

## Librería Shape [KE] – Parte 2

**shape.trajectory( loop, D\_min, D\_max )**: crea una **shape** con una definida cantidad de segmentos (**loop**), que distan entre sí dependiendo de los parámetros **D\_min** y **D\_max** (Distancia mínima y Distancia máxima).

Cada uno de los segmentos es creado de forma aleatoria y dibujados con una **Curva Bezier** de tal manera que, entre todos los segmentos formen una sola trayectoria fluida con cada una de las curvas.

Los parámetros **D\_min** y **D\_max** son opcionales. En el caso de no ponerlos en la función, éstos tienen sus respectivos valores por default:

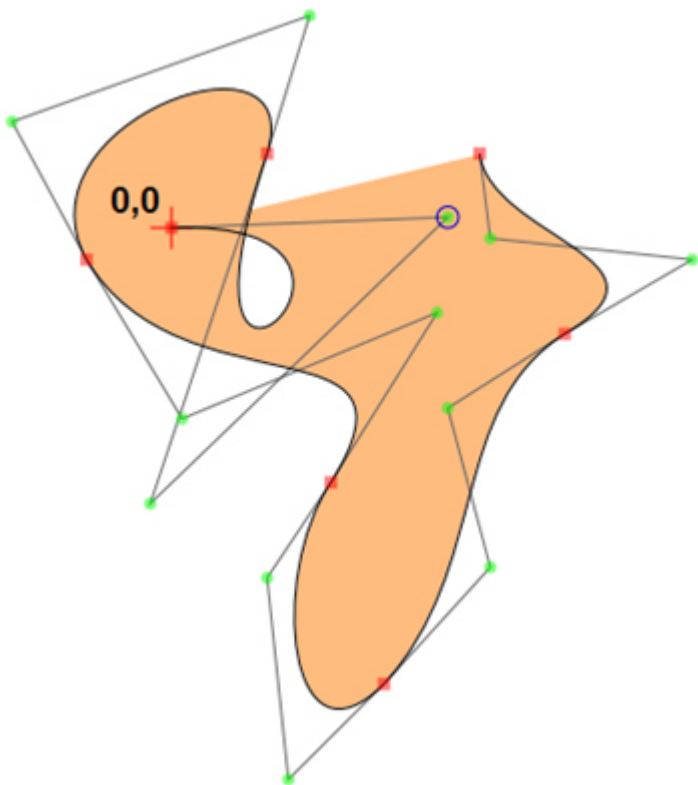
- **D\_min** = 10\*ratio
- **D\_max** = 20\*ratio

Ejemplo:

Return [fx]:

```
shape.trajectory( 5 )
```

Se genera una curva fluida con 5 segmentos de **Curvas Bezier**, y cada una de ellas separada de la otra a una distancia aleatoria entre 10 y 20 px:



Las curvas fluidas que genera esta función se pueden usar como una trayectoria de movimiento dentro de la función **shape.config** en los modos “**move**“, “**move2**” y “**move3**“. Ejemplos:

```
Add Tags:      Add Tags Language: Lua
shape.config( shape.trajetory( 6, 25, 50 ), "move" )
```

```
Add Tags:      Add Tags Language: Lua
shape.config( shape.trajetory( R(4,8) ), "move2" )
```

```
Add Tags:      Add Tags Language: Lua
shape.config( shape.trajetory( 5 ), "move3" )
```

Y para cada uno de los ejemplos anteriores, la función **shape.config** hará que el objeto karaoke de la línea de **fx** se mueva siguiendo como trayectoria a la curva generada por la función **shape.trajetory**.

Las curvas fluidas que genera la función **shape.trajetory** tienen muchas más aplicaciones y opciones de uso. En los próximos artículos, de a poco iremos viendo cómo sacarle el máximo provecho a esta y otras funciones.

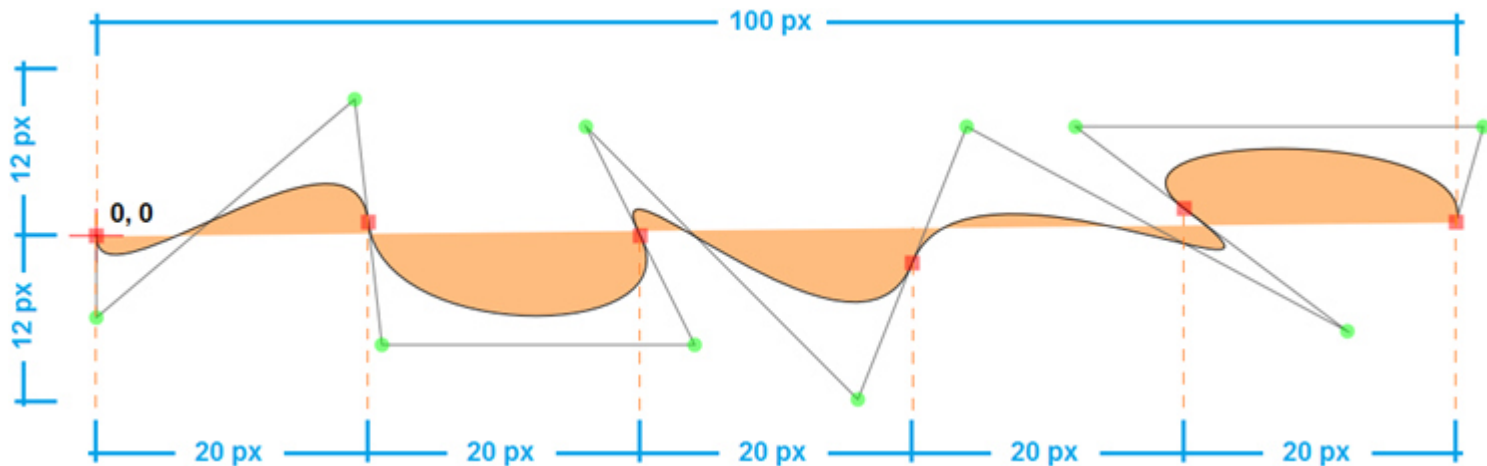
**shape.Ltrajetory( length\_t, length\_c, height\_c )**: es una función similar a **shape.trajetory** con la diferencia de que crea la trayectoria en una sola dirección y con las siguientes especificaciones:

- **length\_t**: es la longitud lineal total de la trayectoria medida en pixeles. Su valor por default equivale a la diferencia entre **xres** y **fx.move\_x1**.
- **length\_c**: es la longitud lineal de cada uno de los segmentos de las **Curvas Bezier** que conforman a la trayectoria. Su valor por default es **xres/4**.
- **height\_c**: equivale a la mitad de la altura promedio máxima que tendrá cada una de las curvas de los segmentos. Su valor por default es **40\*ratio**.

La función crea la trayectoria a partir del punto P = (0,0) y a 0° de dirección, o sea, hacia la derecha de dicho punto.

Ejemplo:

```
Return [fx]:
shape.Ltrajectory( 100, 20, 12 )
```



En la anterior imagen podemos ver una de las trayectorias creadas aleatoriamente, es fluida y sigue las condiciones de los parámetros ingresados en la función:

- Longitud lineal total: **100 px**
- Longitud lineal de los segmentos: **20 px**
- Máximo ascenso y descenso: **12 px**

Las ventajas que tiene cualquier trayectoria creada por una **shape**, es que las podemos modificar usando las funciones de dicha librería. Ejemplos:

- Modificar el ángulo:

```
shape.rotate( shape.Ltrajectory( 100, 20, 12 ), 60 )
```

- Modificar el orden del trazado:

```
shape.reverse( shape.Ltrajectory( 100, 20, 12 ) )
```

- Modificar el “**Ratio**” de alguna de las dimensiones:

```
shape.ratio(shape.Ltrajectory(100, 20, 12),  
1, 0.5 )
```

En fin, las opciones son muchas ya que también podemos combinar dos o más funciones de la librería **shape** para obtener nuevos resultados.

Ejemplo para poner en práctica:

```
Variables:
Trj = shape.reflect( shape.Ltrajectory(100,20,12), "y" )

Add Tags:
Add Tags Language: Lua
shape.config( Trj, "move" )
```

**shape.Ctrajectory( Loop, r\_min, r\_max )**: crea una trayectoria con centro en el punto  $P = (0,0)$  y sin exceder como máximo al radio **r\_max**, ni como mínimo al radio **r\_min**, en donde ambos radios son ingresados en pixeles.

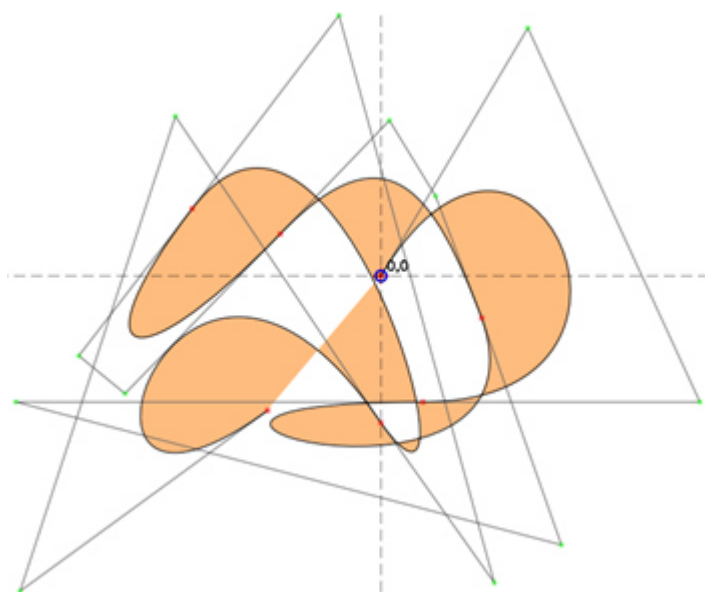
El parámetro **Loop** indica la cantidad de segmentos que tendrá la trayectoria final retornada y su valor por default es **line.duration/720**.

El valor por default de **r\_min** es **xres/40** y el de **r\_max** es **xres/25**, o sea que ambos pueden ser opcionales.

Ejemplo:

```
Return [fx]:
shape.Ctrajectory( 5, 20, 50 )
```

Veamos una de las trayectorias fluidas generadas:



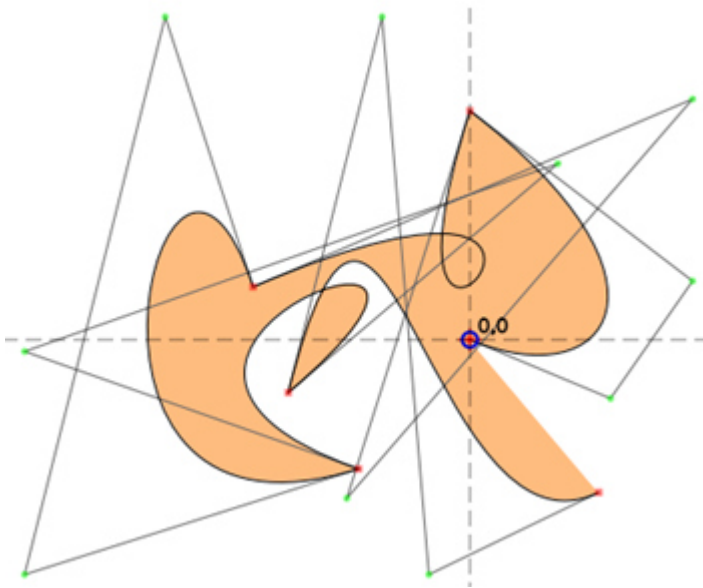
**shape.Rtrajectory( Loop, r\_min, r\_max )**: es una función similar a **shape.Ctrajectory**, pero la trayectoria que genera ya no es fluida sino totalmente aleatoria (random). Esta función crea una trayectoria con centro en el punto  $P = (0,0)$  y sin exceder como máximo al radio **r\_max**, ni como mínimo al radio **r\_min**, con ambos radios son ingresados en pixeles.

