Kara Effector 3.2: Effector Book Vol. II [Tomo XXI]

Kara Effector 3.2:

El **Tomo XXI** es otro más dedicado a la librería **shape**, que como ya habrán notado, es la más extensa hasta ahora vista en el **Kara Effector**. El tamaño de esta librería nos da una idea de la importancia de las Shapes en un efecto karaoke, y es por ello que debemos tomarnos un tiempo en ver y conocer a cada una de las funciones y recursos disponibles para poder dominarlas.

Librería Shape [KE]:

shape.multi5(Shape, width, height, Dxy): genera una **shape** conformada por múltiples Shapes para ser usada en las funciones **shape.movevc y shape.movevci**.

Esta función está apoyada en la función **shape.array** para generar una **shape múltiple** que se retornará, con las siguientes características:

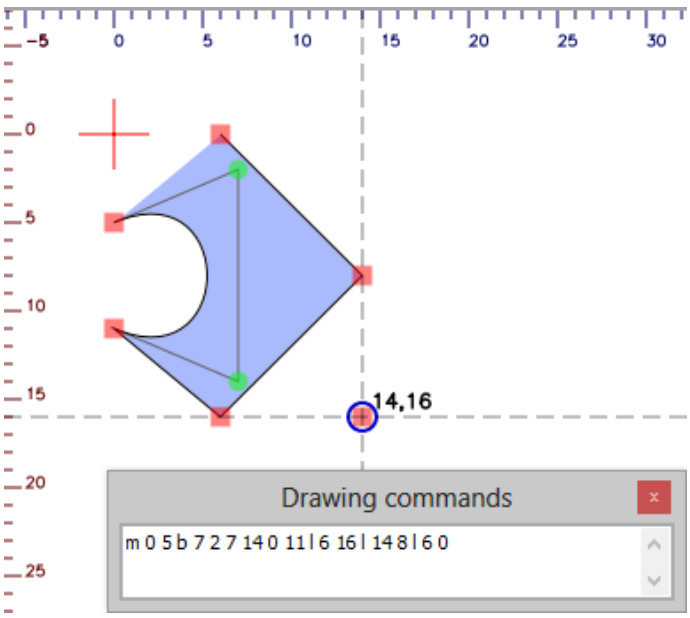
Shape: es la **shape** o la **tabla** de Shapes que se usará como módulo o molde para generar la **shape múltiple**.

width: es el ancho que, a lo mínimo, tendrá la **shape múltiple** generada. Su valor por default es **val_width**, es decir, el ancho del objeto karaoke dependiendo del **Template Type**.

height: es el alto que, a lo mínimo, tendrá la **shape múltiple** generada. El valor que tiene por default es **val_height**, es decir, el alto del objeto karaoke dependiendo del **Template Type**.

Dxy: es la distancia medida en pixeles que separará a una **shape** de la otra. En el caso de ser un número, ésa valor será la distancia horizontal que las separe, y para el caso en que dicho parámetro sea una **tabla** con dos valores numéricos, el primero de ellos indicará la distancia horizontal y el segundo la vertical, ambos en pixeles. Su valor por default es **Dxy** = {0, 0}, o sea, 0 pixeles en cada eje.

Para los próximos ejemplos, usaremos esta simple **shape** que mide 14 px X 16 px:



Y como en los ejemplos de los **Tomos** anteriores, con su código declararemos una variable con el fin de hacer más manejable las funciones en las que la usaremos:

```
Variables:
mi_shape = "m 0 5 b 7 2 7 14 0 11 l 6 16 l 14 8 l 6 0 "
```

Recordemos que el declarar una variable, en este caso, no es obligatorio. Se puede usar la **shape** directamente dentro de la función, siempre y cuando se ponga entre comillas simples o dobles, cualquiera de las dos.

