Designing Applications that See
Lecture 6: Processing

Dan Maynes-Aminzade

24 January 2008

Reminders

- Assignment #1 handed back next Tuesday
- Assignment #2 released next Tuesday

Learn More Tools for Your Project

- CS247L has open lab sessions about many useful tools for physical prototyping
- Wednesdays 6-8PM in Wallenberg 332
- Upcoming Lab Sessions:
  - January 30: Flash
  - February 6: Mobile Interaction
  - February 13: Soldering and Electronics (meet at CCRMA instead of Wallenberg 332)
  - February 20: Physical Computing With d.Tools
- More info: http://cs247.stanford.edu/lab.html

Today's Goals

- Learn the basics of the Processing environment
- Understand how to produce and publish Processing applets
- Learn how to capture and process live video in the Processing framework
- Experiment with color and motion tracking

Outline

- Processing introduction
- Work through some Processing examples
- JMyron introduction
- Look at basic video processing examples
- Build some motion and color tracking examples
- Add interactivity to our examples

What is Processing?

- An easy-to-use Java compiler
- A development environment
- Focused on interactive graphics, sound, and animation
- Produces both locally-run programs and web-embeddable applets
- Can be used together with “real” Java

Processing Perspective

- A *development* tool for exploring multimedia programming
- An *educational* tool for learning programming fundamentals
- An *ideation* tool or "electronic sketchbook" for trying out ideas
- Targeted for designers, artists, beginning programmers

Nice Things about Processing

- Takes care of a lot of the annoying setup logistics for doing video and graphics in Java
- Easy to create interesting dynamic visuals programmatically
- Allows quick experimentation
- Strong focus on graphics, sound, and simple interactivity (unlike traditional Java programming with a text console)

Getting Help on Processing

- Look at the built-in examples
- More examples:

  http://www.processing.org/learning/

- Function reference:

  http://www.processing.org/reference/

- Discussion forums:

  http://www.processing.org/discourse/

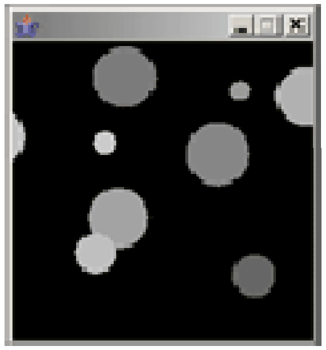
- User-contributed code samples:

  http://www.processinghacks.com/

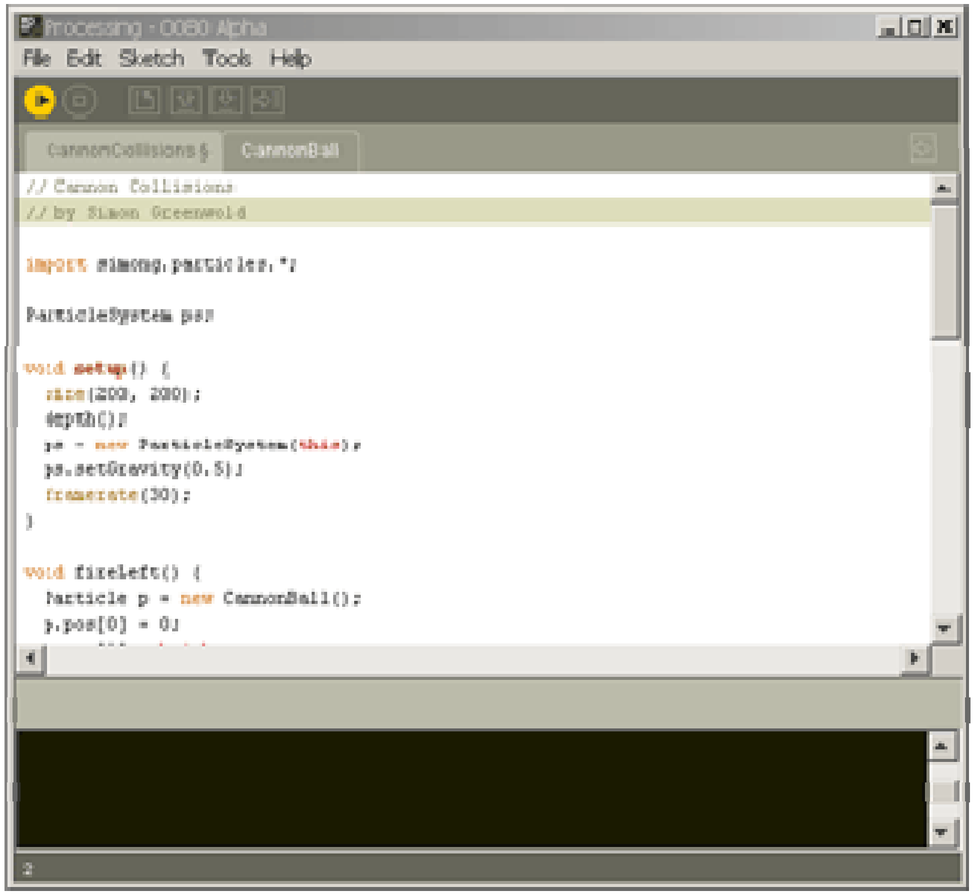
Available Libraries

- Built-In
  - Video
  - Networking
  - Serial Communication
  - Importing XML, SVG
  - Exporting PDF, DXF, etc.
- External Contributions
  - Sound: Ess, Sonia
  - Computer Vision: JMyron, ReacTIVision, BlobDetection
  - Interface: proCONTROLL, Interfascia
  - Many others