El parámetro **Loop** indica la cantidad de segmentos que tendrá la trayectoria final retornada y su valor por default es **line.duration**/720. El valor por default de **r\_min** es **xres**/40 y el de **r\_max** es **xres**/25. Ejemplo:

Return [fx]:

```
shape.Rtrajectory( 5, 30, 40 )
```

Y notamos que la trayectoria esta vez ya no es fluida:



**shape.Strajectory( Loop, Radius )**: esta función es similar a **shape.Rtrajectory**, pero creo la trayectoria con segmentos lineales de forma aleatoria, en vez de usar las **Curvas Bezier** como en las cuatro anteriores funciones.

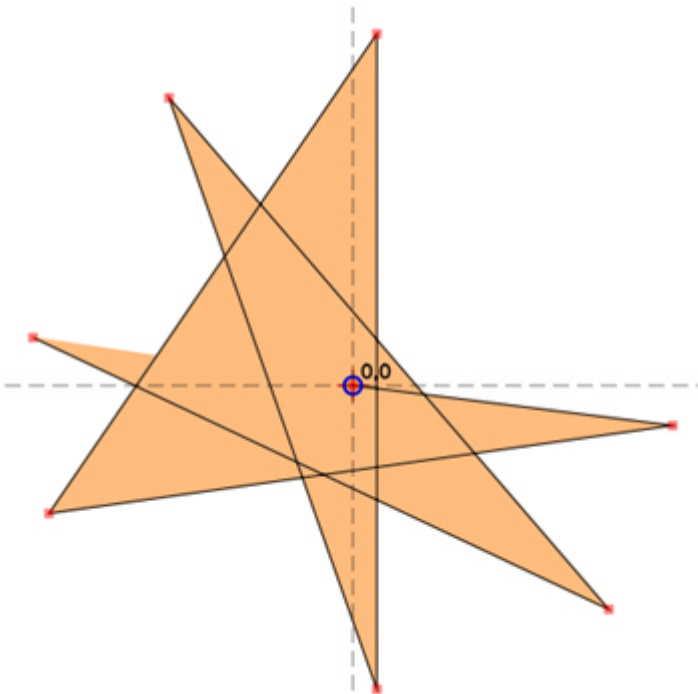
El parámetro **Loop** indica la cantidad total de segmentos lineales que conformarán la trayectoria y su valor por default es **line.duration/820**. El parámetro **Radius** indica la distancia a partir del punto  $P = (0,0)$ , de los extremos de los segmentos. Su valor por default es  **$0.75 * \text{line.height}$** .

Ejemplo:

Return [fx]:

```
shape.Strajectory( 7, 45 )
```

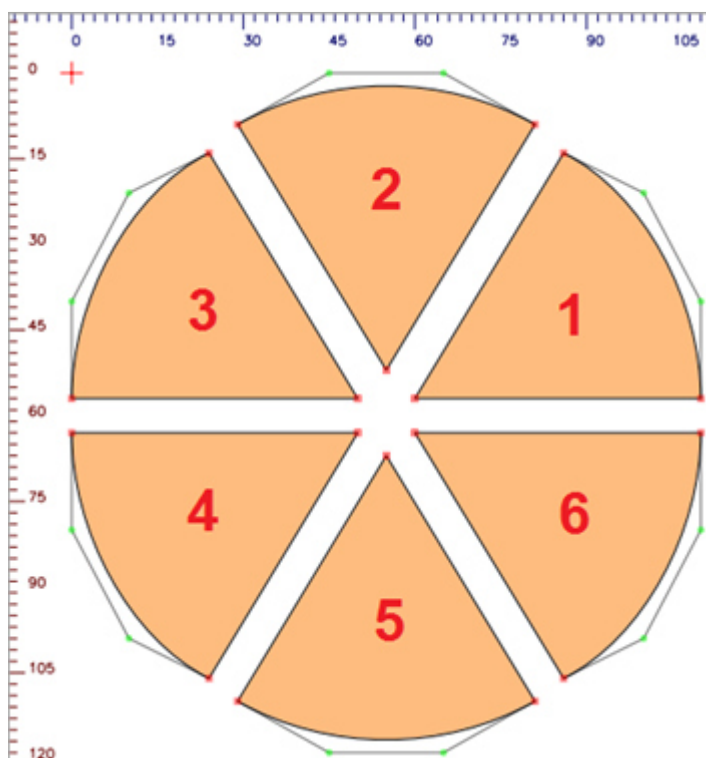
En la siguiente gráfica vemos cómo la trayectoria está conformada por siete segmentos rectos:



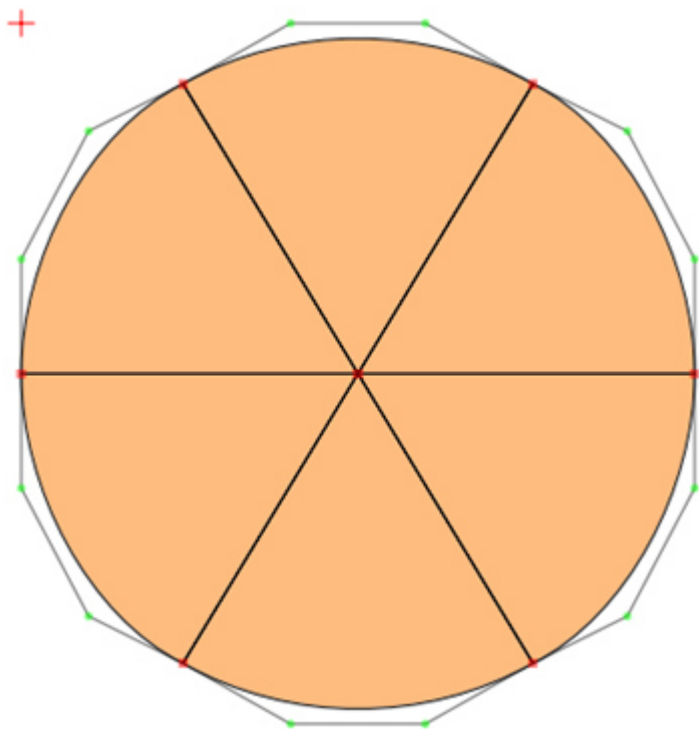
**shape.movevc( Shape, Rtn, width, height, x, y, Dx, Dy, t\_i, t\_f )**: es similar a la función **tag.movevc**, pero con la diferencia que a esta función se le ingresa una **shape** conformada por dos o más Shapes para ser usadas dentro de un tag **\clip**, que posteriormente serán manipuladas de forma individual por el tag **\movevc**.

- **Shape**: es la shape que está conformada por dos o más Shapes individuales.
- **Rtn**: este parámetro decide qué va a retornar la función y tiene tres opciones:
  - “**shape**”: retorna individualmente a cada una de la Shapes que conforma a la shape ingresada en la función.
  - “**loops**”: retorna en número equivalente a la cantidad de Shapes que conforman a la shape ingresada.
  - “**tag**”: retorna una serie de tags equivalente al efecto generado por la función.
- **width**: es el ancho total que abarcarán todos los clip’s que genere la función y su valor por default es **val\_width** (ancho del objeto karaoke: syl, word, char, line y demás. Depende del **Template Type**).
- **height**: es el alto total que abarcarán todos los clip’s que genere la función y su valor por default es **val\_height** (alto del objeto karaoke: syl, word, char, line y demás. Depende del **Template Type**).
- **x, y**: son las coordenadas que ubicarán el centro de la **shape** ingresada respecto al vídeo. Sus valores por default son:
  - **x** = **fx.move\_x1**
  - **y** = **fx.move\_y1**.
- **Dx, Dy**: son las distancias medida en pixeles en las que se moverán cada uno de los clip’s generados, respecto a ambos ejes. Sus valores por default son:
  - **Dx** = **fx.move\_x2** – **fx.move\_x1**
  - **Dy** = **fx.move\_y2** – **fx.move\_y1**
- **t\_i, t\_f**: son los tiempos de inicio y final de los movimientos de cada uno de los clip’s. sus valores por default son:
  - **t\_i** = **fx.movement\_i**
  - **t\_f** = **fx.movement\_f**

Para el ejemplo, usaré el siguiente grupo de Shapes:



Son seis Shapes individuales en total, pero las he juntado de manera que aparenten ser una sola:



A continuación, usaremos el código de la anterior **shape** del **ASSDraw3**, para declarar una variable en la celda de texto “**Variables**“:

Variables:

```
Shapes = "m 50 52 | 100 52 b 100 35 90 16 76 9 | 50 52  
m 50 52 | 76 9 b 60 0 40 0 24 9 | 50 52 m 50 52 | 24 9 b  
10 16 0 35 0 52 | 50 52 m 50 52 | 0 52 b 0 69 10 88 24  
95 | 50 52 m 50 52 | 24 95 b 40 104 60 104 76 95 | 50  
52 m 50 52 | 76 95 b 90 88 100 69 100 52 | 50 52 "
```

He resaltado las letras “**m**” del código de la **shape** con el fin de poder identificar fácilmente a las seis Shapes individuales que conforman a toda la **shape**.

Para el ejemplo usaré un **Template Type: Syl** y la plantilla de efectos: **[001] ABC Template Hilight Syl**. Y en **Add Tags** llamaremos a la función usando casi todos sus parámetros por default, ya que todos ellos hacen referencia a valores ya ingresados, como las posiciones y los tiempos:

Add Tags:

Add Tags Language:

Lua

```
shape.movevc( Shapes, "tag" )
```



Entonces la función generará automáticamente un loop equivalente a la cantidad total de Shapes individuales que conforman a la **shape** ingresada (o sea 6) y generará un clip por cada una de esas Shapes con las siguientes posiciones:



Es decir, como es un **Template Type: Syl**, generará seis líneas de fx por cada sílaba de cada línea a la que se le aplique el efecto:

24	0	0:00:45.92	0:00:54.56	English				Dos corazones
25	1	0:00:02.43	0:00:02.60	Romaji	hi-light	Effector [Fx]	*Ko	
26	1	0:00:02.43	0:00:02.60	Romaji	hi-light	Effector [Fx]	*Ko	
27	1	0:00:02.43	0:00:02.60	Romaji	hi-light	Effector [Fx]	*Ko	
28	1	0:00:02.43	0:00:02.60	Romaji	hi-light	Effector [Fx]	*Ko	
29	1	0:00:02.43	0:00:02.60	Romaji	hi-light	Effector [Fx]	*Ko	
30	1	0:00:02.43	0:00:02.60	Romaji	hi-light	Effector [Fx]	*Ko	
31	1	0:00:02.60	0:00:02.76	Romaji	hi-light	Effector [Fx]	*do	

Pero como no le hemos ordenado ningún movimiento ni tampoco le hemos dado ubicaciones distintas a las que ya tiene por default, en el vídeo veremos a cada sílaba de forma normal:



Pero si manualmente eliminamos a una de esas seis líneas, ya se verá la diferencia, ya que la sílaba que se ve en pantalla está formada por seis partes de la misma:



Al ampliar un segmento de los que conforman la sílaba, veremos algo como esto:



O sea que cada clip usa a una única **shape** para hacer visible una sección de la sílaba y el resto quedará invisible.

