

Creación de animaciones gráficas con Processing

FIGURAS ANIMADAS

Processing es una aplicación en Java que permite al no programador llegar a ser todo un artista gráfico. Os mostramos cómo crear objetos animados y cómo publicarlos con un applet similar a Flash.

POR KRISTIAN KISSLING

Si alguna vez ha intentado programar un cubo que rota con OpenGL, probablemente habrá tenido que emplear varias horas de depuración en el proceso. La falta de librerías o una declaración incorrecta de variables puede hacer la vida imposible al programador aficionado.

Processing

La herramienta de programación de gráficos denominada Processing acerca los efectos gráficos al usuario medio. Processing está orientada a artistas que tienen ideas pero que no se han licenciado en Ciencias de la Computación. La herramienta es ideal para usuarios a los que les gustaría mejorar la presentación de la información sin tener que recurrir a los habituales y aburridos gráficos como los diagramas de tarta multicolor.

El proyecto Code Swarm [1], por ejemplo, usa Processing para visualizar el desarrollo de varios proyectos de software libre en el transcurso del tiempo. Los fascinantes resultados recuerdan a una colmena, que a veces está tranquila y otras veces repleta de gran número de abejas muy activas.

Processing nos permite escribir un sencillo programa y luego simplemente pulsar *Play*. Por supuesto, el código puede que se ejecute o no. En este último caso se nos proporcionará información muy clara del problema para ayudarnos a resolverlo. Si queremos publicar los resultados en Internet, podemos generar un applet Java presionando un

botón, y lo podemos subir a nuestra página Web o a nuestro servidor.

Processing, que vio la luz por primera vez alrededor del año 2001 en los laboratorios del MIT, es comparado a menudo con Flash. Sin embargo, al contrario que Flash, se trata de un proyecto en software libre. En lugar de los plugins Flash, para visualizarlo simplemente se necesita un plugin Java para el navegador, una extensión que generalmente se instala casi instantáneamente.

