Processing, el lenguaje de programación para artistas visuales

Universidad de Montemorelos

Gabriel Trisca

Abstract

Los artistas contemporáneos utilizan toda clase de recursos visuales para crear arte, muchas veces recurriendo a gráficos y contenido generado por computadora. Este contenido es generalmente preparado por el artista y comisionado a un desarrollador, quien lo adapta y lo convierte en la obra final ideada por el artista. Processing surge para disminuir la dificultar y el tiempo requerido para generar contenido visual, dándole más poder al artista visual para realizar sus concepciones y explorar el mundo de los gráficos por computadora de la mano de su creatividad.

Introducción

Desde el principio, la expresión artística está ligada al avance tecnológico de la época:
En el lejano oriente, durante el reinado de la dinastía Qin (en la China moderna), al morir un emperador se realizó una copia de un ejército completo formado por soldados de barro cocido. Luego de ser colocadas las más de 8000 figuras humanas en filas, 130 carruajes y 670 caballos, todos de tamaño natural, fueron enterradas. Más de 2000 años después los restos de esta enorme creación fueron encontrados y bautizados como el "Ejército de Terracota"[1]. La arcilla utilizada, era una de las materias primas más comunes, y su manipulación era bien conocida por los artistas, por lo que la creación de cada pieza utilizaba procesos simples para los artistas.
Griegos, Persas y Egipcios utilizaron la piedra para hacer sus representaciones artísticas más significativas ya que consideraban que éste era el material que permitía que una obra de arte perdurara por la eternidad[2].
La correlación entre la tecnología y el arte son innegables, ejemplo de ello podría ser el famoso Coloso de Rodas, una estatua de 30 metros de alto construida en la ciudad griega del mismo nombre, que colapsó 56 años después de ser construida[3]. La conocida Estatua de la Libertad mide 47 metros de alto y lleva más de 120 años en su locación actual, cosa que podría atribuirse a técnicas más modernas de construcción[4].
Siguiendo esta línea de pensamiento, no es de extrañarse que los artistas contemporáneos utilicen los gráficos generados por computadora para sus expresiones artísticas ya que esa es la tecnología de la cual disfrutamos en la actualidad.
De la misma forma como en la antigüedad se necesitaban expertos en fundición, ingenieros, escultores y mucha mano de obra[5], hoy en día se requieren expertos en distintas áreas para aprovechar los medios de comunicación y representación actuales.
Debido a que gran parte del contenido visual vanguardista es interactivo, algorítmico o

1 http://whc.unesco.org/pg.cfm?cid=31&id_site=441 Accedido el 7/05/2010
2 http://www.visual-arts-cork.com/sculpture/stone.htm Accedido el 7/05/2010
3 http://en.wikipedia.org/wiki/Colossus_of_Rhodes Accedido el 7/05/2010
4 http://www.nps.gov/stli/faqs.htm Accedido el 7/05/2010
5 http://www.greecelogue.com/the-colossus-of-rhodes.html Accedido el 7/05/2010

generativo, se requiere que dicho contenido sea programado, es decir, una computadora debe ser instruida de antemano para crear el contenido visual[6] [7], no es extraño encontrar equipos de que incluyen artistas visuales y desarrolladores de software.

6 http://www.vam.ac.uk/collections/prints_books/features/computer-art/what/index.html Accedido el 12/05/2010

7 Donald Kuspit "Del Arte Analogico al Arte Digital" in Arte Digital Y Videoarte, Kuspit, D. ed., Consorcio del Circulo de Bellas Artes, Madrid

¿Por qué Processing?