Ahora, para darle movimiento a los clip's, simplemente le damos movimiento al objeto karaoke, en este caso, a las sílaba:

Pos in 'X' =	<input type="text" value="fx.pos_x, fx.pos_x + 50"/>	^ v
Pos in 'Y' =	<input type="text" value="fx.pos_y, fx.pos_x - 32"/>	^ v
Times Move =	<input type="text"/>	^ v

Pero no tendría mucho sentido hacerlo de esta forma, ya que todos los clip's se moverán exactamente a la misma dirección y en el mismo tiempo. Entonces si queremos que los clip's se muevan a lugares distintos, debemos usar valores aleatorios (random) para dar la ilusión de que la sílaba se fragmenta en pedazos:

Pos in 'X' =	<input type="text" value="fx.pos_x, fx.pos_x + R(-40,40)"/>	^ v
Pos in 'Y' =	<input type="text" value="fx.pos_y, fx.pos_y + R(-50,50)"/>	^ v
Times Move =	<input type="text"/>	^ v

Así cada trozo se moverá a lugares distintos respecto a los otros, aunque aún lo seguirán haciendo al mismo tiempo, ya que los tiempos del movimiento se dejaron por default:



Siempre que queramos que los clip's se muevan al mismo lugar a dónde lo hará el objeto karaoke y al mismo tiempo que él, entonces lo que debemos hacer es usar la función solo con los dos primeros parámetros, como lo hicimos en el ejemplo anterior.

Es momento para recordarles la gran cantidad de recursos de la **Memoria RAM** que consumen los tags **\clip**, **\iclip** y **\movec**. Lo recomendable es no exceder un **loop** entre 20 o 25 en el efecto, es decir que la **shape** ingresada en la función esté conformada por, a lo máximo, 25 Shapes individuales.

Exceder las 25 Shapes individuales en la **shape** ingresada en la función hará que la computadora empiece a ponerse lenta a medida que se reproduce el efecto, también hará mucho más lento el proceso en encodeo.

**shape.movevci( Shape, Rtn, width, height, x, y, Dx, Dy, t\_i, t\_f )**: es similar a la función **shape.movevci**, pero con la diferencia que retorna iclip's en lugar de clip's como la anterior función.

---

**shape.multi1( Size, Px )**: crea una **shape** formada de múltiples Shapes cuadradas concéntricas para ser usada en las funciones **shape.movevc** y **shape.movevci**.

El parámetro **Size** indica las dimensiones máximas del cuadrado de mayor tamaño y su valor por default es equivalente a la mayor dimensión entre **val\_width** y **val\_height** del objeto karaoke, es decir:

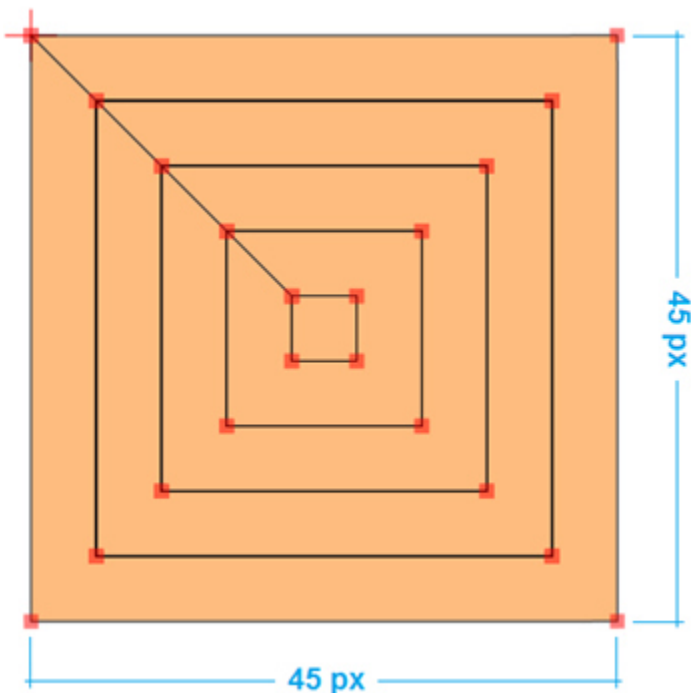
- **Size = math.max( val\_width, val\_height )**

El parámetro **Px** equivale al ancho en pixeles de cada una de las Shapes cuadradas concéntricas que conforman a la **shape** que será retornada. Su valor por default es **4\*ratio**.

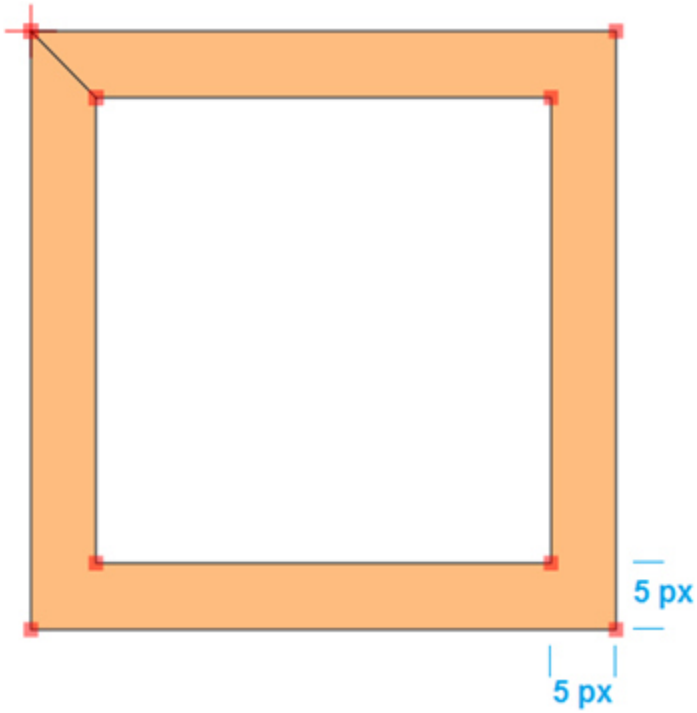
Ejemplo:

```
Return [fx]:
shape.multi1( 45, 5 )
```

Se generará la siguiente **shape**:



Vemos que las dimensiones de la **shape** son 45 X 45 px, y el ancho de cada una de las Shapes que la conforman es de 5 px:



Otra forma de acceder al valor por default del parámetro **Size** es escribiendo la palabra “**default**” en él, Ejemplos:

- **shape.multi1( “default”, 8 )**
- **shape.multi1( “default”, 2 )**

Entonces, para usar la función **shape.multi1** dentro de la función **shape.movevc** es recomendable usar el valor por default del parámetro **Size**, para que el objeto karaoke sea completamente visible en los clip’s generados. Ejemplo:

```
Add Tags:      Add Tags Language: Lua
shape.movevc( shape.multi1( "default", 4 ), "tag" )
```

Lo que generará los siguientes clip’s:



La cantidad de clip’s generados por la función dependerá de las dimensiones del objeto karaoke; entre más grande sea éste, mayor será la cantidad de clip’s generados para poder abarcar toda la dimensión del objeto karaoke.

En algunos casos, el valor por default de **Size** no abarca completamente a las dimensiones del objeto karaoke, ya sea por un borde muy grueso, una sombra muy grande, un **blur** muy marcado u otros factores más; para estos casos, debemos sumar un valor extra que compense el tamaño total de la **shape** generada. Ejemplo:

```
Variables:
new_Size = math.max( val_width, val_height ) + 12
```

O sea que sumamos 12 px para compensar el tamaño de la **shape** generada. Luego usamos esta variable dentro de la función:

```
Add Tags:      Add Tags Language:  Lua
shape.movevc( shape.multi1( new_Size, 8 ), "tag" )
```

Usando un poco de imaginación, usamos las funciones de la librería **shape** para lograr nuevos resultados. Ejemplo:

```
Variables:
Shapes = shape.rotate( shape.multi1( ), 45 )
Add Tags:      Add Tags Language:  Lua
shape.movevc( Shapes, "tag" )
```



**shape.multi2( width, height, Dxy )**: es una función similar a **shape.multi1**, ya que también genera una **shape** conformada por múltiples Shapes para ser usada en las funciones **shape.movevc** y **shape.movevc**.

Esta función genera Shapes diagonales con un ancho de **Dxy**, dentro del rectángulo de medidas **width** X **height**.

- **width**: ancho total de la shape generada.
- **height**: alto total de la shape generada.
- **Dxy**: ancho de las Shapes diagonales

El valor por default del parámetro **width** es **val\_width**, el de **height** es **val\_height**, y el de **Dxy** es **6\*ratio**:

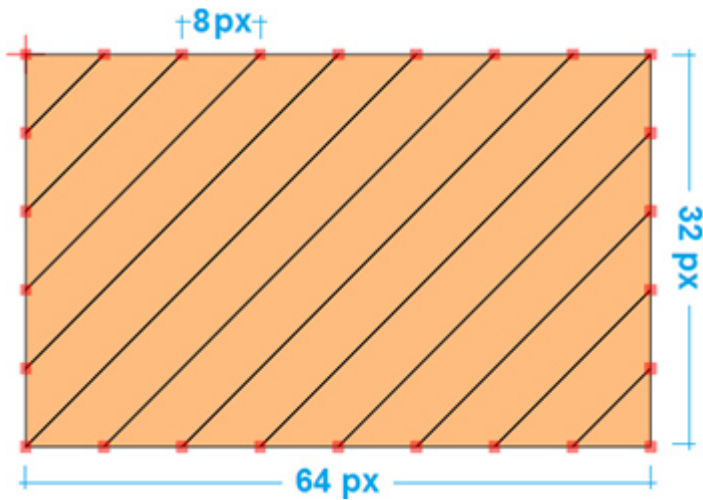
- **width** = **val\_width**
- **height** = **val\_height**
- **Dxy** = **6\*ratio**

Ejemplo:

```
Return [fx]:
shape.multi2( 64, 32, 8 )
```

Ancho   Alto   Grosor

Lo que generará siguiente grupo de Shapes:



Para el próximo ejemplo usaré un **Template Type: Word** y los siguientes parámetros en la función:

```
Variables:
Shapes = shape.multi2( word.width, word.height, 5 )
```

Y luego de declarar la anterior variable, la usamos en la función **shape.movevc**:

```
Add Tags:   Add Tags Language: Lua
shape.movevc( Shapes, "tag" )
```

Lo que generará la siguiente serie de clip's:



Otra variante que podemos usar es:

```
Add Tags:      Add Tags Language:  Lua
shape.movevc( shape.reflect( Shapes, "y" ), "tag" )
```

Lo que invertiría el sentido de las diagonales:



El **loop** total de los clip's generados solo dependerá de las dimensiones del rectángulo, así como del ancho que le demos en la función a las diagonales:

Con el uso de más recursos de la librería **shape** se pueden lograr resultados más complejos como este:



---

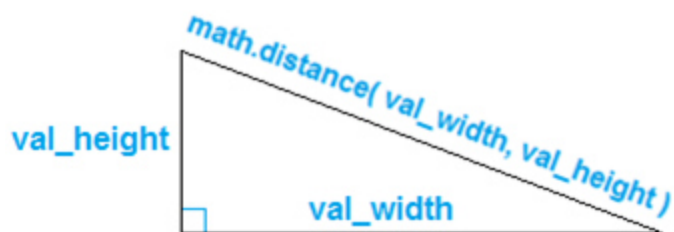
**shape.multi3( Size, Dxy, Shape ):** retorna a una **shape** compuesta por Shapes concéntricas respecto a la **shape** ingresada (**Shape**) con un ancho de **Dxy**, y de un tamaño total **Size**.

- **Size:** tamaño total de la **shape** generada.
- **Dxy:** espesor de las Shapes concéntricas.
- **Shape:**  
**shape** ingresada.