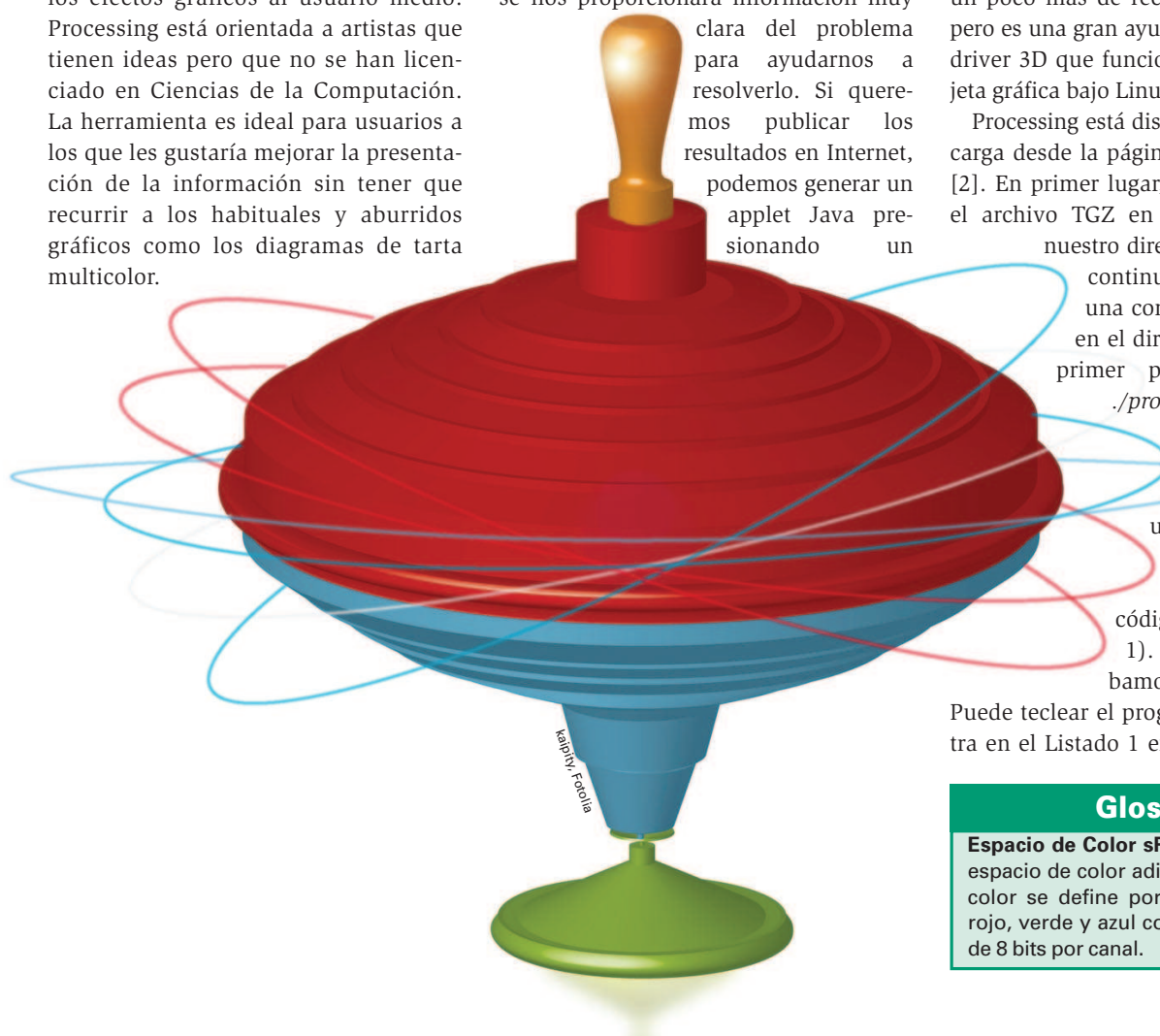
Instalación

Para disfrutar las capacidades 3D del entorno de Processing, necesitamos bien una tarjeta gráfica con aceleración 3D y soporte en los drivers, o bien el motor software 3D que se incluye en la aplicación Java, *P3D*. El motor 3D *P3D* necesita un poco más de recursos que OpenGL, pero es una gran ayuda si no tenemos un driver 3D que funcione con nuestra tarjeta gráfica bajo Linux.

Processing está disponible para su descarga desde la página Web del proyecto [2]. En primer lugar, desempaquetamos el archivo TGZ en un directorio bajo nuestro directorio de usuario, a continuación lanzamos una consola, nos ubicamos en el directorio creado en el primer paso, y tecleamos `./processing` para ejecutar la aplicación.

Tras lanzar el programa veremos un campo vacío en el cual podemos empezar a teclear código (véase la Figura 1). ¿Por qué no lo probamos directamente?

Puede teclear el programa que se muestra en el Listado 1 en la ventana, y pre-



Glosario

Espacio de Color sRGB: Se trata de un espacio de color aditivo en el cual cada color se define por sus componentes rojo, verde y azul con una profundidad de 8 bits por canal.

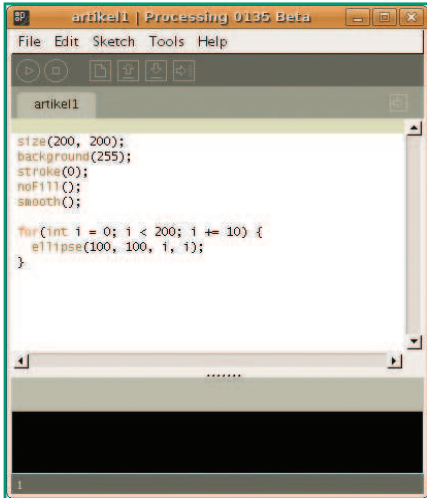


Figura 1: Sólo se necesitan las escasas líneas de código del Listado 1 para que Processing dibuje una figura sencilla.

sionar luego el icono con el triángulo en la parte superior izquierda. Ahora deberíamos ver el objeto mostrado en la Figura 2 (izquierda).