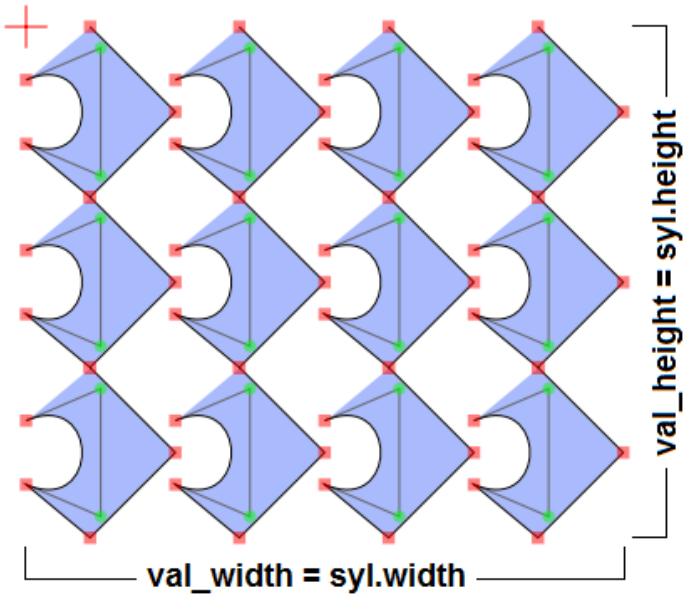
- **Ejemplo 1. Template Type: Syl**

```
Return [fx]:
shape.multi5( mi_shape )
```

- **Shape**: **mi_shape**
- **width**: por default (**syl.width**)
- **height**: por default (**syl.height**)
- **Dxy**: por default (**Dxy** = {0, 0})

Los valores por default de **width** y **height** dependen del **Template Type**, que en este caso es **Syl**.

Vemos en detalle cómo la **shape** ingresada se multiplica en ambos eje hasta alcanzar las medidas **width** y **height**, que en este ejemplo están por default:

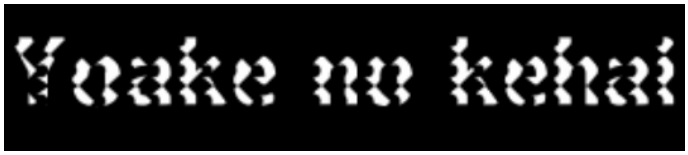


En la imagen anterior la **multi shape** está creada por 12 Shapes individuales (**mi_shape**) y que tanto la distancia horizontal como la vertical que las separa, es de 0 px.

Ahora veamos una pequeña muestra de cómo se vería esta **multi shape** usada en la función **shape.movevc**:

```
Add Tags:   Add Tags Language: Lua
shape.movevc( shape.multi5( mi_shape ) )
```

En aumento:

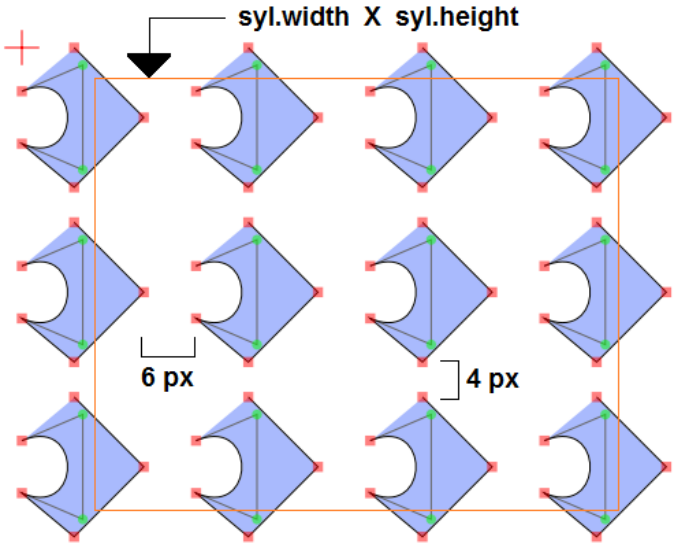


- **Ejemplo 2. Template Type: Syl**
- **width**: **syl.width** + 10
- **height**: **syl.height** + 10
- **Dxy**: {6, 4}

El fin de aumentar las dimensiones de la **multi shape** es para que los clip's generados sean un poco más grande y que dentro de ellos se puedan ver por completo los **objetos karaoke** en el caso en que estén afectados por un **\blur**, de lo contrario, el "brillo" generado por este tag quedaría por fuera de los clip's.