El valor por default del parámetro **Size** equivale a la medida de la diagonal de un rectángulo de dimensiones **val\_width**, **val\_height**. El valor por default de **Dxy** es **5\*ratio** y el de **Shape** es **shape.circle**:

- **Size** = **math.distance( val\_width, val\_height )**



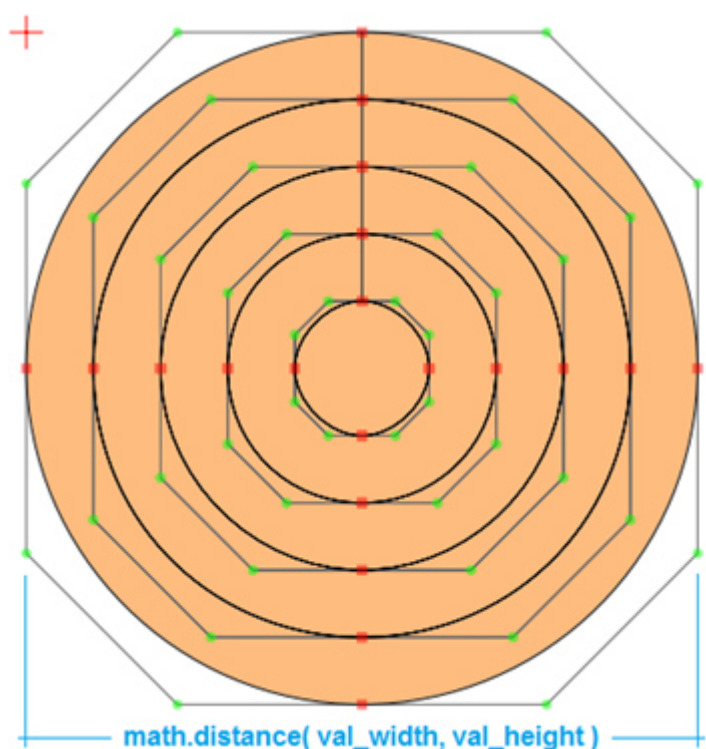
También se puede acceder a este valor si ponemos la palabra “**default**” en este parámetro.

- **Dxy** = **5\*ratio**
- **Shape** = **shape.circle**

Ejemplo 1:

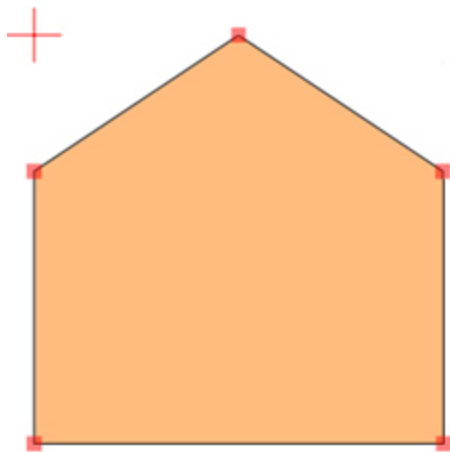
```
Return [fx]:
shape.multi3( "default", 8 )
```

Lo que generará círculos concéntricos de 8 px de espesor cada uno:



Ejemplo 2:

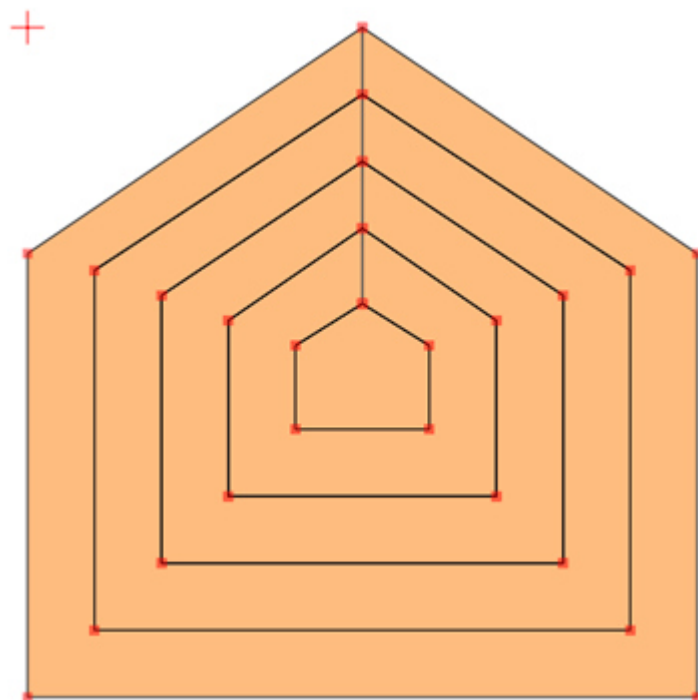
Para este ejemplo, usaremos la siguiente **shape**:



Con un espesor de 6 px:

Return [fx]:

```
shape.multi3( "default", 6, "m 15 0 | 0 10 | 0 30 | 30 30  
| 30 10 | 15 0 " )
```



O sea que las opciones son infinitas, ya que pueden duplicar de forma concéntrica a cualquier **shape** que se imagines. Como ya sabemos, el **loop** total depende de los valores ingresados en la función al igual que el tamaño del objeto karaoke. Una vez decididos los parámetros en la función, ya podemos usar la **shape** generada dentro de las funciones **shape.movevc** y/o **shape.movevci**.

---

**shape.multi4( Size, loop1, loop2 )**: esta función retorna un **Arreglo Radial** de tamaño total **Size** y con una cantidad de repeticiones, en principio determinada por el parámetro **loop1**.

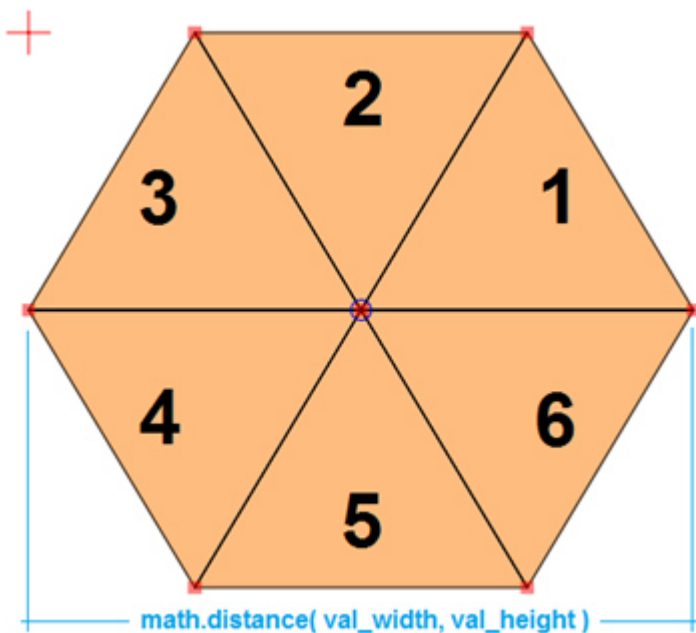
Esta función solo está disponible para la versión **3.2.7** o superior del **Kara Effector**.

Sus valores por default son:

- **Size:**  
**math.distance( val\_width, val\_height )**
- **loop1:** 6
- **loop2:** 1
- **Ejemplo 1.** Todos los parámetros por default:

Return [fx]:

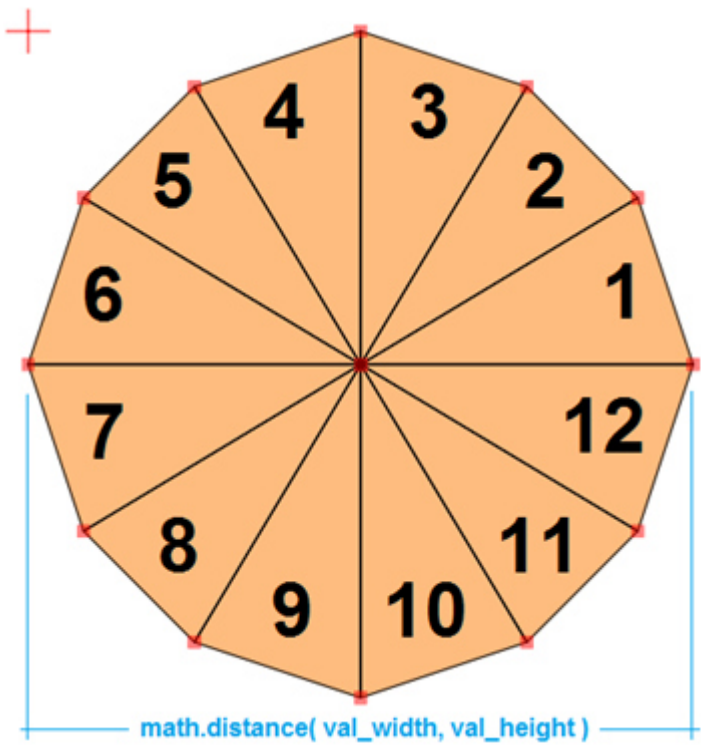
```
shape.multi4( )
```



- **Ejemplo 2.** Modificar a **loop1**:

Return [fx]:

```
shape.multi4( "default", 12 )
```



Del Ejemplo 2 notamos cómo el ancho total de la **shape** es el asignado por default:

**math.distance( val\_width, val\_height )**

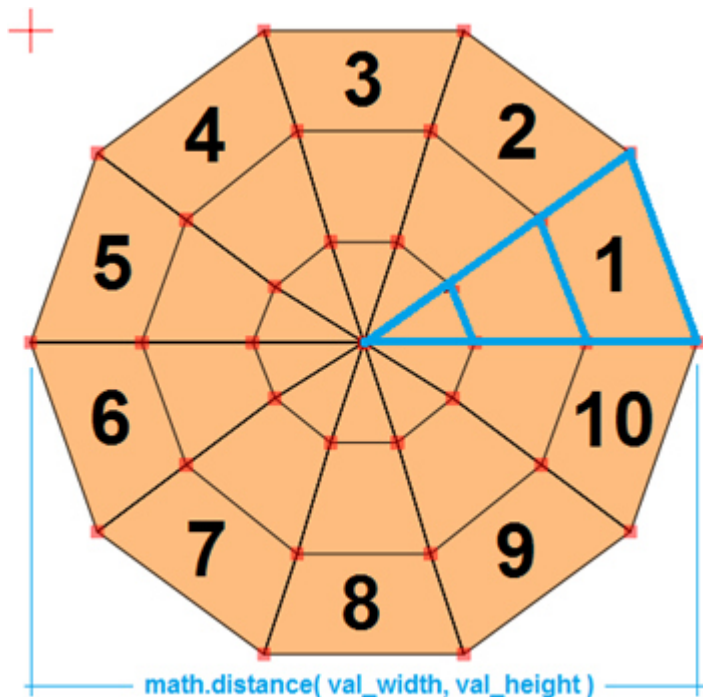
Observamos también que el **Arreglo Radial** está compuesto por doce Shapes individuales, y como **loop2** no está, toma su valor por default que es 1, así que el **loop** total sería de  $12 \times 1 = 12$ .

- **Ejemplo 3.** Modificar el parámetro **loop2**:

Return [fx]:

```
shape.multi4( "default", 10, 3 )
```

Ahora el parámetro **loop2** es 3, lo que hace que el **Arreglo Radial** se multiplique tres veces:



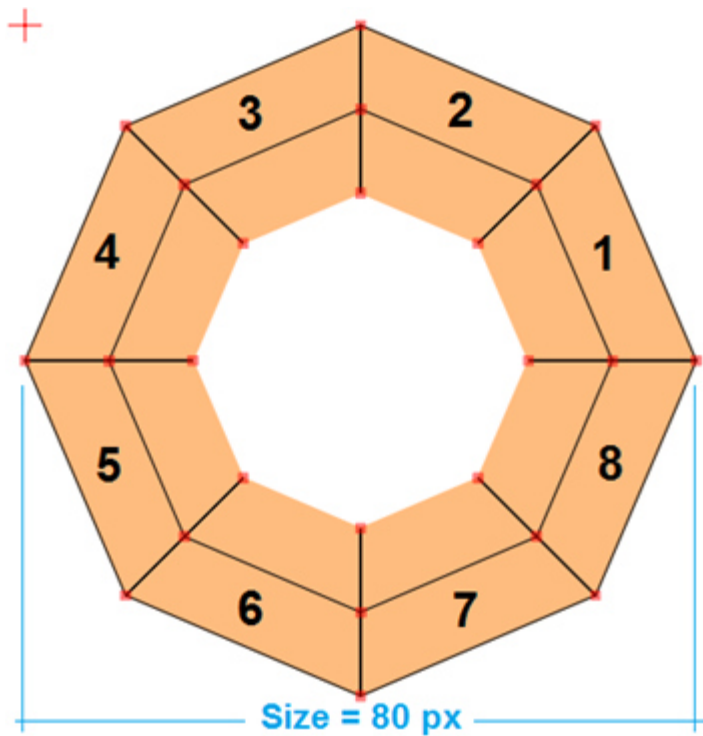
Como **loop1** es 10 en este ejemplo, entonces cada **Arreglo Radial** está compuesto por 10 Shapes individuales, que multiplicados por 3, de **loop2**, hace que el **loop** total sea de  $10 \times 3 = 30$ .

