

Lecture 9

Processing and Arduino topics

IAT267 Introduction to Technological
Systems

Quiz – Week 9

- Available on webct between Friday November 4 and Wednesday, November 9
- Last quiz of the course

Assignments

- Marks for the first two assignments are on webct
- Assignment 3 is on the topic of networking and will be posted in week 11, due in week 13.

Course Project

- Milestone 3 – now in progress
- Marks for Milestone 2 will be up on webct by the end of this week

Final Exam

- December 16, 2011, Friday
- 8:30am -11:30am
- SUR2600

Topics for today

- Interactivity: Data from Keyboard and Mouse
- Images in Processing:
 - Color
 - Transparency
 - Filters
- Processing and Arduino libraries

Mouse Examples

- When a program starts , mouseX and mouseY values are 0.
- If the cursor moves into the display window, the values are set to the current position of the cursor.
- The mouse position is most commonly used to control the location of visual elements on screen.

```
//circle follows the cursor
```

```
void setup()
```

```
{
```

```
  size(100,100);
```

```
  smooth();
```

```
  noStroke();
```

```
}
```

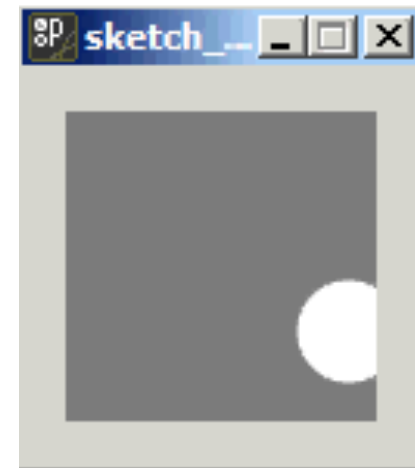
```
void draw()
```

```
{
```

```
  background(126);
```

```
  ellipse(mouseX,mouseY, 33,33);
```

```
}
```

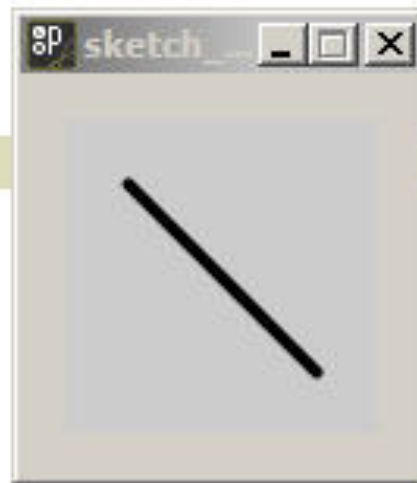
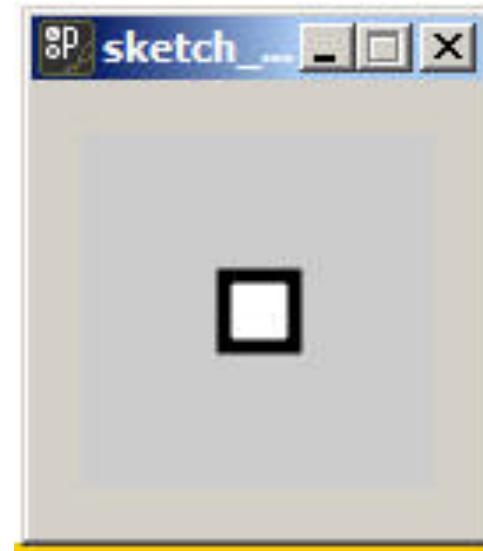


Keyboard Input

- Processing registers the most recently pressed key and whether a key is currently pressed.
 - The boolean variable ***keyPressed*** is *true* if a key is pressed and *false* if not.
 - ***keyPressed*** remains true while the key is held down and becomes false only when the key is released.

```
//draw a rectangle while any key is pressed
```

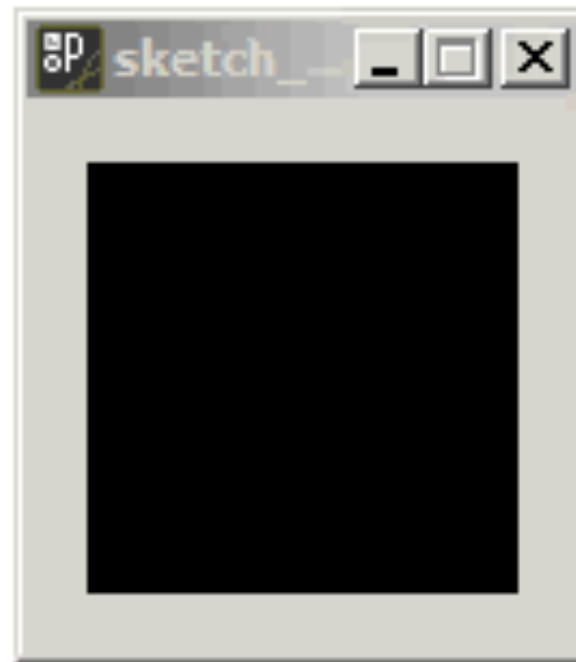
```
void setup()  
{  
  size(100,100);  
  smooth();  
  strokeWeight(4);  
}  
void draw()  
{  
  background(204);  
  if (keyPressed==true)  
  {  
  
    rect(40,40,20,20);  
  }  
  else  
  {  
    line(20,20,80,80);  
  }  
}
```



key variable

- The *key* variable is of the char data type and stores the most recently pressed key .
- The key variable can store only one value at a time.
- A key can be displayed on screen by loading a font and using the text() function.

```
PFont font;  
  
void setup()  
{  
  size(100,100);  
  font=loadFont("ArialMT-48.vlw");  
  textFont(font);  
}  
  
void draw()  
{  
  background(0);  
  text(key,28, 75);  
}
```



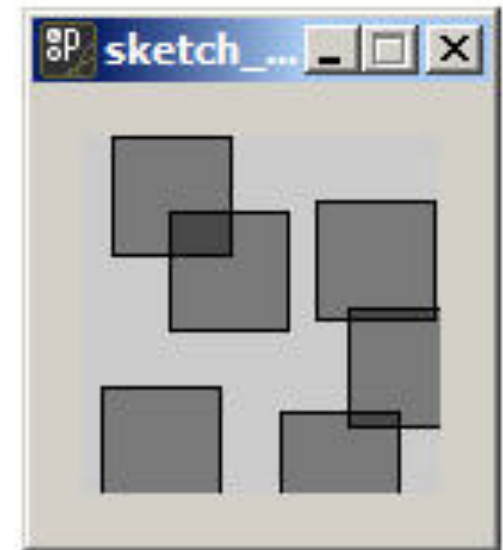
Events

- Mouse and keyboard events for detecting actions and receiving data
- Events alter the normal flow of a program when an action such as a key press or mouse movement takes place.

Mouse Events

- ***mousePressed()***: called every time when a mouse button is pressed
- ***mouseReleased()***: called every time when a mouse button is released
- ***mouseMoved()***: called every time the mouse moves and a mouse button is not pressed
- ***mouseDragged()***: called every time the mouse is moved while a mouse button is pressed

```
void setup() {  
  size(100, 100);  
  fill(0,102);  
}  
  
void draw() {}//empty draw keeps th eprogram running  
  
void mousePressed()  
{  
  rect(mouseX,mouseY, 33,33);  
}
```



- In the previous example rectangles are drawn inside `mousePressed()` and they remain on screen because there is no `background()` inside `draw()`.
- If `background()` is used, visual elements drawn within one of the mouse event functions will appear on the screen for only a single frame.
- The previous example has nothing at all inside `draw()`, but it needs to be there to force Processing to keep listening for events.

Images

- Relevant data type and functions for loading and displaying images
 - PImage, loadImage(), image()
 - tint(), noTint()
- The dimensions of digital images are measured in units of **pixels**.
- Every digital image has a **color depth**.
 - The color depth refers to the number of bits used to store each pixel

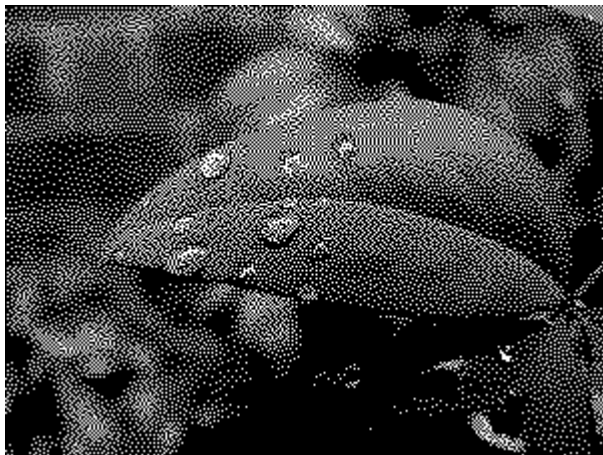
Color Depth

- If the color depth of an image is 1, each pixel can be one of two values (e.g. black or white).
- If the color depth is 4, each pixel can be one of 16 values.
- If the color depth of an image is 8, each pixel can be one of 256 values.