Return [fx]:

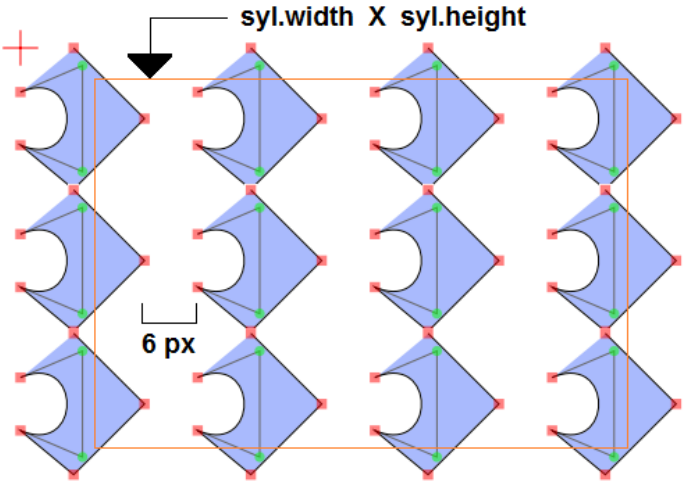
```
shape.multi5( mi_shape, syl.width + 10,
              syl.height + 10, {6, 4} )
```



- **Ejemplo 3. Dxy** = valor numérico:

Return [fx]:

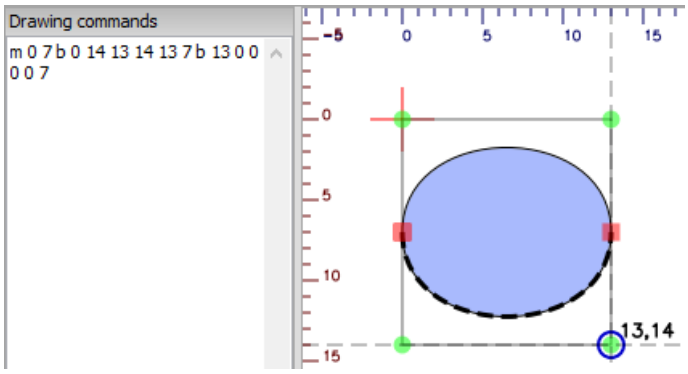
```
shape.multi5( mi_shape, syl.width + 10,
              syl.height + 10, 6 )
```



Como **Dxy** es un valor numérico (6 px), ése será el valor de la distancia que separa las Shapes horizontalmente, y por default, la separación vertical será de 0 px.

- **Ejemplo 4. Shape** = **tabla** de Shapes:

Dibujamos una segunda **shape** que posteriormente estará en la misma **tabla** que la **shape** del ejemplo anterior:

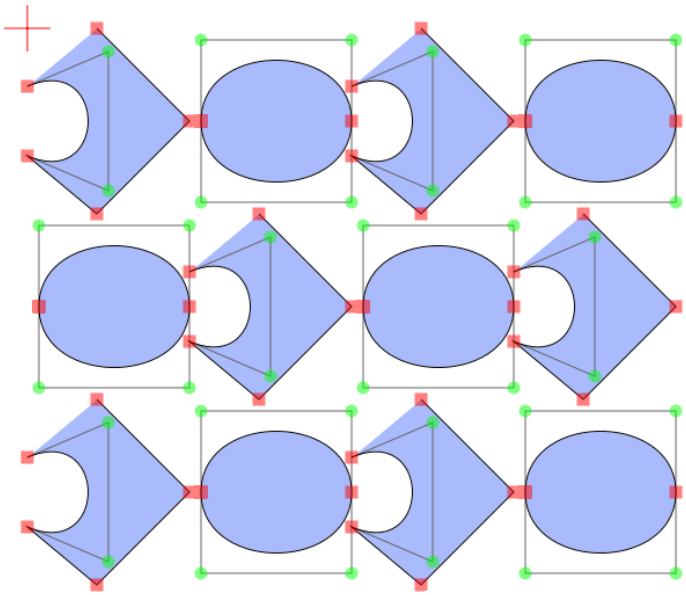


Declaramos la **tabla** con las dos Shapes:

Variables:

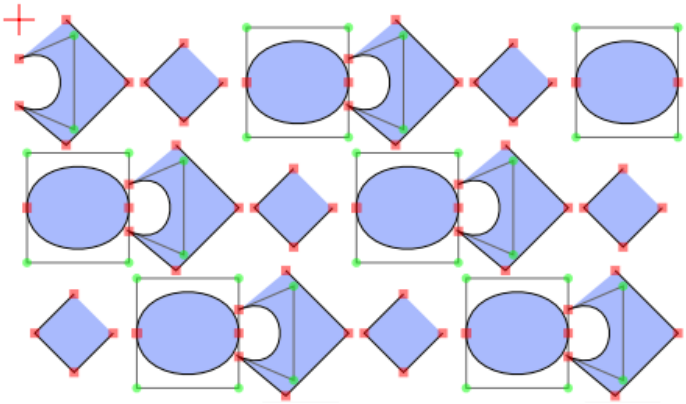
```
mi_shape = {"m 0 5 b 7 2 7 14 0 11 l 6 16 l 14 8 l 6 0 ",
            "m 0 7 b 0 14 13 14 13 7 b 13 0 0 0 0 7 " }
```

En este ejemplo solo hay dos Shapes, pero la cantidad de Shapes en la **tabla** no tiene límite, todo dependerá del efecto que queramos hacer.



Entonces las Shapes de la **tabla mi_shape** se multiplicarán de forma alternada y crearán la **multi shape**.

Un ejemplo de una **multi shape** creada a partir de una **tabla** con tres Shapes:



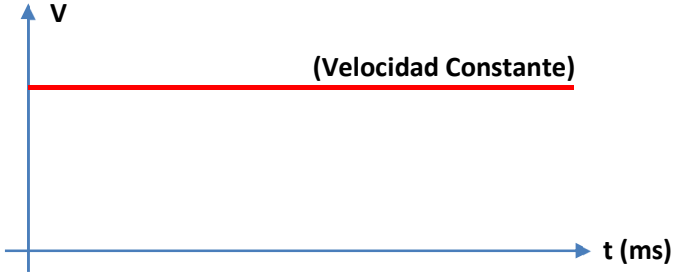
Usada en **shape.movevc**:

Las distancias que separan a las Shapes, tanto vertical como horizontalmente, no necesariamente deben ser valores positivos o cero, también pueden ser valores negativos con el fin de acercar más una **shape** respecto a las otras.

