Processing

Un tutorial originalmente destinado a estudiantes del ITP, suplementario al material del curso "Introduction to Computational Media", "Programming for Non-Programmers", y "Code and Me"

Idiomas: Español, English , Japanese , Korean

Professor del Taller
Josh Nimoy
contacto: jn429 [at] nyu [dot] edu
página web

Traducción al Japanesa de
Hironobu Fujiyoshi and Ayako Takabatake
contacto: hf [at] cs [dot] chubu [dot] ac [dot] jp

Traducción al Coreano de
Koo-Chul Lee
contacto: kclee [at] phya [dot] snu [dot] ac [dot] kr

Traducción al Español de
Gerald Kogler y Angela Precht
contacto: geraldo [at] yuri [dot] at

Creadores del Software
Processing es un proyecto abierto iniciado por Ben Fry y Casey Reas

Descripción

Processing es un contexto para explorar el espacio conceptual emergente que nos entregan los medios electrónicos. Es un entorno para aprender los fundamentos de la programación informática dentro del contexto de las artes electrónicas y es un bloc de notas electrónicas para desarrollar ideas.
www.processing.org

El entorno de Processing es el más fácil compilador de Java / entorno de programación multimedia y gráfico conocido por el hombre. El sistema puede ser usado para producir piezas que arrancan localmente, como también Applets de java incrustados en la web. Deliberadamente, el programa está diseñado para hacer un puente entre la programación gráfica educacional, y el java "real". Processing puede ser utilizado como rueda de entrenamiento, pero no tiene por qué ser eso.

El propósito de este manual es introducir a los usuarios de Flash Macromedia y Director al entorno de Processing a través de la comparación y el contraste de los sistemas. La teoría dice que el conocimiento adquirido a través de las herramientas de Macromedia puede ser fácilmente transferido, reduciendo la cantidad de clases necesarias. Se asume que tienes un conocimiento básico de alguno de los productos de Macromedia. Hacia el final de este manual deberías ser capaz de producir tus propias piezas de Processing (Java).

Índice

Introducción

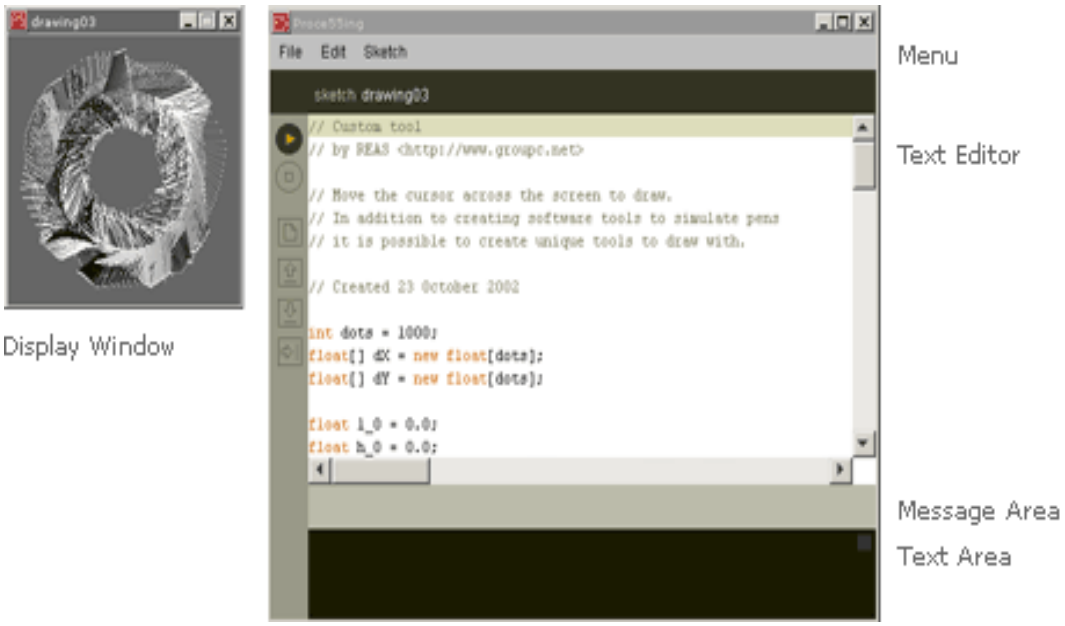
Actualmente, en clases interactivas de diseño digital, el vehículo educativo dominante ha sido Flash o Director. Los estudiantes están empezando a realizar trabajos geométricamente dinámicos y algorítmicamente más complejos influenciados por trabajos realizados en entornos diferentes a los suyos. En ITP, atestigüé un experimento en una clase (claro, que desde entonces están usando Processing). En medio de la enseñanza de Director Lingo, un intensivo de una semana a la programación en Java introduce a los alumnos a un lenguaje diferente a Lingo – con la esperanza que adquieran una visión más diversa de distintos sistemas de programación. Se les entregó una plantilla y ellos simplemente cambiaron el código. Luego de una semana de confusión algunos de los estudiantes quedaron con una sensación de vacío, anhelando saber más de Java. No hubo manera de explicarles que un curso de Java promedio en una escuela, frecuentemente te tiene trabajando en una consola de texto y que tiene poco o nada de relación con gráfica de Applets al menos que sea un curso específicamente enfocado para enseñar eso. En la lección siguiente, espero hacer un puente sobre este hueco académico con la ayuda del entorno de Processing. No está pensado como un reemplazo a aquellos cursos de Java sino un suplemento que se preocupa de la logística, sin profundizar en los matices sintácticos. Además, Processing y Java no son presentados como el siguiente nivel después de Macromedia, tampoco como un sistema de nivel inferior. Son simplemente una alternativa capaz de hacer cosas de modo distinto. Si actualmente asistes a un curso de Java, es posible que uses Processing en tus trabajos, dependiendo de la flexibilidad de tu profesor. Este manual es una mezcla entre mi redacción e imágenes y otras encontradas en el sitio de Processing que mantienen Casey Reas y Ben Fry. Te familiarizarás con estos nombres en la medida que te hagas usuario de Processing.

Obtener el software de Processing

Processing es libre (libre como la cerveza gratis, libre como la libertad de expresión, libre como un país libre) y todavía en desarollo. Seguirá siendo libre incluso después de ser terminado. Este software está actualmente en la fase BETA. Los errores están siendo reparados y otros aspectos están siendo agregados. Para descargarte el programa de instalación de multiplataforma, puedes mandar un email a los desarrolladores para unirte a la comunidad de testeo. En la página de Processing, pincha **Download** para más instrucciones. Además hay un fluido sistema de mensajes entre los testeadores. Es altamente recomendable que te crees una identificación de entrada (login) para ti mismo. La comunidad que se ha creado a su alrededor es la mejor manera de recibir ayuda en cualquier materia, de otros miembros de la comunidad de testeo, de los mismos autores y de otros empollones informáticos como el autor de este artículo. Es importante señalar que esta comunidad en línea ayuda a desarrollar Processing a través de la discusión de aspectos en los foros. En la página de Processing pincha **Discourse**. Tanto la página de Processing como el software están constantemente siendo actualizados. Ve revisando por nuevas actualizaciones y versiones del software. Justamente ahora es un momento muy emocionante!

