

Media Processing in Processing

Media Processing in Processing

Collection edited by: Davide Rocchesso

Content authors: Davide Rocchesso, Pietro Polotti, and Anders Gjendemsjø

Online: <<http://cnx.org/content/col10268/1.14>>

This selection and arrangement of content as a collection is copyrighted by Davide Rocchesso.

It is licensed under the Creative Commons Attribution License: <http://creativecommons.org/licenses/by/3.0/>

Collection structure revised: 2010/11/10

For copyright and attribution information for the modules contained in this collection, see the "[Attributions](#)" section at the end of the collection.

Media Processing in Processing

Table of Contents

- [Chapter 1. Programming in Processing](#)
 - [1.1.](#)
 - [Introduction](#)
 - [Data Types](#)
 - [Variables](#)
 - [Programming Structures](#)
 - [Conditional Instructions](#)
 - [Iterations](#)
 - [Functions](#)
 - [Objects and Classes](#)
- [Chapter 2. Media Representation in Processing](#)
 - [2.1.](#)
 - [Visual Elements](#)
 - [Coordinates](#)
 - [Images](#)
 - [Colors](#)
 - [The RGB model](#)
 - [HSB Model](#)
 - [Alpha channel](#)
 - [Tinting an image](#)
 - [Translations, Rotations, and Scale Transformations](#)
 - [Typographic Elements](#)
 - [Auditory Elements](#)
 - [Sounds](#)
 - [Timbre](#)
- [Chapter 3. Graphic Composition in Processing](#)
 - [3.1.](#)
 - [Graphic primitives](#)
 - [0D](#)
 - [1D](#)
 - [2D](#)
 - [3D](#)
 - [The stack of transformations](#)
 - [Lighting](#)
 - [Projections](#)
 - [Perspective projections](#)
 - [Parallel views](#)

- [Orthographic projection](#)
 - [Oblique projection](#)
 - [Casting shadows](#)
 - [Pills of OpenGL](#)
 - [References](#)
- [Chapter 4. Signal Processing in Processing: Sampling and Quantization](#)
 - [4.1.](#)
 - [Sampling](#)
 - [1-D: Sounds](#)
 - [2-D: Images](#)
 - [Quantization](#)
 - [References](#)
- [Chapter 5. Signal Processing in Processing: Convolution and Filtering](#)
 - [5.1.](#)
 - [Systems](#)
 - [Impulse response and convolution](#)
 - [Properties](#)
 - [Frequency response and filtering](#)
 - [Causality](#)
 - [2D Filtering](#)
- [Chapter 6. Convolution - Discrete time](#)
 - [6.1.](#)
 - [Introduction](#)
 - [Derivation of the convolution sum](#)
 - [Graphical illustration of convolution properties](#)
 - [Convolution Sum](#)
 - [Convolution Through Time \(A Graphical Approach\)](#)
- [Chapter 7. Signal Processing in Processing: Elementary Filters](#)
 - [7.1.](#)
 - [FIR filters](#)
 - [Averaging filter](#)
 - [Symmetric second-order FIR filter](#)
 - [High-pass filters](#)
 - [FIR filters in 2D](#)
 - [Non-linear filtering: median filter](#)
 - [IIR Filters](#)
 - [Resonant filter](#)
- [Chapter 8. Textures in Processing](#)
 - [8.1.](#)
 - [Color Interpolation](#)
 - [Bilinear Interpolation](#)
 - [Texture](#)
 - [Texture mapping](#)

- [Texture Generation](#)
- [Chapter 9. Signal Processing in Processing: Miscellanea](#)
 - [9.1.](#)
 - [Economic Color Representations](#)
 - [Palette](#)
 - [Dithering](#)
 - [Floyd-Steinberg's Dithering](#)
 - [Economic Sound Representations](#)
 - [Histogram-Based Processing.](#)
 - [Translation and Expansion of an Histogram](#)
 - [Non Linear Scaling](#)
 - [Equalization of an Histogram](#)
 - [Segmentation and Contour Extraction](#)
 - [Contours](#)
 - [Regions](#)
 - [Audio Dynamic Compression](#)
- [Index](#)

Chapter 1. Programming in Processing

Introduction

This introduction is based on [Daniel Shiffman's tutorial](#).

is a language and development environment oriented toward . In the course **Media Processing in Processing (MPP)**, processing is one of the main instruments used to introduce some fundamentals in sound and image processing. Processing is an extension of Java that supports many Java structures with a simplified syntax.

