The Animated Android: Graphical Animation in Processing 2

@stevebattle
http://blog.stevebattle.me
Sketching is a way of thinking

- The idea is to write a few lines of code and have something show up on-screen.

- Processing is Free/Libre/Open Source Software (FLOSS).

- Runs on almost everything: Windows, Android, Mac OS, iOS, Linux

- [http://processing.org](http://processing.org)
Workshop goals

- Introduce programming in Processing 2.0
- Focus on graphics and animation
Getting started

- Download Processing:
  http://www.processing.org

- Installation instructions on the forum:
  http://processing.freeforums.org

- Start Processing

- Create a new project:
  File > New

- Save as ‘AnimatedAndroid_work’
  File > Save
Hello World

1. Type your code in here:
   
   ```
   println("hello world");
   ```

2. Press 'Run'.

3. Output appears in the 'console':
   
   `hello world`
2D shapes

`ellipse(50, 50, 20, 20);`

- **X, Y Coordinates**
- **Width & Height**

![Diagram of an ellipse with coordinates and dimensions labeled]
Co-ordinates

Y increases as you move down-screen
y = 50

0 100
0 x = 50 100
Variables

`int x = 50, y = 50;`  
`background(0);`  
`fill(255);`  
`ellipse(x, y, 20, 20);`

VARIABLES HAVE A TYPE

VARIABLE IS INITIALIZED TO 50.

THIS LINE IS A VARIABLE DECLARATION

VARIABLE NAMES 'x' AND 'y'.

50

x
Data types

Short for ‘integer’
Whole numbers

```
int i = 0;

float greyscale = 3.14159265
```

Numbers with a floating point

*NOTE THE U.S. SPELLING*

```
color orange = color(255,128,0);

boolean selected = true;

byte b = -128; // 2^8 = 256 different values
               // -128 to +127

char input = key; // e.g. 'h' character

String data = "hello world";
```

A string of characters
Download Data files

Go to...
http://github.com/stevebattle/TheAnimatedAndroid

1. Click ‘Download ZIP’ (bottom right-hand corner)
2. Extract and copy to your sketchbook folder.
4. In Processing save as ‘TheAnimatedAndroid_work’
5. Copy ‘data’ folder from ‘TheAnimatedAndroid’
to ‘TheAnimatedAndroid_work’.
# Motion Capture


<table>
<thead>
<tr>
<th>Dots per frame, frames</th>
<th>13</th>
<th>200</th>
</tr>
</thead>
<tbody>
<tr>
<td>Head x,y</td>
<td>333</td>
<td>110</td>
</tr>
<tr>
<td>Left shoulder x,y</td>
<td>341</td>
<td>160</td>
</tr>
<tr>
<td>Left elbow x,y</td>
<td>325</td>
<td>212</td>
</tr>
<tr>
<td>Left wrist x,y</td>
<td>287</td>
<td>245</td>
</tr>
<tr>
<td>Right shoulder x,y</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Right elbow x,y</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Right wrist x,y</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Left hip x,y</td>
<td>341</td>
<td>257</td>
</tr>
<tr>
<td>Left knee x,y</td>
<td>352</td>
<td>352</td>
</tr>
<tr>
<td>Left Ankle x,y</td>
<td>411</td>
<td>396</td>
</tr>
<tr>
<td>Right hip x,y</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Right knee</td>
<td>307</td>
<td>344</td>
</tr>
<tr>
<td>Right ankle</td>
<td>302</td>
<td>416</td>
</tr>
</tbody>
</table>
Setup()

```java
String[] data;
int points, frames;
int frame=0;

void setup() {
    size(650,500);
    data = loadStrings("motion/walk");
    points = int(data[0]);
    frames = int(data[1]);
    frameRate(20);
    print(points);
    print(" ");
    println(frames);
}
```

Explanation coming next.

Setup() is called once at the start.

We want to draw() (coming up) at 20 frames per second.

Output in the console

http://gist.github.com/stevebattle/9655130#file-motion-capture
Arrays

String[] data;

```
data
∅  "13"
1  "200"
2  "333"
3  "110"
...```

This declares a string array
draw()

Processing expects to find setup() and draw() functions. (Though not defining them is not an error).

But you can also define additional functions like point() here.

Use a loop to display all the points.

PVector is a point containing x, y coordinates.

http://gist.github.com/stevebattle/9655130#file-motion-capture
Expressions

\[ 2 + f \times \text{points} \times 2 + p \times 2 \]

\[(\text{frame}+1) \mod \text{frames}\]

\[
\begin{aligned}
\text{Low Precedence} & \quad \{ \\
+ & \quad \text{add} \\
- & \quad \text{subtract} \\
\times & \quad \text{times} \\
/ & \quad \text{divide} \\
\% & \quad \text{modulo (remainder)}
\end{aligned}
\]

**Do Multiplication (and Division) Before Addition (and Subtraction)**

Otherwise work from Left to Right.
The *for* loop

**Initialization happens once at the start of the loop.**

```java
for (INITIALIZE; TEST; INCREMENT) {
    ...
}
```

**The test is evaluated at the start of each iteration.**

**The increment happens at the end of each iteration.**

---

Declare and initialize the loop variable, `i`.