Un paseo por la interfaz

La siguiente imagen fue sacada de www.processing.org. Para verla en contexto pincha **Reference** y luego **Environment**.



Probablemente estás pensando: "Jo, qué simple esta interfaz. ¿Cómo puede ser tan potente como Director o Flash?" Ambos, Director y Flash, poseen todo tipo de interfases de importación y edición de medios, basados en funciones comunes de multimedia comercial. En Processing todo esto se hace o usando otro programa o programando en Java. Por ejemplo, Flash tiene su propio mini Ilustrator, mientras Director viene con su propio mini Photoshop. En consecuencia, un largo trozo del trabajo realizado en ambos programas ha asemejado las restricciones de sus editores integrados. En Processing (y en Java), tu provees tu propia lista de gráficas vectoriales o archivos GIF, y los interpretas usando programación. Eres libre para usar tus propias formas y estructuras, usando el lenguaje para controlar los píxeles en la pantalla más directamente. Para aquellos que gustan de experimentar y desean producir nuevas formas de manera independiente o delante del status quo y sus herramientas de automatización, Processing puede ser más conveniente.

Aquí hay una introducción de seis botones del costado izquierdo de la ventana.

El botón de **reproducir** (play) es el mismo que en Director y Flash. Cliquéalo para ver tu código ejecutado como un programa.

El botón de **parar** (stop) es el mismo que en Director o Flash. Cliquéalo para detener tu programa.

Crea un **nuevo** (new) archivo. Processing los llama sketches (bosquejos). los puedes llamar también Applets, programas o piezas interactivas. Director y Flash los llamanmovies (*películas*).

Abre (open) un sketch preexistente. Un menú aparecerá y podrás elegir dentro de tu propia colección, guardada en la carpeta especial de Processing que te enseñaré más adelante. También puedes elegir dentro de una amplia gama de ejemplos de sketches realizados por famosos diseñadores/artistas de nuevos medios, para aprender de ellos y usarlos como código de referencia.

Guarda (save) en sketch actual dentro de la carpeta de sketches de Processing. Si quieres darle un nombre distinto vete a guardar como (save as) en el menú de archivo (*File*).

Exporta (export) el sketch a la carpeta de sketch de Processing, esta vez como un Applet de java, completo con su propio archivo html. Esta propiedad será tratada con más profundidad más adelante.

Para mayor y más detallada información acerca del entorno de Processing, ver la referencia de Entorno de Processing.

Manipulación de medios a bajo nivel

En Director, uno importa o crea medios dentro de un reparto, luego lo arrastra hacia un escenario donde existirá como un actor. En Flash, uno también importa o crea medios dentro de una librería y luego los instancia como clips de películas a un escenario similar. En Processing (y en Java) esta importación de medias está realizada en código, de manera similar como trabaja HTML. Adicionalmente, cualquier medio personalizado que tu inventas (sistema de vectores, datos DNA, muestras de colores de una película) todos pueden ser insertados como parte del código Java. De hecho, no estás restringido a tener cualquier imagen externa o sonidos si quieres mantener todo en un archivo ordenado, porque el píxel de tu imagen también puede convertirse para ser parte de tu código, y los datos de sonidos también pueden ser almacenados como un largo array de datos. El beneficio de una librería o reparto es para tener mayor control sobre el formato, para ahorrar espacio en el disco / memoria, y también para agregar características apuntables&cliqueables hacia un metafórico sistema de archivo común. El beneficio del actor o de la película es que un objeto visible y tangible puede estar en la pantalla y ser una manera fácil para que la gente cree botones, video juegos, caracteres, elementos individuales gráficos y otros elementos visuales y controlables de espacio positivo. De todas maneras, creando grupos de cooperación de elementos y objetos que no son ni de espacio negativo ni positivo, esta metáfora se ha convertido en carga para algunos. En Processing, esta complicada capa no existe; hay solo teclado, ratón y una serie de eventos en conjunción con rutinas básicas de dibujo. Uno se

ocupa de redibujar la escena repetidas veces para introducir cambios y tiempo. En esta materia, es tu responsabilidad el escribir tu propio sistema de actor o película, pero no te lo requerirán. Inventa otra metáfora que será más útil para ti como artista. También contribuirás para diversificar con profundidad la estética de aquellos trabajos en cuales usualmente vemos la influencia de Macromedia.

En las siguientes secciones de esta guía, introduciré métodos para renderizar la imaginería a la pantalla. Luego introduciré una animación. Finalmente, te mostrare como agregar animación con el ratón, teclado y el puerto serial. Estos son los pilares básicos para cualquier cosa que quieras hacer con las herramientas de mayor nivel. Serás capaz de hacerlos en Java si te focalizas en construirlos tu mismo.

Estructura sintáctica

Para los que usan Flash MX esto es un repaso. Lo siguente se llama StatementsComments en los ejemplos.

```
// Statements & Comments
// by REAS

// Statements are the elements that make up programs.
// The ";" is used to end statements. It is called the "statement terminator."
// Comments are used for making notes to help people better understand programs.
// A comment begins with two forward slashes ("//").

// Created 1 September 2002

// The size function is a statement that tells the computer
// how large to make the window.
// Each function statement has zero or more parameters.
// Parameters are data passed into the method
// and used as values for specifying what the computer will do.
size(200, 200);

// The background function is a statement that tells the computer
// which color to make the background of the window
background(102);
```

Y las variables de Java son como a continuación:

```
int x = 0;
println(x);
x=x+1;
println(x);
x=x+1;
println(x);
```

Pincha reproducir (play) y ves eso:

```
0
1
2
3
```

Personas que usen Flash: No hay nada como var. Para mayor información de variables, consultar la guía oficial de lenguaje Java.
Aquí está la parte de la variables.

Qué ocurre con if-then?

```
int a = 1;
int b = 2;

if(a==b){
  println("same");
}else{
  println("different");
}
```

Pincha reproducir (play) y ves eso:

```
different
```

Las cosas son un poco diferentes a Lingo si hay que comparar. El programador usa un sólo "=" para asignar a una variable un valor. Uno usa "==" (doble igual) cuando intenta determinar que si un número es igual a otro número o no. "No igual a " no es más "<>", ahora es "!=" y los demás son los mismos ("<" , ">" , ">=" , y "<="). Para información en condicionales, consulte a la autoridad.