En la página oficial de Processing podemos leer la siguiente definición: “Processing es un lenguage libre[8] para personas que desean programar imágenes, animaciones e interacción”[9]
La pregunta que surge es: ¿Como se pueden programar imágenes? O ¿Las animaciones pueden ser programadas? La respuestas es sí.
Las imágenes se forman en la pantalla de un ordenador como resultado de la suma de tres colores: Rojo, Verde y Azul, y esta forma de crear color es conocida como RGB por sus siglas en inglés[10][11]. Estos tres colores se mezclan para formar un pixel, que es la unidad mínima de información que puede ser representada por un monitor. Ahora, esto solo nos da información sobre el color, sin embargo necesitamos también la posición. Los pixeles se ordenan en filas y columnas numeradas comenzando por la esquina superior izquierda del monitor, y continuan de izquierda a derecha, de arriba hacia abajo hasta llegar a la esquina inferior derecha, esto quiere decir que el primer pixel se encuentra en la posición (0,0) y que el último pixel estaría en la posición (1024,768) en un monitor de resolución de 1024 pixeles horizontales y 768 verticales.
Con estos dos conceptos, color y posición, programar una imagen suena más factible, ya que podemos asumir que cualquier imagen puede ser “descompuesta” en pixeles y reconstruida en un monitor de computadora.
Processing nos facilita este proceso al permitirnos definir miles de pixeles simultáneamente y no tener que hacerlos uno por uno. Además, nos facilita la creación de formas geométricas y transparencia, que es considerada parte del color, aunque en realidad la transparencia es una propiedad que el ordenador simula[12].
La pregunta de este capítulo queda contestada al agregar que Processing no solamente nos permite manipular el color de cada pixel de forma sencilla si no que además no se requiere conocimientos avanzados de programación para poder manipular y crear contenido visual en la computadora.

8 The Free Software Definition, http://www.gnu.org/philosophy/free-sw.html Accedido el 20/05/2010
9 Processing http://processing.org/ Accedido el 20/05/2010
10 RGB http://en.wikipedia.org/wiki/RGB Accedido el 20/05/2010
11 Additive Color http://en.wikipedia.org/wiki/Additive_color Accedido 20/05/2010
12 Alpha Compositing http://en.wikipedia.org/wiki/Alpha_channel Accedido el 20/05/2010

Estructura de un programa en Processing

Un programa en Processing se compone de dos secciones principales: *setup* y *draw*. En la sección de *setup* como su nombre lo indica, se realiza la inicialización del programa. Esta incluye las dimensiones de la ventana donde se generará el contenido, el modo de gráficos que se utilizará, así como propiedades de *antialiasing*[13].

Por otro lado, en la sección *draw* se crean las imágenes. Este bloque de código se ejecuta una y otra vez, por lo tanto, es posible mediante pequeños cambios en el contenido gráfico que se presenta, dar la ilusión que se está observando una animación.

El *Diagrama 1* muestra el flujo de un programa hecho en Processing.

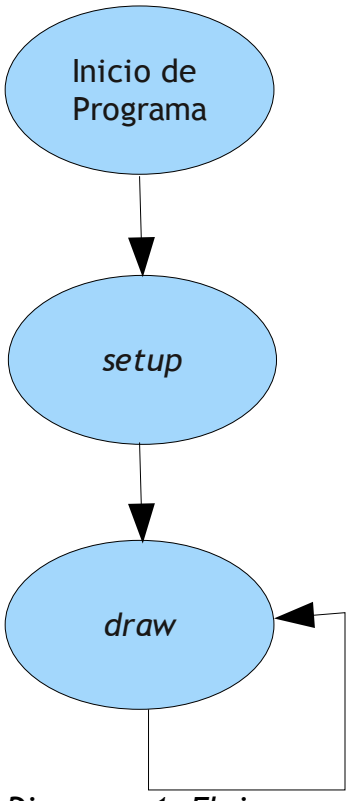


Diagrama 1: Flujo

13 Antialias http://en.wikipedia.org/wiki/Antialias Accedido el 20/05/2010

Es importante notar que aunque dentro del bloque *setup* se puede colocar código que imprima información a la pantalla, este sólo se ejecutará una vez.

La ejecución termina automáticamente cuando el programa se cierra, cuando la función "*exit()*" es llamada, o cuando el proceso java de Processing es finalizado.

