HTML5 Graphical Programming with Processing.js

About This Reference

This write-up was written to help explain what the code does not. While it includes all the code and resource links, a document is not a good place to check out the results of a program or to click on web links. Please see the index.html file instead as it will link to all the examples below and provide links to the listed resources.

Background

The Processing Language was created by Ben Fry and Casey Reas at MIT in the year 2000. It's goal was to simplify graphics programming for people who did not have a really strong programming background.

> *"Processing is an open source programming language and environment for people who want to create images, animations, and interactions. Initially developed to serve as a software sketchbook and to teach fundamentals of computer programming within a visual context, Processing also has evolved into a tool for generating finished professional work." -- Processing.org*

Processing is Java based. The programs written using Processing (called sketches) were deployed as an applet. When the language was first created, Java was expected to be come the client side programming language of the web (applets everywhere!). However, that is not what happened. Initializing and sandboxing the JVM takes time and Java plug-ins are often out of date or disabled. On some platforms plug-ins are not even an option (iOS for example).

When Processing was created, JavaScript was not very fast. It was viewed as a language that you could use to change a bit of content here and there but nothing serious. However, over the last decade JavaScript has gotten amazingly fast. It is THE language of the web, understood by all the major browsers and across every platform whether it is desktop or mobile. Today, there are serious applications being developed using JavaScript.

Processing.js was a project initiated in 2008 by John Resig who also started JQuery. He wanted to try to write something that used the <canvas> element. Thus, he started Processing.js which would translate Processing code into JavaScript and render it using in the

<canvas> element. He proved that it was possible to do this and released his work as an open source project called Processing.js.

In the fall of 2009, a group of students from Seneca College led by their Professor David Humphrey began the job of bringing Processing.js to parity with Processing as part of their open source course. Over the next year, they along with a few other outside contributors, added all the 3D functionalities of Processing.js, fixed many bugs, increased its efficiency and tested the code over multiple platforms. In 2010, Processing.js 1.0 was release. Today, Processing.js is at version 1.4.

An Introduction to the Processing Language

This section is for those who have never used the Processing Language before. If you are comfortable with Processing, please feel free to jump ahead to the next section.

I have made all sketches below are available at:

http://seneca-cdot.sketchpad.cc

Please feel free to sign-up, clone and alter it. You are also welcome to copy them and use it for your own class. If you wish to set up a private studio for your class, you can contact studio sketchpad and they will make one for you.

The Processing Language is Java based and thus it essentially follows the same syntactic rules as Java. While it is possible to create objects, it does not have to be written in an object-oriented manner. It is possible to write an entire sketch without objects.

The Processing Language has a fairly large library of functions that draws/alters basic shapes. It would not be possible to fully cover them all in this workshop. Please refer to the reference pages on these functions and their usage:

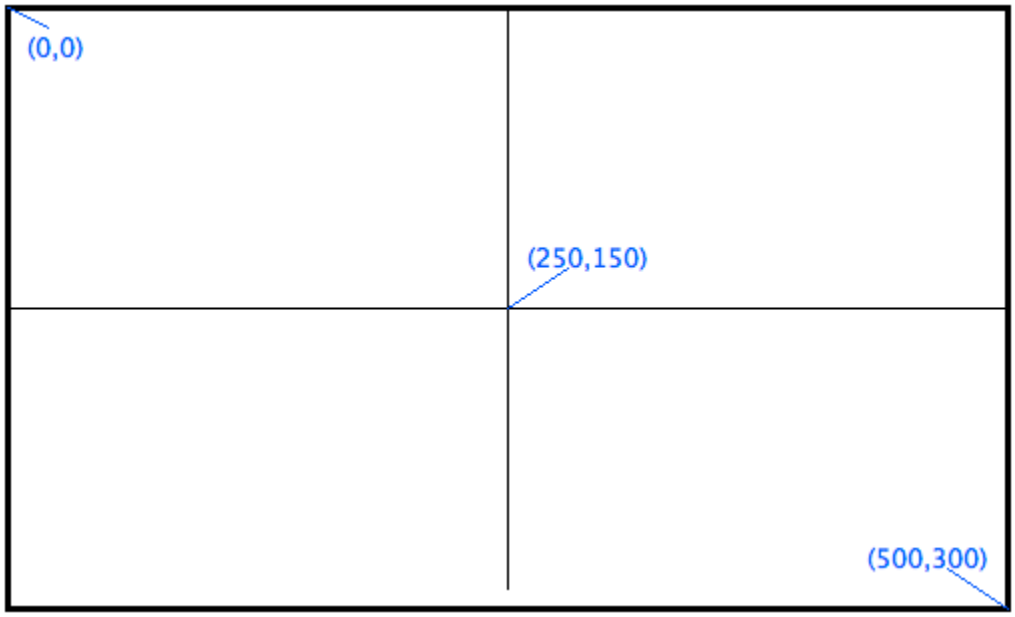
http://processingjs.org/reference/

NOTE: I am including the Processing.js reference instead of the Processing reference here. For the most part there aren't any major differences between them however, there are some functionalities that are not implemented for Processing.js (mostly related to files which do not make sense in a web context). It is best to be aware of this and avoid their usage.

Workshop Example 1: A Simple Sketch

```
size(500,300);            //sets the size of the sketch
                          //500 px wide, 300 px tall

ellipse(250,150,50,50);   //draws a 50px diameter circle in the
                          //middle of the sketch.
                          //ellipse(x,y,width,height)
```

The Processing coordinate system places 0,0 at the top left corner with increases values of x and y. The bottom right corner will therefore have coordinates equal to the size of the sketch. If you are trying to simply create an image programmatically you can do so simply by typing in the api commands to draw what you wish



Colours

Processing allows you to specify colours in different ways depending if you want greyscale, opaque colours or colours with transparency.

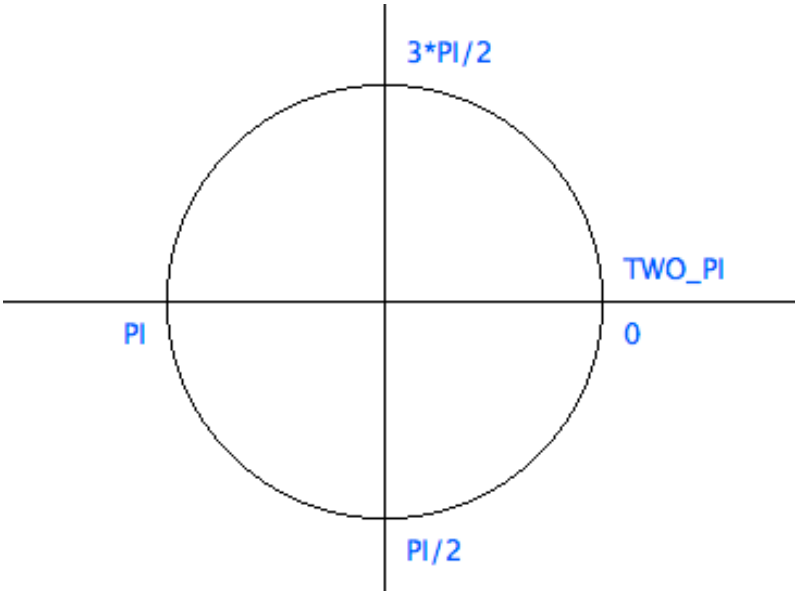
- greyscale: specified with a single value - 0 is black, 255 is white
- opaque colours: specified as 3 values for amount of red, green and blue light. This web resource may be useful to find the colour you want: http://cloford.com/resources/colours/500col.htm
- colours with alpha: same as opaque colours but also includes a 4th value for transparency. 0 is completely transparent, 255 is completely opaque

Workshop Example 2a: The Olympic Rings

```
size(500,300);
background(255,255,255);  //colors are represented as 3 values
                           //from 0 to 255 representing RGB
                           //Optionally you can also include an
                           //alpha channel as the 4th parameter

strokeWeight(5);           //use a thicker line
noFill();                  //these are rings, so hollow them out

stroke(0,0,255);           //blue ring
ellipse(190,100,50,50);

stroke(0,0,0);             //black ring
ellipse(250,100,50,50);

stroke(255,0,0);           //red ring
ellipse(310,100,50,50);

stroke(255,255,0);         //yellow ring
ellipse(220,120,50,50);

stroke(0,255,0);           //green ring
ellipse(280,120,50,50);
```

The above is not quite correct because the way that the sketch works is to draw things in order according to the code. Thus the yellow ring sits on top of the blue ring completely when what we want is to create a sketch where the rings appear to intertwine. One way to do this is to redraw a small arc over the pieces where necessary.