A Quick Tour



Display Window



```
// Cannon Collisions
// by Simon Greenwold

import simong.particles.*;

ParticleSystem ps;

void setup() {
  size(200, 200);
  depth();
  ps = new ParticleSystem(this);
  ps.setGravity(0.5);
  framerate(30);
}

void fireLeft() {
  Particle p = new CannonBall();
  p.pos[0] = 0;
```

Menu
Toolbar
Tabs

Text Editor

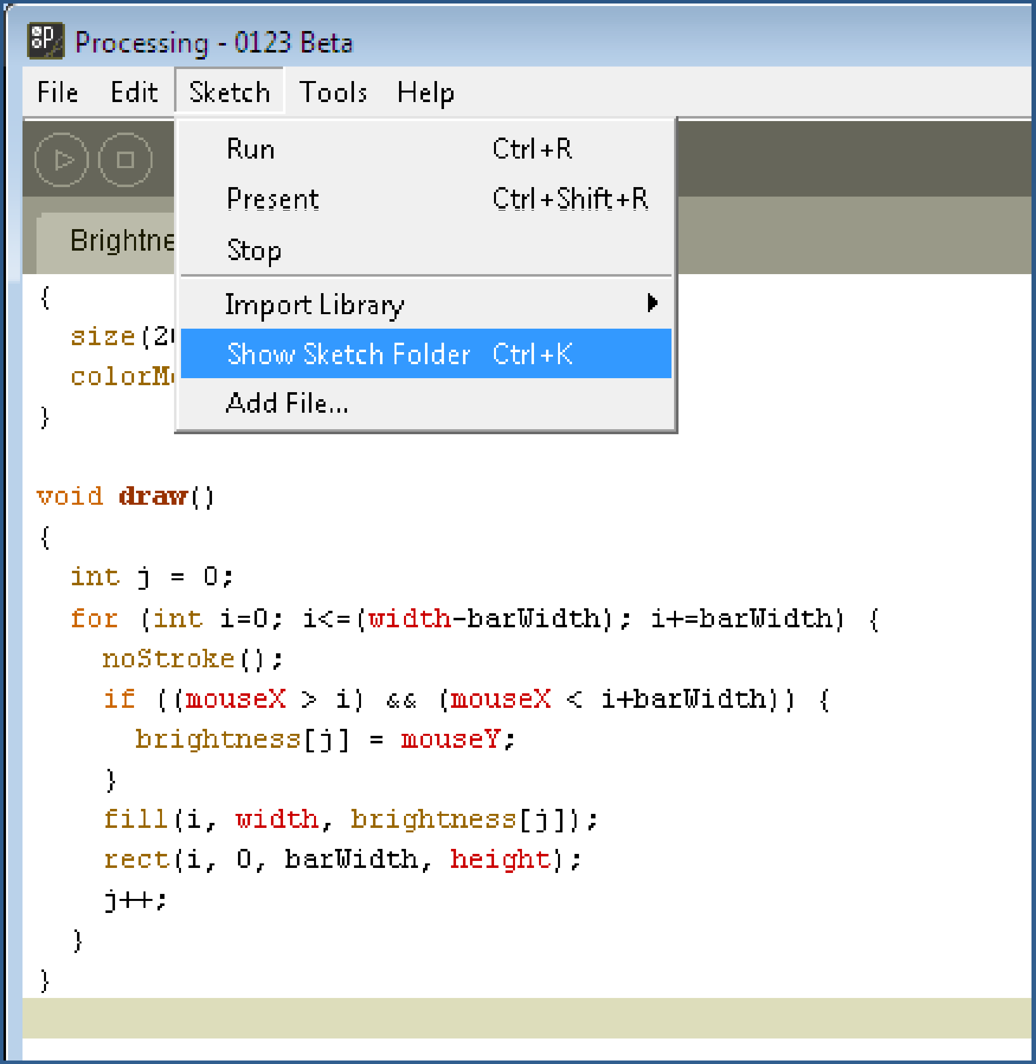
Message Area

Text Area