Cuando ejecutemos el código del Listado 1 deberíamos ver una pequeña ventana que muestra una serie de círculos concéntricos. Pulsamos *Help* | *Reference* en la interfaz gráfica de Processing en busca de una referencia de comandos que nos proporcione una pequeña pero clara explicación de cada función.

Si preferimos una descripción más detallada, podemos seleccionar *File* | *Examples* y descubriremos uno de los muchos programas de ejemplo que se incluyen en los archivos de Processing. El código está comentado en todos los casos.

Igualmente, podemos encontrar algunos libros sobre Processing. El tomo de Ira Greenberg [3] que recomiendo está dirigido a artistas y no programadores.

Las primeras dos líneas del programa de ejemplo del Listado 1 abren un espacio de trabajo de 200 por 200 píxeles con

Listado 1: Dibujar una Figura Sencilla

```
01 size(200, 200);
02 background(255);
03 stroke(0);
04 noFill();
05 smooth();
06 for(int i = 0; i < 200; i
  +=10) {
07 ellipse(100, 100, i, i);
08 }
```

el fondo blanco *background(255)*. Si sólo introducimos un valor, estamos seleccionando uno de los 255 posibles tonos de grises que van desde el 0 (negro) hasta el 255 (blanco). Si introducimos tres valores separados por comas, estamos especificando el espacio de color sRGB. Una entrada como *background(255, 0, 0)* proporcionaría fondo rojo puro, por ejemplo. Para añadir una pincelada negra, seguimos el mismo procedimiento (*stroke(0)*).

La función *noFill()* indica a Processing que no rellene las figuras, de otra manera el círculo más grande cubriría al resto. El efecto anti-aliasing proporciona bordes más suaves y se habilita con la palabra clave (denominada con acierto) *smooth()*, el cual actúa sobre todas las figuras siguientes. A continuación, un bucle *for()* ejecuta una función específica hasta que ocurre la condición de parada (en este caso, cuando *i* alcanza el valor de 200).

El bucle comienza fijando un contador entero a cero (*int i = 0*) y a continuación se incrementa en pasos de 10 (*i += 10*). El bucle continúa mientras el contador está por debajo de 200 (*i < 200*). Tan pronto como alcanza el valor objetivo, finaliza. En cada vuelta ejecuta la función *ellipse()* de la línea 7. Esta función dibuja un círculo con las especificaciones proporcionadas entre paréntesis. Los valores 100, 100 posicionan el centro del círculo en el centro del espacio de trabajo de 200 por 200 píxeles. Las dos variables *i* definen el diámetro vertical y horizontal del círculo. Debido a que *i* se incrementa en pasos de 10, Processing comienza dibujando un círculo con un diámetro de 10, luego de 20, etc.

Cambios

Pequeños cambios pueden tener efectos drásticos en la programación. Por ejemplo, si cambiamos la segunda *i* entre

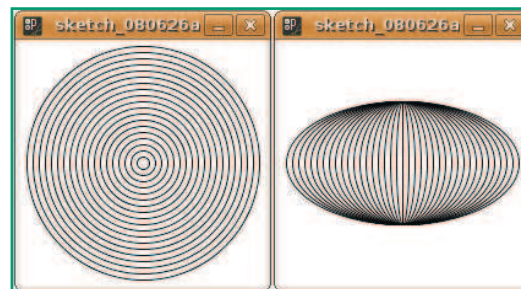


Figura 2: Al cambiar un simple parámetro la figura cambia visiblemente.

Listado 2: Cambiamos un Parámetro

```
01 for(int i = 0; i < 200; i
  +=10) {
02 float r = random(200);
03 ellipse( 100, 100, r, 100);
04 }
```

paréntesis a continuación de *ellipse* y la reemplazamos con 100, es decir

```
ellipse(100, 100, i, 100)
```

estamos cambiando de un círculo a una elipse (Figura 2, derecha). Mientras que el diámetro horizontal continúa creciendo, gracias a la *i* restante, el diámetro vertical permanece constante en 100 píxeles.