¿Y repetir bucles?

```
for(int i=0 ; i<5 ; i++){
   println(i);
}
```

Pincha reproducir (play) y ves eso:

```
0
1
2
3
4
```

Usuarios de Lingo, esto es los mismos que "repeat with i=0 to 4". Dentro de estos paréntesis hay tres declaraciones separadas por dos punto y coma. La primera declaración crea una variable temporal. La segunda declaración especifica la condición que permite al bucle continuar su repetición. A penas i no es más menor que 5, el bucle se detendrá. La tercera declaración te dará la posibilidad de incrementar i como desees. "i++" es la abreviación de "i = i+1". Sun podrá contarte más acerca de los bucles.

Mientras los bucles son similares a la estructura de un if-then.

```
while(6!=2){
   println("muhuhaha!");
}
```

No ejecutes este programa!

:)

Si estás curioso de saber más de la sintaxis de control de flujos, aquí hay un link del manual de lenguaje de Java. La parte de control de flujos.

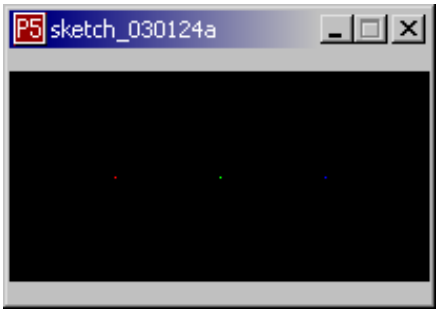
Me meteré en funciones en un momento. Entrego estás instrucciones básicas de sintaxis no para ser cuidadoso sino para que comprendas las instrucciones que vienen a continuación.

Para una mayor introducción a las estructuras, mira la Comparación de Lenguaje Processing y los Ejemplos de Estructura de Processing.

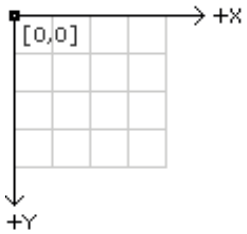
Dibujo 2D estático

```
size(200,100);
background(0,0,0);
stroke(255,0,0);
point(50,50);
stroke(0,255,0);
point(100,50);
stroke(0,0,255);
point(150,50);
```

Arranque este código y obtendrás la siguiente imagen. Esta es la gran ventana negra con tres píxeles de colores rojo, verde y azul.



Permítenos descomponer este código línea por línea.



Primero que nada conviene saber que la pantalla es un gráfico de píxeles, cada uno registrado para una única coordenada (X,Y). El origen (0,0) está en la esquina izquierda arriba del rectángulo. Al sumar a Y, te mueves hacia abajo. Al sumar a X, te mueves a la derecha. Es parecido a jugar el juego de tablero, **Combate Naval**. No tiene diferencia a Director o Flash.

```
size(200,100);
```

Nombrando la función de tamaño (**size**), permite el tamaño del lienzo a 200 píxeles de altura, 100 píxeles de ancho. Si no defines esto al inicio quedará determinado como 100x100.

```
background(0,0,0);
```

Llamando la función de fondo (**background**) te permite cambiar el color de todo el lienzo. En Director esto es el color del escenario. En Flash, esto es el color de fondo del documento. 0,0,0 significa negro. Si nunca llamas negro al fondo, quedará predeterminado como gris.

```
stroke(255,0,0);
```

Llamar la función de trazo (**stroke**) te permite cambiar el color del dibujo actual de tal manera que cada comando de color llamado luego será dibujado usando ese color. 255,0,0 significa rojo. Si nunca llamas trazo en Processing, por definición será negro.

```
point(50,50);
```

Llamar la función de punto (**point**) fijará el pixel en 50,50 al color del lienzo actual. En este caso, rojo.

```
stroke(0,255,0);
point(100,50);
```

Este código hace un punto verde en el centro.

```
stroke(0,0,255);
point(150,50);
```

Este código dibuja el punto azul a la derecha.

Como puedes ver, esto es similar a **draw()** en Lingo, o aquellos métodos de dibujo en ActionScript. Tú controlas la pantalla como si fuera un lienzo que entiende ciertas operaciones de dibujos. Permítenos desarrollar formas más complejas.

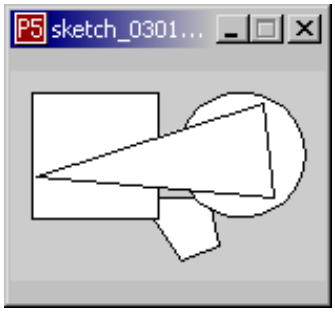
```
background(0,0,0);
stroke(255,255,255);
line(0,0,60,40);
stroke(255,255,0);
line(30,50,100,100);
```

Aquí estoy dibujando dos líneas. La primera blanca y la segunda, amarilla.

En la línea de funciones, los dos primeros parámetros son las primeras coordenadas y los últimos dos parámetros son las segundas coordenadas x,y. La línea se dibuja desde la primera coordenada a la segunda.

Aquí hay unas formas prefabricadas.

```
size(150,100);
quad(61,60, 94,60, 99,83, 81,90);
rect(10,10,60,60);
ellipse(80,10,60,60);
triangle(12,50, 120,15, 125,60);
```



triangle(triángulo) dibujará un polígono de tres puntas. Tiene seis parámetros. Parámetros 1 y 2 son los primeras coordenadas X,Y. Parámetros 3 y 4 son las segundas coordenadas X,Y. Los parámetros 5 y 6 son las terceras coordenadas X,Y.

triangle(x1, y1, x2, y2, x3, y3);

quad (cuadrado) dibujará un polígono de 4 puntas. La estructura de los parámetros es similar a las del triángulo, pero esta vez un cuarto par de parámetros se agregan para especificar una cuarta coordenada X,Y.

quad(x1, y1, x2, y2, x3, y3, x4, y4);

rect dibujará un rectángulo. El primer y segundo parámetro especificarán la posición, mientras que el tercero y el cuarto lo harán con la altura y el ancho.

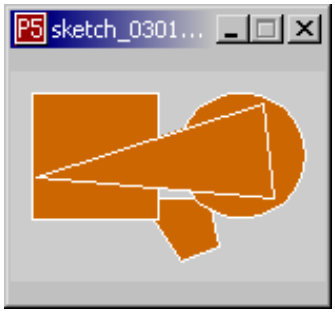
rect(x, y, width, height);

ellipse dibujará un óvalo. Sus parámetros trabajan de la misma manera que los de rect.

ellipse(x, y, width, height);

Ahora modificaré el programa para mostrarte algo nuevo. El nuevo código **está marcado**.