Toolbar Buttons

- Run
- Stop
- New
- Open
- Save
- Export

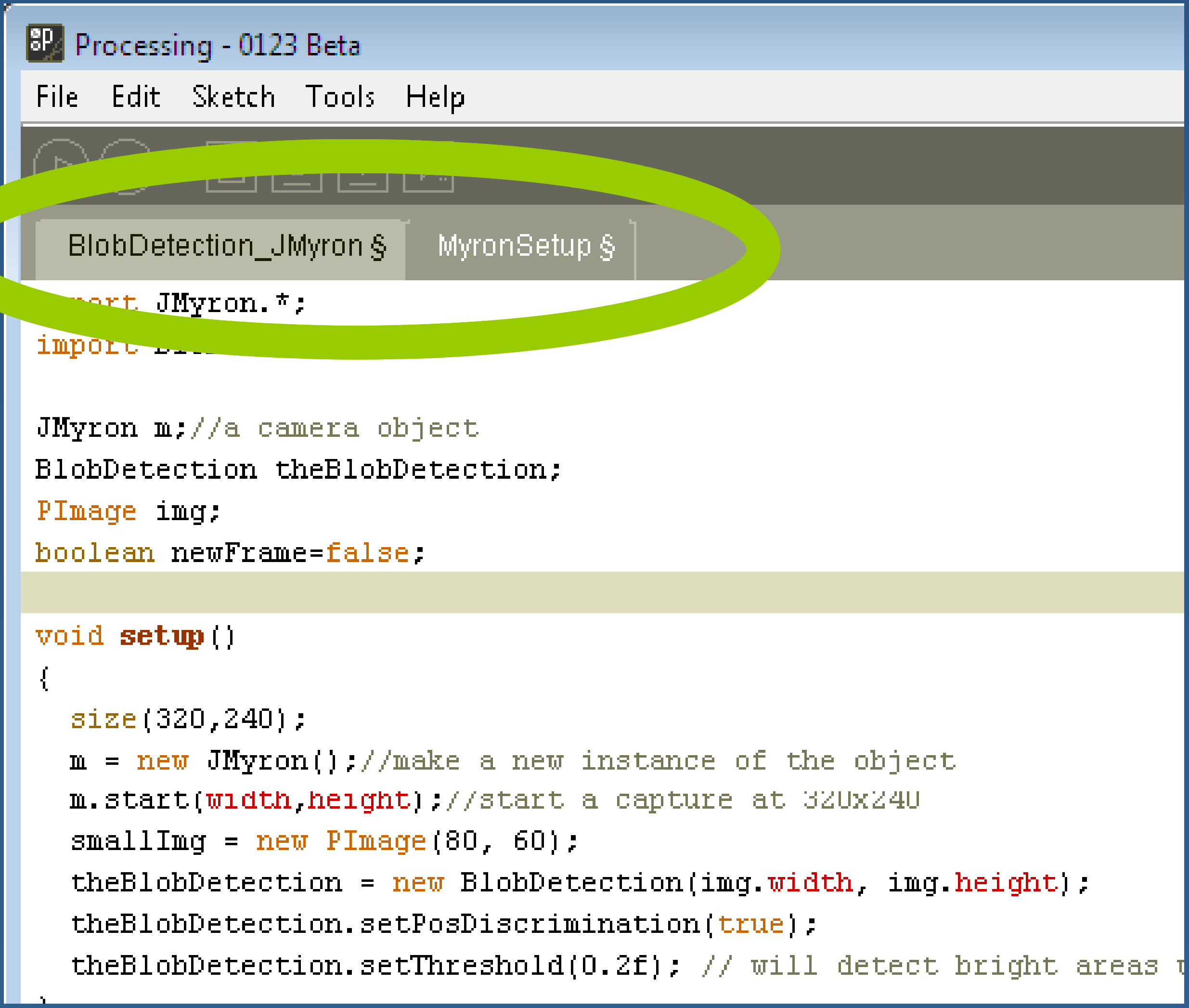
Sketches



```
{
  size(2
  colorM
}

void draw()
{
  int j = 0;
  for (int i=0; i<=(width-barWidth); i+=barWidth) {
    noStroke();
    if ((mouseX > i) && (mouseX < i+barWidth)) {
      brightness[j] = mouseY;
    }
    fill(i, width, brightness[j]);
    rect(i, 0, barWidth, height);
    j++;
  }
}
```