- **Ejemplo 4.** Usar un cuarto parámetro en la función que condiciona los anteriores resultados:

Return [fx]:

```
shape.multi4( 80, 8, 4, 2 )
```

- **Size** = 80 px
- **loop1** = 8
- **loop2** = 4
- Repeticiones a tener en cuenta: 2



Entonces, de las cuatro repeticiones del **Arreglo Radial**, solo se tendrán en cuenta las dos primeras, gracias al cuarto parámetro de la función **shape.multi4**.

Ya teniendo una mejor idea de cómo usar esta función, se nos hace un poco más simple poderla emplear dentro de la función **shape.movevc**. Ejemplo:

```
Variables:  
Shape = shape.multi4( "default", 6, 2 )  
Add Tags: Add Tags Language: Lua  
shape.movevc( Shape, "tag" )
```

O sino, de forma directa:

```
Add Tags: Add Tags Language: Lua  
shape.movevc( shape.multi4( "default", 6, 2 ), "tag" )
```

Crea un **Arreglo Radial** de 6 Shapes individuales y dicho Arreglo se repite 2 veces, para un total de 12 clip's:



Hecho este ejemplo, ya se podrán imaginar la gran cantidad de opciones que nos ofrece esta función. Todo es cuestión de ensayar y experimentar con las diversas combinaciones hasta que se familiaricen con esta y las demás funciones.

---

**shape.múltiple5( 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:

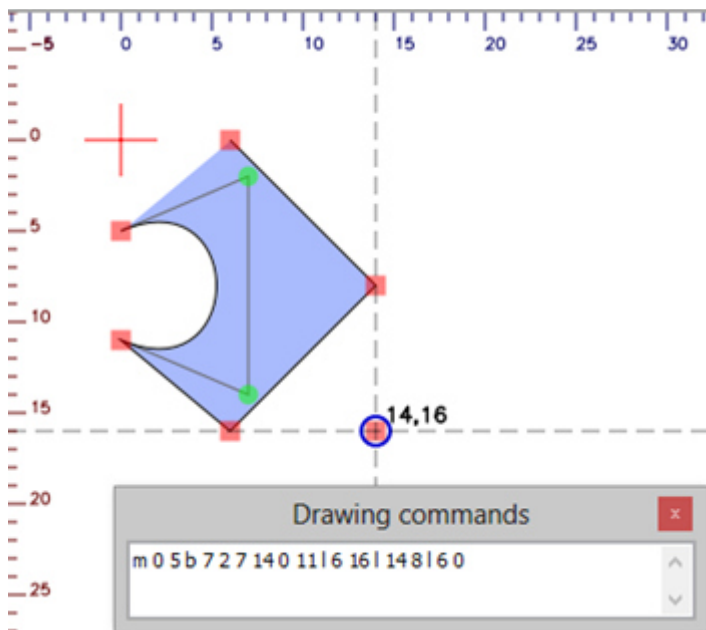
**Shape:** es la **shape** o la **tabla** de Shapes que se usará como módulo o molde para generar la **shape múltiple**.

**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 6 16 14 8 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**

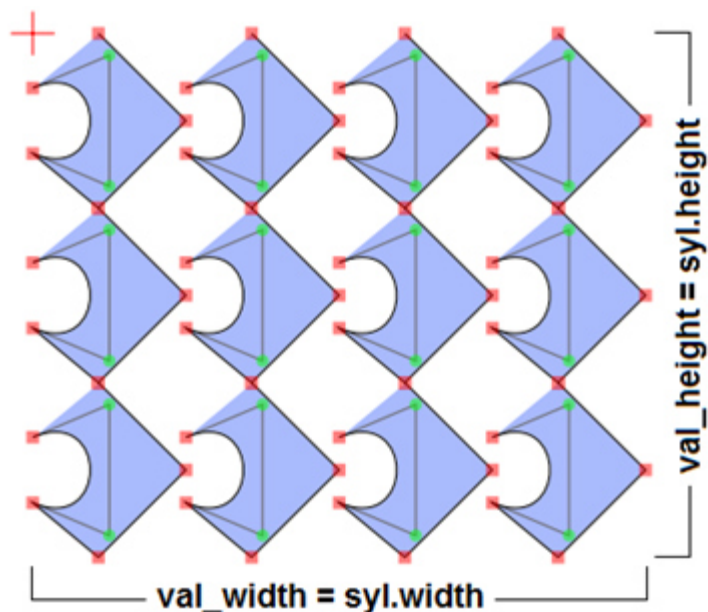
```
Return [fx]:  
shape.multi5( mi_shape )
```

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



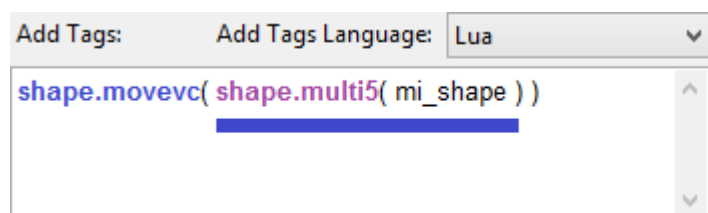
Los valores por default de **width** y **height** dependen del **Template Type**, que en este caso es **Syl**.

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:

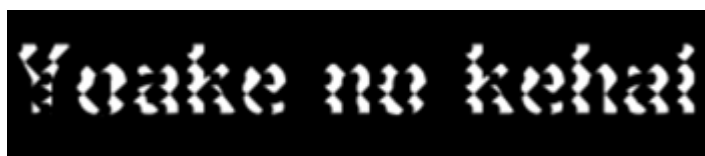


En la imagen anterior la **multi shape** está creada por 12 Shapes individuales (**mi\_shape**) y que tanto la distancia horizontal como la vertical que las separa, es de 0 px.

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



En aumento:

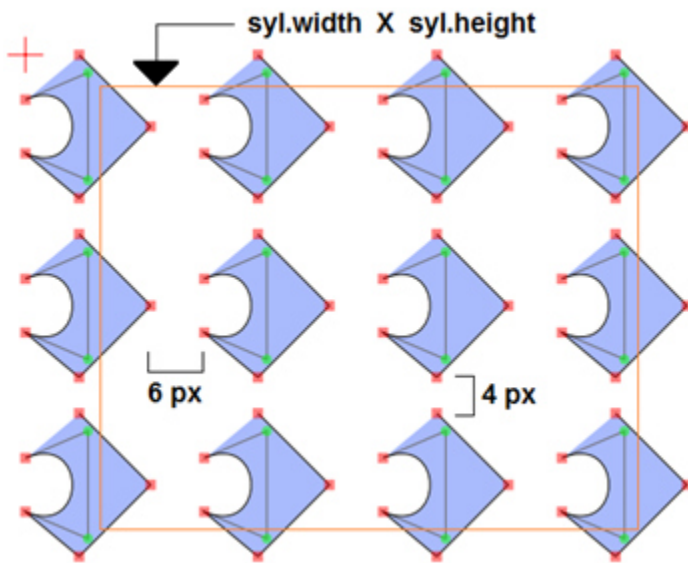


- Ejemplo 2. Template Type: Syl
- width: syl.width + 10
- height: syl.height + 10
- Dxy: {6, 4}

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]:

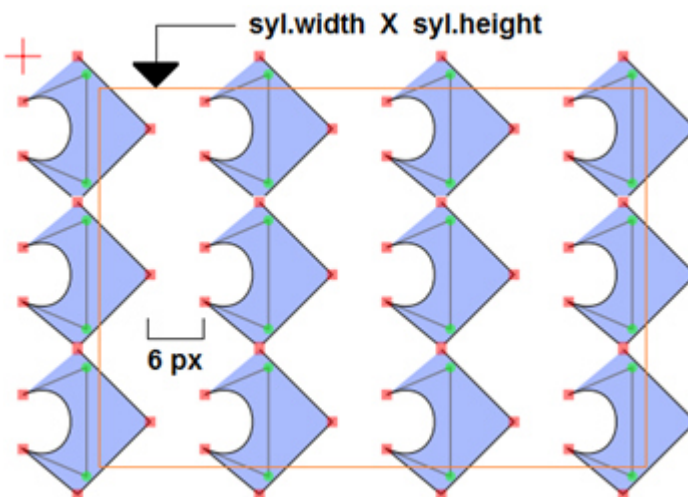
```
shape.multi5( mi_shape, syl.width + 10,  
              syl.height + 10, {6, 4} )
```



- Ejemplo 3. Dxy = valor numérico:

Return [fx]:

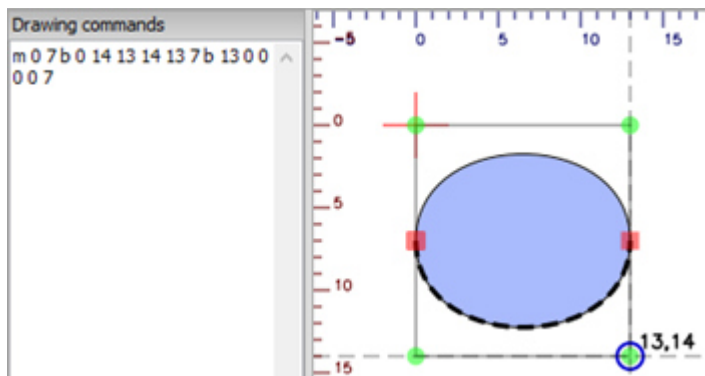
```
shape.multi5( mi_shape, syl.width + 10,  
              syl.height + 10, 6 )
```



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:**

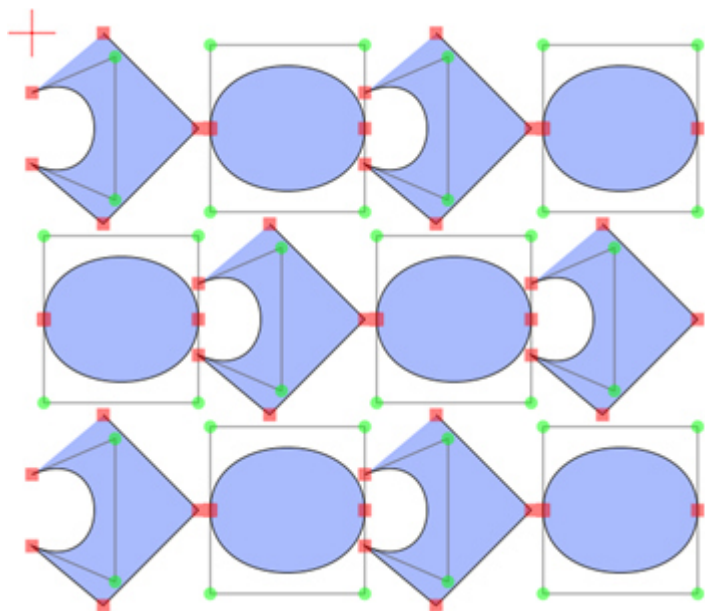
Dibujamos una segunda **shape** que posteriormente estará en la misma **tabla** que la **shape** del ejemplo anterior:



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

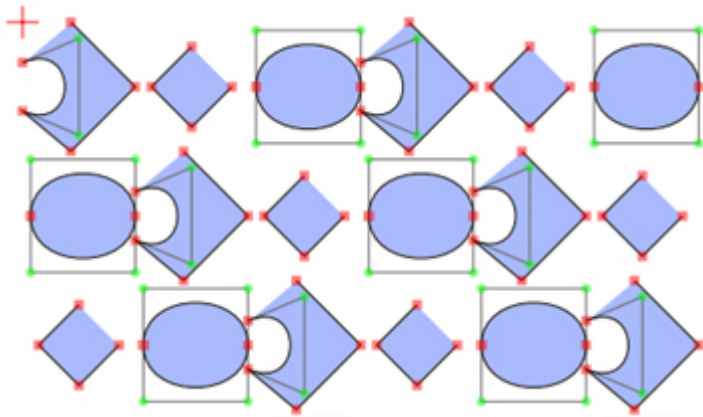
```
Variables:
mi_shape = {"m 0 5 b 7 2 7 14 0 11 | 6 16 | 14 8 | 6 0 ", ^
            "m 0 7 b 0 14 13 14 13 7 b 13 0 0 0 7 " }
```

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.