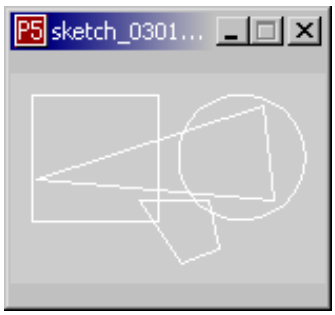
```
size(150,100);
fill(#CC6600);
stroke(#FFFFFF);
quad(61,60, 94,60, 99,83, 81,90);
rect(10,10,60,60);
ellipse(80,10,60,60);
triangle(12,50, 120,15, 125,60);
```



(Nota aqui, que estoy especificando colores de una manera diferente que antes -esta vez en estilo HTML)

Fill (llenar) se introduce como un primo del trazo. Fill es lo que hace que el polígono verde, mientras que el trazo es lo que hace que los bordes sean rojos. Los parámetros de fill especifican el color, como el trazo. Está predeterminado como blanco. Pero qué pasa si no quieres llenar?

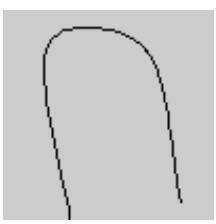
```
size(150,100);
noFill( );
stroke(#FFFFFF);
quad(61,60, 94,60, 99,83, 81,90);
rect(10,10,60,60);
ellipse(80,10,60,60);
triangle(12,50, 120,15, 125,60);
```



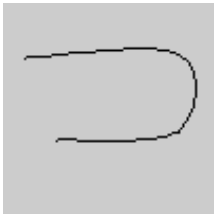
Ahora puedes ver el cuadrado debajo del óvalo porque sólo los trazos han sido dibujados. De manera similar, existe un **noStroke** (no trazo), que deshabilita los bordes de ser dibujados . Para habilitar el trazo o el llenar una vez más debes llamar el trazo o el llenar especificando un color.

Dibujar con curvas es ligeramente más complicado que dibujando con lineas rectas. Especificando una curva requiere proveer información no visual que ayude a definir la severidad y dirección de la curvatura. Processing provee ambos métodos, curve( ) (curva) y la curva de bezier( ).

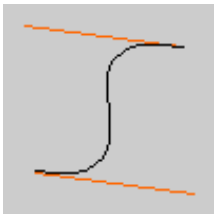
```
curve(84, 91, 68, 19, 21, 17, 32, 100);
```



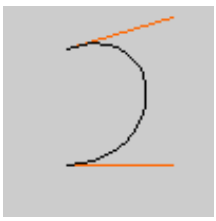
```
curve(10, 26, 83, 24, 83, 61, 25, 65);
```



```
stroke(255, 102, 0);
line(85, 20, 10, 10);
line(90, 90, 15, 80);
stroke(0, 0, 0);
bezier(85, 20, 10, 10, 90, 90, 15, 80);
```



```
stroke(255, 102, 0);
line(30, 20, 80, 5);
line(80, 75, 30, 75);
stroke(0, 0, 0);
bezier(30, 20, 80, 5, 80, 75, 30, 75);
```



curve(x1, y1, x2, y2, x3, y3, x4, y4);
bezier(x1, y1, x2, y2, x3, y3, x4, y4);

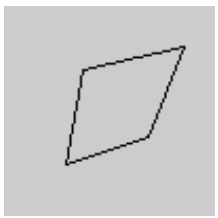
Para la función de **curve( )**, el primer y segundo parámetro especifica el punto de la curva y los dos últimos parámetros especifican el segundo punto de la curva. Los parámetros de en medio establecen los puntos para definir la forma de la curva.

Para la función **bezier( )**, los primeros dos parámetros especifican el primer punto de la curva y los dos últimos parámetros especifican el último punto. Los parámetros de en medio otorgan el contexto para definir la forma de la curva.

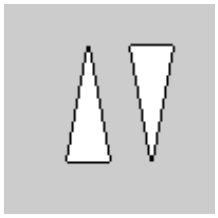
En los ejemplos de **bezier( )** de arriba, las lineas naranjas revelan los puntos de control escondidos de las curvas.
A pesar que Processing otorga estos primitivos rápidos, eres libre para construir tus propias formas.
Usando los métodos de **beginShape( )** y el **endShape( )** son la llave para crear formas más complicadas. **beginShape( )** empieza registrando vértices para una forma y **endShape( )** termina ese registro. El comando **beginShape( )** requiere un parámetro para decirle qué tipo de forma para crear desde los vértices otorgados. Los parámetros disponibles para **beginShape( )** son LINES, LINE_STRIP, LINE_LOOP, TRIANGLES, TRIANGLE_STRIP, QUADS, QUAD_STRIP, y POLYGON.
Luego de dar el comando **beginShape( )**, una serie de comandos **vertex( )** (vértices) deben seguir. Para dejar de dibujar la forma, usa el comando **endShape( )**. Los comandos **vertex ( )** con dos parametros especifican la posición 2D y los comandos **vertex( )** con tres parámetros especifican la posición en 3D. Cada forma será delineada con su correspondiente color de trazo y llenada con el color de llenado (ver la sección de Color para más información.

Aquí hay algunos ejemplos de Processing.org

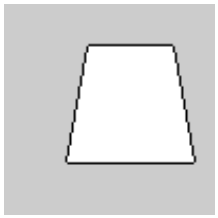
```
beginShape(LINE_LOOP);
vertex(30, 20);
vertex(85, 20);
vertex(85, 75);
vertex(30, 75);
endShape( );
```



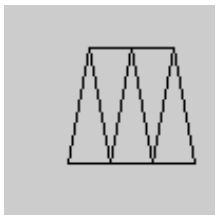
```
beginShape(TRIANGLES);
vertex(30, 75);
vertex(40, 20);
vertex(50, 75);
vertex(60, 20);
vertex(70, 75);
vertex(80, 20);
vertex(90, 75);
endShape( );
```



```
beginShape(TRIANGLE_STRIP);
vertex(30, 75);
vertex(40, 20);
vertex(50, 75);
vertex(60, 20);
vertex(70, 75);
vertex(80, 20);
vertex(90, 75);
endShape( );
```

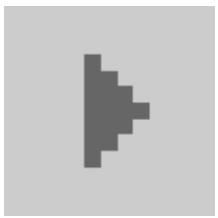


```
noFill( );
beginShape(TRIANGLE_STRIP);
vertex(30, 75);
vertex(40, 20);
vertex(50, 75);
vertex(60, 20);
vertex(70, 75);
vertex(80, 20);
vertex(90, 75);
endShape( );
```