El siguiente paso añade un elemento aleatorio. Vamos a reemplazar el bucle *for()* del Listado 1 con el bucle del Listado 2.

El nuevo bucle introduce un nuevo elemento con la variable aleatoria *r*. En cada iteración, la línea *float r = random(200);* crea un número en punto flotante *float* entre 0 y 200 y lo guarda en la variable *r*. Este número continuamente cambiante explica las instancias horizontales irregulares de la Figura 3. Si añadimos un segundo número aleatorio (Listado 3) y dejamos el diámetro vertical al azar (Figura 4) se añade más aleatoriedad. La figura aparece más espacial, pero en ningún caso es tridimensional.

Entrando en la Tercera Dimensión

Entrar en la tercera división es realmente fácil con Processing. Para demostrarlo, el Listado 4 dibuja un sencillo cubo. Lo bueno de esto es que reacciona a los movimientos del ratón – podemos empujar y arrastrarlo en un espacio de trabajo definido (Figura 5). Desafortunadamente, la herramienta de captura de pantallas de Gnome no muestra dónde está el ratón, pero doy mi palabra de que estaba en el centro del cubo en las tres figuras.

El código comprende dos funciones básicas: *setup()* y *draw()*, las cuales contienen otras funciones. La función de configuración define los parámetros básicos del programa, y la



Figura 3: Los números aleatorios ayudan a la figura a escapar de los estrictos patrones geométricos.

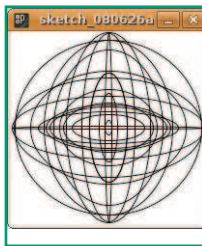


Figura 4: Si usamos dos números aleatorios, la figura comienza a parecer un poco 3D.

función de dibujo dibuja un objeto específico y ejecuta los comandos del interior de las llaves antes de reiniciar.

La primera función de la línea 2 inicializa el espacio de trabajo. El alcance está seguido por una palabra clave de *P3D* con aspecto siniestro que inicializa el motor 3D (basado en software) interno de Processing. El motor calcula los gráficos 3D y los envía a la tarjeta gráfica, la cual los renderiza en pantalla. Para usar OpenGL a la hora de renderizar por hardware los gráficos mediante el uso del soporte para tarjetas gráficas 3D, simplemente tenemos que añadir la siguiente línea al comienzo del código:

```
import processing.*;
opengl.*;
```

A continuación reemplazamos *P3D* por *OPENGL* en el Listado 4. Con una máquina potente no apreciaremos ninguna diferencia para figuras simples.

Para conseguir un rendimiento mejorado en las figuras más complejas necesitaremos una tarjeta gráfica con aceleración 3D. En otras palabras, tendremos que instalar los drivers Nvidia, ATI o Intel.

La línea 3 especifica el número de fotogramas por segundo. Un valor de 30 debería ser suficiente, y no notaremos ningún movimiento brusco.

El siguiente paso está relacionado con las propiedades del cubo. En este ejem-

Listado 3: Añadimos Aleatoriedad

```
01 for(int i = 0; i < 200; i
  +=10) {
02 float r = random(200);
03 float s = random(200);
04 ellipse( 100, 100, r, s);
05 }
```

plo se usa un gris claro como color de fondo (*background(200)*).

Las funciones *pushMatrix()* y *popMatrix()* de las líneas 8 y 14 dibujan un marco de trabajo alrededor de la figura. Mientras que *pushMatrix()* crea un nuevo sistema de coordenadas y lo carga en el Matrix Stack [4] (véase el cuadro titulado “¿Qué es la Matriz?”), *popMatrix()* restaura la matriz usada anteriormente.

Aunque el programa funcionaría sin ninguna de estas dos funciones, tiene sentido usarlas, como muestra el siguiente ejemplo.

Dentro de la matriz, la primera función es *translate()*. Lee los tres parámetros y posiciones del cubo en las intersecciones de los ejes *x*, *y* y *z*, es decir, en el espacio virtual.

