Kara Effector 3.2:

El **Tomo XIX** 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.Ltrajectory(length_t, length_c, height_c): es una función similar a **shape.trajectory** con la diferencia de que crea la trayectoria en una sola dirección y con las siguientes especificaciones:

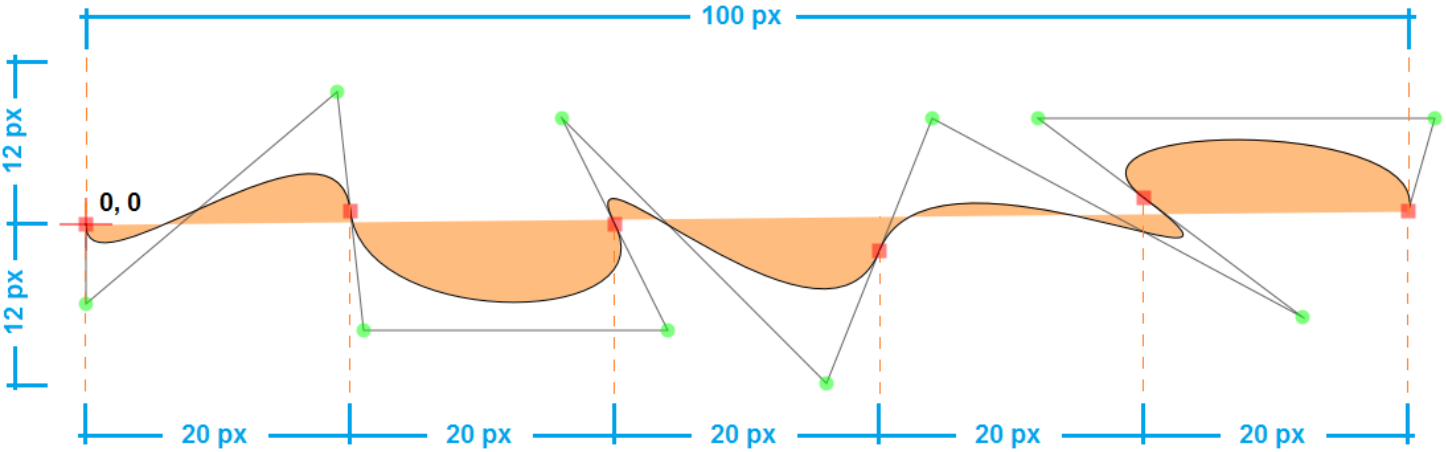
- **length_t:** es la longitud lineal total de la trayectoria medida en pixeles. Su valor por default equivale a la diferencia entre **xres** y **fx.move_x1.**
- **length_c:** es la longitud lineal de cada uno de los segmentos de las **Curvas Bezier** que conforman a la trayectoria. Su valor por default 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.

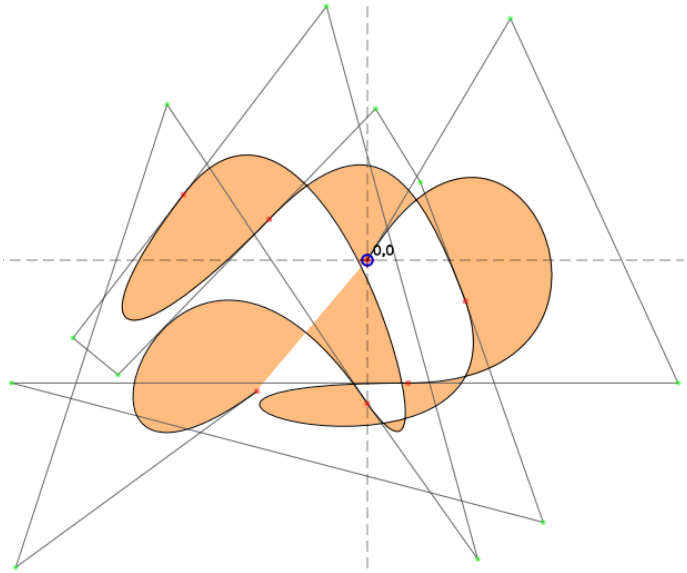
El valor por default de **r_min** es **xres**/40 y el de **r_max** es **xres**/25, o sea que ambos pueden ser opcionales.