Rotation Angle



Workshop 2b: Olympic Rings Refined

```
size(500,300);
background(255,255,255);  //colors are represented as 3 values
                           //from 0 to 255 representing RGB
                           //Optionally you can also include
                           //alpha channel as the 4th parameter

strokeWeight(5);           //use a thicker line these are
noFill();                  //rings, so hollow them out

stroke(0,0,255);           //blue ring
ellipse(190,100,50,50);

stroke(255,225,0);         //yellow ring
ellipse(220,120,50,50);

stroke(0,0,0);             //black ring
ellipse(250,100,50,50);

stroke(0,255,0);           //green ring
ellipse(280,120,50,50);

stroke(255,0,0);           //red ring
ellipse(310,100,50,50);

stroke(0,0,255);           //blue over yellow
arc(190,100,50,50,-PI/8,PI/8);

stroke(255,225,0);         //yellow over black
arc(220,120,50,50,3*PI/2,3*PI/2+PI/4);

stroke(0,0,0);             //black over green
arc(250,100,50,50,-PI/8,PI/8);

stroke(0,255,0);           //green over red
arc(280,120,50,50,3*PI/2,3*PI/2+PI/4);
```

Animations

The examples thus far have been to draw using Processing's drawing functions. However, Processing not only allows you to draw an image programmatically but it also allows you to create animations. To do this, there are two main functions:

void setup() - a function that is called once when the sketch is first loaded
void draw() - a function that is called once every frame. To create an animation, you can use variables to control how something is drawn in each frame

Workshop 3a: Draw a triangle that flies from left to right (frame based)

```
//This sketch draws a triangle that floats across the  sketch from
//left to right.  When it reaches the right it restarts at the left

float positionX=0;  //x position of the triangle

//code in setup runs once only.
void setup(){
  size(500,300);   //this is something we only need to do once
}

//code in draw() function runs every frame
//to do animation, use variables to control the things you draw.
void draw(){
  background(100);      //redraw the backgorund each time to erase
                        //previous frame.  Use a single value
                        //between 0 and 255 for greyscale

  triangle(20+positionX,140,10+positionX,160,30+positionX,160);

  positionX=positionX+1;
  if(positionX>500){
    positionX=0;
  }
}
```

While frame based animations are simple, it is often better to account for systems with lower performance and perform time based animation instead. This can be accomplished by determining the displacement you wish an object to travel per sec and then determining its position based on the elapsed time.

displacement= velocity*time

Workshop 3b: Draw a triangle that flies from left to right (time based)