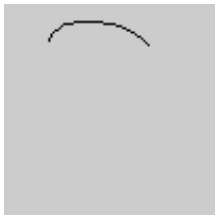


```
noStroke( );
fill(153, 153, 153);
beginShape(TRIANGLE_STRIP);
vertex(30, 75);
vertex(40, 20);
vertex(50, 75);
vertex(60, 20);
vertex(70, 75);
vertex(80, 20);
vertex(90, 75);
endShape( );
```

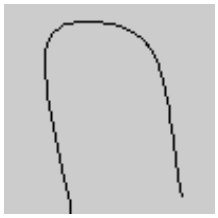
```
noStroke( );
fill(102);
beginShape(POLYGON);
vertex(38, 23);
vertex(46, 23);
vertex(46, 31);
vertex(54, 31);
vertex(54, 38);
vertex(61, 38);
vertex(61, 46);
vertex(69, 46);
vertex(69, 54);
vertex(61, 54);
vertex(61, 61);
vertex(54, 61);
vertex(54, 69);
vertex(46, 69);
vertex(46, 77);
vertex(38, 77);
endShape( );
```



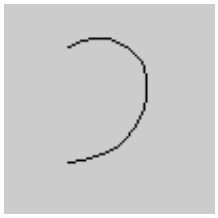
```
beginShape(LINE_STRIP);
curveVertex(84, 91);
curveVertex(68, 19);
curveVertex(21, 17);
curveVertex(32, 100);
endShape( );
```



```
beginShape(LINE_STRIP);
curveVertex(84, 91);
curveVertex(84, 91);
curveVertex(68, 19);
curveVertex(21, 17);
curveVertex(32, 100);
curveVertex(32, 100);
endShape( );
```



```
beginShape(LINE_STRIP);
vertex(30, 20);
bezierVertex(80, 0, 80, 75, 30, 75);
endShape();
```



Para más detallada información acerca de vectores de dibujos, ver los ejemplos **Form** de Processing, la referencia de formas de Processing.

Hay mucho más para dibujar e interpretar en la pantalla, pero yo sólo te he dado las rutinas de dibujo 2D para así poder cubrir animación e interactividad. Luego volveremos a los otros métodos de dibujo.

Tiempo y movimiento

En Director hay un marcador. Te asignan una cabeza lectora y métodos tweening para actores. Cosas como el vídeo, incrustaciones de Flash, QTVRs y sonido parecen animar en su propio espacio-tiempo. Si trabajas en algo más que una animación dinámica puedes utilizar un fotograma y código en respuesta a un evento **ExitFrame** o **PrepareFrame**. En Flash, te dan una linea de tiempo y un soporte de tweening más complejo que en Director. Aquellos que eligen trabajar completamente en ActionScript comúnmente usan dos marcos − uno para ajustar y para llamar un marco de repetición y otro para repetir para siempre. Actionscript también te permite responder en código con un **onClipEvent (enterFrame)**. Processing no tiene línea de tiempo, marcador o métodos de tweening, a menos que decidas estructurar tu código así. Como Lingo y Actionscript, Processing te permite responder a un evento de progresión de fotogramas con tu propia rutina de dibujo. Hasta ahora te he mostrado el código de Processing en **Modo Básico**, que es para imágenes estáticas. Es un lista de elementos visuales. Processing tiene tres modos de operación: básico, estándar y avanzado. **Modo Java** es Java convencional, sin las ruedas de entrenamiento. Para comenzar con tiempo y movimiento iremos a **Modo Continuo**. Si has estado pinchando los enlaces de este texto, debes haber visto uno o dos programas en modo estándar Processing. Aquí hay un ejemplo simple:

```
int x = 0;

void setup( ){
   noStroke( );
}

void draw( ){
   background(190);
   rect(x, 0, 5, 100);
   x=x+1;
}
```

En este ejemplo, un rectángulo blanco se mueve de izquierda a derecha sólo una vez.

Un GIF animado muestra este movimiento a la derecha.

Una sección **setup( )** opcional arranca cunado el programa empieza. La sección de bucle **draw ( )** arranca para siempre hasta que termina el programa. En Lingo **setup( )** es similar al **beginSprite** o **startMovie** - y **draw( )** es similar a **ExitFrame** o **PrepareFrame**. En Flash,

setup( ) es similar al primer marco de animación que sólo se ejecuta una vez, luego llama al bucle. **setup( )** y **draw( )** son ambas **funciones**. También puedes escribir tus propias funciones para organización y encapsulación de complejidad . Para mayor información sobre como escribir funciones personalizadas en Java, mira el tutorial de lenguaje de Java - Implementing Methods section.

Una vez que hayas escrito la primera función en Processing, ese programa cambiará automáticamente al modo estándar. Cualquier declaración fuera de su función que no sea variable de inicio no funcionará más cuando presiones play. Puedes mover este código hacia setup() o draw(). Si quieres que una variable sea global (significando que conserva su valor fuera del alcance de las funciones) entonces declara la al comienzo del programa, fuera de ambos draw() y setup(). En el ejemplo arriba, la variable x fue declarada global.

En Processing, la función de **framerate**(n) se puede utilizar para retrasar o para acelerar el bosquejo entero, pero es ciertamente posible mover cosas en diferentes velocidades simplemente variando la cantidad que incrementas, o usando floats y solamente agregando una fracción a ellos. Para un control más a largo plazo y a tiempo más exacto, Processing da el acceso completo a la medidas occidental de tiempo.

year( ) // año actual, p.ej. 2002, 2003, etc.
month( ) // mes actual, de 1..12
day( ) // día del mes, de 1..31
hour( ) // hora actual, de 0..23
minute( ) // minuto actual, de 0..59
second( ) // segundo actual, de 0..59

Una función especial llamada **millis( )** devuelve el número de milisegundos (miles de un segundo) desde comenzar el applet. Esto es comúnmente usado para secuencias animadas temporizadas.

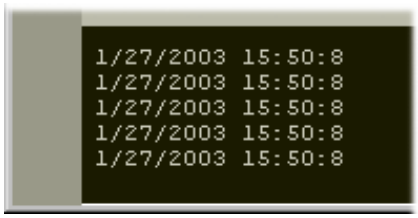
millis( ) // número de milisegundos desde el comienzo del applet.

También es posible hacer que tu applet espere al usar la función e demora (delay). Usar esta función puede ajustar efectivamente los tempos (frame rates).

delay(40); // hace una siesta de 40 milisegundos

```
void draw( ){
   print(month( )+"/");
   print(day( )+"/");
   print(year( )+" ");
   print(hour( )+":");
   print(minute( )+":");
   println(second( ));
}
```

Aquí hay un ejemplo en el que el tiempo actual y la fecha son constantemente escritas a la siguiente área al final de la ventana de Processing.