Processing can be used in three

Basic : Sequence of commands for simple drawing by graphic primitives. --

Table 1.1.

applet without nose	<pre>size(256,256); background(0); stroke(255); ellipseMode(CORNER); ellipse(72,100,110,130); triangle(88,100,168,100,128,50); stroke(140); strokeWeight(4); line(96,150,112,150); line(150,150,166,150); line(120,200,136,200);</pre>
---	--

Intermediate : Procedural programming --

Table 1.2.

	<pre>void setup() { size(256,256);</pre>
--	--

applet
with
nose

```
        background(0);
    }

    void draw() {
        stroke(255);
        strokeWeight(1);
        ellipseMode(CORNER);
        ellipse(72,100,110,130);
        triangle(88,100,168,100,128,50);
        stroke(140);
        beginShape(TRIANGLES);
        vertex(114, 180);
        vertex(mouseX, mouseY);
        vertex(140, 180);
        endShape();
        strokeWeight(4);
        line(96,150,112,150);
        line(150,150,166,150);
        line(120,200,136,200);
    }
```

Complex : Object-Oriented Programming (Java) --

Table 1.3.

applet

```
Puppet pinocchio;

void setup() {
    size(256,256);
    background(0);
    color tempcolor = color(255,0,0);
    pinocchio = new Puppet(tempcolor);
}

void draw() {
    background(0);
    pinocchio.draw();
}

class Puppet {
    color colore;
    Puppet(color c_) {
```

with
colored
nose

```
    colore = c_;  
  }  
void draw () {  
  stroke(255);  
  strokeWeight(1);  
  ellipseMode(CORNER);  
  ellipse(72,100,110,130);  
  stroke(colore);  
  beginShape(TRIANGLES);  
  vertex(114, 180);  
  vertex(mouseX, mouseY);  
  vertex(140, 180);  
  endShape();  
  strokeWeight(4);  
  line(96,150,112,150);  
  line(150,150,166,150);  
}  
}
```

The Processing programs can be converted into Java applets. In order to do that, one just goes to the **File** menu and chooses **Export**. As a result, five files will be created and put in an `applet` folder:

- **index.html**: html code to visualize the applet
- **filename.jar**: the compiled applet, including all data (images, sounds, etc.)
- **filename.pde**: the Processing source code
- **filename.java**: the Java code embedding the Processing source code
- **loading.gif**: an image to be displayed while the applet is being loaded.

Moreover, by means of **Export Application** it is possible to generate an executable application for Linux, MacOS, or Windows platforms.

Data Types

Variables

A variable is a pointer to a memory location, and it can refer either to primitive values (`int`,

float, ecc.) or to objects and arrays (tables of primitive-type elements).

The operation of assignment `b = a` produces

- The copy of the content of `a` into `b`, if the variables refer to primitive types.
- The creation of a new reference (pointer) to the same object or array, if the variables refer to objects or arrays.

To have a clear understanding of computer science terms such as those that follow, we recommend looking at [Wikipedia](#)

Definition: scope

within a program, it is a region where a variable can be accessed and its value modified

Definition: global scope

defined outside the methods `setup()` and `draw()`, the variable is visible and usable anywhere in the program

Definition: local scope

defined within a code block or a function, the variable takes values that are local to the block or function, and any values taken by a global variable having the same name are ignored.