- Looking at the same image displayed with different color depths:



[24-bit color](#) (16.7m colors)



[1-bit color](#) ($2^1 = 2$ colors)
monochrome



[4-bit color](#) ($2^4 = 16$ colors)



[8-bit color](#) ($2^8 = 256$ colors)

Display

- Processing can load images, display them on the screen, and change their size, position, opacity, and tint.
- There is a data type for images called **PImage**.
 - The same way that *integers* are stored in variables of the *int* data type and values of *true* and *false* are stored in the *boolean* data type, images are stored in variables of the *PImage* data type.

loadImage()

- Before displaying an image, it's necessary to first load it with the **loadImage()** function
- Syntax:
 - loadImage(filename)
 - loadImage(filename, extension)
 - No matter which syntax to use, be sure to include the file format extension as a part of the *filename* (e.g., “pup.gif”, “cat.jpg”, “trees.png”).
- For the image to load, it must be in the data folder of the current program.
- Add the image by selecting the “Add File” option in the Sketch menu of the Processing environment.

- As a shortcut, you can also drag and drop an image to the Processing window.
- To make sure the image was added, select “Show Sketch Folder” from the Sketch menu. The image will be inside the *data folder*.

- With the image file in the right place, you can load and then display it with the **image()** function:
- Syntax:
 - *image(name, x, y)*
 - *image(name, x, y, width, height)*

image() parameters

- The parameters for `image()` determine the image to draw and its position and size.
- The *name* parameter must be a ***PImage*** variable.
- The ***x*** and ***y*** parameters set the position of the upper-left corner of the image
- By default, the image will display at its actual size (in units of pixels), but you can change the size by adding the additional ***width and height*** parameters.


```
test$
```

```
PImage img;  
// Image must be in the sketch's "data" folder  
img = loadImage("greens.jpg");  
image(img, 0, 0);
```



```
test$
```

```
PImage img;  
// Image must be in the sketch's "data" folder  
img = loadImage("greens.jpg");  
image(img, 20, 20, 60, 60);
```

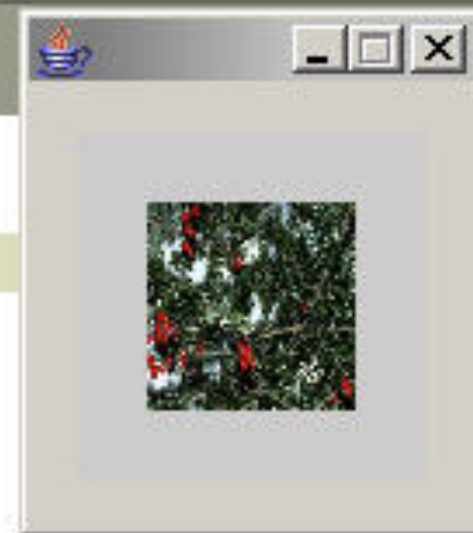


Image Color, Transparency

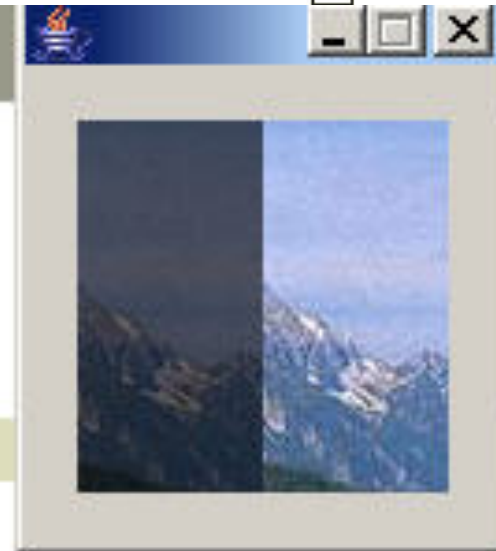
- Images are colored with the `tint()` function.
- This function is used the same way as `fill()` and `stroke()`, but it affects only images:
 - `tint(gray)`
 - `tint(gray, alpha)`
 - `tint(value1, value2, value3)`
 - `tint(value1, value2, value3, alpha)`
 - `tint(color)`

Image Color, Transparency

- All images drawn after running `tint()` will be tinted by the color specified in the parameters.
- This has no permanent effect on the images, and running the `noTint()` function disables the coloration for all images drawn after it is run.

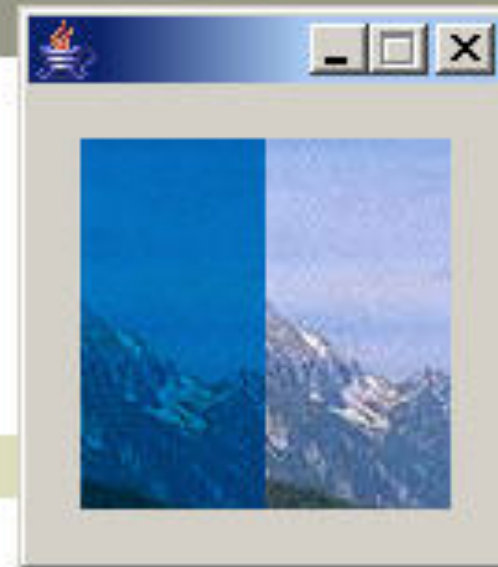
```
test$
```

```
PImage img;  
img = loadImage("castle.jpg");  
tint(102); // Tint gray  
image(img, 0, 0);  
noTint(); // Disable tint  
image(img, 50, 0);
```



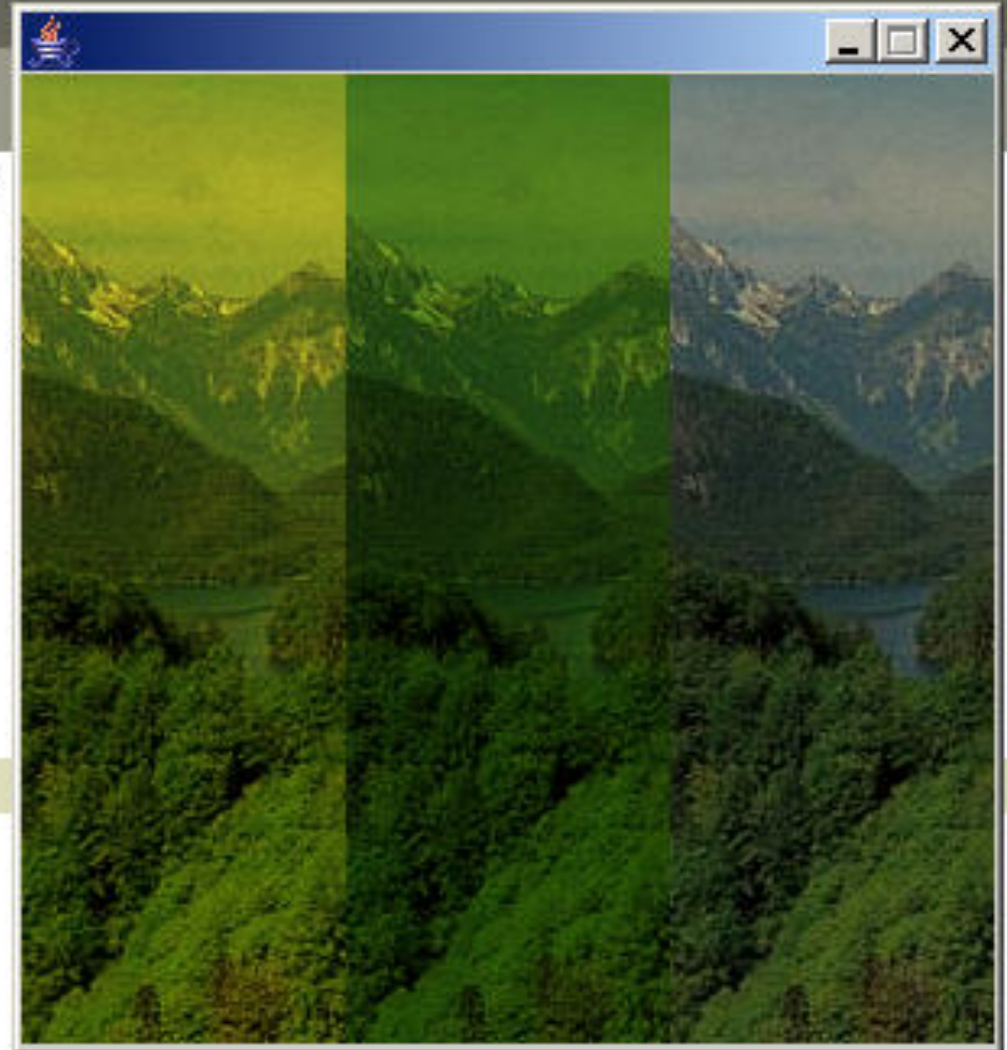
```
test$
```

```
PImage img;  
img = loadImage("castle.jpg");  
tint(0, 153, 204); // Tint blue  
image(img, 0, 0);  
noTint(); // Disable tint  
image(img, 50, 0);
```



```

test$
color yellow = color(220, 214, 41);
color green = color(110, 164, 32);
color tan = color(180, 177, 132);
PImage img;
size(300,300);
img = loadImage("castle.jpg");
tint(yellow);
image(img, 0, 0);
tint(green);
image(img, 100, 0);
tint(tan);
image(img, 200, 0);
    
```