Esto no es un ejemplo muy bonito pero es fácil de entender. Para ejemplos de tiempo más bonitos (y complejos) mira Milliseconds, de REAS. Para más ejemplos de movimiento animado, véase los ejemplos **Motion** de Processing.

Ratón & teclado

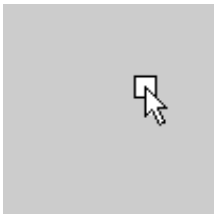
El acceso al ratón y el teclado son similares al modo que lo hacen Flash y Director . En Lingo, el ratón es direccionable con **the mouseLoc**, **the mouseH**, y **the mouseV**. Además hay también manejadores de suceso de ratón como **on mouseDown**. En Flash, hay **onClipEvent (mouseDown)**, etc. En Processing, la función **mousePressed( )** se llama cada vez que el ratón es presionado y el método **mouseReleased( )** es llamado cada vez que el ratón es liberado o dejado de presionar. Lo que debes hacer es agregar la función a tu código, así como en **draw( )**.

```
void draw( ) {
   background(190);
   rect(mouseX-5, mouseY-5, 10,
10);
}

void mousePressed( ) {
   fill(0);
}
void mouseReleased( ) {
   fill(255);
}
```

En este sencillo ejemplo, un cuadrado es dibujado por donde el ratón vaya.

Si aprietas el ratón el cuadrado se tornará negro.

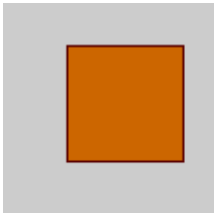


Para más información de ratón, mira la referencia del raton de Processing, y no te olvides de revisar estos ejemplos **Mouse** de Processing.

La entrada del teclado es igual a Flash y Director.

```
void draw( ) {
   if(keyPressed) {
      fill(102, 0, 0);
   } else {
      fill(204, 102, 0);
   }
   rect(30, 20, 55, 55);
}
```

En este simple ejemplo el cuadrado cambia a rojo oscuro si cualquier tecla está siendo presionada. No hay necesidad de redibujar de fondo!



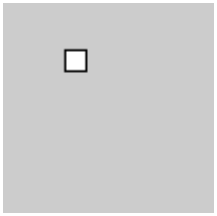
La entrada del teclado también puede ser distribuida como un event handling function.

```
int x = 50;
int y = 50;

void draw( ){
   background(190);
   rect(x,y,10,10);
}

void keyPressed( ){
   if(key=='w'||key=='W'){
      y--;
   }else if(key=='s'||key=='S'){
      y++;
   }else if(key=='a'||key=='A'){
      x--;
   }else if(key=='d'||key=='D'){
      x++;
   }
}
```

En este ejemplo, las teclas del teclado moverán el cuadrado alrededor.



Para mayor información de entrada de teclado, mira la referencia del teclado de Processing, y no olvides de revisar estos ejemplos **Keyboard** de Processing.

Presentación / Exportar

En Flash y Director, hay controles en las teclas e funciones en el menú que pueden seleccionarse para arrancar tu programa, de un modo que usa toda la pantalla y cubre todo los ornamentos presentes en un sistema operativo. La modalidad de pantalla completa es muy útil para la instalación y presentación. En Processing, puedes escoger en el menú **Sketch > Present**, o puedes apretar Ctrl+P (⌘+P en un Mac). Además intenta apretar el botón de play mientras presionas SHIFT. La pantalla completa irá a gris oscuro y verás tu creación en el medio. Para volver a los normal, presionas ESC. Si no funciona hay un botón de "detener" a la esquina final izquierda.

Cualquier programa de Processing puede ser publicado como un applet de Java. Primero asegúrate que el bosquejo (sketch) está salvado, luego escoge **File -> Export**, o presiona Ctrl+E (o presiona el botón de exportar). Verás el área de mensajes de Processing diciendo "Exporting. . ." por un momento y luego dirá "Done Exporting." Para tener los archivos de la web, entra a la **carpeta de bosquejos** de Processing. Busca la carpeta con el nombre de tu bosquejo y ábrela. En esa carpeta verás otra llamada **applet**. Esta carpeta puede ser subida a la web. Recomiendo altamente editar el **index.html** predeterminado que se genera desde Processing. Debes mantener todos los archivos relativos al HTML en esta carpeta applet, ya que están relacionadas de un modo que ninguna otro medio HTML está relacionado.

sketches/
_your_sketch_name_/
applet/
_your_sketch_name_.java
_your_sketch_name_.class
_your_sketch_name_.jar
index.html

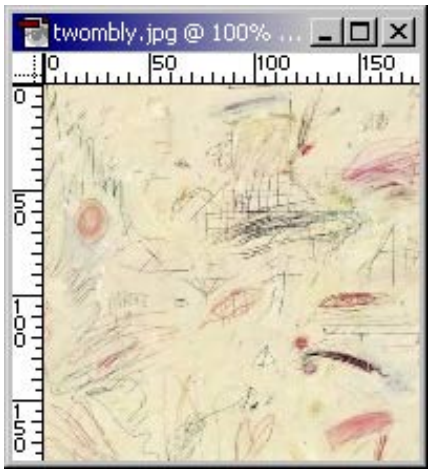
save( ) and saveFrame( )
Si necesitas exportar formatos no-interactivos, es posible hacer archivos .tif de la ventana de Processing usando la función **saveFrame( )** (guardar fotograma). Ubicando este método al final del bucle **draw( )** guardará la imagen en la pantalla. Si **saveFrame( )** es llamado numerosas veces, creará una secuencia de imágenes como la siguiente: screen-0001, screen-0002, screen-0003, etc. Usando **save( )** (guardar) te permitirá elegir un nombre de archivo. Es simple importar estas imágenes al Quicktime y otro programa de vídeo para hacer una documentación animada de un programa de Processing. Aunque Processing tiene incorporado esta fácil función de guardar imágenes, también es posible exportar otros formatos con un poco más de trabajo. Por ejemplo, aquí hay un programa de Processing que exporta a Adobe Illustrator.