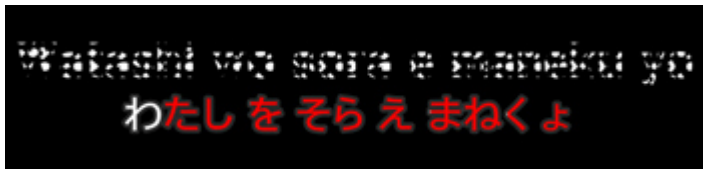


Entonces las Shapes de la **tabla mi\_shape** se multiplicarán de forma alternada y crearán la **multi shape**.

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



Usada en **shape.movevc**:



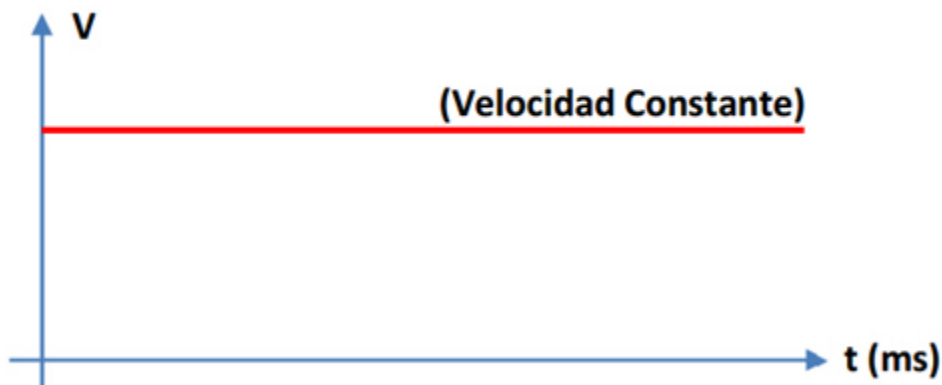
Las distancias que separan a las Shapes, tanto vertical como horizontalmente, no necesariamente deben ser valores positivos o cero, también pueden ser valores negativos con el fin de acercar más una **shape** respecto a las otras.

---

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 momento 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<sub>1</sub> = ( x1, y1 )** hasta **P<sub>2</sub> = ( x2, y2 )**, desde el tiempo **t1** hasta el tiempo **t2** y con una aceleración **accel**. **Lmove** significa **Movimiento Lineal**.

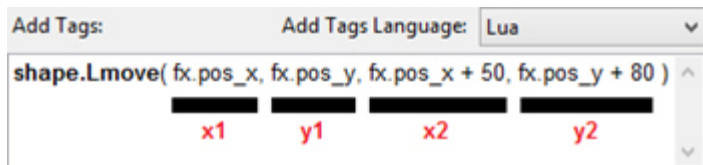
**x1** y **x2** son las coordenadas respecto al eje “x” y ambas tienen el mismo valor por default en el caso de no usar estos parámetros: **fx.move\_x1**

**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.movement\_i** y **fx.movement\_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**

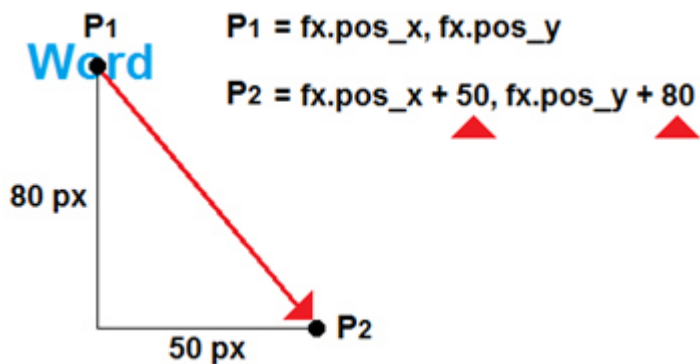


Así, los valores de **t1** y **t2** serán los que tienen por default:



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:

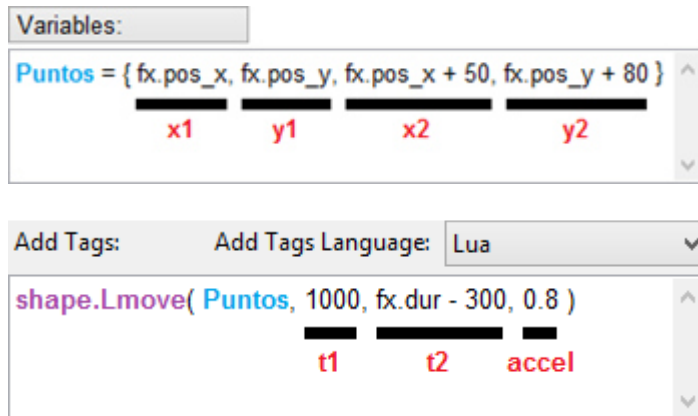


En la siguiente imagen vemos la **shape invisible** que hace que cada palabra (Word) se mueva según los parámetros que hemos ingresado en la función:

0:00:08.16	Romaji	lead-in	Effector [Fx]	*m 0 0   100 100 *Kodoku
0:00:08.16	Romaji	lead-in	Effector [Fx]	*m 0 0   100 100 *na
0:00:08.16	Romaji	lead-in	Effector [Fx]	*m 0 0   100 100 *hoho
0:00:08.16	Romaji	lead-in	Effector [Fx]	*m 0 0   100 100 *wo
0:00:08.16	Romaji	lead-in	Effector [Fx]	*m 0 0   100 100 *nurasu
0:00:08.16	Romaji	lead-in	Effector [Fx]	*m 0 0   100 100 *nurasu
0:00:08.16	Romaji	lead-in	Effector [Fx]	*m 0 0   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**”.

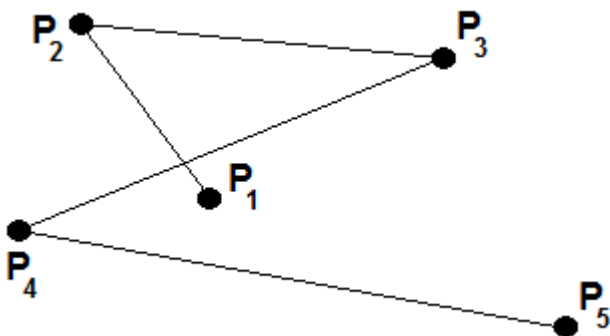
- **Ejemplo 2:**



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

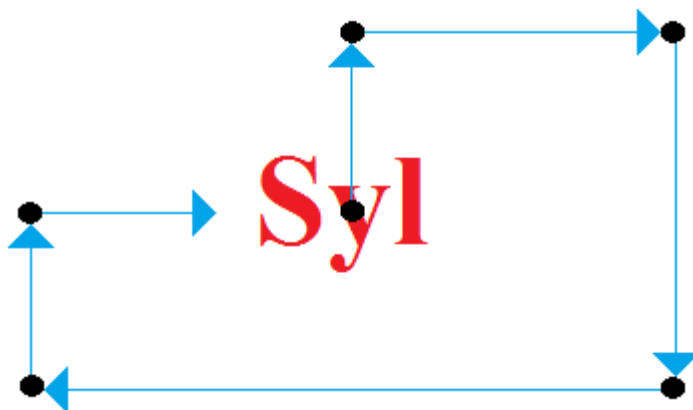
**Puntos:** { x1, y1, x2, y2, x3, y3,... ..xn, yn }

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.

---

### **shape.Pmove(x(s), y(s), dom, t1, t2, acc, off\_t):**

Esta función hace que el objeto karaoke se mueva siguiendo la trayectoria de la gráfica de la función definida por los parámetros **x(s)** y **y(s)** en el dominio **dom**.

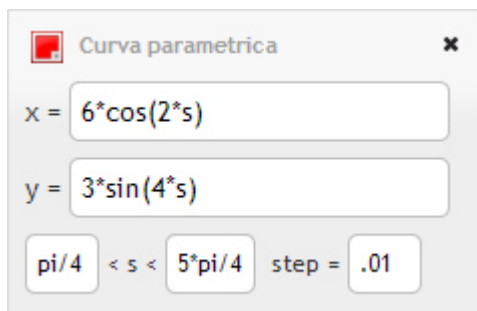
Los parámetros **t1** y **t2** son los tiempos de inicio y final del movimiento del objeto karaoke y sus valores por default son 0 y **fx.dur**.

El parámetro **dom** es el dominio de las funciones paramétricas **x(s)** y **y(s)**. Cuando es un valor numérico, el dominio será el intervalo cerrado entre 0 y dicho valor. Cuando es una **tabla**, el dominio va desde el valor del primer elemento hasta el segundo: **{dom\_i, dom\_f}**

El parámetro **acc** es la aceleración del movimiento en las transformaciones que genera la función, y su valor por default es 1.

El parámetro **off\_t** hace referencia al tiempo a añadir o restar de la duración de las transformaciones. Cada una de las transformaciones que genera esta función tiene una duración por default de  $2.4 * \text{frame\_dur}$ , y con **off\_t** podemos añadir o quitar tiempo a esa duración.

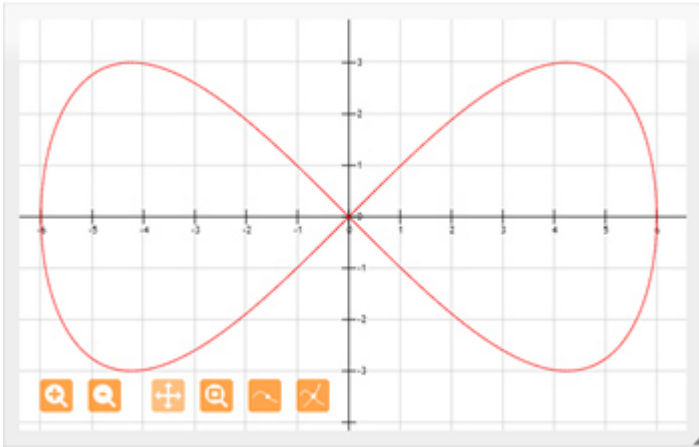
**Ejemplo:** [www.fooplot.com](http://www.fooplot.com)



The image shows a web interface titled "Curva parametrica" with a close button (X). It contains two input fields for parametric equations: "x =" with the value "6\*cos(2\*s)" and "y =" with the value "3\*sin(4\*s)". Below these, there are three input fields for the domain: "pi/4", "< s <", and "5\*pi/4", followed by a "step =" field with the value ".01".