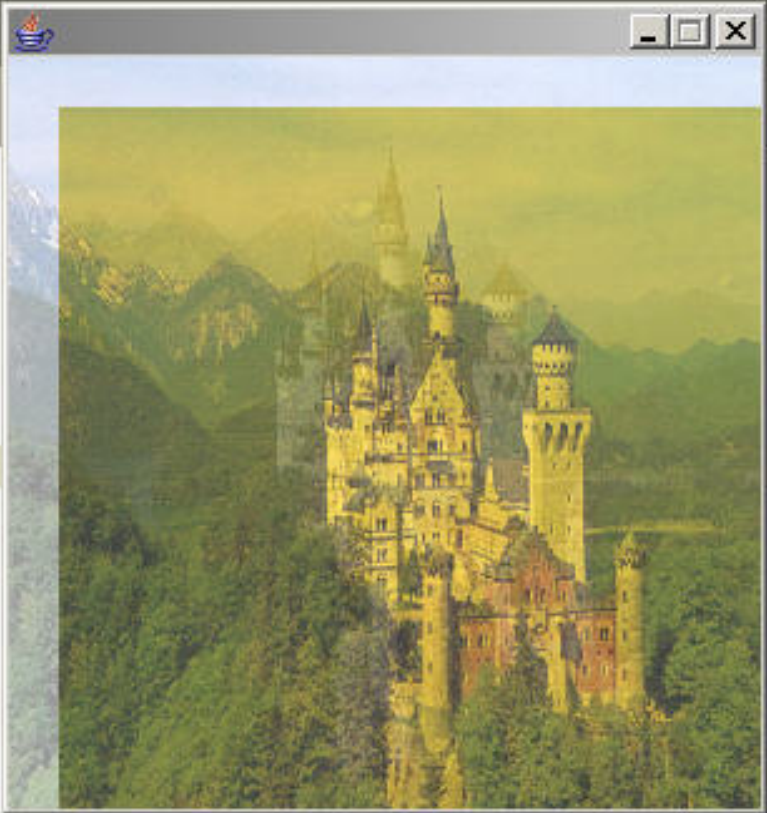
Transparent Image

- To make an image transparent without changing its color, set the *tint* to white, with an alpha value < 255


```
File Edit Sketch Tools Help
```

test\$

```
PImage img;  
size(300,300);  
img = loadImage("castle.jpg");  
background(255);  
tint(255, 102); // Alpha to 102 without changing the tint  
image(img, 0, 0, 300, 300);  
tint(255, 204, 0, 153); // Tint to yellow, alpha to 153  
image(img, 20, 20, 300, 300);
```

The image shows a software interface with a code editor on the left and a preview window on the right. The code editor has a menu bar with 'File', 'Edit', 'Sketch', 'Tools', and 'Help'. Below the menu bar is a toolbar with icons for play, stop, save, copy, paste, and zoom. The code editor contains the following code:

```
PImage img;  
size(300,300);  
img = loadImage("castle.jpg");  
background(255);  
tint(255, 102); // Alpha to 102 without changing the tint  
image(img, 0, 0, 300, 300);  
tint(255, 204, 0, 153); // Tint to yellow, alpha to 153  
image(img, 20, 20, 300, 300);
```

The preview window shows a castle image with a yellow tint and semi-transparent background. The castle is a large, multi-towered structure with a prominent central tower and several smaller towers. The background is a light greenish-yellow color, and the castle is positioned in the center of the frame.

filter()

- Filters the display window as defined by one of several modes:
 - THRESHOLD** - converts the image to black and white pixels
 - depending if they are above or below the threshold defined by the level parameter. The level must be between 0.0 (black) and 1.0(white). If no level is specified, 0.5 is used.
 - **GRAY** - converts any colors in the image to grayscale equivalents
 - **INVERT** - sets each pixel to its inverse value
 - **POSTERIZE** - limits each channel of the image to the number of colors specified as the level parameter
 - **BLUR** - executes a Gaussian blur with the level parameter specifying the extent of the blurring. If no level parameter is used, the blur is equivalent to Gaussian blur of radius 1.


```
test$
```

```
size(400,400);
```

```
PImage b;
```

```
b = loadImage("castle.jpg");
```

```
image(b, 0, 0);
```

```
filter(GRAY);
```





copy()

- Copies a region of pixels from the display window to another area of the display window.
- `copy(x, y, width, height, dx, dy, dwidth, dheight)`
- If the source and destination regions aren't the same size, it will automatically resize the source pixels to fit the specified target region.