Esta estructura está diseñada para que el programa mantenga una estructura interna y sea más fácil la detección de errores, así mismo permite que el programa sea más modular, sin embargo, Processing también permite que el programa no siga este patrón y que todo el código sea escrito de forma continua; ésto puede resultar muy útil a la hora de hacer pruebas, o de crear contenido estático. No se puede crear contenido animado utilizando este método, ya que cuando no existe un método *draw,* en cuanto el código termina de ejecutarse, no se pueden hacer cambios adicionales a la imagen generada.

Instalación y Primeros Pasos

Para obtener una copia de Processing, todo lo que debemos hacer es dirigirnos a http://processing.org/download/ donde se puede seleccionar la versión y sistema operativo del sistema donde se pretende usar Processing.

Una vez instalado, al ejecutar el programa veremos la siguiente pantalla:

Ilustración 1: Primera pantalla

En la parte superior izquierda, se encuentra un botón circular con un triángulo: este es el

botón que se presiona para ejecutar nuestro código en Processing. Una forma alternativa, es presionando la tecla “Ctrl” y la letra “R” simultáneamente.

El primer programa y siguiendo la tradición del “Hola Mundo”, podría ser escrito de la siguiente forma:

```
line(25, 10, 25, 90);
line(25, 55, 70, 55);
line(70, 10, 70, 90);
```

Para ejecutarlo, copiamos el código en el área blanca de Processing, y presionamos el botón de “Play” o las teclas Ctrl+R y veremos la siguiente ventana:

Ilustración 2: "H" de "Hola mundo"

Este código está creando tres líneas: dos verticales y una horizontal, y como podemos ver, la sintaxis está explicada en el *Diagrama 2*

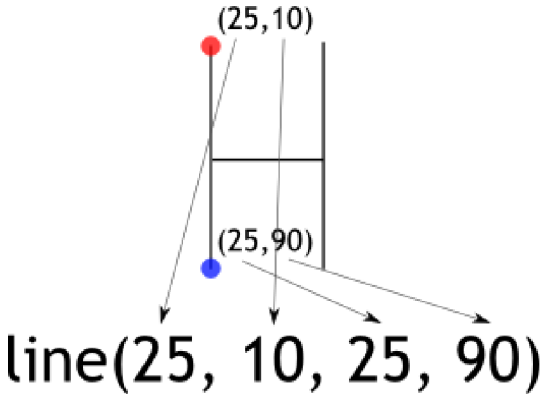


Diagrama 2: Explicación del comando "line"

Como podemos ver, la función *line* dibuja una línea desde la posición x1, y1 a la posición x2, y2, así como se observa en el *Diagrama 2*

Animaciones en Processing

Las animaciones se realizan presentando distintas imágenes a medida que la sección *draw* se ejecuta. Para probar ésto, se puede ejecutar el siguiente código:

```
void setup(){
  size(400,100);
  smooth();
}
int x = 0;
void draw(){
  x++;
  ellipse(50+x,50,90,90);
  if(x>350)exit();
}
```

Cuyo resultado es:

En este caso usamos los bloques *setup* y *draw,* en el bloque setup configuramos el tamaño de nuestro área de trabajo (de dimensiones 400x100) y especificamos que las figuras geométricas deben ser “alisadas” (antialiasing).

Más adelante definimos una variable de tipo entero, **x**, y le asignamos el valor 0.

Ahora en la parte importante del programa, lo que hacemos es incrementar en uno el valor de equis y dibujar un elipse con centro en la posición “50+**x**”, que como podemos imaginarnos, al cambiar el valor de **x**, también cambiará su centro. Los últimos dos valores de la función *ellipse* determinan el ancho y alto del elipse.