Tabs

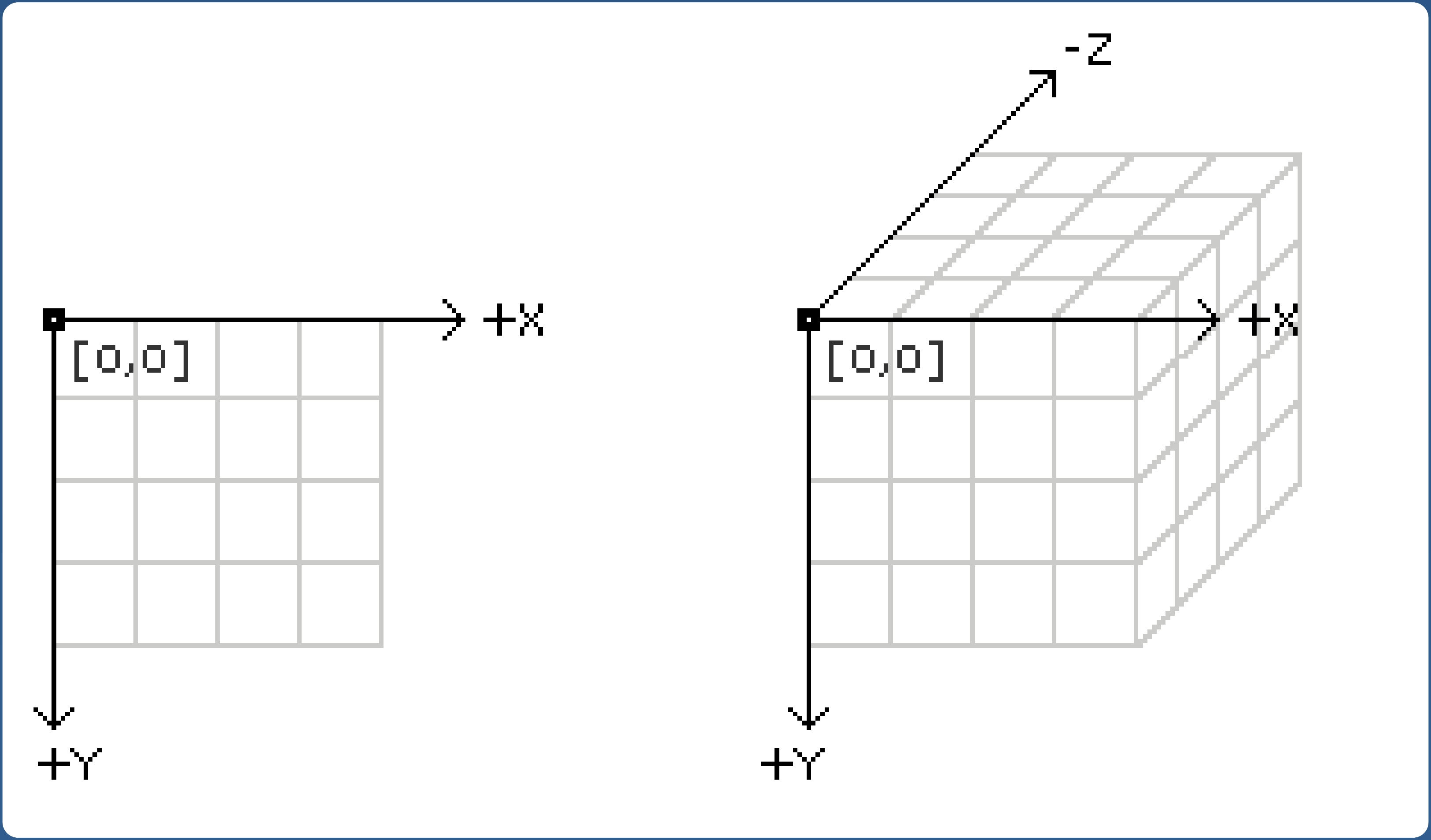


```
    ort JMyron.*;
import Bl

JMyron m;//a camera object
BlobDetection theBlobDetection;
PImage img;
boolean newFrame=false;

void setup()
{
  size(320,240);
  m = new JMyron();//make a new instance of the object
  m.start(width,height);//start a capture at 320x240
  smallImg = new PImage(80, 60);
  theBlobDetection = new BlobDetection(img.width, img.height);
  theBlobDetection.setPosDiscrimination(true);
  theBlobDetection.setThreshold(0.2f); // will detect bright areas w
}
```

Coordinates



Programming Modes

- Basic
  - For drawing static images and learning programming fundamentals
- Continuous
  - Provides a setup() and draw() structures and allows writing custom functions and classes and using keyboard and mouse events
- Java
  - Most flexible mode, giving access to the full Java programming language

Basic Mode

```
size(200, 200);
background(255);
noStroke();
fill(255, 204, 0);
rect(30, 20, 50, 50);
```

Continuous Mode

```
void setup() {
  size(200, 200);
  noStroke();
  background(255);
  fill(0, 102, 153, 204);
  smooth();
  noLoop();
}
void draw() {
  circles(40, 80);
  circles(90, 70);
}
void circles(int x, int y) {
  ellipse(x, y, 50, 50);
  ellipse(x+20, y+20, 60, 60);
}
```

Continuous Mode

```
void setup() {
  size(200, 200);
  rectMode(CENTER);
  noStroke();
  fill(0, 102, 153, 204);
}

void draw() {
  background(255);
  rect(width-mouseX, height-mouseY, 50, 50);
  rect(mouseX, mouseY, 50, 50);
}
```

Java Mode

```
public class MyDemo extends PApplet {
  void setup() {
    size(200, 200);
    rectMode(CENTER);
    noStroke();
    fill(0, 102, 153, 204);
  }
  void draw() {
    background(255);
    rect(width-mouseX, height-mouseY, 50, 50);
    rect(mouseX, mouseY, 50, 50);
  }
}
```

Some Basic Setup Statements

```
// specifies window size
size(200, 200);
// specifies background color
background(102);
// disables filling in shapes
noFill();
// disables drawing lines
noStroke();
// set fill color
fill(255,100,100);
// set stroke color
stroke(100,255,100);
```

Some Basic Drawing Functions

```
// draw a point in the middle
// width and height store the
// window size
point(width/2, height/2);
// draw a 20x20 rectangle
rect(10,10,20,20);
// draw an ellipse
ellipse(50,50,30,30);
// draw an irregular shape
beginShape();
vertex(60, 40); vertex(160, 10);
vertex(170, 150); vertex(60, 150);
endShape();
```