test | Processing 0135 Beta

File Edit Sketch Tools Help



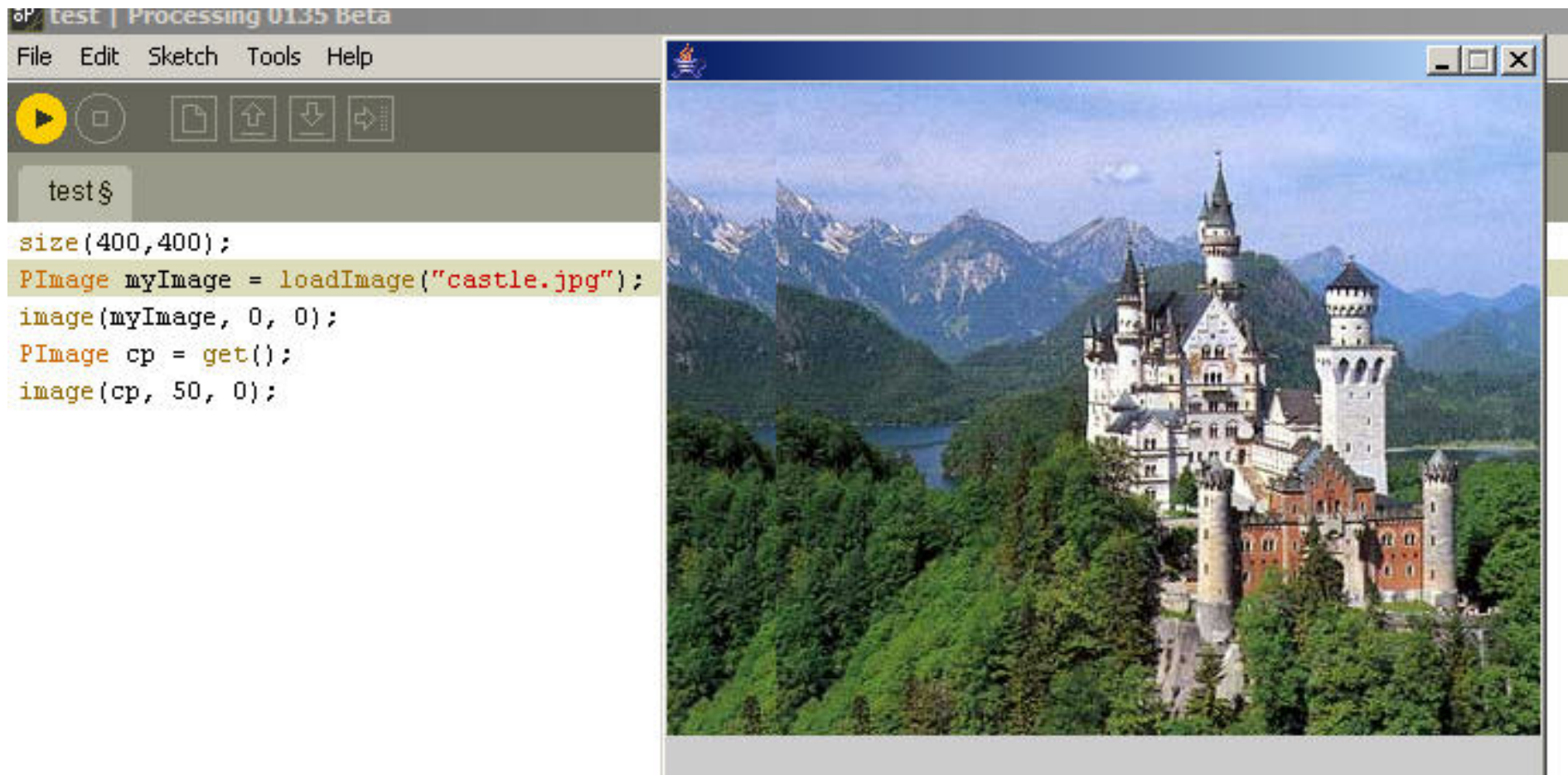
test\$

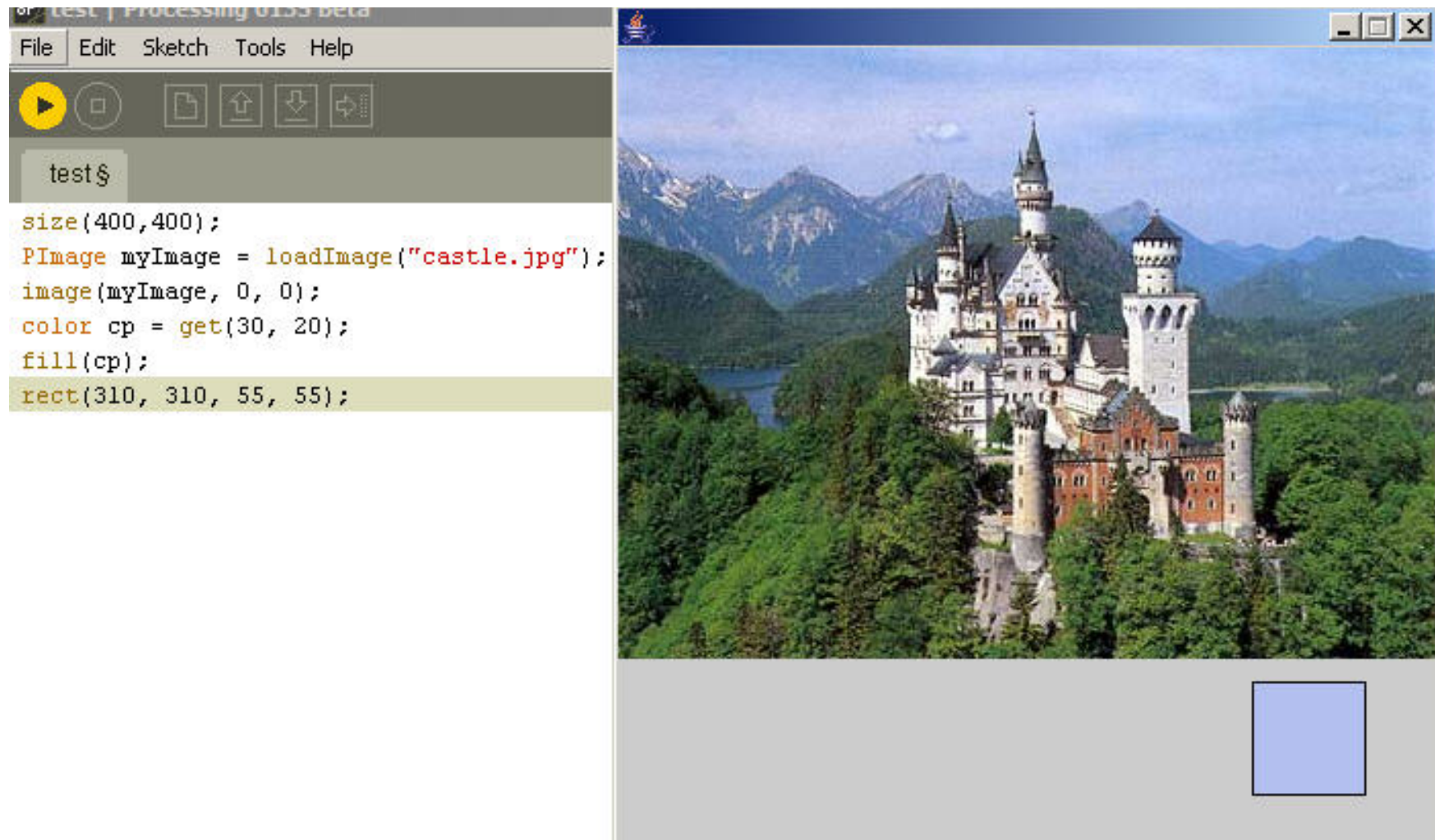
```
size(400,400);  
PImage img = loadImage("castle.jpg");  
image(img, 0, 0);  
copy(15, 25, 40, 40, 300, 200, 80, 80);  
noFill();  
// Rectangle shows area being copied  
rect(15, 25, 10, 10);
```



get()

- Reads the color of any pixel or grabs a section of an image. If no parameters are specified, the entire image is returned.
- Get the value of one pixel by specifying an x, y coordinate.
- Get a section of the display window by specifying an additional **width** and **height** parameter.
- If the pixel requested is outside of the image window, black is returned.





The screenshot displays a Processing IDE window titled "test | Processing 0.133 beta". The menu bar includes "File", "Edit", "Sketch", "Tools", and "Help". Below the menu is a toolbar with icons for play, stop, save, copy, paste, and zoom. The code editor contains the following code:

```
test $  
size(400,400);  
PImage myImage = loadImage("castle.jpg");  
image(myImage, 0, 0);  
color cp = get(30, 20);  
fill(cp);  
rect(310, 310, 55, 55);
```

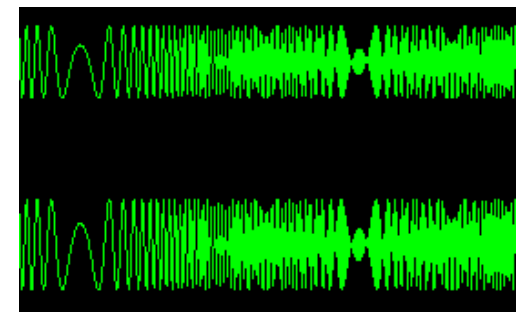
The right side of the window shows a rendered image of a large, ornate castle with multiple towers and spires, situated on a hillside with a lake and mountains in the background. A small, light blue square is visible in the bottom right corner of the rendered area, representing the color sampled from the image at coordinates (30, 20).

Processing - libraries

- Wide and varied range of libraries
- Many of the Processing libraries are available for download at the Processing website
 - www.processing.org
- Examples of libraries:
 - 3D
 - Bluetooth communication
 - Gesture recognition libraries

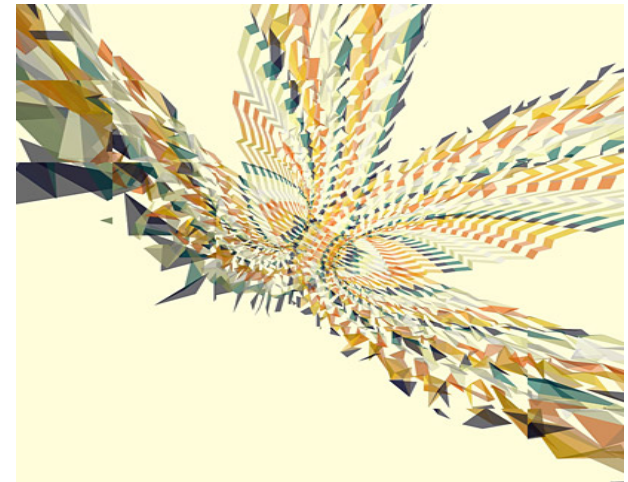
Minim Library

- By Damian Di Fede
- Uses the JavaSound API to provide an easy-to-use **audio library**
- Provides a reasonable amount of flexibility for more advanced users
- Highly recommended for beginners
- Clear, well documented



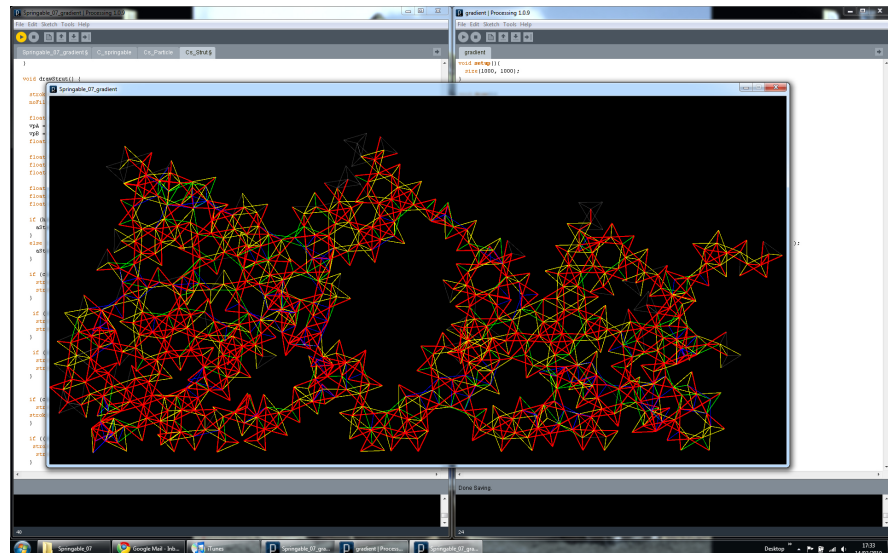
surfaceLib library

- By Andreas Koberle and Christian Riekoff
- Offers an easy way to create different 3D surfaces
- It contains a library of surfaces and a class to extend



Physics Library

- By Jeffrey Traer Bernstein
- It is a nice and simple particle system physics engine that helps you get started using particles, springs, gravity and drag.



Bluetooth Desktop library

- By Patrick Meister
- This library lets you send and receive data via Bluetooth wireless networks

proMidi Library

- By Christian Riekoff
- This library lets Processing send and receive MIDI information

oscP5 Library

- By Andreas Schlegel
- This library is an OpenSound Control (OSC) implementation for Processing.
- OSC is a protocol for communication among computers, sound synthesizers, and other multimedia devices.