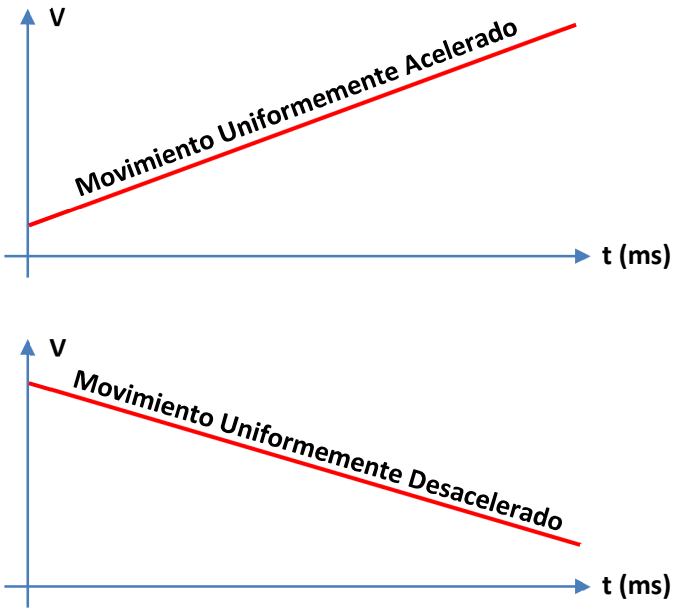
Las siguientes funciones de la **librería shape** están basadas en los conceptos de movimiento y aceleración. Los tags de movimiento en los filtros **VSFilter 2.39** y **VSFilterMod** son:

- \move
- \moves3
- \moves4
- \mover
- \movevc

Y ninguno de los anteriores cinco tags puede acelerar o desacelerar al memento de ejecutar el movimiento. Todos ellos generan un movimiento con velocidad constante:



Es decir que la velocidad es la misma siempre, a medida que transcurre el tiempo. Para los dos casos de **Movimientos Uniformemente Acelerados** tenemos:



Para generar este tipo de movimientos en el **Aegisub** hay varios tipos de combinaciones de tags que lo pueden hacer posible. Ejemplos:

- \org(Px, Py) \t(t1, t2, **aceleración**, \frz<**ángulo**>)
- \t(t1, t2, **aceleración**, \fsp<**distancia horizontal**>)
- \t(t1, t2, **aceleración**, \fsvp<**distancia vertical**>)

Cada una de ellas con cierto nivel de complejidad e incluso de imprecisión. Los siguientes tipos de movimientos que veremos están basados en una **shape** "**invisible**" que hará que el **objeto karaoke** se desplace de un punto a otro con la aceleración que nosotros decidamos:

shape.Lmove(x1, y1, x2, y2, t1, t2, accel): genera una **shape** invisible que mueve al **objeto karaoke** en línea recta desde el punto P_1 = (**x1, y1**) hasta P_2 = (**x2, y2**), desde el tiempo **t1** hasta el tiempo **t2** y con una aceleración **accel**. **Lmove** significa **Movimiento Lineal**.

x1 y **x2** son las coordenadas respecto al eje "**x**" y ambas tienen el mismo valor por default en el caso de no usar estos parámetros: **fx.move_x1**

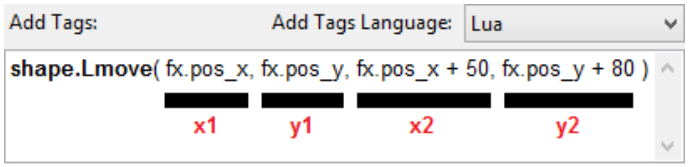
y1 y **y2** son las coordenadas respecto al eje "**y**" y ambas tienen el mismo valor por default en el caso de no usar estos parámetros: **fx.move_y1**

t1 y **t2** son los tiempos de inicio y final del movimiento medidos en milisegundos (**ms**), y sus valores por default son: **fx.movet_i** y **fx.movet_f**

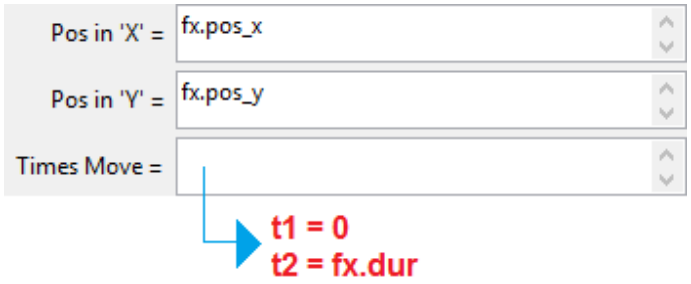
accel es la aceleración del movimiento y su valor por default es 1. Cuando la aceleración es 1, entonces la velocidad es constante. Para valores menores que 1 el movimiento es uniformemente desacelerado y para valores mayores que 1, el movimiento es uniformemente acelerado:

- **accel** < 1: Movimiento Uniforme Desacelerado
- **accel** = 1: Movimiento con Velocidad Constante
- **accel** > 1: Movimiento Uniforme Acelerado

- **Ejemplo 1. Template Type: Word**

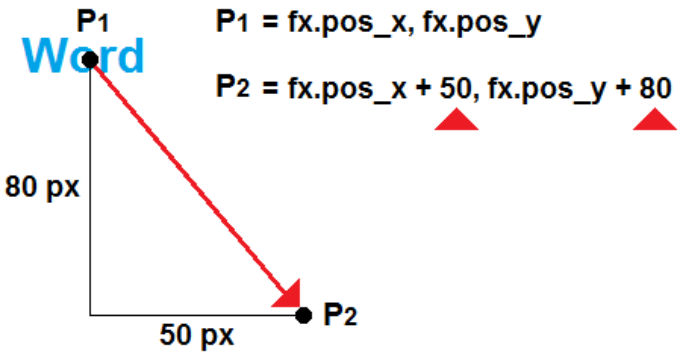


Así, los valores de **t1** y **t2** serán los que tienen por default:



El parámetro **accel** también tendría su valor por default: 1

Entonces la función hará que cada palabra (Word) se mueva desde su posición original hacia un punto ubicado **50 px** a su derecha y **80 px** hacia abajo, en la duración total de la línea de fx y sin ninguna aceleración:

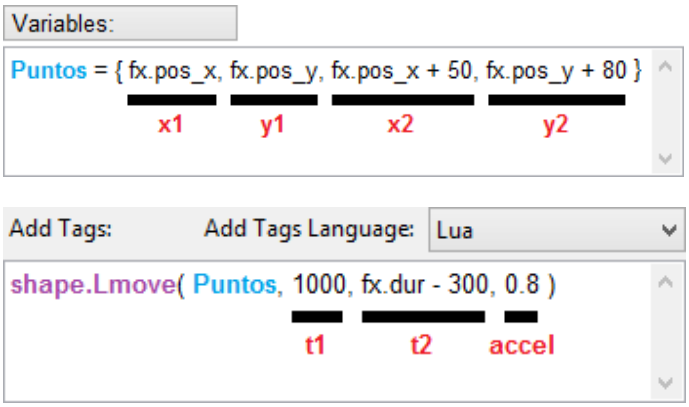


En la siguiente imagen vemos la **shape invisible** que hace que cada palabra (Word) se mueva según los parámetros que hemos ingresado en la función:

0:00:08.16	Romaji	lead-in	Effector [Fx]	✱m 0 0 l 100 100 ✱Kodoku
0:00:08.16	Romaji	lead-in	Effector [Fx]	✱m 0 0 l 100 100 ✱na
0:00:08.16	Romaji	lead-in	Effector [Fx]	✱m 0 0 l 100 100 ✱hoho
0:00:08.16	Romaji	lead-in	Effector [Fx]	✱m 0 0 l 100 100 ✱wo
0:00:08.16	Romaji	lead-in	Effector [Fx]	✱m 0 0 l 100 100 ✱nurasu
0:00:08.16	Romaji	lead-in	Effector [Fx]	✱m 0 0 l 100 100 ✱nurasu
0:00:08.16	Romaji	lead-in	Effector [Fx]	✱m 0 0 l 100 100 ✱kedo

Las coordenadas de los dos puntos ingresados en la función también pueden ser primero declaradas en una **tabla**, en la celda de texto "**Variables**".