La función supone un valor de 0, 0 para el borde superior izquierdo del espacio de trabajo y fija el centro de la figura a esas coordenadas.

Cualquier otra función de traslación que esté dentro de la función de dibujo usará las coordenadas de traslación previas como su punto de referencia. Esto significa que podemos mover el cubo fácilmente con el ratón porque *mouseX*, *mouseY* siempre hace referencia a las coordenadas *x/y* previas del puntero del ratón.

El siguiente paso es definir un color en la variable *c1*, gris claro en este caso, aunque igualmente siempre podemos recurrir a valores sRGB para ello. El script rellena el cubo con este color en la línea 11 y colorea los bordes en gris oscuro mediante la llamada a *stroke()*.

Pero no se ensambla realmente hasta la línea 13. La longitud del borde se fija a

Listado 4: Un Cubo Sencillo

```
01 void setup() {
02 size(400, 400, P3D);
03 frameRate(30);
04 }
05
06 void draw() {
07 background(200);
08 pushMatrix();
09 translate(mouseX,mouseY,10);
10 color c1 = color(180);
11 fill(c1);
12 stroke(128);
13 box(150);
14 popMatrix();
15 }
```

150 píxeles. Para dibujar un cubo necesitamos introducir tres valores para la longitud, anchura y altura.

La Estrella de la Muerte

Finalmente, el Listado 5 presenta un script más complejo. Éste muestra dos cubos verdes que rotan en sentidos contrarios (véase la Figura 6).

Los breves comentarios hacen que el código sea más inteligible, pero las finalidades más importantes son el uso de *pushMatrix()* y de *popMatrix()*, y los comandos para rotar las figuras.

En las líneas 3 a 7, se comienza por la inicialización de dos variables para cada cubo *-xdirection* y *ydirection* – con valores de 1 y -1 respectivamente. La función de configuración ya nos es familiar, pero la función de dibujo cambia.

El script llama a *directionalLight()* para arrojar algo de luz al cubo. La función tiene seis parámetros, los tres primeros destinados a los valores sRGB o HSV, y los otros tres son las posiciones de los ejes *x*, *y* y *z* del sistema de coordenadas. Los tres primeros valores especifican el color de la luz, y los últimos tres definen la dirección desde la cual llega la luz.

El script define el tamaño, color y rotación de los dos cubos entre las funciones *pushMatrix()* y *popMatrix()*. Éstas son las mismas en ambos cubos excepto por un pequeño detalle: las funciones *rotateX()* y *rotateY* usan sentidos opuestos de rotación.

Las líneas 42 y 43 rotan el cubo 2 un grado en sentido contrario a las agujas

¿Qué es la Matriz?

El siguiente ejemplo nos dará una idea aproximada de lo que es la Matrix Stack. Si creamos una serie de tres secuencias (cada una de ellas con múltiples movimientos), cada secuencia comenzará con la función *pushMatrix()*, que crea un nuevo sistema de coordenadas. Para volver a la secuencia anterior, con las coordenadas usadas anteriormente, simplemente tenemos que hacer un *popMatrix()*. La cosa funciona como una especie de botón deshacer, aunque no podemos volver hacia adelante.

Glosario

HSV: Un espacio de color que define el color como referencia al tono, saturación y valor.



Figura 5: El cubo sigue los movimientos del ratón y puede moverse fácilmente por la pantalla.

del reloj en los ejes x e y . Ambas variables comienzan con un valor de -1 . El cubo 1 rota en sentido contrario. Debido a que están centrados en la misma posición, son contra-rotativos. Tras esta mínima rotación, se asignan nuevos valores a las variables en las líneas 48 a 54.

El par de torsión se incrementa para el cubo 1 y se decrementa para el cubo 2. Processing ejecuta la función de dibujo nuevamente, y ambos cubos contra-rotan otro grado. Como la función de dibujo usa una tasa de 30 fotogramas por segundo, los cubos rotan con suavidad.

