Processing with Spde

What is Processing?

Processing is a programming language, development environment, and online community that since 2001 has promoted software literacy within the visual arts.

— processing.org

The Programming Language

P5

Java 1.4 with some syntax sugar

Not evoloving

The Development Environment

The PDE

Very simplified IDE

1-click run, applet export

No syntax completion

Not extensible

And yet…

Processing is used with great success in many fields.

Education and Design

NYU's ITP program shows student projects at Big Screens

Urban Planning

Political Analysis

The corpus of all the State of the Union addresses from 1790 to 2006, visualized

Art

Experimental organic interface allows people to control a computer while playing in the mud

Through an API built with Processing, artists and designers develop mud-controlled games, physics simulations, and expressive tools.

And Industry

Flocking behavior used in television spots for FOX Japan

Where does Scala fit in?

Its usual benefits over Java

Type inference

Greater expressivity

Access to functional programming

... all *without* sacrificing compatibility with the wealth of Java and JNI libraries built for Processing

But the PDE isn't flexible enough to support other languages.

(I tried that.)

Fortunately there is Simple Build Tool.

The sbde-sbt plugin

An sbt plugin may

Preprocess sources

Obtain dependencies

Run programs

Export applets

Or anything else, really

Plugins are an ecosystem

Projects can use multiple plugins

Projects and plugins can spin off more plugins

No user-interface code, no cry

People may choose their own IDE poison

Learning Scala with Spde

"Spde" is like "Processing"—it's the project, libraries, and environment. Loosely speaking.

Plotting functions

In Processing you work with pixels directly, but these can also be abstracted away.

```
size(500, 500)
def squared = points(-50, 50) { x => x * x }
def draw {
  lineplot(squared)
}
```

```
size(500, 500)
def slope = points(-250, 250) {
  x => ((mouseY * 2.0 / height) * x)
}
def draw {
  background(128)
  lineplot(slope)
}
```

```
size(500, 500)
def sliding_sine = points(0, 10) {
  x => sin( x * mouseX * 5 / width )
} map {
  case (x, y) => (x, y * mouseY * 250 / height)
}
def draw {
  background(128)
  lineplot(sliding_sine)
}
```

```
val d= 200
size(400, 400)
background(255)
val a=List(List(d/2,0,255,0,0),List(d,d,0,255,0),List(0,d,0
var p=List(d,0,255,0,0)
def draw() {
  for(i<-1 to 10) {
    p=p zip a.random map{case (x,y)=>x/2+y}
    stroke(p(2),p(3),p(4))
    point(p(0), p(1))
  }
}
```

```
size(500, 200)
frameRate(20)
val items = { 0 to width }.view.map { (_, random(255).toInt

def draw {
  for ((x, color) <- items) {
    stroke(color)
    line (x, 0, x, height)
  }
}
```

```
size(465, 190)
frameRate(25)

def random_seq = (0 until width) map { (_, random(height)) }
var h: Seq[(Int, Float)] = random_seq
override def mouseClicked { h = random_seq }

def draw() {
  background(255)
  h foreach { case (x, h) => line(x, 0, x, h) }
  h = (h :\ List[(Int, Float)]()) {
    case ((x1, h1), Nil) => (x1, h1) :: Nil
    case ((x1, h1), (x2, h2) :: t) =>
      (x1, (h1 + h2) /2) :: (x2, h2) :: t
  }
}
```

```
import scala.util.continuations._
size(500, 200)
frameRate(15)
var cur: Unit => Unit = { (u: Unit) =>
  reset {
    var x = 0
    while(true) {
      x = (x + 1) % width
      background(255)
      color(0)
      line(width - x, 0, x, height)
      shift { k: (Unit => Unit) => cur = k }
    }
  }
}
def draw() { cur() }
```

And you can translate neat Processing sketches

```
size(640, 360, P3D)
val rSize = width / 6 // rectangle size
noStroke()
fill(204, 204)
var a = 0.0
def draw() {
  background(0)
  a = (a + 0.005) % TWO_PI
  translate(width/2, height/2)
  rotateX(a)
  rotateY(a * 2.0)
  rect(-rSize, -rSize, rSize*2, rSize*2)
  rotateX(a * 1.001)
  rotateY(a * 2.002)
```

```
var index=0
def index_++ = {
  index += 1
  index - 1
}
```

```
for (int i=0;i<cuantos;i++){
  lista[i].dibujar();
}
```

```
lista foreach { _.dibujar() }
```

And one more thing...

Android

Spde needs YOU

I've got the tools under control, but...

Spde could use more

Exemplary sketches in `spde-examples`

Full-fledged, world-changing projects

Serious attempts at writing a core library

THANK YOU

http://technically.us/spde