Esta es la gráfica de los anteriores parámetros:



Esto es lo que debemos hacer para pasar la información desde el graficador a la función:

Curva paramétrica ✕

x =  → = "6\*cos( 2\*%s )"

y =  → = "3\*sin( 4\*%s )"

< s <  → = { pi/4, 5\*pi/4 }

- Poner las ecuaciones paramétricas entre comillas, ya sean simples y dobles.
- No olvidar poner siempre el símbolo del producto (\*), o sea: 6\*cos en vez de 6cos.
- Añadir el símbolo de porcentaje (%) antes de cada "s" que aparezca en las ecuaciones paramétricas.
- Hacer una **tabla** con los valores de inicio y final del dominio de las funciones paramétricas **x(s)** y **y(s)**.

```
Add Tags: Add Tags Language: Lua
shape.Pmove( "6*cos(2*%s)", "3*sin(4*%s)",
              {pi/4, 5*pi/4} )
```

Entonces la función genera una shape invisible y una serie de transformaciones que harán que el objeto karaoke se mueva siguiendo la trayectoria de la gráfica generada por las dos ecuaciones paramétricas y el dominio asignado.

En el caso en que notemos que el movimiento sea poco perceptible, es porque las proporciones de las funciones son muy pequeñas. Las proporciones son los valores por el cual multiplicamos las funciones del anterior ejemplo:

- "cos(2\*%s)" → "6 \* cos(2\*%s)"
- "sin(4\*%s)" → "3 \* sin(4\*%s)"

### Ejemplo:

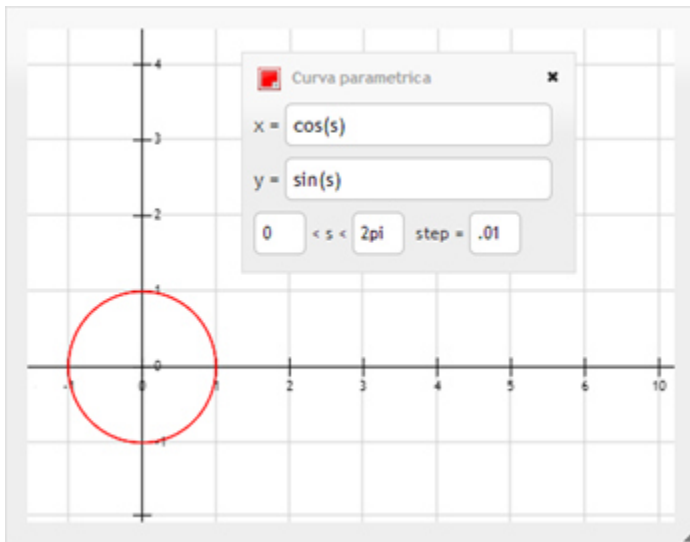
Este sería un ejemplo usando todos los parámetros de la función:

```
Add Tags: Add Tags Language: Lua
shape.Pmove( "6*cos(2*%s)", "3*sin(4*%s)",
             {pi/4, 5*pi/4}, 0, 300, 160 )
```

O sea que el movimiento empezará en 0 ms y terminará a los 300 ms. A parte de eso le estamos sumando 160 ms a la duración de las transformaciones generadas.

### Ejemplo:

Los siguientes parámetros corresponden a la gráfica de un círculo de 1 px de radio, de modo que si los usamos para la función **shape.Pmove**, entonces no se notaría mucho el movimiento en el vídeo:

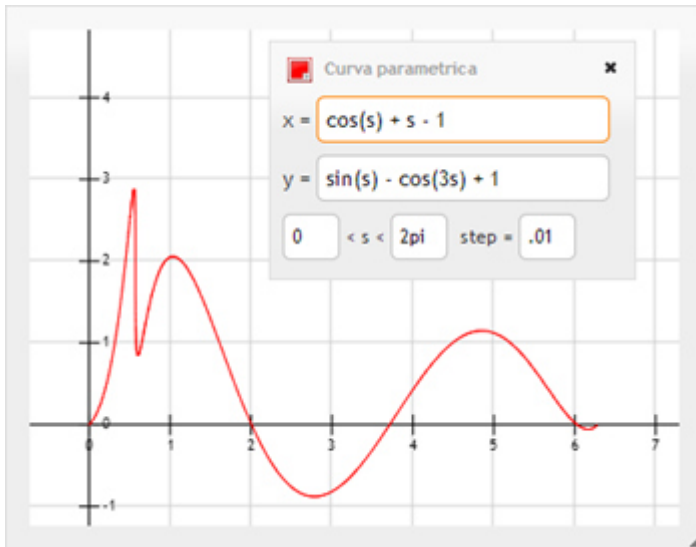


Lo que debemos hacer es multiplicar a cada ecuación paramétrica por un valor tal, que haga que el círculo sea más grande y así sea evidente el movimiento. Ejemplo:

- “100 \* cos( %s )”
- “100 \* sin ( %s )”

## Ejemplo:

Podemos definir las ecuaciones paramétricas en el graficador online y una vez que obtengamos una gráfica que sea de nuestro agrado, la podemos usar tal cual y luego ir modificando ambas proporciones con el fin de que el movimiento se ajuste a las necesidades de nuestro efecto:



Las ecuaciones para usar en la función quedarían:

- “cos( %s ) + %s - 1”
- “sin( %s ) - cos ( 3\*%s) + 1”

**Dominio:**

- { 0, 2\*pi }

Y para modificar las proporciones, debemos encerrar entre paréntesis a las ecuaciones paramétricas y así podremos multiplicar a cada una de ellas por el valor que necesitemos. Ejemplo:

- “45 \* ( cos( %s ) + %s - 1 )”
- “72 \* ( sin( %s ) - cos ( 3\*%s) + 1 )”

**Shape.Smove( Shape, t1, t2, off):**

Esta función hace que el objeto karaoke se mueva siguiendo el contorno del perímetro de la shape ingresada “Shape”, desde **t1** a **t2**.

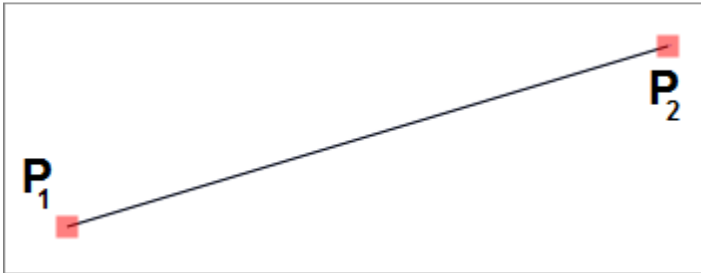
Los parámetros **t1** y **t2** son los tiempos de inicio y final del movimiento del objeto karaoke y sus valores por default son 0 y **fx.dur**.

El parámetro **off** hace referencia al tiempo a añadir o restar de la duración de las transformaciones. Cada una de las transformaciones que genera esta función tiene una duración por default de 2\***frame\_dur**, y con **off** podemos añadir o quitar tiempo a esa duración.

El parámetro **Shape** puede ser o una **tabla** o un **string**. Para el caso del **string**, debe ser el código de la **shape** en formato **.ass**, es decir el código que obtenemos de la **shape** en el **ASSDraw3**; o sea que podemos utilizar cualquiera de las Shapes que por default ya vienen en el **Kara Effector**. Y para el caso en que el parámetro **Shape** sea una **tabla**, ésta debe contener valores numéricos que cumplan con las siguientes características:

- Coordenadas de 2 puntos:

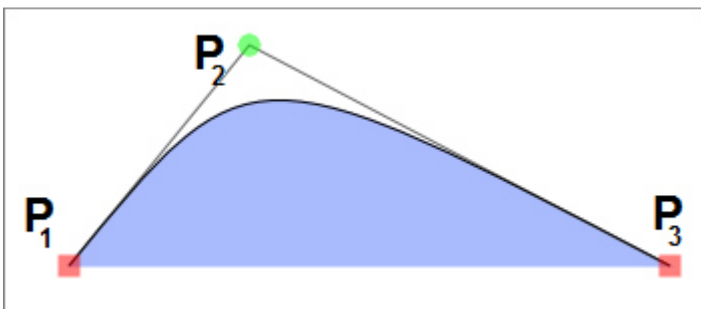
$$\text{Shape} = \{ P_{x1}, P_{y1}, P_{x2}, P_{y2} \}$$



El objeto karaoke se moverá siguiendo la trayectoria de la línea recta que forman los dos puntos ingresados.

- Coordenadas de 3 puntos:

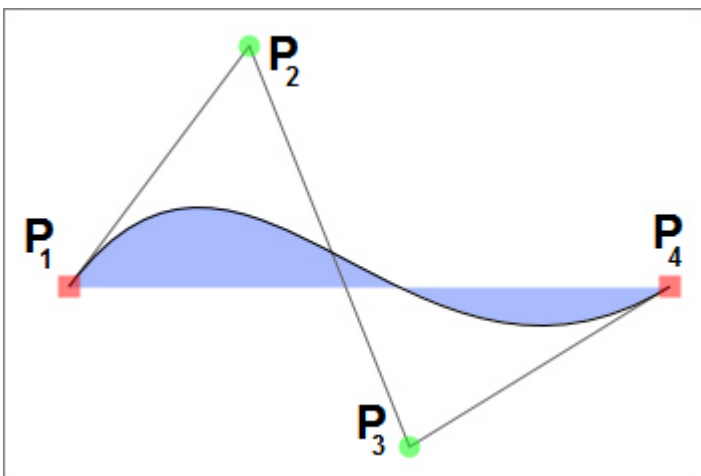
$$\text{Shape} = \{ P_{x1}, P_{y1}, P_{x2}, P_{y2}, P_{x3}, P_{y3} \}$$



El objeto karaoke se moverá siguiendo la trayectoria de la **Curva Bezier** que forman los tres puntos ingresados.

- Coordenadas de 4 puntos:

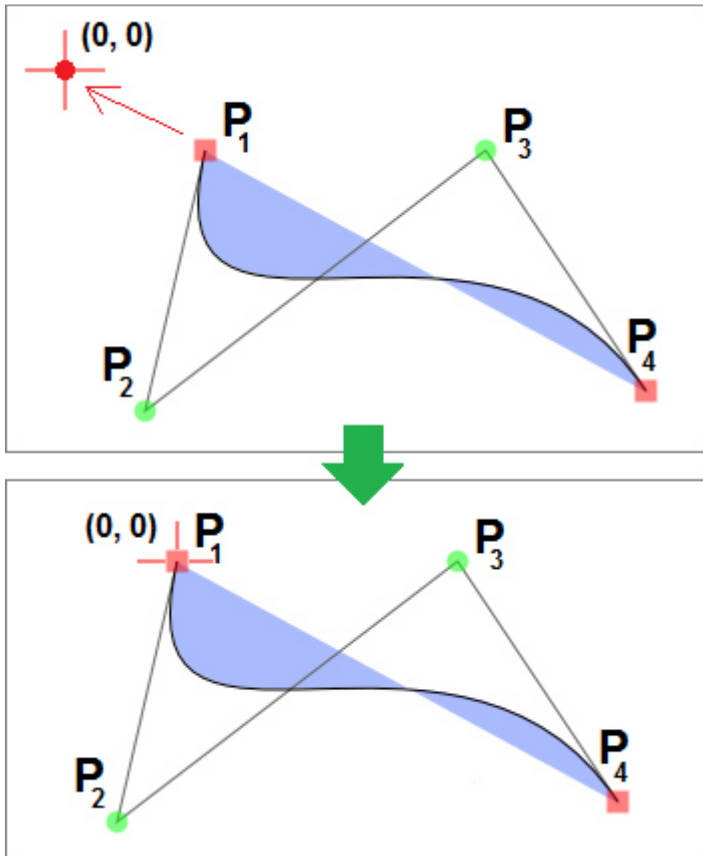
$$\text{Shape} = \{ P_{x1}, P_{y1}, P_{x2}, P_{y2}, P_{x3}, P_{y3}, P_{x4}, P_{y4} \}$$



El objeto karaoke se moverá siguiendo la trayectoria de la **Curva Bezier** que forman los cuatro puntos ingresados.

