- Better: Check button without delay

```c
void loop() {
    static byte ledOn[4][4][4];
    static bool running = false;
    static long nextUpdateTime = millis();
    if (millis() > nextUpdateTime) {
        if (running) {
            updatePattern(ledOn);     // updates pattern
            nextUpdateTime += 1000;
        }
        if (digitalRead(BUTTON) == HIGH) {
            running = !running;
        }
    }
}```
Complexity requirements

• Minimum requirement:
  cubes: 25; 6×6 planes: 35; larger planes: 30
• Must do one “additional handout”, or propose your own
• Details are in the “overview” handout

<table>
<thead>
<tr>
<th>Points</th>
<th>Hardware</th>
<th>Software</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>No additional hardware (includes serial data)</td>
<td>Simple response</td>
</tr>
<tr>
<td>10</td>
<td>Minor additional hardware</td>
<td>Minor complexity</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Raindrop pattern</td>
</tr>
<tr>
<td>15</td>
<td>Moderate additional hardware</td>
<td>Moderate complexity</td>
</tr>
<tr>
<td></td>
<td>• Pushbuttons</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Audio non-frequency</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>Complex additional hardware</td>
<td>Impressive complexity</td>
</tr>
</tbody>
</table>
Breadboard style

Please don’t:

It's like a leet-speak translation of a manifesto by a survivalist cult leader who's for some reason obsessed with memory allocation.

I can get someone else to review my code.

Not more than once, I bet.

https://www.xkcd.com/1833/