Ejemplo:

Return [fx]:

```
shape.Ctrajectory( 5, 20, 50 )
```

Veamos una de las trayectorias fluidas generadas:

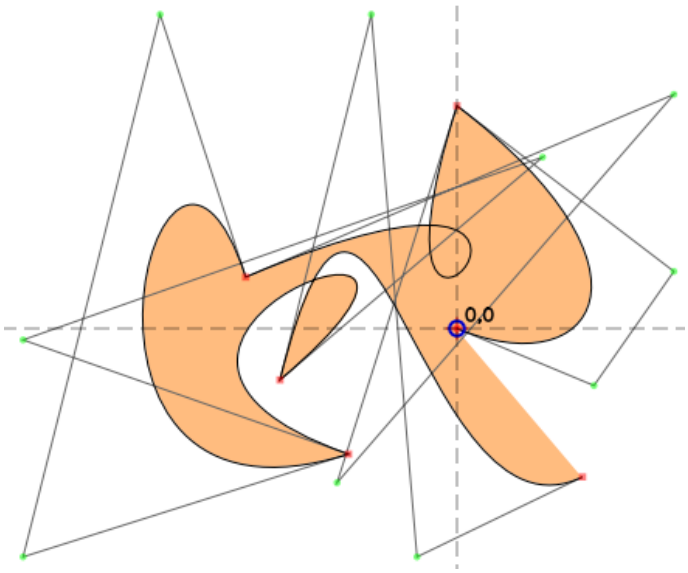


shape.Rtrajectory(Loop, r_min, r_max): es una función similar a **shape.Ctrajectory**, pero la trayectoria que genera ya no es fluida sino totalmente aleatoria (random). Esta función crea una trayectoria con centro en el punto P = (0,0) y sin exceder como máximo al radio **r_max**, ni como mínimo al 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:



---

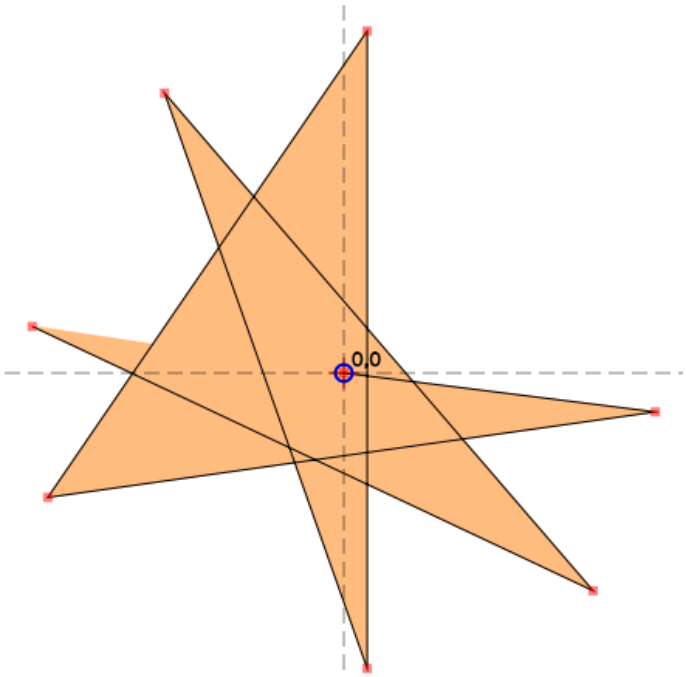
shape.Strajectory(Loop, Radius): esta función es similar a **shape.Rtrajectory**, pero creo la trayectoria con segmentos lineales de forma aleatoria, en vez de usar las **Curvas Bezier** como en las cuatro anteriores funciones.

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*line.height**.