Audio in Processing

- Processing has a great deal of support for working with graphics, video and OpenGL built into the core libraries
- Minim library: one of the best known and complete library for audio (several other audio libraries are available as well)

Using Minim library

- Core: a class called Minim
- Every time you use the Minim library, you need to instantiate a Minim object
- **Four tasks** with the Minim object:
 - Play an audio file that you load into your application
 - Play audio that you create in your program
 - Monitor audio and get data about it
 - Record audio to disk

Load MP3 file and play it back

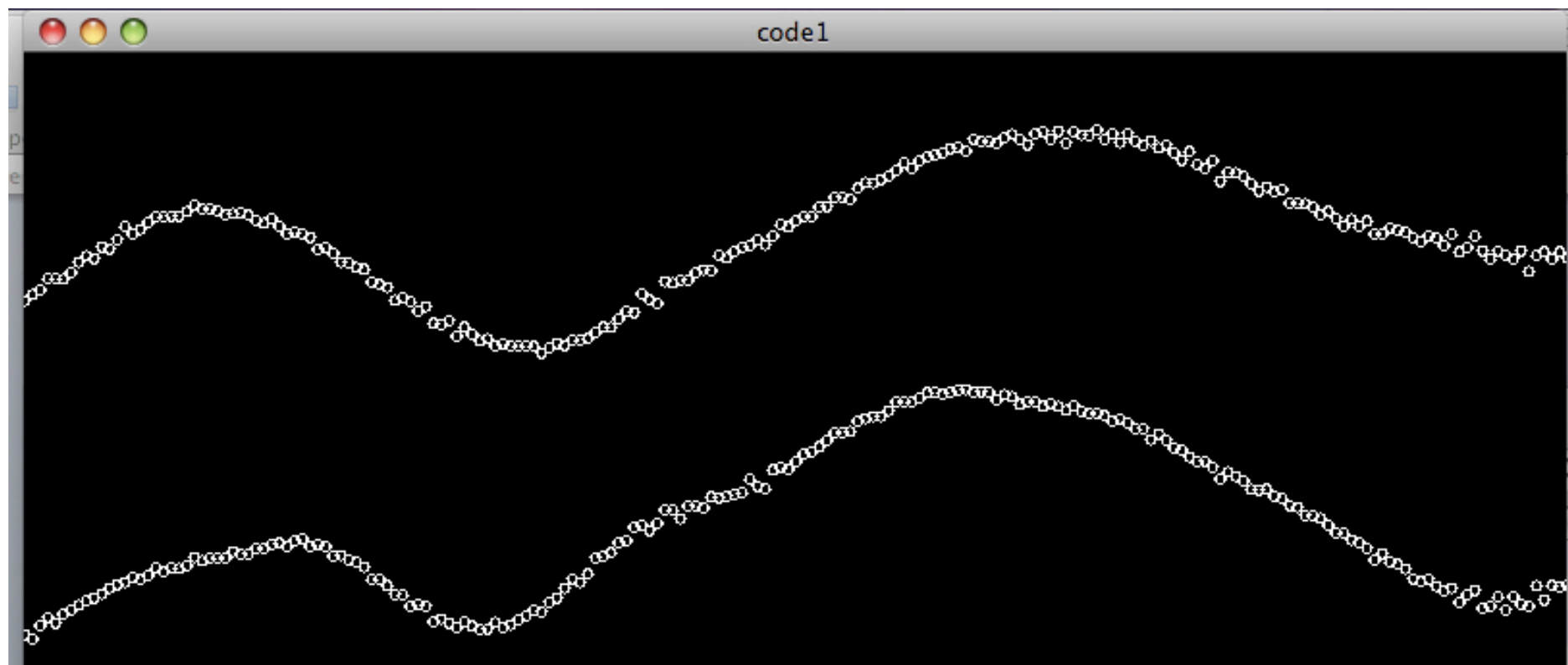
```
import ddf.minim.*;  
AudioPlayer song;  
Minim minim;  
  
void setup()  
{  
  size (800, 800);  
  minim = new Minim (this);  
  song = minim.loadFile("song.mp3");  
  //this loads song.mp3 from the data folder  
} //cont'd on next slide
```

Steps

- Core class instantiated
- New objects are created from the core object to do specific tasks like playing files, creating filters, generating tones, etc
- The `AudioPlayer` class provides mono and stereo playback – instantiated by the `loadFile()` method
- Next slide: `draw()`

```
void draw()
{
  background(0);
  stroke(255);
  noFill();
  for (int i=0; i<song.bufferSize() -1; i++)
  {
    ellipse(i*4, 100+song.left.get(i)*100, 5, 5);
    ellipse(i*4, 250+song.right.get(i)*100, 5, 5);
  }
}
```

```
boolean isPlaying = false;  
void mousePressed()  
{  
    if (isPlaying)  
    {  
        song.pause();  
        isPlaying = false;  
    } else {  
        song.play();  
        isPlaying = true;  
    }  
}
```



AudioPlayer class

- *left, right*: are arrays filled with floating-point numbers that represent the left and right channels in the audio file.
- In the `draw()` method of our code, we have used these arrays to draw a line of small ellipses.

Generating sounds with Minim

- Minim defines methods to generate new sounds from equations
- Four fundamental kind of waves can generate sounds:
 - Triangle wave
 - Square wave
 - Sine wave
 - Sawtooth wave

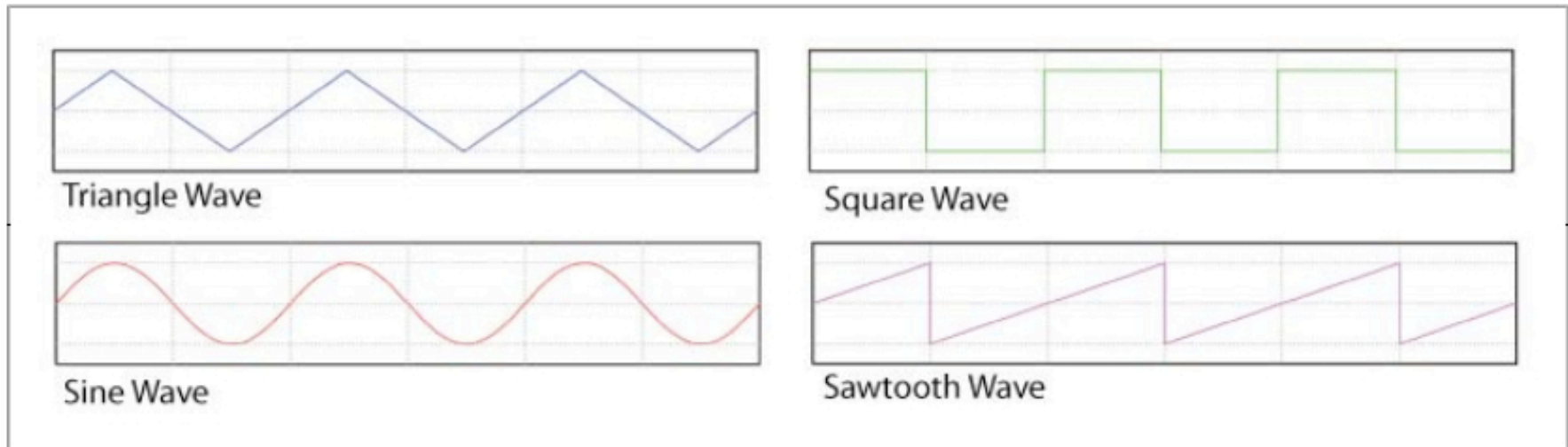


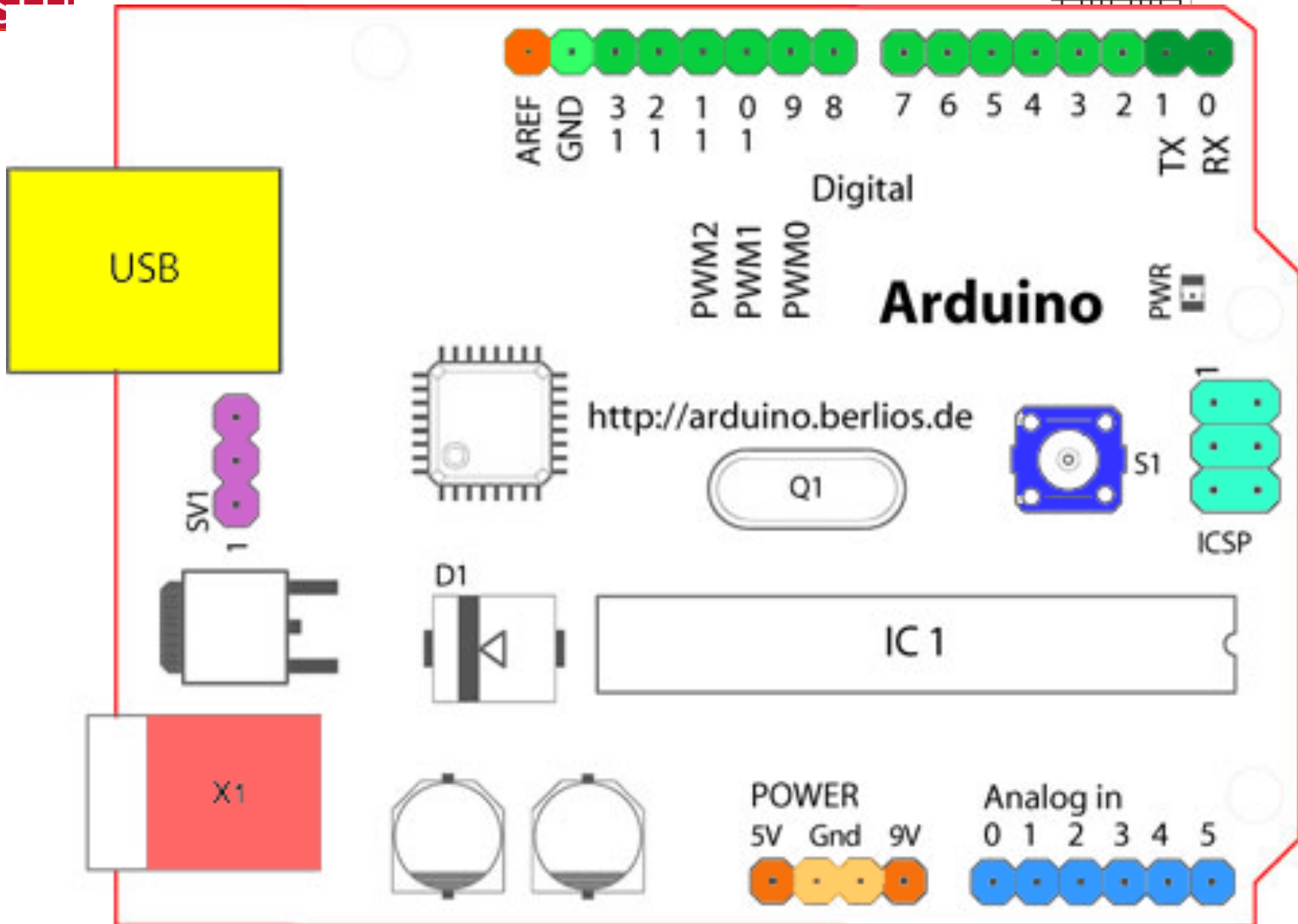
Figure 7-4. Sound wave pattern types

Arduino

- Connecting multiple LEDs to Arduino pins
- LCD displays
- Servomotors
- Play sound with a piezo speaker
 - (these could be needed in some of the projects)