Dibujar archivos de imagen

Poner una imagen dentro de un bosquejo de Processing es simple. Processing acepta .gif, .jpg, .tga, y .png (a menos que quieras hacer trabajo extra). Puedes poner la imagen dentro de tu bosquejo usando el sistema de archivos y un par de líneas de código. Primero guarda el bosquejo. Luego puedes encontrar el archivo del bosquejo chequeando en la carpeta de bosquejos definida en las preferencias (**File -> Preferences**). Allí encontrarás una carpeta llamada **sketches** (libro de bosquejos). Allí probablemente encontrarás la carpeta con el mismo nombre que tu bosquejo. Dentro encontrarás otra carpeta llamada **data**. Aquí es la carpeta donde debes ubicar tu archivo de imagen para un Processing más fácil. Aquí , otra manera de decirlo:

sketches/
_your_sketch_name_/
data/
_your_imagefile.gif_

Asumamos que tengo un bosquejo llamado **image_example_1** y quiero dibujar la siguiente imagen llamada **twombly.jpg**, un dibujo de Cy Twombly:

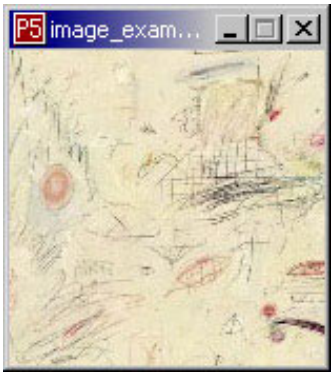


Lo guardaré en una carpeta apropiada:

Sketchbook location / sketches /
image_example_1 **/**
data /
twombly.jpg

Y sé que estoy listo para agregar el código.

```
size(150,150);
PImage b = loadImage("twombly.jpg");
image(b,0,0,150,150);
```



PImage es un objeto que mantendrá tu archivo cargado hasta que lo dibujes. **b** es el nombre que escojí. **image( )** es lo que dibuja la imagen a la pantalla.

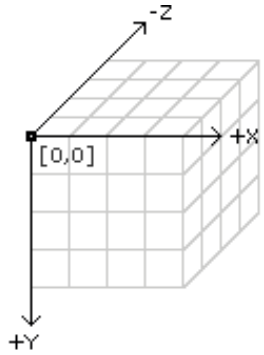
image(PImage, x, y, width, height);

También puedes elegir el omitir el ancho y alto y la imagen se dibujará a escala normal.

Importar archivos no son el único camino, también funcionan los URLs.

Para mayor información de imágenes, consulta el manual de Processing. Aquí está la parte explicando cargando y mostrando imagenes. Construir a través de este conocimiento, también es posible mostrar images sequenciales (video footage). Para más información mira los ejemplos **Image** de Processing.

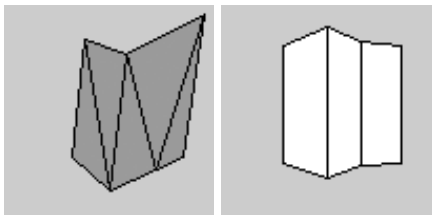
Formas 3D



Ha habido mucha preocupación y quejas por introducir 3D a entornos de 2D. En Flash, muchas herramientas de terceras partes han sido desarrolladas. En Director, un actor gráfico de 3D vectorial se introdujo recientemente. El sistema es tan complejo que muchos han desistido incluso de aprenderlo.

En Processing, 3D sólo significa definir un rendering 3D y agregar una eje z.

```
vertex(x, y, z);
line(x1, y1, z1, x2, y2, z2);
bezierVertex(x, y, z);
curveVertex(x, y, z);
```



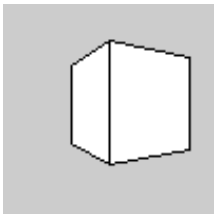
```
box(size);
box(width, height, depth);
sphere(size);
```

Processing actualmente tiene tres **modos de rendering**. Si no especificamos ningún modo en la función size() – como hecho en todos los ejemplos hasta ahora - usamos el el rendering de Java 2D. Para usar funciones de 3D hace falta especificar el rendering de la siguiente manera:

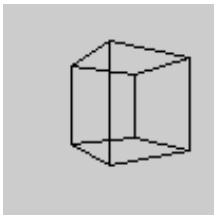
size(100,100,P3D); //P3D define el modo de rendering para 3D

Este modo también da soporte a todas las funciones de 2D y además supone ser más rápida que el rendering 2D. Además existe otro modo de rendering llamado OpenGL. Más información sobre el los modos de rendering encuentras en esta página de Processing.

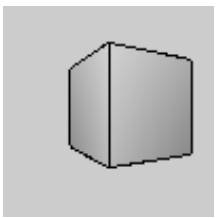
```
size(100,100,P3D);
translate(58, 48, 0);
rotateY(0.5);
box(40);
```



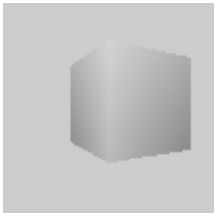
```
size(100,100,P3D);
noFill( );
translate(58, 48, 0);
rotateY(0.5);
box(40);
```



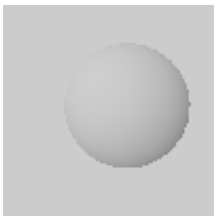
```
size(100,100,P3D);
lights( );
translate(58, 48, 0);
rotateY(0.5);
box(40);
```



```
size(100,100,P3D);
noStroke( );
lights( );
translate(58, 48, 0);
rotateY(0.5);
box(40);
```



```
size(100,100,P3D);
noStroke( );
lights( );
translate(58, 48, 0);
sphere(28);
```



Nótese que **box** (caja) y **sphere** (esfera) no te piden que especifiques coordenadas. En estos ejemplos, es necesario usar **translate** y **rotate**. También hay **scale**, y un par de funciones llamadas **push** y **pop** que te permiten marcar tus traslados en una moda muy organizada. Para mayor información ver la referencia de transformaciones de Processing y los ejemplos **Transform** de Processing. Si no te importan estas transformaciones, siempre hay una solución.

Nótese el uso de **lights( )** y **noLights( )**. Usar luces renderizarán la forma 3D de un modo que sugerirá sombreado. Para más información, mira la referencia de luzes de Processing.

"¿Qué? ¿Esto es para 3D?"

Si crees que esto no es suficiente 3D para permitirte hacer cosas interesantes, entonces mira las cosas que han hecho en la exposición de Processing. Y esto es sólo el comienzo.