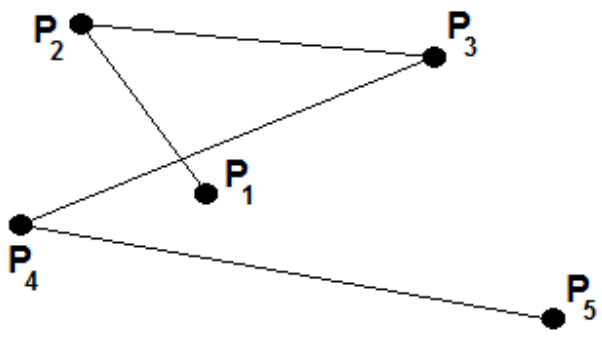
- **Ejemplo 2**:



Cuando se remplazan a los cuatro primeros parámetros de la función, con una **tabla** de puntos, ésta puede tener la cantidad de puntos que queramos:

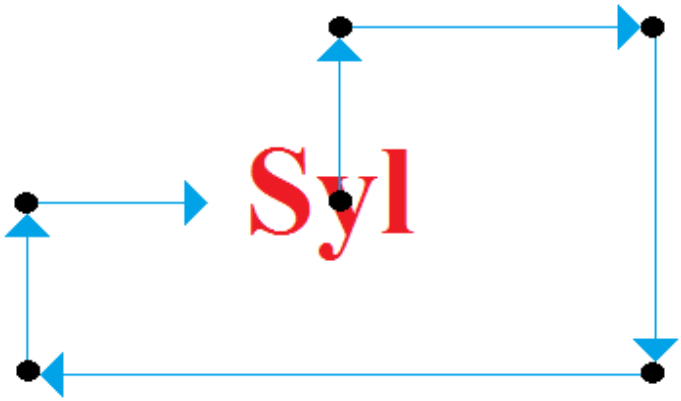
Puntos: { x1, y1, x2, y2, x3, y3,... ...xn, yn }

Lo que generará un movimiento lineal entre cada uno de los puntos ingresados en la **tabla**:



En tiempo de desplazamiento entre un punto y otro es proporcional a la distancia entre ellos.

Si por ejemplo quisiéramos que el objeto karaoke se moviera varias veces alrededor de su centro y luego retorne a su posición original, como en la siguiente imagen:



Lo que debemos hacer es declarar a cada una de las coordenadas de esos puntos en una **tabla**, y luego usar dicha **tabla** en la función **shape.Lmove** con los tiempos por default o asignados por nosotros mismos, al igual que la aceleración del movimiento. Los valores de aceleración que recomendamos son entre 0.3 y 2.5

Entonces, a parte de la ventaja que nos da esta función de poder hacer un movimiento acelerado, también podemos mover al **objeto karaoke** en n cantidad de puntos en un tiempo determinado por nosotros mismos. Esta es la primera de ocho funciones que generan movimiento, y cada una de ellas con características distintas que iremos viendo en los próximos **Tomos**.

De estas ocho funciones mencionadas, cuatro de ellas pertenecen a la **librería shape**, y hasta el momento solo hemos visto la referente a movimientos lineales, pero esto es apenas la punta del iceberg, ya que las que vienen son igual o hasta más atractivas como herramienta para crear nuevos efectos.

Es todo por ahora para el **Tomo XXI**, pero la **librería shape** continuará en el próximo **Tomo**. Intenten poner en práctica todos los ejemplos vistos en este **Tomo** y no olviden descargar la última actualización disponible del **Kara Effector 3.2** y visitarnos en el **Blog Oficial**, lo mismo que en los canales de **YouTube** para descargar los nuevos Efectos o dejar algún comentario, exponer alguna duda o hacer alguna sugerencia. Pueden visitarnos y dejar su comentario en nuestra página de **Facebook**:

- www.karaeffector.blogspot.com
- www.facebook.com/karaeffector
- www.youtube.com/user/victor8607
- www.youtube.com/user/NatsuoDCE
- www.youtube.com/user/karalaura2012