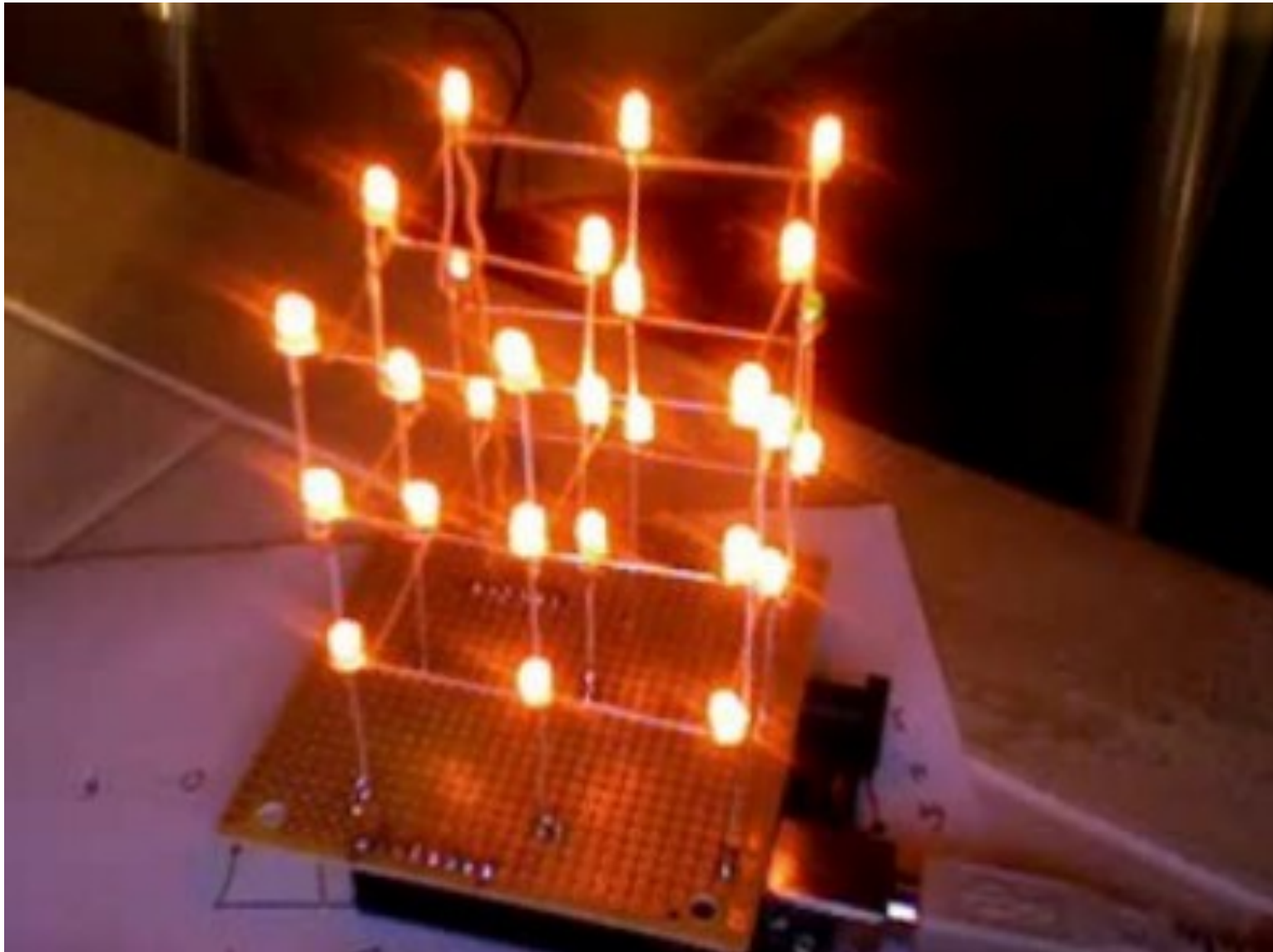
Using an LED matrix

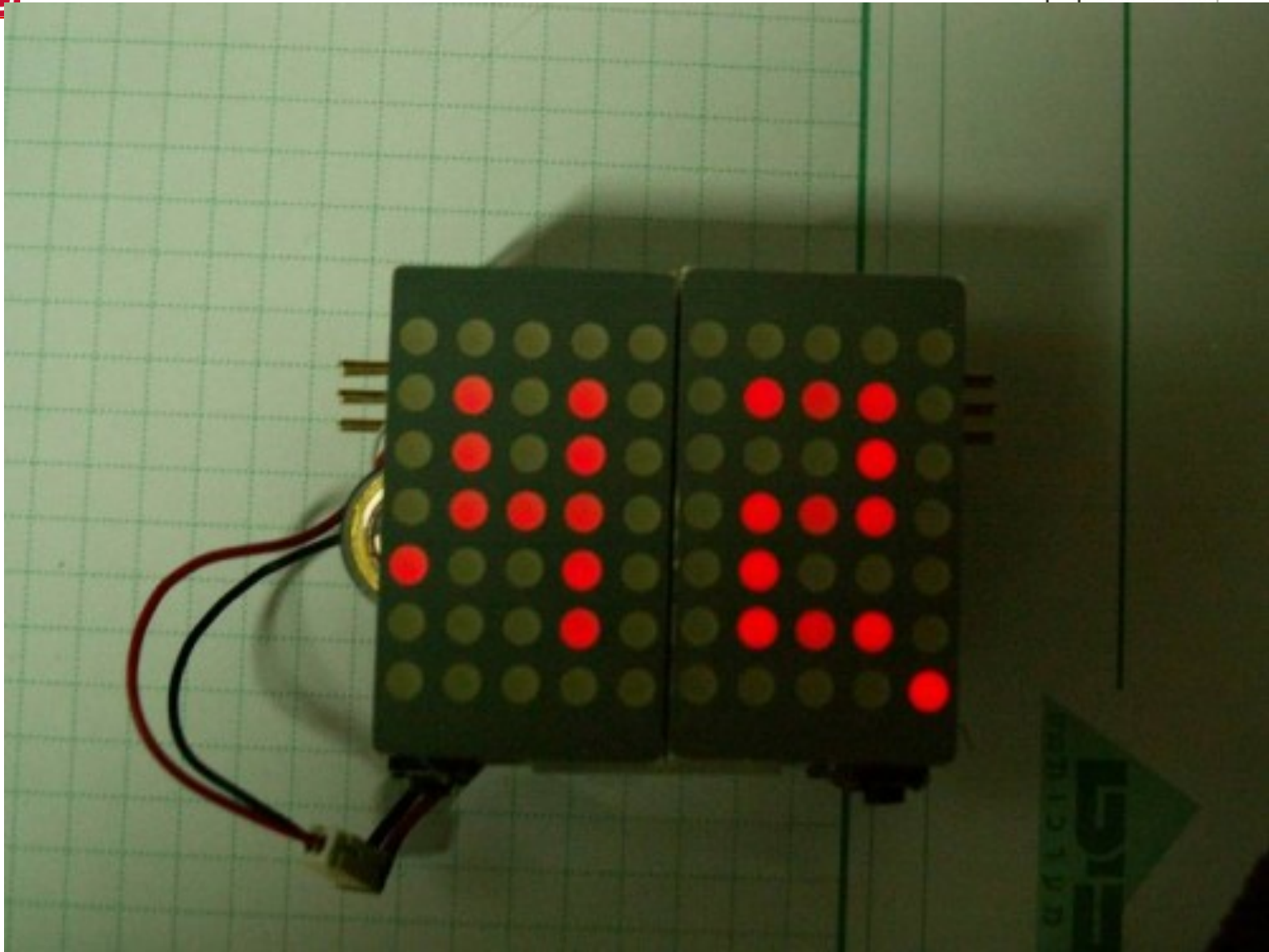
- LED:
 - Can indicate something to a user
 - Can provide recognition of a user action
 - Can act as an alert
- Using multiple LEDs with Arduino is limited by the number of digital out pins that the Arduino has



LED driver chip

- You can control up to an 8x8 matrix of LEDs or an eight-digit LED display
- This lets you draw simple shapes, characters and digits to begin creating simple animations



