

Building a Visualization Language

John Resig

<http://ejohn.org/> - <http://twitter.com/jeresig>

Topics

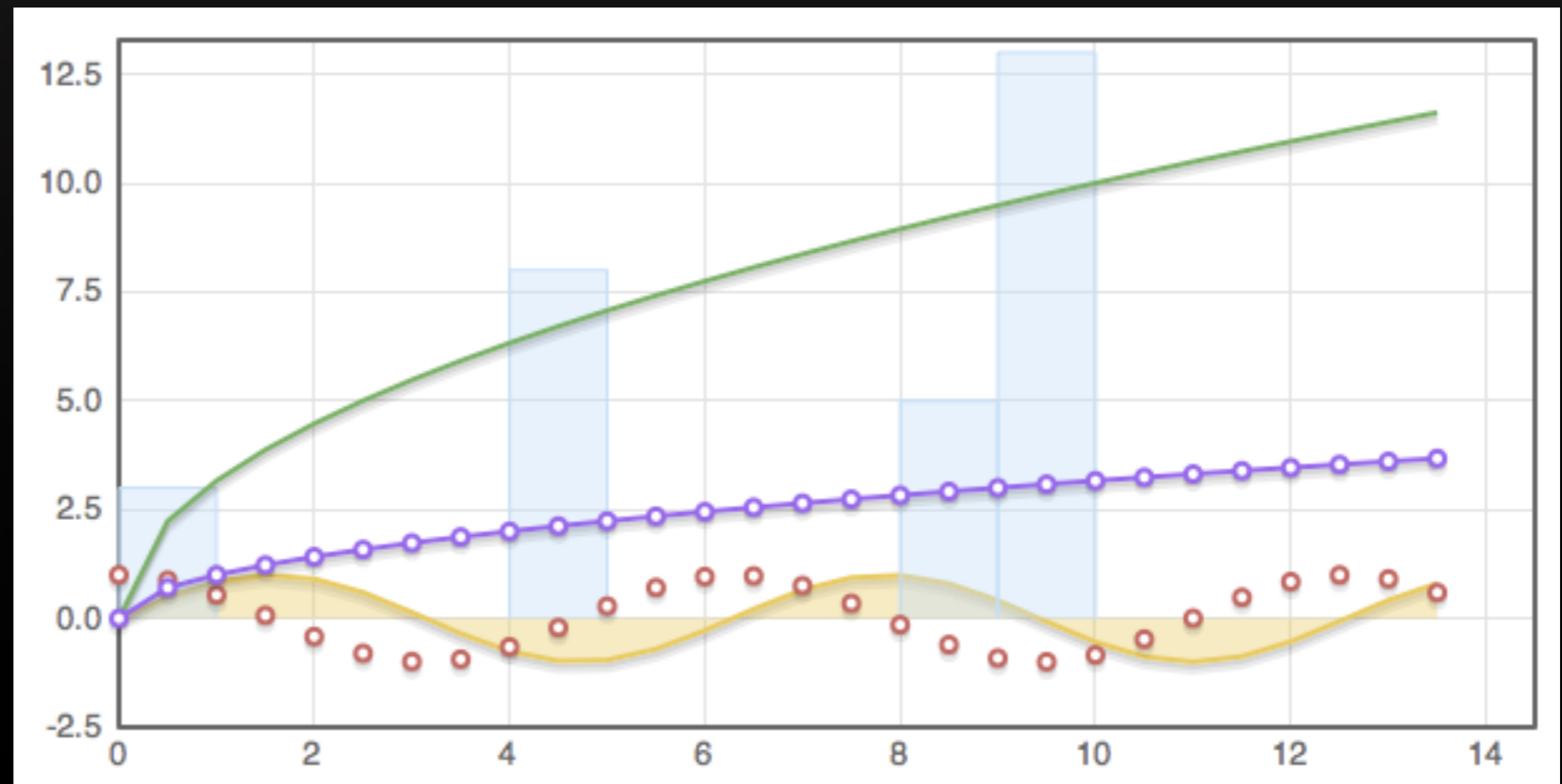
- ◆ Canvas
- ◆ Processing
- ◆ Processing.js

Canvas

- ◆ Proposed and first implemented by Apple in WebKit
- ◆ A 2d drawing layer
 - ◆ With possibilities for future expansion
- ◆ Embedded in web pages (looks and behaves like an img element)

Shapes and Paths



- ◆ NOT vectors (unlike SVG)
- ◆ Rectangles
- ◆ Arcs
- ◆ Lines
- ◆ Curves
- ◆ Charts:







Fill and Stroke


- ◆ All can be styled (similar to CSS)
- ◆ Stroke styles the path (or outline)
- ◆ Fill is the color of the interior
- ◆ Sparklines:

Examples

Generate inline  line graphs 

Customize size and colours 

Bar charts too:   negative values: 

Tristate charts (think games won, lost or drawn): 

Canvas Demo

Canvas Embedding

- ◆ Canvases can consume:
 - ◆ Images
 - ◆ Other Canvases
- ◆ Will be able to consume (Firefox 3.1):
 - ◆ Video
- ◆ In an extension:
 - ◆ Web Pages

Processing

- ◆ Data Visualization programming language
- ◆ Built on top of Java
- ◆ Can do 2d and 3d
- ◆ Many libraries (manipulate video, images, audio)

The Language

- ◆ Strictly typed
- ◆ Has classes, inheritance
- ◆ A bunch of globally-accessible functions
 - ◆ (Very flat API)
- ◆ `setup()` and `draw()` methods
 - ◆ Very OpenGL-like
 - ◆ `draw()` is called continually at a specific framerate