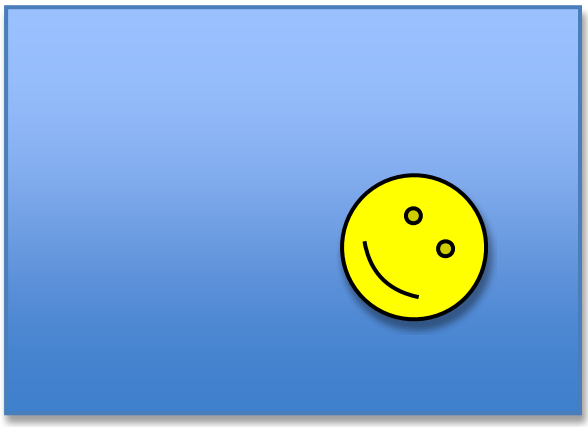
```
//This sketch draws a triangle that floats across the  sketch from
//left to right.  When it reaches the right it restarts at the left

float positionX=0;  //x position of the triangle
float oldTime;      //variable that holds a timestamp
float velocity=10;  //how fast the triangle travels

//code in setup runs once only.
void setup(){
  size(500,300);      //this is something we only need to do once
  oldTime = millis(); //initialize with time at start of sketch
                      //note millis() returns time in milliseconds
}

void draw(){
  background(100);      //redraw the backgorund each time to erase
                        //previous frame. Here, a single value
                        //between 0 and 255 for greyscale is used
                        //you can also use other color values

  float currTime=millis();  //current time in milliseconds

  //elapsedTime is the amount of time between the last time
  //draw() was called and the current time.  The difference is
  //divided by 1000 to get time in seconds
  float elapsedTime = (currTime - oldTime)/1000;

  //calculate the new position == oldposition+displacement
  //displacement = elapsedTime * velocity
  positionX=positionX+elapsedTime*velocity;

  triangle(20+positionX,140,10+positionX,160,30+positionX,160);

  positionX=positionX+1;
  if(positionX>500){
    positionX=0;
  }
  oldTime=currTime;
}
```

To spin an object, we need to alter the code a little bit. Instead of drawing the triangle at the position specified, the simplest way to rotate an object is to draw the triangle such that it is centered at the origin and use the translate() and rotate() functions to spin and position it.

Think of the translate() function as moving the sketch so that the origin is placed at the specified coordinate.

rotate() then rotates the object

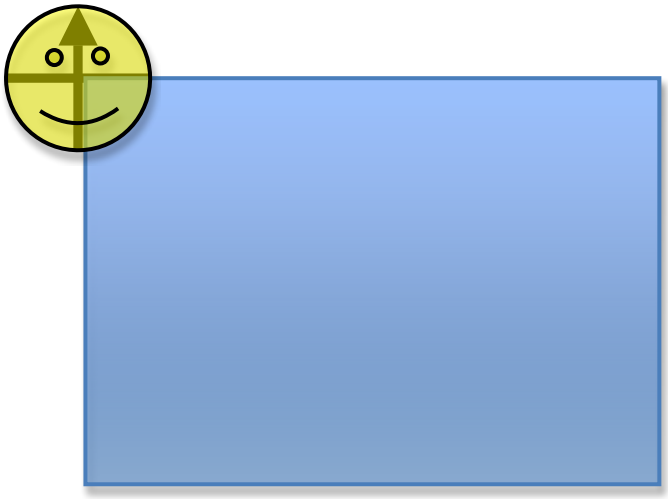
How this works:



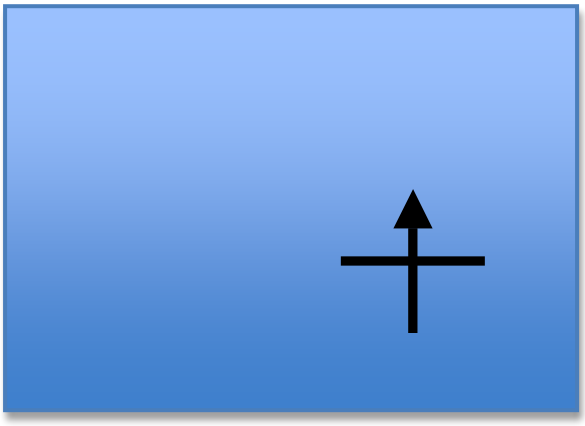
Suppose I wanted to draw a rotated smiley face like the picture on left.

Think of the process of doing this as follows:

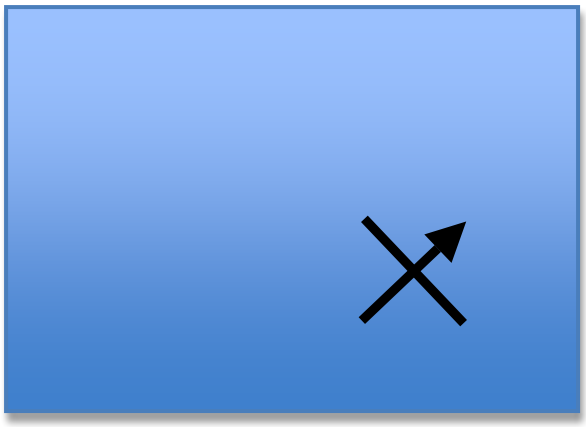
```
imageMode(CENTER);
translate(x,y);
rotate(PI/4);
image(smiley,0,0);
```