Ejemplo:

```
Return [fx]:
shape.Strajectory( 7, 45 )
```

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



---

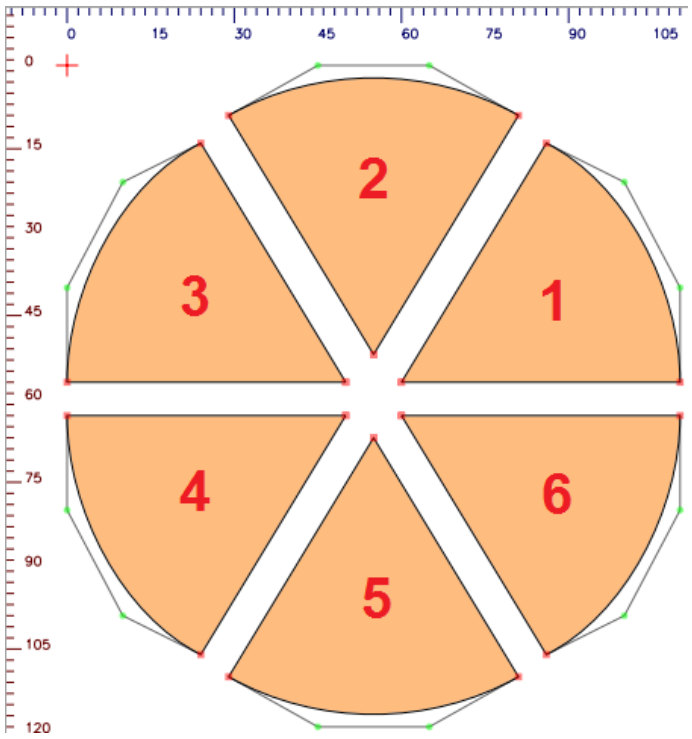
shape.movevc(Shape, Rtn, width, height, x, y, Dx, Dy, t_i, t_f): es similar a la función **tag.movevc**, pero con la diferencia que a esta función se le ingresa una **shape** conformada por dos o más Shapes para ser usadas dentro de un tag **\clip**, que posteriormente serán manipuladas de forma individual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

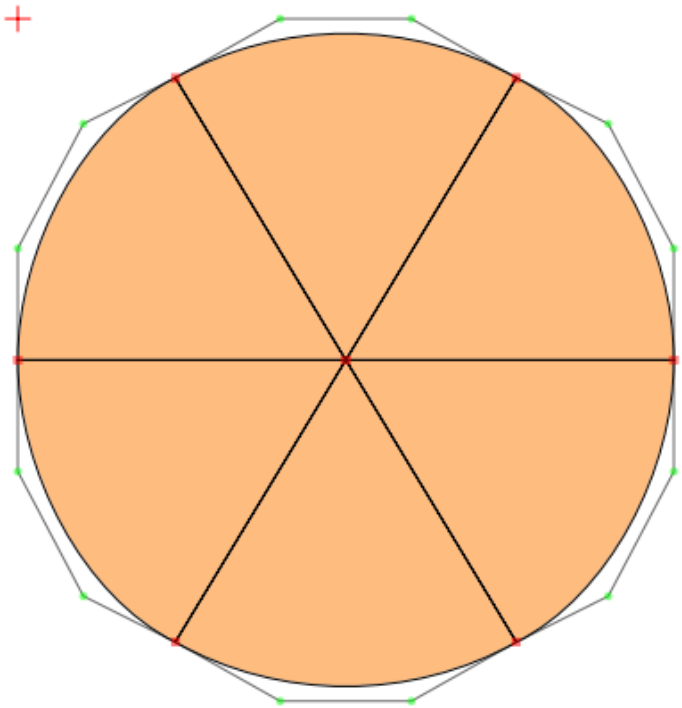
- "**tag**": retorna una serie de tags equivalente al efecto generado por la función.

- **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.movet_i**
  - **t_f = fx.movet_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 l 100 52 b 100 35 90 16 76 9 l 50 52 m 50 52 l 76 9 b 60 0 40 0 24 9 l 50 52 m 50 52 l 24 9 b 10 16 0 35 0 52 l 50 52 m 50 52 l 0 52 b 0 69 10 88 24 95 l 50 52 m 50 52 l 24 95 b 40 104 60 104 76 95 l 50 52 m 50 52 l 76 95 b 90 88 100 69 100 52 l 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 corazone
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

6

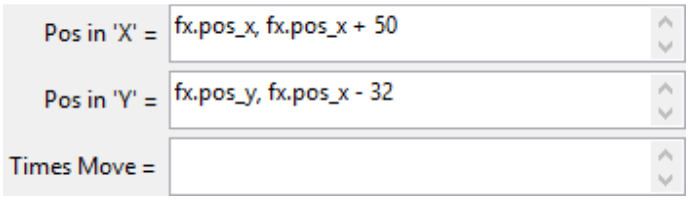
Pero como no le hemos ordenado ningún movimiento ni tampoco le hemos dado ubicaciones distintas a las que ya tiene por default, en el vídeo veremos a cada sílaba de forma normal:

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:

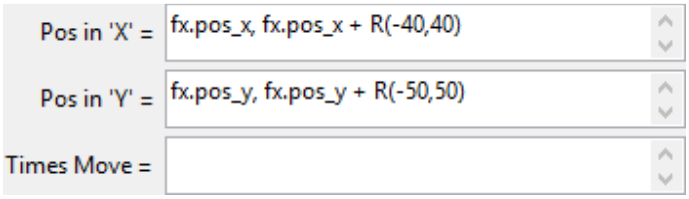
Al ampliar un segmento de los que conforman la sílaba, veremos algo como esto:

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:



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:



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)**

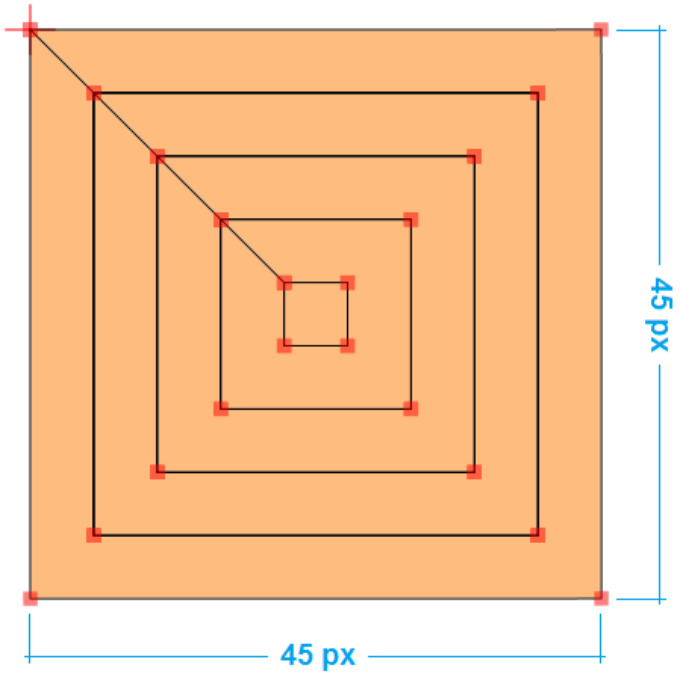
El parámetro **Px** equivale al ancho en pixeles de cada una de las Shapes cuadradas concéntricas que conforman a la **shape** que será retornada. Su valor por default es 4***ratio**.

Ejemplo:

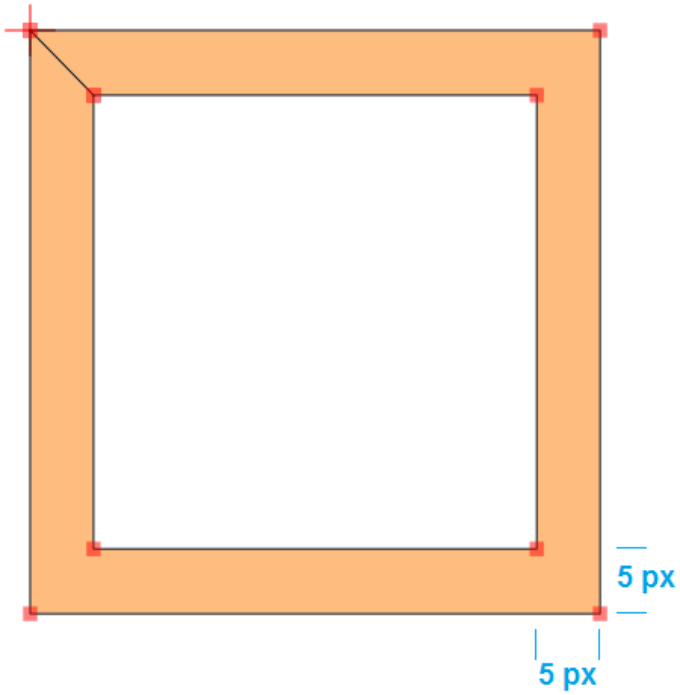
Return [fx]:

```
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" )
```

Es todo por ahora. En el **Tomo XX** continuaremos viendo más de las funciones de la librería **shape**. 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.facebook.com/karaeffector