Draw A Line w/ Mouse

- ◆ While you hold the mouse down, draw a line from the previous mouse point

- ◆ Topics > Drawing > Continuous Line

- ◆

```
void setup() {  
    size(200, 200);  
    background(102);  
}
```

```
void draw() {  
    stroke(255);  
    if(mousePressed) {  
        line(mouseX, mouseY, pmouseX, pmouseY);  
    }  
}
```

Initialization

- ◆ `setup()` is called initially
- ◆ `size(...)` set the width/height of the drawing area
- ◆ Call any other number of methods, such as:
 - ◆ `background(...)` - draw and fill a background
- ◆

```
void setup() {  
    size(200, 200);  
    background(102);  
}
```

draw() loop

- ◆ draw() gets called as fast as possible, unless a frameRate is specified
- ◆ stroke() sets color of drawing outline
- ◆ fill() sets inside color of drawing
- ◆ mousePressed is true if mouse is down
- ◆ mouseX, mouseY - mouse position
- ◆

```
void draw() {  
  stroke(255);  
  if(mousePressed) {  
    line(mouseX, mouseY, pmouseX, pmouseY);  
  }  
}
```

Drawing

- ◆ Different drawing methods:
 - ◆ `line()`
 - ◆ `rect()`
 - ◆ `arc()`
 - ◆ `ellipse()`
 - ◆ `point()`
 - ◆ `quad()`
 - ◆ `triangle()`
 - ◆ `bezier()`
- ◆ All use `stroke()`, `strokeWeight()`, `fill()`

The Drawing Area

- ◆ Just like Canvas!
- ◆ Mutate the canvas rendering:
 - ◆ `translate()`
 - ◆ `scale()`
 - ◆ `rotate()`
- ◆ Save and Restore the state of the canvas:
 - ◆ `pushMatrix()`
 - ◆ `popMatrix()`
- ◆ Demo: Basics > Transform > Arm

Shapes

- ◆ A series of vertices, built into a shape

- ◆

```
fill(127);
beginShape();
for (int i=0; i<segments; i++){
    vertex(ground[i].x1, ground[i].y1);
    vertex(ground[i].x2, ground[i].y2);
}
vertex(ground[segments-1].x2, height);
vertex(ground[0].x1, height);
endShape(CLOSE);
```

Classes

- ◆ Hold data, do inheritance
- ◆ Demo: Topics > Motion > Reflection 2
- ◆

```
class Ground {  
    float x1, y1, x2, y2, x, y, len, rot;  
    Ground() { }  
    Ground(float x1, float y1, float x2, float y2) {  
        this.x1 = x1; this.y1 = y1; this.x2 = x2; this.y2 = y2;  
        x = (x1+x2)/2;  
        y = (y1+y2)/2;  
        len = dist(x1, y1, x2, y2);  
        rot = atan2((y2-y1), (x2-x1));  
    }  
}
```

Math functions

- ◆ `dist()`, `map()`, `constrain()`, `abs()`,
`floor()`, `ceil()`
- ◆ `random()`, `noise()`
- ◆ `atan2()`, `cos()`, `sin()`, `pow()`, `sqrt()`
- ◆ `radians()`

Images

- ◆ Load in external images

- ◆ Demo: Topics > Animation > Sequential

- ◆

```
int numFrames = 12; // The number of frames in the animation
int frame = 0;
PImage[] images = new PImage[numFrames];
void setup(){
    size(200, 200);
    frameRate(30);
    for(int i=0; i<numFrames; i++) {
        String imageName = "PT_anim" + nf(i, 4) + ".gif";
        images[i] = loadImage(imageName);
    }
}
void draw() {
    frame = (frame+1)%numFrames;
    image(images[frame], 0, 0);
}
```

Processing Demos

- ◆ <http://acg.media.mit.edu/people/fry/zipdecode/>
- ◆ <http://complexification.net/gallery/machines/substrate/>
- ◆ http://sublimeguile.com/processing/tree_071017b/
- ◆ <http://vimeo.com/311550>

Processing.js

- ◆ How do we run Processing on web pages?
- ◆ Applets suck
- ◆ Processing: Convert Processing to JavaScript and run on Canvas
- ◆ Two stages:
 - ◆ Language Conversion
 - ◆ Processing API

Ported to JS

- ◆ Released in May:
<http://ejohn.org/apps/processing.js/>
- ◆ Port of the Processing Language +
the 2d Processing API
- ◆ All runs in JavaScript on top of HTML 5
Canvas
- ◆ Works in all browsers (IE with excanvas)

Porting

- ◆ The API
 - ◆ Rather straight-forward
 - ◆ 1-to-1 mapping of Processing to Canvas
- ◆ The Language
 - ◆ Much more complicated
 - ◆ Parse the full language
 - ◆ (lots of Regular Expressions)

Previous Demos

- ◆ <http://ejohn.org/apps/processing.js/examples/topics/continuouslines.html>
- ◆ <http://ejohn.org/apps/processing.js/examples/basic/arm.html>
- ◆ <http://ejohn.org/apps/processing.js/examples/topics/reflection2.html>
- ◆ <http://ejohn.org/apps/processing.js/examples/topics/sequential.html>

Processing.js Demos

- ◆ <http://ejohn.org/apps/processing.js/examples/custom/snake.html>
- ◆ <http://ejohn.org/apps/processing.js/examples/custom/molten.html>
- ◆ <http://ejohn.org/apps/processing.js/examples/basic/>
 - ◆ Recursion
 - ◆ Distance 2D
 - ◆ Random
- ◆ <http://ejohn.org/apps/processing.js/examples/topics/>
 - ◆ Soft Body
 - ◆ Flocking
 - ◆ Tree
- ◆ <http://ejohn.org/apps/hero/>