Setup and Draw

```
void setup()  {
  size(200, 200);
  stroke(255);
  frameRate(30);
}
float y = 100;
void draw() {
  background(0);
  y = (y+1) % height;
  line(0, y, width, y);
}
```

noLoop

```
void setup()  {
  size(200, 200);
  stroke(255);
  frameRate(30);
  noLoop();
}
float y = 100;
void draw() {
  background(0);
  y = (y+1) % height;
  line(0, y, width, y);
}
```

Loop

```
void mousePressed() {
  loop();
}
```

Redraw

```
void mousePressed() {
  redraw();
}
```

Event Handlers

```
mouseDragged()
mouseMoved()
mousePressed()
mouseReleased()
...
keyReleased()
keyPressed()
```

Mouse Drawing

```
void setup() {
  size(200, 200);
  background(50);
}
void draw() {
  stroke(255);
  if(mousePressed) {
    line(mouseX, mouseY,
         pmouseX, pmouseY);
  }
}
```

Functions

```
void draw_target(int xloc,
    int yloc, int size, int num) {
  float grayvalues = 255/num;
  float steps = size/num;
  for(int i=0; i<num; i++) {
    fill(i*grayvalues);
    ellipse(xloc, yloc,
     size-i*steps, size-i*steps);
  }
}
```

Other Basic Concepts

- These behave how you would expect (exactly as they do in Java)
  - Data types (int, float, boolean)
  - Arrays
  - Loops
  - Conditionals and Logical Operators
  - Strings
  - Variables and Scoping

Images

```
size(200, 200);
PImage img;
img = loadImage("tennis.jpg");
image(img, 0, 0);
image(img, 0, 0, img.width/10,
  img.height/10);
```

Color Spaces

```
noStroke();
colorMode(RGB, 100);
for(int i=0; i<100; i++) {
  for(int j=0; j<100; j++) {
    stroke(i, j, 0);
    point(i, j);
  }
}
colorMode(HSB, 100);
for(int i=0; i<100; i++) {
  for(int j=0; j<100; j++) {
    stroke(i, j, 100);
    point(i, j);
  }
}
```

Reading Pixel Data

```
PImage img;
size(300,300);
noStroke();
img = loadImage("monzy.jpg");
noLoop();
for (int x=0; x<img.width; x+=5) {
  for (int y=0; y<img.height; y+=5) {
    int pixelcolor =
  img.pixels[x+y*img.width];
    fill(pixelcolor);
    ellipse(x,y,4,4);
  }
}
```

Loading Video

```
import processing.video.*;
Movie myMovie;
void setup() {
  size(320, 240);
  myMovie = new Movie(this, "ball.mov");
  myMovie.loop();
}
void draw(){
// tint(255, 20);
  image(myMovie, mouseX, mouseY);
}
void movieEvent(Movie m) {
  m.read();
}
```

Capturing Video

```
import processing.video.*;
Capture myCapture;
void setup() {
  size(160, 120);
  String s = "Logitech QuickCam Pro 4000-WDM";
  myCapture = new Capture(this, s, width,
    height, 30);
}
void captureEvent(Capture myCapture) {
  myCapture.read();
}
void draw() {
  image(myCapture, 0, 0);
}
```

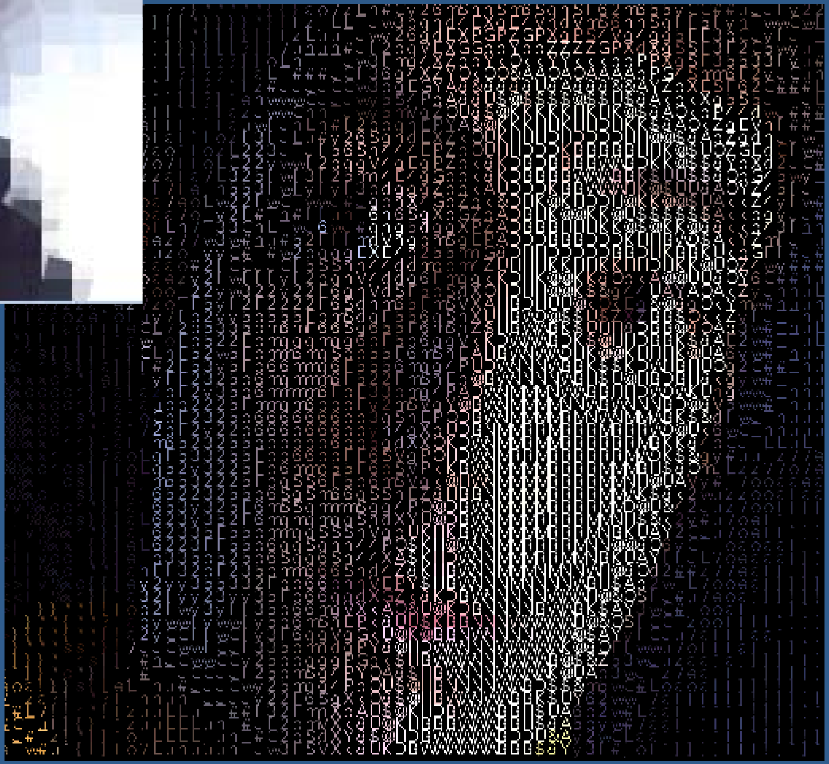
Process Video (Simple)

```
void draw() {
  for (int i=0; i<width; i+=5) {
    for (int j=0; j<height; j+=5) {
      int pixel =
        myCapture.pixels[i+width*j];
      fill(pixel);
      ellipse(i,j,5,5);
    }
  }
}
```