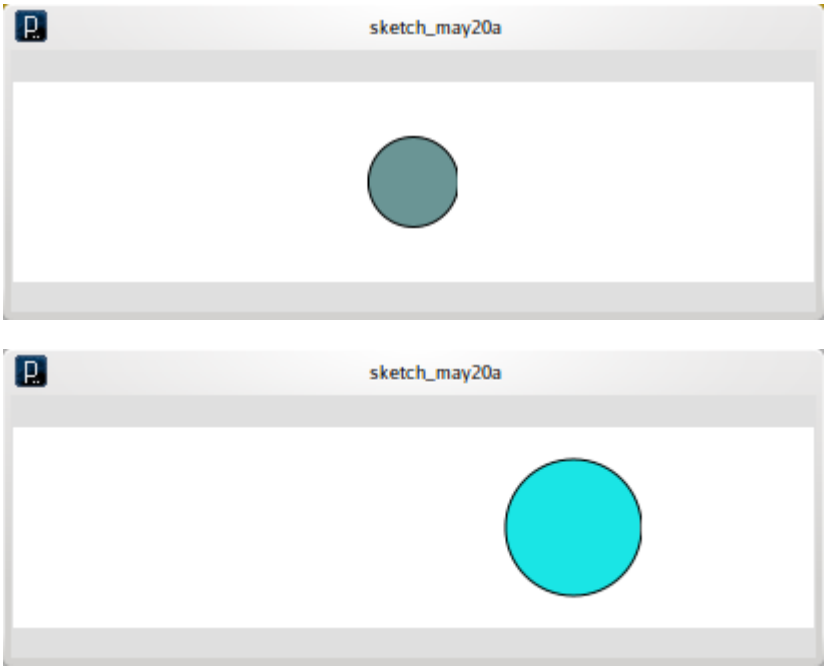
En la penúltima línea colocamos una estructura *if* para que si el valor de **x** es mayor que 350, el programa termine la ejecución.

Sin embargo, aunque se puede observar la animación, podemos ver como ésta se va generando sobre la anterior, dejando una “estela” de las imágenes anteriores. Para solucionar este problema, debemos eliminar todo contenido de la imagen antes de dibujar nuevamente. Para lograr esto, se utiliza la función *background()* y basta con colocarla en cualquier línea antes de la línea que dibuja el elipse. Una versión funcional de este ejemplo, agregando funciones trigonométricas que cambien el tamaño de el elipse, y

cambiando el color de la forma por medio de la función *fill()* y además sustituido el *exit()* por una asignación de **x** a 0, puede verse a continuación:

```
    void setup(){
    size(400,100);
    smooth();
  }
  int x = 0;
  void draw(){
    background(#ffffff);
    fill(Math.max(255-
x,0),Math.min(x,255),Math.min(x,255));
    x++;
    ellipse(x,50,(int)
(90*Math.sin(x/57.3)),(int)
(90*Math.sin(x/57.3)));
    if(x>400)x=0;
  }
```

El resultado es el siguiente, capturas de pantalla tomadas a los 2 segundos y a los 3 segundos de haber comenzado el programa:



Ejemplo completo

No sería factible explicar todas las funciones y técnicas existentes para desarrollar contenido visual en Processing, sin embargo, en Internet existen cientos de manuales, ejemplos y artículos sobre este tema. La mejor manera de empezar, es accediendo a http://processing.org/learning/ y dándole un vistazo a los diferentes temas que se tocan en esta página.

Como despedida, a continuación está el código de un programa que genera una animación hipnótica de colores.

```
import processing.opengl.*;

void setup(){
  size(400,400,OPENGL);
  fill(255,0,0);
  smooth();
}

int a,G,B=0;
int dirG=1,dirB=1;
  float sc;
void draw(){
  background(45,45,45);
  a+=4;
  if(a>360)a=0;
  for(int i=0;i<529;i++){
    pintar(i,a,-25+(i%23)*20,-25+
(i/23)*20);
  }
  G+=5*dirG;
  B-=7*dirB;
  if(G>255 || G<0)dirG*=-1;
  if(B>255 || B<0)dirB*=-1;
}

void pintar(int index,int a,int x,int y){
  pushMatrix();
  translate(x,y);
  fill(color(255-(index/40),(G*index)/255,
(B*50)/255,127));
  scale(1+abs(cos(radians(index+a))));
  if(index%2==0){
    rotateZ(radians(index*2+a));
    rect(0,0,20,20);
  }else if(index%3==0){
    ellipse(0,0,20,20);
  }else{
```

```
    rotateZ(radians(index*4-a));
    triangle(0,sqrt(10),-10,20,10,20);
  }
  popMatrix();
}
```