Es decir, que la función toma las coordenadas de los puntos ingresados y las convierte en una **shape**, para hacer que el objeto karaoke se mueva siguiendo dicha trayectoria.

Explicado esto, podemos decir que la función siempre toma como base a la trayectoria del perímetro de una **shape** para hacer que el objeto karaoke se mueva en el contorno de la misma. Lo siguiente que hace la función es mover la **shape** de tal forma que el primer punto quede en las coordenadas  $P = (0, 0)$  del plano en el **ASSDraw3**. Ejemplo:



Una vez se desplaza de la anterior forma a la shape, la función hace coincidir al centro del objeto karaoke con ese punto  $P = (0, 0)$ , y ahí sí genera las transformaciones que hacen posible el movimiento.

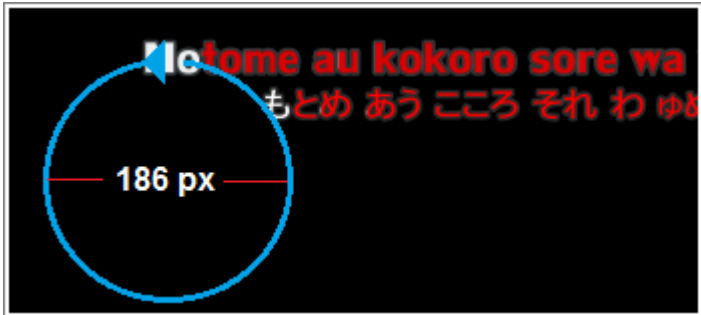
**Ejemplo:**

```
Variables:
mi_shape = shape.size( shape.circle, 186 )
```

Declaramos una **shape** en la celda de texto “**Variable**“, aunque sabemos que no es necesario, ya que lo podemos hacer directamente dentro de la función. La **shape** para este ejemplo es un círculo de 186 px de diámetro.

```
Add Tags:      Add Tags Language:  Lua
shape.Smove( mi_shape, fx.dur/2, fx.dur )
```

- $t1 = fx.dur/2$
- $t2 = fx.dur$



El objeto karaoke, en este caso las Sílabas, se moverán siguiendo como trayectoria al perímetro del círculo de 186 px de diámetro a partir de la mitad de la duración total de la línea de **fx** ( $t1 = fx.dur/2$ ), hasta el tiempo final de la misma, como se ve en la imagen anterior.

### Shape.Rmove( Dx, Dy, t1, t2, acc, off )

Esta función hace que el objeto karaoke se mueva aleatoriamente de forma lineal, desde su centro por default (**fx.move\_x1**, **fx.move\_y1**), sin exceder los límites **Dx** y **Dy**. El movimiento que se genera es perpetuo, dependiendo de los límites de tiempo **t1** y **t2**.

Los parámetros **t1** y **t2** son los tiempos de inicio y final del movimiento del objeto karaoke y sus valores por default son 0 y **fx.dur**.

El parámetro **acc** es la aceleración del movimiento en las transformaciones que genera la función, y su valor por default es 1.

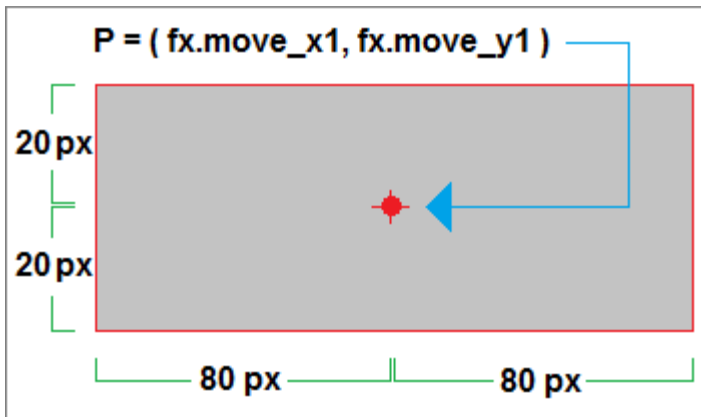
El parámetro **off** hace referencia al tiempo a añadir o restar de la duración de las transformaciones. Cada una de las transformaciones que genera esta función tiene una duración por default de  $3.6 * frame\_dur$ , y con **off** podemos añadir o quitar tiempo a esa duración.

### Ejemplo:

- $Dx = 80$
- $Dy = 20$

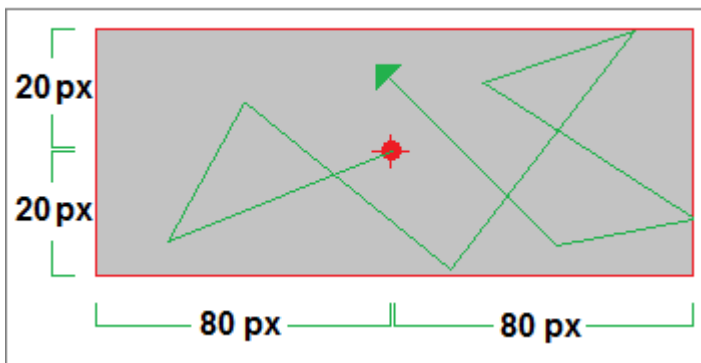
```
Add Tags:      Add Tags Language:  Lua
shape.Rmove( 80, 20 )
```

Entonces la función crea un rectángulo imaginario con el doble de los valores ingresados en **Dx** y **Dy**, como ancho y alto de dicho rectángulo:



Y las transformaciones generadas harán que una **shape** invisible cree el **Movimiento Lineal Aleatorio** sin salirse nunca de los límites creados por el rectángulo imaginario:

### Movimiento Lineal Aleatorio Delimitado



Esta función tiene una particularidad, que sin importar la cantidad de movimientos generados, la posición final será el mismo punto donde empezó, es decir que se moverá de forma aleatoria para finalmente terminar en donde estaba posicionado inicialmente:  $P = (fx.move\_x1, fx.move\_y1)$

### Ejemplo:

```
Add Tags: Add Tags Language: Lua
shape.Rmove( 30, 25, 0, 400, 0.8 )
```

Entonces el objeto karaoke se moverá de forma aleatoria en un rectángulo imaginario de 60 X 50 px (el doble de cada valor ingresado en **Dx** y **Dy**), desde los 0 ms hasta los 400 ms y con una aceleración de 0.8; pero una vez transcurrido ese lapso de tiempo, volverá a su posición inicial de origen.

Las cuatro funciones que generan movimiento por medio de una shape invisible son:

- **shape.Lmove**
- **shape.Pmove**
- **shape.Smove**
- **shape.Rmove**

Aparte de **shape.Smove**, las otras tres funciones pueden generar movimiento con aceleración y cada una de ellas con las características que hasta acá hemos visto.

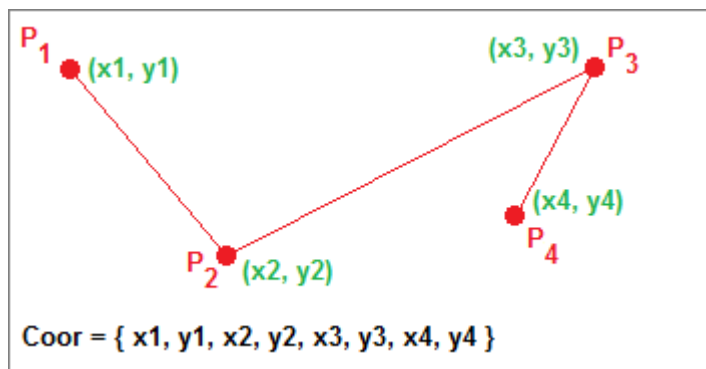
El movimiento que generan las transformaciones de estas cuatro funciones es similar al generado por **shape.config**, con la diferencia que esta última genera un **loop** del objeto karaoke para dar la ilusión de movimiento.

Ya para terminar, el uso de estas cuatro funciones puede consumir una cantidad considerable de recursos de nuestra PC y volverla un poco lenta, lo que hará que no podamos apreciar en tiempo real a nuestro efecto. Todo dependerá del tipo de PC que cada uno tenga, pero pensando es este consumo de recursos, más adelante veremos otras cuatro funciones que también nos dan la posibilidad de generar movimiento acelerado con características similares a las que recientemente hemos visto y sin el efecto secundario de la lentitud de nuestra PC. Estas funciones pertenecen a la librería **tag** y en próximos **artículos** las veremos en detalle.

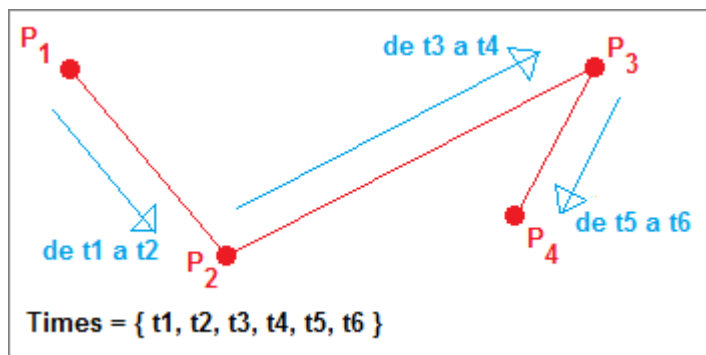
### **shape.Lmove2( Coor, Times) :**

Esta función es similar a **shape.Lmove**, pero con la diferencia que necesita dos tablas como parámetros para llevar a cabo su labor.

El parámetro **Coor** es la **tabla** de las coordenadas de los puntos en los que se desplazará nuestro objeto karaoke:



El parámetro **Times** es la **tabla** de los tiempos iniciales y finales de los movimientos entre punto y punto de la **tabla** del parámetro **Coor**:



### **> Ejemplo:**

Este será un sencillo ejemplo de su aplicación y puede ser usado en las líneas de traducción, de manera que usaremos el efecto con **Template Type: Translation Line**



Primero definimos nuestras dos **tablas** en la celda de texto “**Variables**“, aunque este procedimiento no es del todo necesario, ya que podemos ingresar directamente las tablas en la función sin tener de definirlas, pero lo hago por simple organización:

```
>> Variables:
puntos = { line.center - 500, line.middle,
           line.center, line.middle,
           line.center + 500, line.middle };
tiempos = { 0, 200, line.dur - 200, line.dur }
```

Entonces tenemos:

- **P1 = line.center – 500, line.middle**
- **P2 = line.center, line.middle**
- **P3 = line.center + 500, line.middle**
- **t1 = 0**
- **t2 = 200**
- **t3 = line.dur – 200**
- **t4 = line.dur**

Y la función hará la siguiente:

- Moverá la línea de texto de P1 a P2 desde t1 a t2, o sea, desde los 0 ms hasta 200 ms.
- Desde t2 a t3, la línea de texto permanecerá estática, o sea que la línea no se moverá desde los 200 ms hasta line.dur – 200 ms.
- Moverá la línea de texto de P2 a P3 desde t3 a t4, o sea, desde line.dur – 200 ms hasta line.dur

Y para ello ponemos en **Add Tags** así:

```
Add Tags Language: >> Lua
shape.Lmove2( puntos, tiempos )
```

La cantidad de puntos que podemos ingresar en la función **shape.Lmove2** también es ilimitada, lo que conlleva a una cantidad ilimitada de movimientos lineales. Las aplicaciones son múltiples y todo dependerá de la imaginación.

La ventaja de esta función es que podemos hacer la cantidad de movimientos lineales que deseemos, y no necesariamente deben ser consecutivos ni durar la misma cantidad de tiempo, ya que controlamos los tiempos de cada desplazamiento, así como los lugares a los que se moverá.