Process Video (More Complex)

- Declare some new global variables:
```
int numPixels;
int blockSize = 10;
color myMovieColors[];
```
- Initialize variables in `setup()`:
```
noStroke();
background(0);
numPixels = width / blockSize;
myMovieColors = new color[numPixels * numPixels];
```
- Add to `captureEvent`:
```
for(int j=0; j<numPixels; j++) {
  for(int i=0; i<numPixels; i++) {
    myMovieColors[j*numPixels + i] =
      myCapture.get(i*blockSize, j*blockSize); } }
```
- Replace `draw()` event:
```
for(int j=0; j<numPixels; j++) {
  for(int i=0; i<numPixels; i++) {
    fill(myMovieColors[j*numPixels + i]);
    rect(i*blockSize, j*blockSize,
         blockSize-1, blockSize-1); } }
```

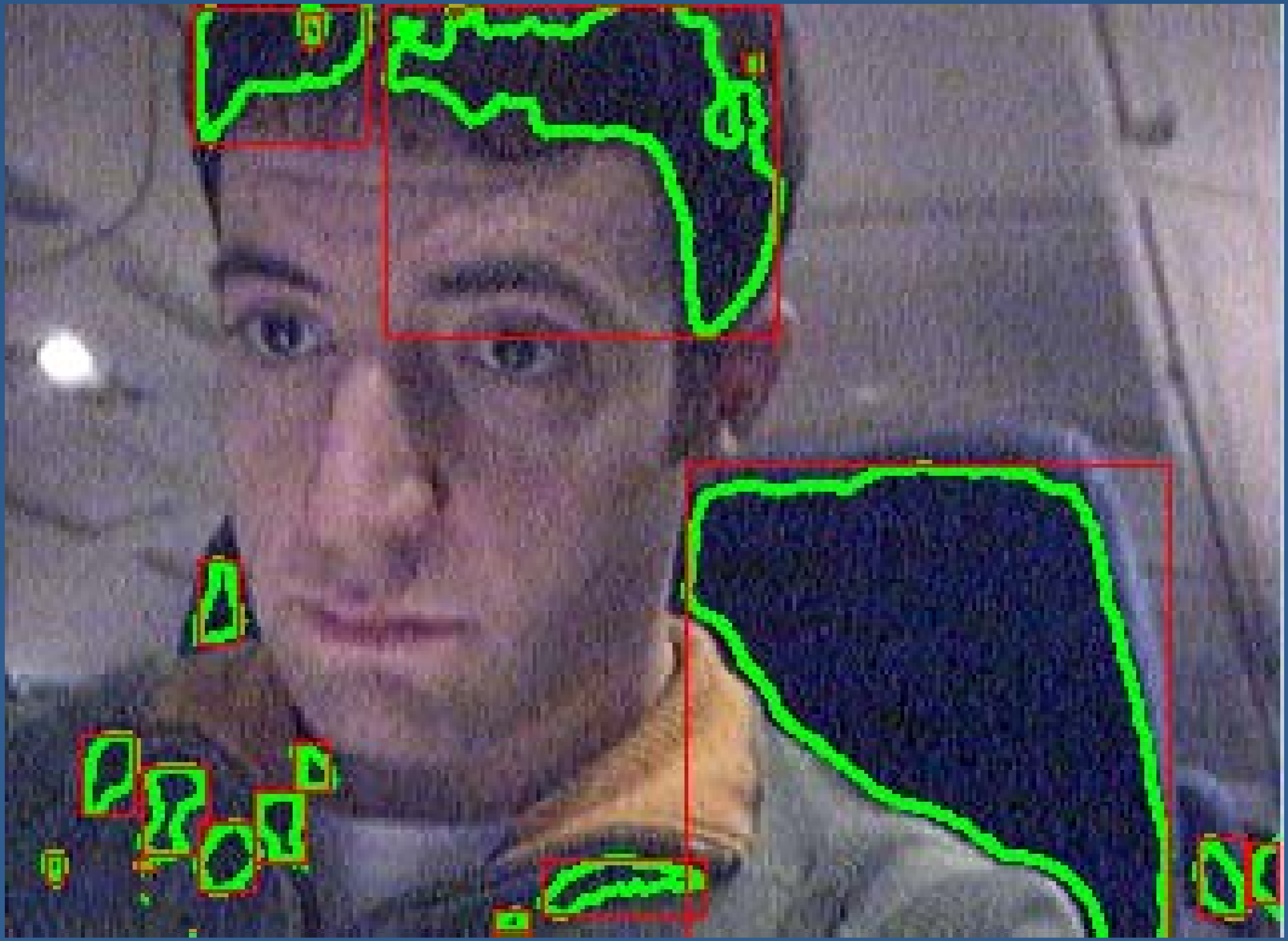
Other Examples



Basic Color Tracking

```
for ( int x=0;x<video.width;x++) {
  for ( int y=0;y<video.height;y++) {
      int loc = x + y*video.width;
      color currentColor =
        video.pixels[loc];
      float r1 = red(currentColor);
      float g1 = green(currentColor);
      float b1 = blue(currentColor);
      float r2 = red(trackColor);
      float g2 = green(trackColor);
      float b2 = blue(trackColor);
      float d = dist(r1,g1,b1,r2,g2,b2);
      if (d < closestDiff) {
        closestDiff = d;
        closestX = x;
        closestY = y;
      }
    }
  }
```