```java
for (int i=0; i<points; i++) {
    ...
}
```

Add 1. Same as `i = i + 1`.

---

Exit the loop when the test is false.
Relational Operators

i<points

<  less than
>
>=  less than or equals
>=  greater than or equals
!=  not equals
==  equals

The result of a relational operator is boolean: true/false.
Robot images from my dad’s toy-robot collection
Drawing images

```
PIimage img;
void setup()
{
    size(650,500);
    data = loadStrings("motion/walk");
    img = loadImage("part/head1.gif");
    points = int(data[0]); frames = int(data[1]);
    frameRate(20);
}

void draw()
{
    background(0);
    imageMode(CENTER);
    for (int i=0; i<points; i++)
    {
        PVector p = point(frame,i);
        ellipse(p.x,p.y,5,5);
    }
    PVector h = point(frame,0);
    image(img,h.x,h.y);
    frame = (frame+1) % frames;
}
```

http://gist.github.com/stevebattle/9655130#file-drawing-images
Scaling

scale(0.5)
Translation

\[ \text{translate}(8,8) \]
Class Head

**Classes are recipes for creating new objects.**

This has no return type, not even 'void' so it's the class constructor.

This function, in the context of a class, is called a method.

```java
class Head {
    PImage img;
    float scaling;

    Head(String filename, float scaling) {
        img = loadImage(filename);
        this.scaling = scaling;
    }

    void draw(PVector head) {
        pushMatrix();
        translate(head.x, head.y);
        scale(scaling);
        image(img, 0, 0);
        popMatrix();
    }
}
```

To create a new class, click here and input the class name.

Saves existing scale, translate (and rotate) settings. Recover earlier scale, translate, (rotate) settings.

[Link](http://gist.github.com/stevebattle/9655130#file-class-head)
Meanwhile, back in the main code

Our 'Head' class defines a new type

Construct a new head object.

Call the draw method on the head object

Object.method()
Rotate

If this circle were a clock, then zero degrees (0°) is at 3 o'clock.

Conventionally, positive angles sweep anti-clockwise so 90° is at 12 o'clock, and 180° is 9 o'clock.

But because the y axis is reversed, positive angles sweep clockwise.
A little Geometry

The midpoint between the elbow (a) and wrist (b) is the axis of rotation.

The angle between the elbow (a) and wrist (b) is the angle at which we draw the forearm.

```
class Part {
    PImage img;
    float scaling;

    PVector midpoint(PVector a, PVector b) {
        return new PVector((a.x+b.x)/2,(a.y+b.y)/2);
    }

    float angleBetween(PVector a, PVector b) {
        return atan2(b.y-a.y, b.x-a.x);
    }
}
```

http://gist.github.com/stevebattle/9655130#file-class-part
Radian angles and their conversions:

- $180° = \pi \text{ radians}$
- $90° = \frac{\pi}{2} \text{ radians}$
- $360° = 2\pi \text{ radians}$

When we use these radial ratios as angles, call them radians, in these diagrams, we can use special names:

- $\pi$ (Pi)
- $\tau$ (Tau)

**The ratio of the circumference of a circle to its radius is $2\pi$ (two Pi) or $\tau$ (Tau), or one full turn.**

**The arm graphic runs vertically, from 12 o'clock to 6 o'clock.**

**Subtract $90° = \pi/2 \text{ radians}$ to get the angle from the vertical at 6 o'clock.**
class Part

Part(String filename, float scaling) {
    img = loadImage(filename);
    this.scaling = scaling;
}

draw(PVector a, PVector b) {
    pushMatrix();
    PVector m = midpoint(a,b);
    translate(m.x,m.y);
    scale(scaling);
    rotate(angleBetween(a,b)-PI/2);
    image(img,0,0);
    popMatrix();
}

* TRANSLATE THE ORIGIN TO THE MIDPOINT OF THE FOREARM BEFORE SCALING AND ROTATING.
* TRANSLATE, SCALE AND ROTATE BEFORE DRAWING THE IMAGE.
* POPMATRIX RESETS THE COORDINATE SYSTEM BACK TO THE PREVIOUS PUSHMATRIX.

http://gist.github.com/stevebattle/9655130#file-class-part
Drawing order

- Add remaining parts.

```java
void draw() {
    background(255); fill(0);
    imageMode(CENTER);
    for (int i=0; i<points; i++) {
        PVector p = point(frame,i);
        fill(0);
        if (p!=null) ellipse(p.x,p.y,5,5);
    }
    head.draw(point(frame,HEAD));
    forearm.draw(point(frame,LEFT_ELbow),point(frame,LEFT_WRIST));
    upperArm.draw(point(frame,LEFT_SHOULDER),point(frame,LEFT_ELbow));
    frame = (frame+1) % frames;
}
```

http://gist.github.com/stevebattle/9655130#file-drawing-order
Data glitches

- Empty data points (0,0) in the motion data cause jitter.

```java
int RIGHT_HIP = 10;
int RIGHT_KNEE = 11;
int RIGHT_ANKLE = 12;

PVector[] cache = new PVector[13];

PVector point(int f, int p) {
    int i = 2 + f*points*2 + p*2;
    PVector v = new PVector(int(data[i]),int(data[i+1]));
    if (v.x==0 && v.y==0 && cache[p]!=null)
        return cache[p];
    cache[p] = v;
    return v;
}
```

http://gist.github.com/stevebattle/9655130#file-fix-glitches
if statement

if (CONDITION) ...
else ...

if (v.x==0 && v.y==0 && cache[p]!=null) return cache[p];
Boolean Operators

"If both co-ordinates are zero and the cache isn't empty"

\[ v.x == 0 \land v.y == 0 \land \text{cache}[p] \neq \text{null} \]

\&\& and

\| or

! not

* The inputs (operands) to a boolean operator must be boolean.
* The output of a boolean operator is boolean.
Background

- To complete the sci-fi setting, add a suitable background image.
- the size() command must use the dimensions of the background image.

```java
frameRate(20);
bg = loadImage("background/optics_angel.jpg");
}

PImage bg ;

void draw() {
  background(bg);
  imageMode(CENTER);
```

http://gist.github.com/stevebattle/9655130#file-background
Links

- Demo: http://stevebattle.github.io/animated-android/

- Forum: http://processing.freeforums.org

- Processing 2: http://www.processing.org