Example 1.1. Array declaration and allocation

```
int[] arrayDiInteri = new int[10];
```

Programming Structures

Conditional Instructions

- if:

```
if (i == NMAX) {
    println("finished");
}
else {
    i++;
}
```

Iterations

- while:

```
int i = 0; //integer counter
while (i < 10) { //write numbers between 0 and 9
    println("i = "+ i);
    i++;
}
```

- for:

```
for (int i = 0; i < 10; i++) { //write numbers between 0 and 9
    println("i = "+ i);
}
```

Example 1.2. Initializing a table of random numbers

```
int MAX = 10;
float[] tabella = new float[MAX];
for (int i = 0; i < MAX; i++)
    tabella[i] = random(1); //random numbers between 0 and 1
println(tabella.length + " elements:");
println(tabella);
```

Functions

Functions allow a modular approach to programming. In Processing, in the **intermediate** programming mode, we can define functions other than `setup()` and `draw()`, usable from within `setup()` and `draw()`.

Example 1.3. Example of function

```
int raddoppia(int i) {
```

```
    return 2*i;
}
```

A function is characterized by the entities (with reference to the [example](#)) :

- return type (`int`)
- name (`raddoppia`)
- parameters (`i`)
- body (`return 2*i`)

Objects and Classes

A class is defined by a set of data and functions. An object is an instance of a class. Vice versa, a class is the abstract description of a set of objects.

For an introduction to the concepts of object and class see [Objects and Classes](#).

Example 1.4. Example of class

```
Dot myDot;
void setup() {
    size(300,20);
    colorMode( RGB, 255, 255, 255, 100 );
    color tempcolor = color(255, 0, 0);
    myDot = new Dot(tempcolor, 0);
}

void draw() {
    background(0);
    myDot.draw(10);
}

class Dot
{
```

```

color colore;
int posizione;

/**CONSTRUCTOR***/
Dot(color c_, int xp) {
    colore = c_;
    posizione = xp;
}

void draw (int ypos)    {
    rectMode(CENTER);
    fill(colore);
    rect(posizione, ypos, 20, 10);
}
}

```

A class is characterized by the following entities (with reference to the [example](#)) :

- name (Dot)
- data (colore, posizione)
- constructor (Dot())
- functions (or methods, draw())

An object (instance of a class) is declared in the same way as we declare a variable, but we have to allocate a space for it (as we did for the arrays) by means of its constructor (with reference to the [example](#)).

- Declaration: (Dot myDot;)
- Allocation: (myDot = new Dot(tempcolor, 0))
- Use: (myDot.draw(10);)

For a quick introduction to the Java syntax see [Java Syntax Primer](#)

Exercise 1.

With the following `draw()` method we want to paint the window background with a gray whose intensity depends on the horizontal position of the mouse pointer.

```
void draw() {  
    background( (mouseX/100) *255);  
}
```

However, the code does not do what it is expected to do. Why?

The variable `mouseX` is of `int` type, and the division it is subject to is of the integer type. It is necessary to perform a from `int` to `float` by means of the instruction `(float)mouseX`.

Exercise 2.

What does the following code fragment print out?

```
int[] a = new int[10];  
a[7] = 7;  
int[] b = a;  
println(b[7]);  
b[7] = 8;  
println(a[7]);  
int c = 7;  
int d = c;  
println(d);  
d = 8;  
println(c);
```

```
7  
8  
7  
7
```

Exercise 3.

The following sketch generates a set of 100 moving circles and draws all chords linking the intersection points of all couples of intersecting circles.

```
/*
```

Structure 3

A surface filled with one hundred medium to small sized circles. Each circle has a different size and direction, but moves at the Display:

- A. The instantaneous intersections of the circles
- B. The aggregate intersections of the circles