Píxeles

El control sobre los píxeles está actualmente lejos de Flash .**SetPixel** y **GetPixel** recién han sido añadidos a Flash y Director (y un conocido artista interactivo, amante de Director ha adoptado el apodo de SetPixel). Estudiantes del ITP van a un curso dictado por Danny Rozin llamado El Mundo – Píxel por Píxel,(The World - Pixel by Pixel), y continúan programando en C porque es la única manera suficientemente rápida para alcanzar las metas conceptuales (Siendo Actionscript, Lingo y MAX las únicas alternativas). Es común entre estos estudiantes de ITP prepararse para el curso de Danny's yendo a un curso C. Trabajar en píxeles de Processing es considerablemente más rápido que en Lingo, y mucho menos complejo. Aunque Java no se compara a la velocidad de C, pronto los estudiantes de Danny podrán explorar la posibilidad usando Processing para acelerar la curva de aprendizaje.

```
get(x, y); // Returns an integer
set(x, y, color);
pixels[index]; // Array containing the display window
```

```
int width = 100;
int height = 100;
PImage b; // declare variable "b" of type PImage
b = loadImage("basel.gif");

image(b, 0, 0);
for (int i=30; i<(width-15); i++) {
   for(int j=20; j<(height-25); j++) {
      color here = get(30, j);
      set(i, j, here);
   }
}
```

Controlando los píxeles, también puedes implementar tus propias rutinas de dibujo. Por ejemplo, aquí hay transparencia. Escribir el resto de las tintas (inks) de Director no será duro. Aquí hay una función de línea de puntos (dotted line).

Para más información de píxeles mira la Preferencia de imagenes de Processing y los ejemplos **Image** de Processing.

Tipografía

El sistema de renderizado de tipografía usa un formato de archivo de carácter específico de Processing. Los creadores del software han incluido un tipo de función de importación de tipo de letra en el menú para ayudarte, y además han dotado al programador de Processing con una gama de tipos de letra para escoger. Pincha aquí para ver los tipos actuales. Estas fuentes están almacenadas como imágenes bitmap. Aquí hay un ejemplo muy sencillo de renderizar texto al bosquejo. Arranca este programa y espera equivocarte.

```
size(200,100);
background(#FFFFFF);
fill(#000000);
PFont f = loadFont("Bodoni-Italic.vlw");
textFont(f, 50);
text("handglove", 14, 60);
```

Os saldrá un error diciendo que no puede encontrar **Bodoni-Italic.vlw**. Esto es porque no has importando aún la fuente a tu carpeta de data. (la sección de imagen de este manual tiene más información de la carpeta de data). Usa la función de creación de fuentes del menu **Tools -> Create Font...** para copiar fisicamente el archivo de fuentes a tu carpeta data. Ahora el programa arrancará bien luego de haber hecho eso.

PFont f = loadFont("Bodoni-Italic.vlw" **);** carga esa fuente dentro de la variable f.

textFont(f, size**);** determina la fuente actual y su tamaño antes de dibujar el texto.

text("handglove", x, y**);** renderiza el texto en el lugar.

Este ejemplo incorpora rotación, y un simple bucle **for**.

```
size(200,100);
noStroke( );
PFont f = loadFont("Univers66.vlw");
textFont(f, 50);
fill(#FFFFFF);
ellipse(-50,-55,150,150);
fill(#CC6600);
for(int i=0;i<20;i++){
   rotateZ(0.2);
   text("dizzy", 90,0);
}
```

Si no te interesa esto porque no te va el rollo de tipografía hay un modo más simplificado. Si necesitas algo como un campo de entrada de texto es útil que escribas tu propio. Muchos widgets de usuarios de interfaces no son difíciles de agregar al proyecto si piensas de ellos como ejercicios pequeños, modulares e interactivos más que algo que el sistema operativo proveerá exclusivamente. Lo que ganas es el control último sobre el diseño. Para más reinvenciones de interfases vernaculares mira los ejemplos **GUI** de Processing.

Mira los fantásticos ejemplos **Typography** de Processing. Processing viene de un grupo de gente preocupada por la estética y la computación. Nuevas formas de tipografía es una de las cosas que por las que este MIT Media Lab research group es famoso.

El futuro

Processing es un trabajo muy nuevo y continuo sujeto a constantes actualizaciones. Una nueva entrega puede ser fácilmente instalada en el lugar de la vieja y puedes mantener tu carpeta de sketches ajustando las preferencias de Processing. Al traducir este documento, Processing está en su **versión BETA 0115**. Processing en sus inicios se llamó Proce55ing así que no te sorprendas si encuentras documentos bajo este nombre. Desde la versión 0085 Processing esta en estado BETA. Todavía tiene errores por corregir y asuntos pendientes antes que salga la versión 1.0, hasta entonces van a mantener este sistema de nombramiento con los cuatro dígitos.

A diferencia con Flash y Director, que están escritos mayormente en C, Processing fue escrito en Java, el mismo lenguaje en el que se le pide a los usuarios que programen. Gracias a la popularidad de Java podemos reutilizar fácilmente programas en este lenguaje. Hay muchas librerías especialmente adaptadas, algunas de ellas están incorporadas en Processing: **Video** para reproducir Quicktimes y incorporar cámaras web, **Serial** para comunicarse con dispositivos conectadas al puerto serial, **Net** para mandar y recibir datos por Internet usando clientes y servidores, además de las librarías **OpenGL**, **PDF**, **DXF** y **JavaScript**. Además hay librerías como **Sonia** para crear sonidos, **proMidi** para mandar y recibir datos midi y muchos otros para comunicarte con servicios web como Google, del.icio.us, flickr entre otros.

Con todo esto en mente, espero encuentren que Processing es útil en suma a las herramientas de Macromedia que estás aprendiendo. Por favor, publica cualquier ejemplo de Processing para que podamos aprender de tu exploración y no te olvides de ser parte de la comunidad online. ¡Feliz creación!

Josh Nimoy se graduó en el Interactive Telecommunications Program de la New York University's Tisch School of the Arts. Crea y exhibe obras digitales relacionadas con interactividad, naturaleza y sistemas tipográficos experimentales. También dicta clases en Design and Media Arts en la UCLA School of Arts and Architecture, especializado en culturas digitales y tecnologías. Josh ha sido investigador visitante en el MIT Media Laboratory en 1999 en el Aesthetics and Computation Group, liderado por John Maeda, donde trabajó con Ben Fry y Casey Reas.
website

Ben Fry se doctoró en el Aesthetics and Computation Group en el MIT Media Laboratory donde investigó métodos de visualización de grandes cantidades de datos dinámicos. Su trabajo actual esta dividido entre consultas y la visualización de datos genéticos para Eric Lander en el Eli & Edyth Broad Institute de MIT & Harvard.
website

Casey Reas se graduó en Media Arts and Sciences en el MIT Media Laboratory donde fue miembro del Aestetics and Computation Group de John Maeda. Después fue profesor fundador del Interaction Design Institute Ivrea en Italia. Actualmente ejerce como profesor asistente en el departamento de Design & Media Arts en UCLA. En sus clases procura entregar bases para la reflexión de ordenadores e internet, como un medio de exploración y definir una estructura para crear una síntesis avanzada de cultura, tecnología y estética.
website

última actualización 30 de Mayo, 2006