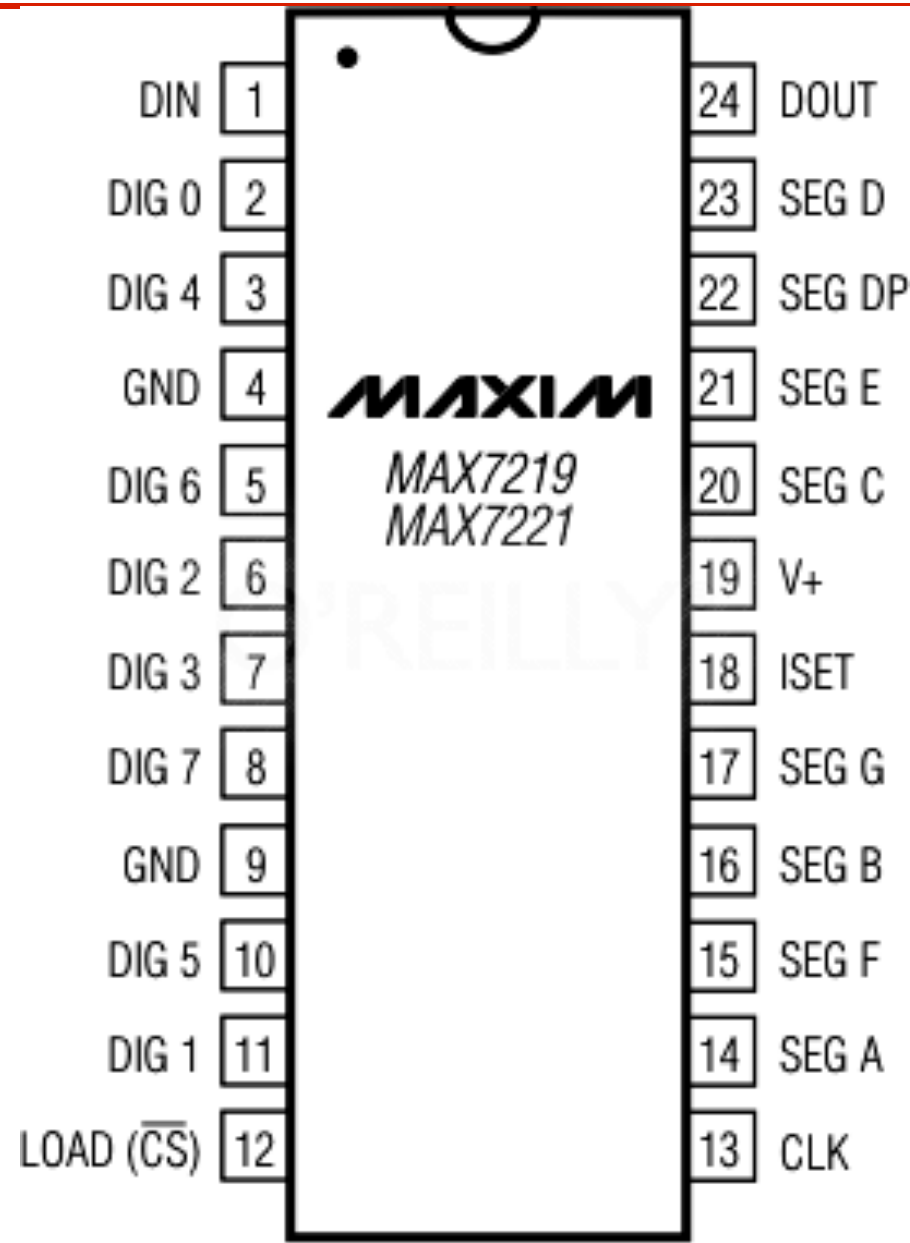


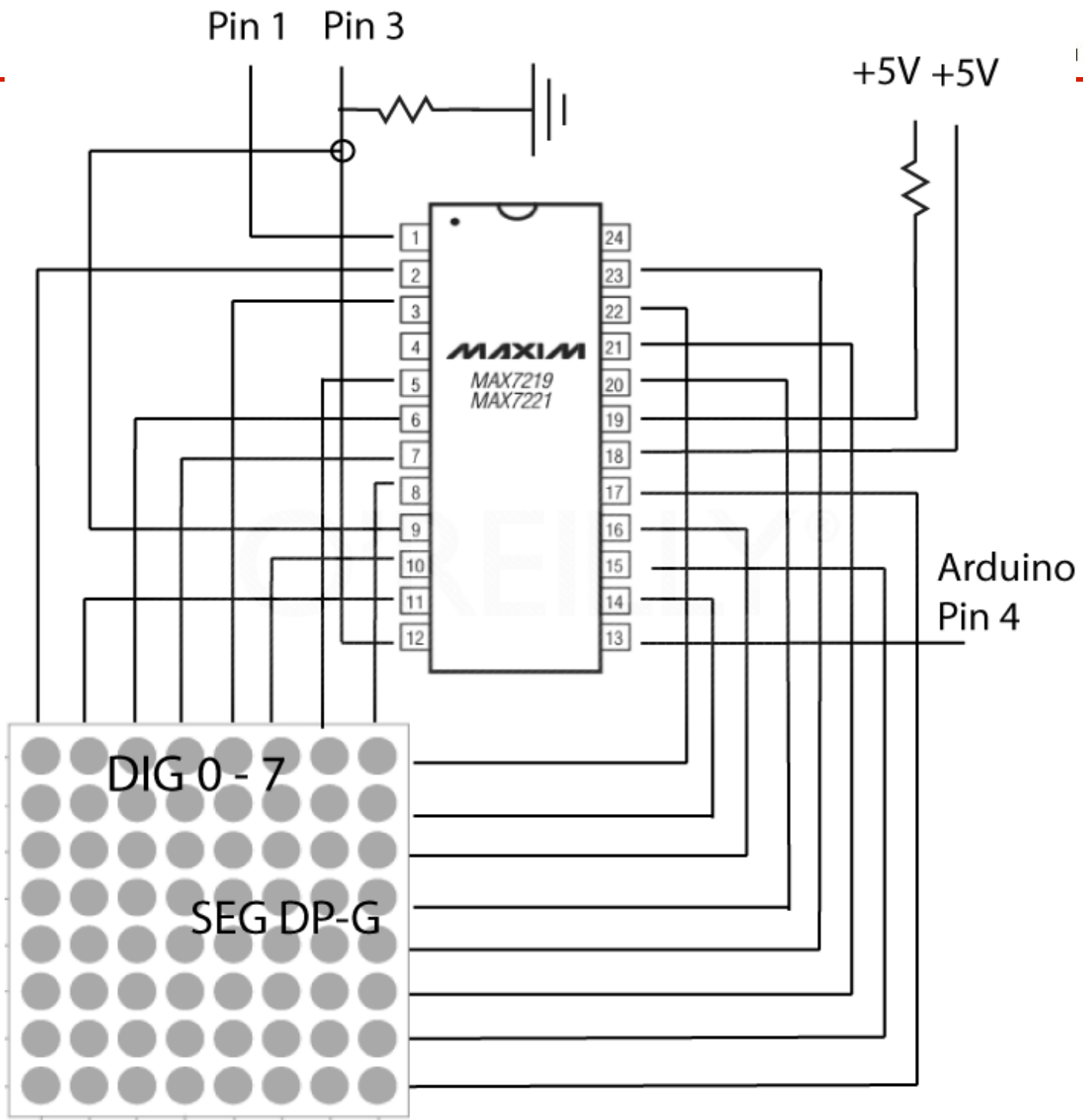
Libraries : LED and Arduino

- **Matrix:** this library enables you to work with a single LED driver
- **LedControl:** enables you to work with multiple LED drivers and newer drivers as well
- **Sprite:** this library allows you to create image sprites to use with the Matrix library

Arduino: Using the Matrix library

- The most common approach to drive an LED matrix is to use a chip called the Maxim MAX7221 to drive the LEDs
- This chip can be wired to an 8x8 Led matrix and an Arduino

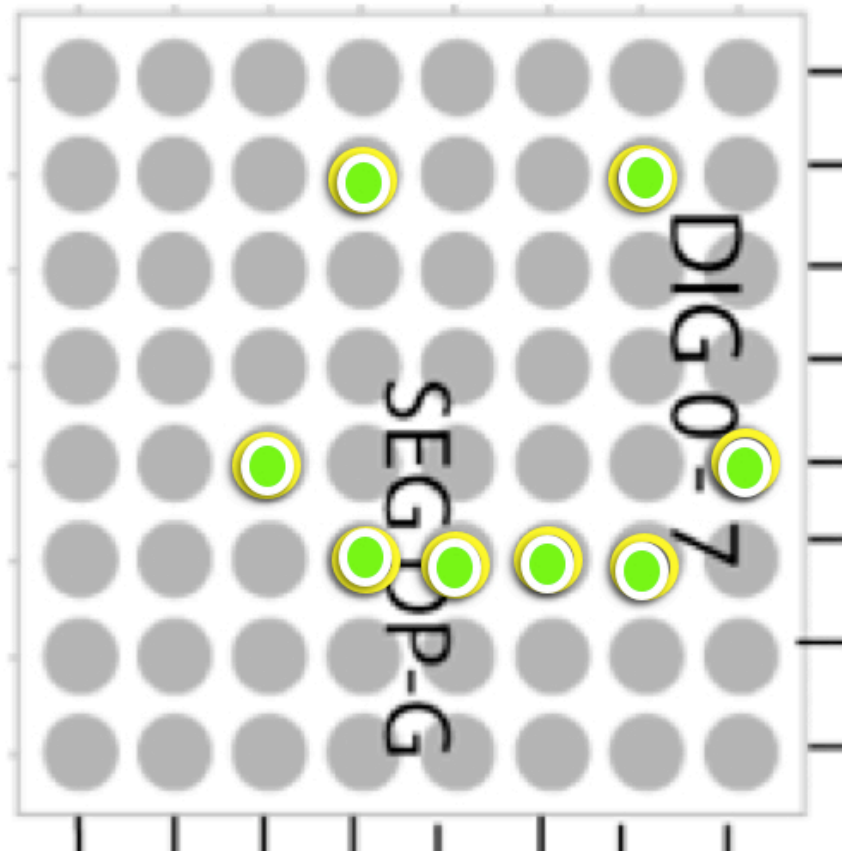




- The Matrix library allows you to communicate easily with the MAX7221 chip
- Individual LEDs can be turned on / off, as well as the brightness or clearing the LED matrix.
- Creating a single LED matrix is easy, the wiring is the only thing that might get a bit tricky...)