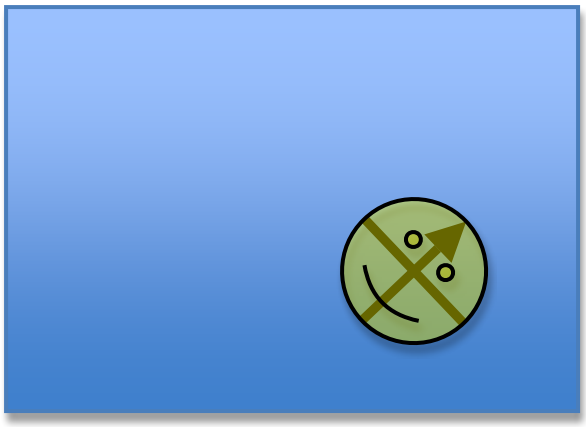
imageMode(CENTER);
image(smiley,0,0); would normally draw the lower right quandrant of the smiley in the top left corner.



translate moves the origin to the specified coordinates



rotate() rotates the orientation



now, drawing the image results in what we want

workshop 3C: Spinning moving triangle

```
//This sketch draws a triangle that floats across the  sketch from
//left to right.  When it reaches the right it restarts at the left

float positionX=0;  //x position of the triangle
float rotation=0;
float oldTime;
float velocity=10;
float angularSpeed=5;

//code in setup runs once only.
void setup(){
  size(500,300);   //this is something we only need to do once
  oldTime = millis();
}

//code in draw() function runs every frame
//to do animation, use variables to control the things you draw.
void draw(){
  background(100);      //redraw the backgorund each time to erase
                        //previous frame.  Use a single value
                        //between 0 and 255 for greyscale

  float currTime=millis();  //current time in milliseconds
  float elapsedTime = (currTime - oldTime)/1000;

  positionX = positionX + elapsedTime * velocity;
  rotation = rotation + elapsedTime * angularSpeed;

  translate(positionX,150);
  rotate(rotation);
  triangle(0,-10,-10,10,10,10);


  positionX=positionX+1;
  if(positionX>500){
    positionX=0;
  }
  oldTime=currTime;
}
```

Image based animations

While drawing from primitives is useful, some applications benefit from graphics with more details (games for example). Processing can be used to animate still images.

```
//Walk sequence animation
//Art work by "TheZakMan" at OpenGameArt.org
//http://opengameart.org/content/plataform-walk-jump-animation-side-view
//Licenced under CC-BY 3.0 license

/*@pjs preload="data/walk01.png, data/walk02.png, data/walk03.png,
                data/walk04.png, data/walk05.png, data/walk06.png,
                data/walk07.png, data/walk08.png, data/walk09.png,
                data/walk10.png, data/walk11.png, data/walk12.png,
                data/walk13.png, data/walk14.png, data/walk15.png,
                data/walk16.png, data/walk17.png, data/walk18.png,
                data/walk19.png, data/walk20.png, data/walk21.png,
                data/walk22.png, data/walk23.png, data/walk24.png,
                data/walk25.png, data/walk26.png, data/walk27.png,
                data/walk28.png, data/walk29.png, data/walk30.png,
                data/walk31.png"; */

PImage [] walk = new PImage[31];  //31 frames in the walk cycle
int sequenceid=0;

void setup(){
  size(500,500);
  for(int i=0;i<31;i++){
    walk[i]=loadImage("data/walk"+(i<9?"0":"") + (i+1) + ".png");
  }
  imageMode(CENTER);
}

void draw(){
  background(255);
  image(walk[sequenceid], width/2, height/2);
  sequenceid++;
  if(sequenceid==31)
    sequenceid=0;
}
```

This workshop barely scratches the surface of what is possible with the Processing Language. If you wish to learn more there are several books on Processing available and their website is a great resource with lots of examples of what people have done with it.