Better Tracking with JMyron



JMyron Setup

```
import JMyron.*;
JMyron m;
void setup(){
  size(320,240);
  m = new JMyron();
  m.start(width,height);
}
```

JMyron Drawing

```
void draw(){
  m.update();//update the camera view
  int[] img = m.image();

  loadPixels();
  for(int i=0;i<width*height;i++) {
    pixels[i] = img[i];
  }
  updatePixels();
}
```

JMyron Cleanup

```
public void stop(){
  m.stop();
  super.stop();
}
```

JMyron Color Tracking

- Setup the color tracking

```
m.trackColor(255,255,255,200);
m.minDensity(100);
```

- Draw boxes around the detected regions

```
int[][] b = m.globBoxes();
for(int i=0;i<b.length;i++) {
  rect(b[i][0],b[i][1],
       b[i][2], b[i][3]);
}
```

Drawing "Globs"

```
int list[][][] = m.globPixels();
for(int i=0; i<list.length;i++) {
  int[][] pixellist = list[i];
  if(pixellist!=null) {
    beginShape(POINTS);
    for(int j=0;j<pixellist.length;j++) {
      vertex(pixellist[j][0],
             pixellist[j][1]);
    }
    endShape();
} }
```

Other Useful JMyron Functions

- Get the average pixel value across a region:

```
int c = m.average(mouseX-20, mouseY-20,
                  mouseX+20, mouseY+20);
```

- Get the center points of the globs:

```
int[][] gcs = m.globCenters();
```

- Get the bounding quads of the globs:

```
int[][] bqs = m.globQuads(20,200);
```

Background Subtraction

- Set rate of adaptivity:

```
m.adaptivity(10);
```

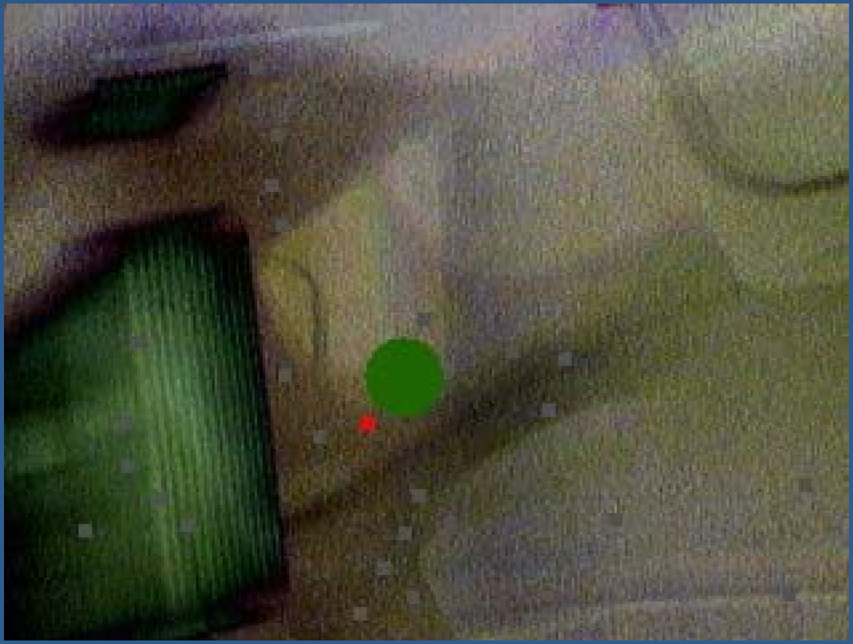
- Take a snapshot of the background for differencing:

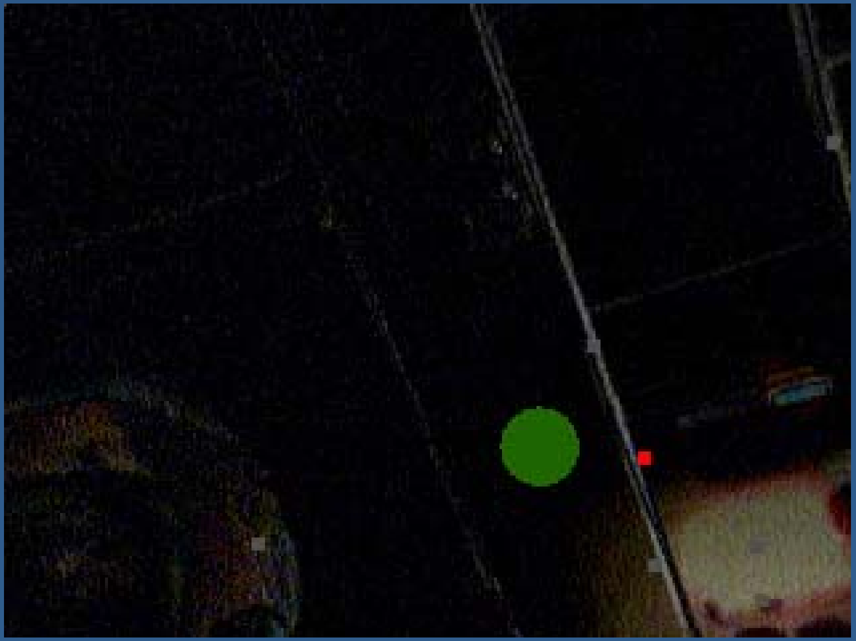
```
m.adapt();
```

- Get the difference image:

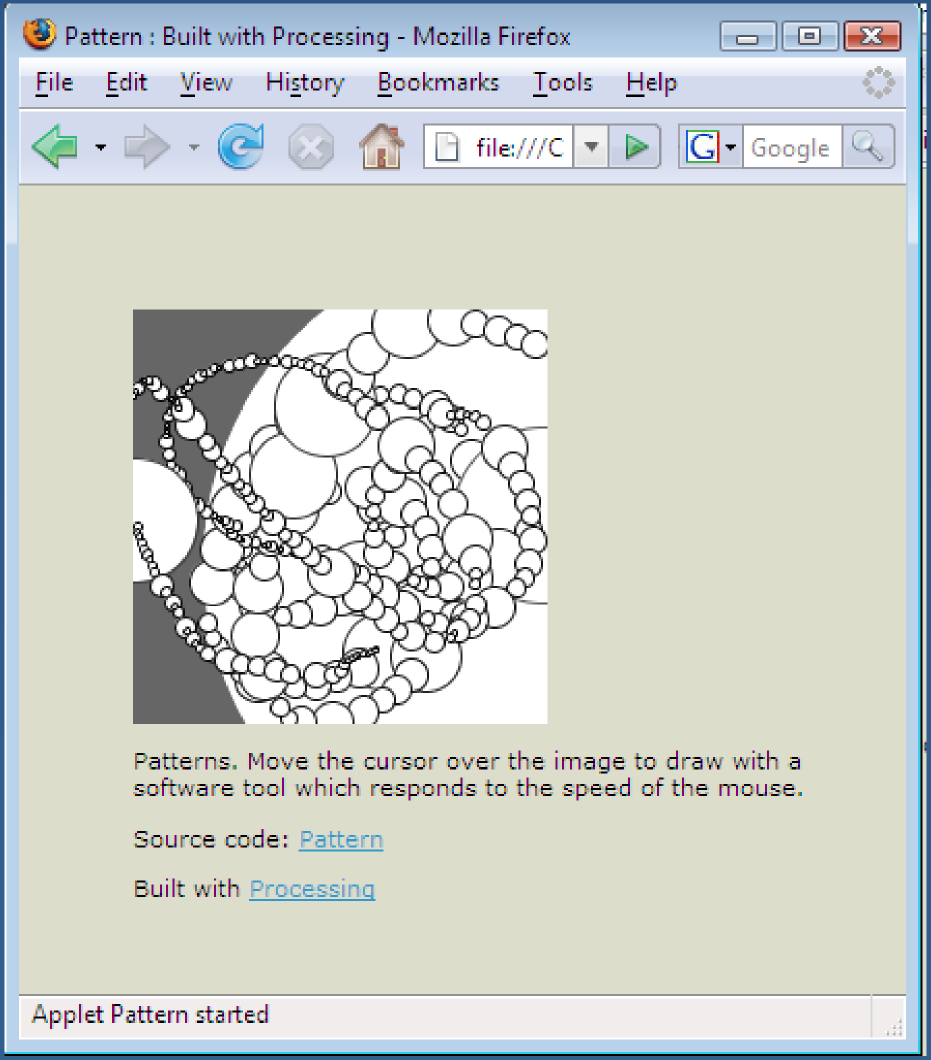
```
int[] img = m.differenceImage();
```

Controlling a Cursor





Exporting an Applet


Pattern : Built with Processing - Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///C
Google

Patterns. Move the cursor over the image to draw with a software tool which responds to the speed of the mouse.

Source code: Pattern

Built with Processing

Applet Pattern started

Signing an Applet

- Generate a keystore:

```
$ keytool -genkey -alias signFiles
  -keystore mystore
  -keypass thepassword
  -dname "CN=projname, OU=name,
   O=company, L=location, S=state,
   C=country" -storepass thepassword
```

- Export a certificate file (optional):

```
$ keytool -export -keystore mystore
  -storepass thepassword
  –alias signFiles
  -file mycertificate.cer
```

- Sign your jar file:

```
$ jarsigner -keystore mystore
  -storepass thepassword -keypass thepassword
  -signedjar output.jar input.jar signFiles
```

(courtesy of Kevin Cox)

Summary

- Processing provides a fun, easy, visual way to program interactive graphics
- Built-in computer vision capabilities are somewhat limited, but you can still do many interesting things (and you could always try doing your own pixel wrangling)
- Check out the examples and take a look at the various external libraries