Example: creating a smiley face

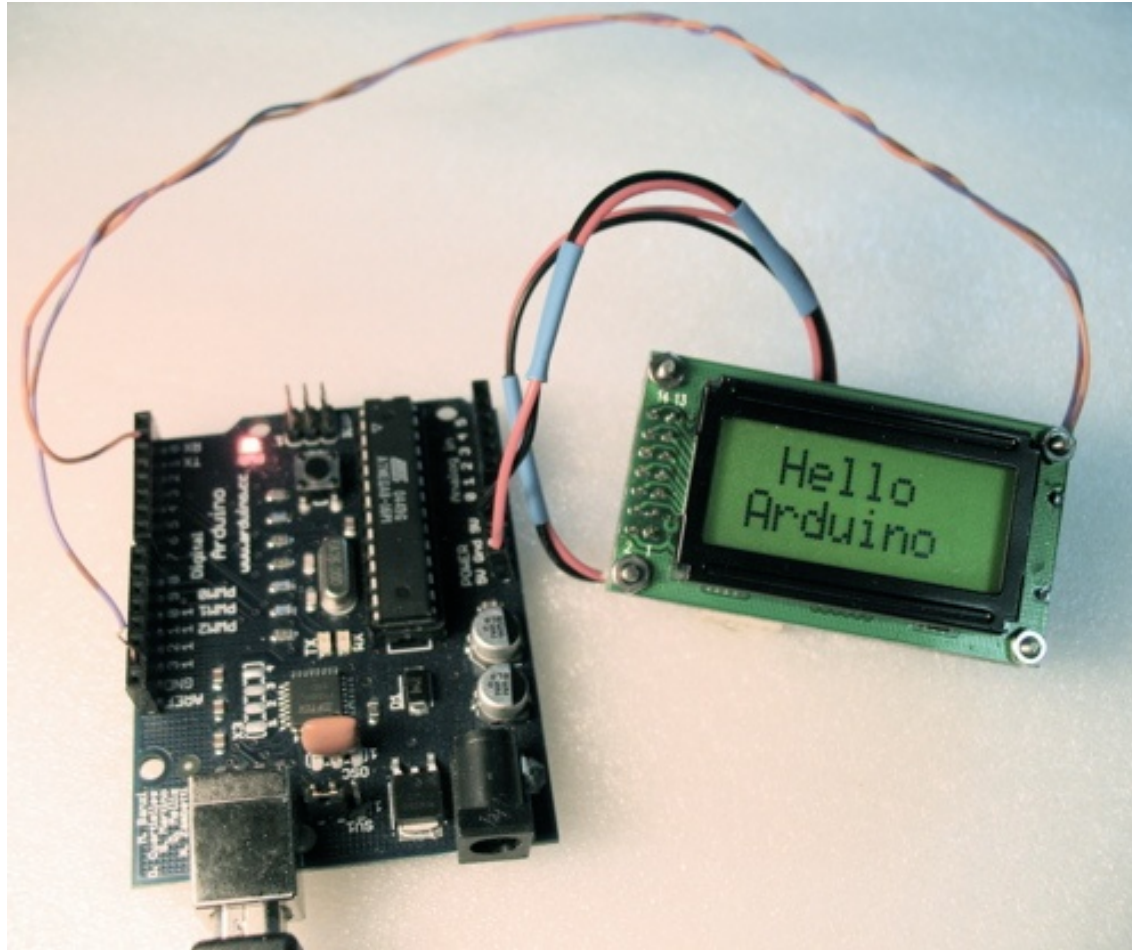
```
mat.write(1, 5, HIGH);  
mat.write(2, 2, HIGH);  
mat.write(2, 6, HIGH);  
mat.write(3, 6, HIGH);  
mat.write(4, 6, HIGH);  
mat.write(5, 2, HIGH);  
mat.write(5, 6, HIGH);  
mat.write(6, 5, HIGH);
```



write() method

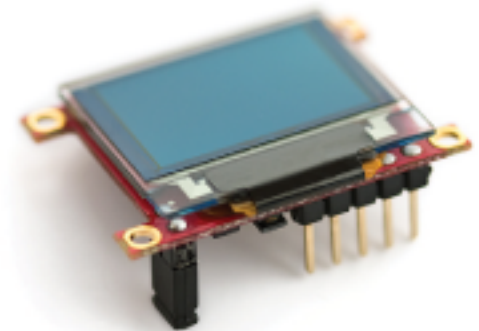
- This method takes two parameters:
 - column of the LED
 - row of the LED

Connecting Arduino to an LCD Display



LCD

- Liquid Crystal Display
- Allows return of data that is more complex than an analog range or a digital value



LCD Displays

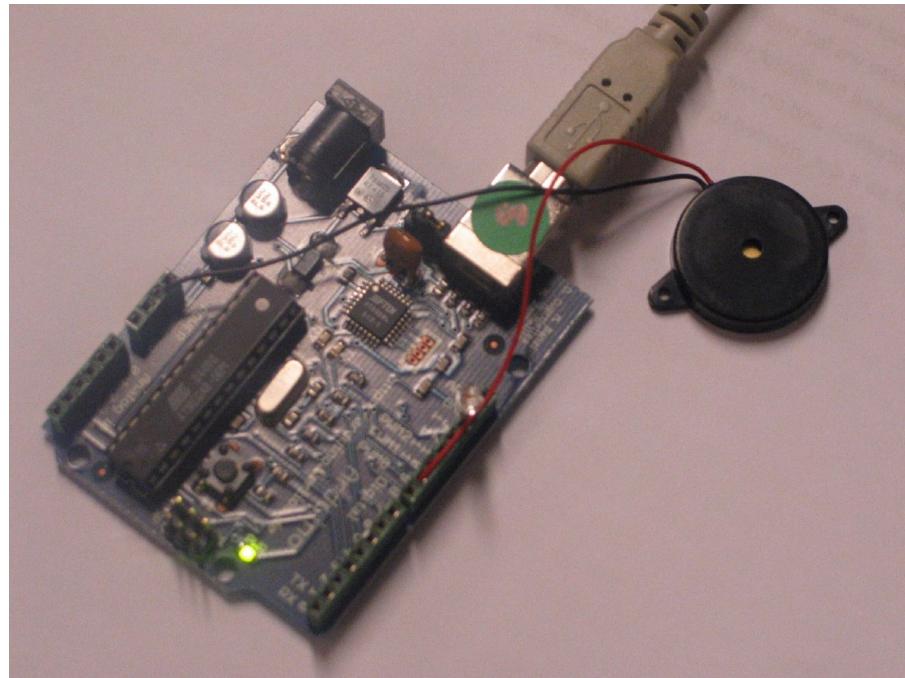
- LCD displays are most of the times driven using an industrial standard established by Hitachi.
- According to it there is a group of pins dedicated to **sending data** and locations of that data on the screen
- The user can choose to use 4 or 8 pins to send data.
- On top of that three more pins are needed to synchronize the communication towards the display.

LCD Arduino Libraries

- LCD - control LCDs (using 8 data lines)
- LCD 4 Bit - control LCDs (using 4 data lines)
- <http://www.arduino.cc/en/Reference/Libraries> - has also an Arduino LCD tutorial available (in case you want to have an LCD for your project)

Arduino and Sound

- Play melodies using a piezospeaker: operates on the principle of the piezoelectric effect
- <http://www.arduino.cc/en/Tutorial/PlayMelody>



Arduino and sound

- Piezospeaker
- Playing melodies makes use of PWM (same principle we used to control the brightness of an LED connected to one of digital pins 9, 10 or 11).
- Piezos have polarity, commercial devices usually having a red and a black wires indicating how to plug it to the board. We connect the black one to ground and the red one to the output.
- Sometimes it is possible to acquire Piezo elements without a plastic housing, then they will just look like a metallic disc.

Connecting multiple sensors to Arduino

- Walk-through code – see code on webct
- Exercise in the workshop on Friday will deal with two sensors

Thank you!

Questions?