Implemented by Casey Reas <<http://groupc.net>>

8 March 2004

Processing v.68 <<http://processing.org>>

modified by Pietro Polotti

28 March, 2006

Processing v.107 <<http://processing.org>>

```
*/
```

```
int numCircle = 100;
Circle[] circles = new Circle[numCircle];

void setup()
{
  size(800, 600);
  frameRate(50);
  for(int i=0; i<numCircle; i++) {
    circles[i] = new Circle(random(width),
      (float)height/(float)numCircle * i,
      int(random(2, 6))*10, random(-0.25, 0.25),
      random(-0.25, 0.25), i);
  }
  ellipseMode(CENTER_RADIUS);
  background(255);
}

void draw()
{
  background(255);
  stroke(0);
```

```

for(int i=0; i<numCircle; i++) {
    circles[i].update();
}
for(int i=0; i<numCircle; i++) {
    circles[i].move();
}
for(int i=0; i<numCircle; i++) {
    circles[i].makepoint();
}
noFill();
}

```

```

class Circle
{
    float x, y, r, r2, sp, ysp;
    int id;

    Circle( float px, float py, float pr, float psp, float pyp, int
        x = px;
        y = py;
        r = pr;
        r2 = r*r;
        id = pid;
        sp = psp;
        ysp = pyp;
    }

    void update() {
        for(int i=0; i<numCircle; i++) {
            if(i != id) {
                intersect( this, circles[i] );
            }
        }
    }

    void makepoint() {
        stroke(0);
        point(x, y);
    }

    void move() {
        x += sp;
    }
}

```

```

y += ysp;
if(sp > 0) {
    if(x > width+r) {
        x = -r;
    }
} else {
    if(x < -r) {
        x = width+r;
    }
}
if(ysp > 0) {
    if(y > height+r) {
        y = -r;
    }
} else {
    if(y < -r) {
        y = height+r;
    }
}
}
}

```

```

void intersect( Circle cA, Circle cB )

```

```

{
    float dx = cA.x - cB.x;
    float dy = cA.y - cB.y;
    float d2 = dx*dx + dy*dy;
    float d = sqrt( d2 );

    if ( d>cA.r+cB.r || d<abs(cA.r-cB.r) ) {
        return; // no solution
    }

    // calculate the two intersections between the two circles cA and
    // whose coordinates are (paX, paY) and (pbX, pbY), respectively

    stroke(255-dist(paX, paY, pbX, pbY)*4);
    line(paX, paY, pbX, pbY);
}

```


1. Complete the missing part that is expected to compute the intersections of the circles, in such a way to draw the chords linking the intersection points. It is possible to use the computation of intersection coordinates in a ad-hoc reference system (), then converting the result into the Processing window coordinate system.
2. Make the chords time-variable by giving different speeds to different circles.

```
/*
```

```
Structure 3
```

```
A surface filled with one hundred medium to small sized circles.  
Each circle has a different size and direction, but moves at the  
Display:
```

- A. The instantaneous intersections of the circles
- B. The aggregate intersections of the circles

```
Implemented by Casey Reas <http://groupc.net>
```

```
8 March 2004
```

```
Processing v.68 <http://processing.org>
```

```
modified by Pietro Polotti
```

```
28 March, 2006
```

```
Processing v.107 <http://processing.org>
```

```
*/
```

```
int numCircle = 100;
```

```
Circle[] circles = new Circle[numCircle];
```

```
void setup()
```

```
{
```

```
  size(800, 600);
```

```
  frameRate(50);
```

```
  for(int i=0; i<numCircle; i++) {
```

```
    circles[i] = new Circle(random(width),
```

```
      (float)height/(float)numCircle * i,
```

```
      int(random(2, 6))*10, random(-0.25, 0.25),
```

```
      random(-0.25, 0.25), i);
```

```
  }
```

```
  ellipseMode(CENTER_RADIUS);
```

Thank You for previewing this eBook

You can read the full version of this eBook in different formats:

- HTML (Free /Available to everyone)
- PDF / TXT (Available to V.I.P. members. Free Standard members can access up to 5 PDF/TXT eBooks per month each month)
- Epub & Mobipocket (Exclusive to V.I.P. members)

To download this full book, simply select the format you desire below