Workshop 6: mouse interaction

```
//this sketch draws a circle every time you click the mouse

/*A simple class that stores info needed to redraw all circles*/
class Circle{
  float x_, y_;
  float radius_;

  Circle(float x,float y, float r){
    x_=x;
    y_=y;
    radius_=r;
  }
  void draw(){
    ellipse(x_,y_,radius_,radius_);
  }
}

boolean spawnCircle=false;            //this variable is true if mouse
                                      //was clicked

Circle [] circleList=new Circle[50];  //Create an array of Circle
                                      //References

int numCircles=0;                     //number of Circles in list

void setup(){
  size(500,500);
}
void draw(){
  background(255);

  //if the mouse had been clicked, create a new circle
  if(spawnCircle==true){
    if(numCircles<50){
      circleList[numCircles]=new
                Circle(random(10,490),random(10,490),random(50,100));
      numCircles++;
    }
    spawnCircle=false;
  }

  //draw the circles
  for(int i=0;i<numCircles;i++){
    circleList[i].draw();
  }

}
/*
  Function gets called once, as soon as the
  mouse button is pressed.  It sets the spawnCircle variable
  to true
*/
void mousePressed(){
  spawnCircle=true;
}
```

How to put your sketch on a web page

Once you have a Processing Sketch, adding it to a web page is a fairly simple process. The processing sketch is rendered using <canvas>. Follow these steps to place it on a page:

1) download processing.js from the Processing.js site (http://processingjs.org). You can use either the development or production version. The difference is that the development version allows you to read the processing.js code that does the actual translation while the minified version has been obfuscated. For the most part, it is better to use the minified version unless you are trying to contribute to the Processing.js project.

2) put your sketch, html file and processing.js file into one directory. Suppose that they are named as follows:

- processing.js (the file you download from processingjs.org renamed)
- mysketch.pde (the processing sketch)
- index.html (the html file)

3) create your index.html file as follows

```
<html>
  <head>
    <script src="processing.js></script>
  </head>
  <body>
    <canvas data-processing-source="mysketch.pde"></canvas>
  </body>
</html>
```

If you have more than one file, you can list them all insides the " ".

For example

```
<html>
  <head>
    <script src="processing.js></script>
  </head>
  <body>
    <canvas data-processing-source="file1.pde files2.pde"></canvas>
  </body>
</html>
```

Useful Resources

Resource Name URL/Location	Description
Processing.js website http://processingjs.org	This is the place to get the latest version of Processing.js. It has links to many resources on Processing.js including many examples.
Processing website http://processing.org	This is Processing's home page. A good online resources with many examples of how to create processing sketches starting from the very beginning. This is where you can get the Processing IDE (NOTE that the 2.0 alpha release actually lets you publish a sketch as processing.js directly from the IDE. pre 2.0, you would have to create the html separately by hand)
Studio Sketchpad http://sketchpad.cc	An online tool that allows multiple people to work on a sketch together and immediately share the results. It is based on etherpad. If you want to use it for teaching, you can even ask them to setup a private studio for you
Follow Processing.js on twitter: @Processingjs	Processing.js is an open community and twitter is the best way to follow its happenings. Since much of the development was done at Seneca, there is a large community in Toronto and with regular in-person meetups.
Bug Tracker https://processing-js.lighthouseapp.com	If you find a bug with Processing.js please report it to their bug tracker here.
IRC Channel irc://moznet , channel #processingjs	The IRC channel where Processing.js devs hang out. The community is quite active and you will often be able to find help here.
Mike "Pomax" Kamermans's Processing.js Super Mario Tutorial http://processingjs.nihongoresources.com/test/PjsGameEngine/docs/tutorial/mario.html	This tutorial shows how to build a Super Mario level using a game engine made with Processing.js
Khan Academy Computer Science http://www.khanacademy.org/cs/tutorials/all-tutorials	This site includes video tutorials and a programming environment right in the browser.
Popcorn.js http://popcornjs.org/	Popcorn.js is a JavaScript library that allows time based media to control the contents of a web page.