Otro aspecto curioso puede ser de su interés: si se eliminan las funciones `pushMatrix()` y `popMatrix()` del código del ejemplo, el segundo cubo orbita súbitamente en torno al primero como si fuera un planeta. Cada cubo necesitaría un sistema de coordenadas diferentes, a menos que queramos diseñar un planetario virtual.

Publicar en la Web

Por defecto, Processing guarda nuestros trabajos en un directorio denominado `sketchbook` en nuestro directorio de usuario. Si guardamos nuestras creaciones

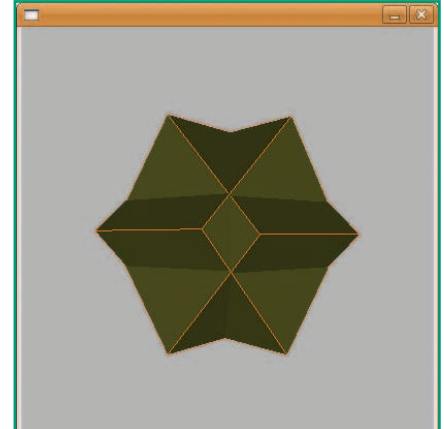


Figura 6: Los cubos contra-rotativos iluminados por un punto de luz virtual.

como `experimento_0815`, las encontraremos más tarde en `~/sketchbook/experimento_0815/`.

Para mostrar nuestras habilidades en programación a los amigos y compañeros, puede que queramos publicar nuestras figuras en la Web. Para ello debemos seleccionar `File | Export`. Processing crea un subdirectorio `applet` en el directorio mencionado anteriormente y guarda los archivos necesarios dentro. Si abrimos el archivo `index.html` desde nuestro navegador, veremos las figuras.

Para publicarlas en nuestra propia página Web, podemos usar un cliente FTP para subir el directorio `applet` al servidor con nuestra página Web y añadir un enlace a `index.html` en nuestra página de inicio.

Si queremos acceder al archivo directamente, simplemente podemos dirigirnos a `http://www.midominio.com/applet/index.html`

Listado 5: Cubos Contra-Rotativos

```

01 import processing.opengl.*;
02
03 float ydirection1 = 1;
04 float xdirection1 = 1;
05
06 float ydirection2 = -1;
07 float xdirection2 = -1;
08
09 void setup() {
10 size(400, 400, OPENGLE);
11 frameRate(30);
12 }
13
14 void draw() {
15 background(0);
16
17   directionalLight(255,255,128,0,0,-1);
18 /* Cubo 1 */
19 pushMatrix();
20
21 translate(200,200,100);
22 color c1 = color(102, 102,0);
23 fill(c1);
24 stroke(204,102,0);
25
26
27 rotateY(radians(ydirection1));
28 rotateX(radians(xdirection1));
29
30 box(100);
31 popMatrix();
32
33 /* Cubo 2 */
34 pushMatrix();
35
36 translate(200,200,100);
37 color c2 = color(102, 102,0);
38 fill(c2);
39 stroke(204,102,0);
40
41
42 rotateY(radians(ydirection2));
43 rotateX(radians(xdirection2));
44
45 box(100);
46 popMatrix();
47
48 /* cube 1 clockwise */
49 ydirection1 = ydirection1 +1;
50 xdirection1 = xdirection1 +1;
51
52 /* cube 2 counterclockwise */
53 ydirection2 = ydirection2 -1;
54 xdirection2 = xdirection2 -1;
55
56 }

```

RECURSOS

- [1] El proyecto The Code Swarm usa Processing: <http://code.google.com/p/codeswarm/>
- [2] Página de descarga de Processing: <http://processing.org/download/index.html>
- [3] Greenberg, Ira A. "Processing: Creative Coding and Computational Art". Computer Bookshops, New York, 2007.
- [4] `pushMatrix()` y `popMatrix()` explicados: http://processing.org/discourse/yabb_beta/YaBB.cgi?board=Syntax;action=display;num=1177279317