Kara Effector 3.2: Effector Book Vol. I [Tomo XX]

Kara Effector 3.2:

El **Tomo XX** 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.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.movevci**.

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**

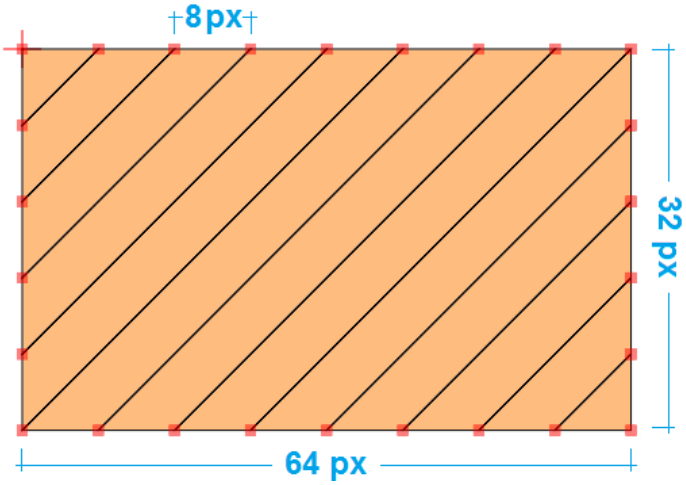
Ejemplo:

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:

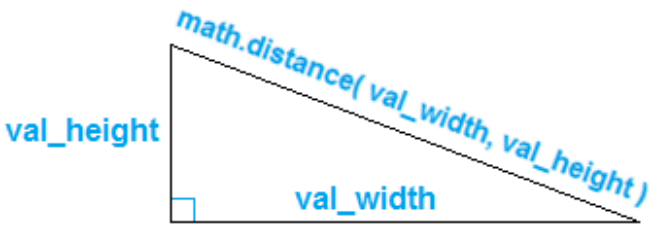
---

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)**



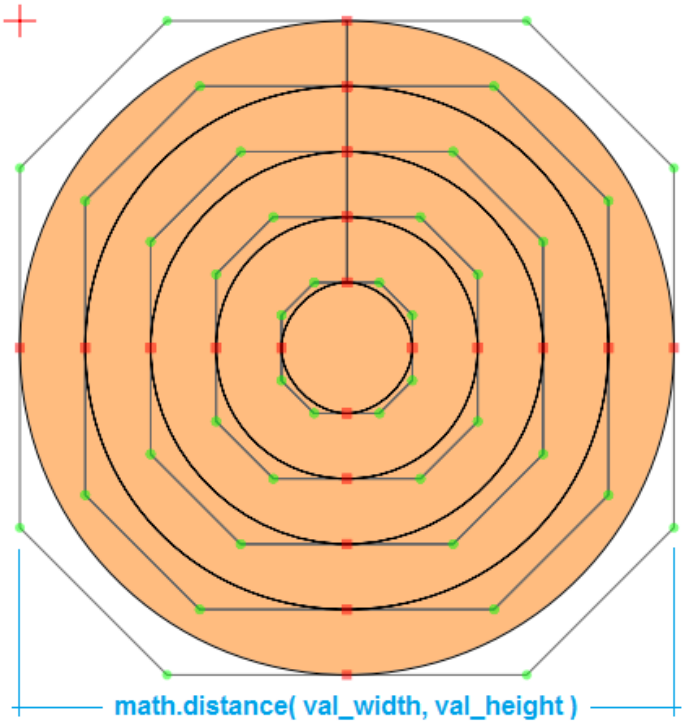
También se puede acceder a este valor si ponemos la palabra "**default**" en este parámetro.

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

Ejemplo 1:

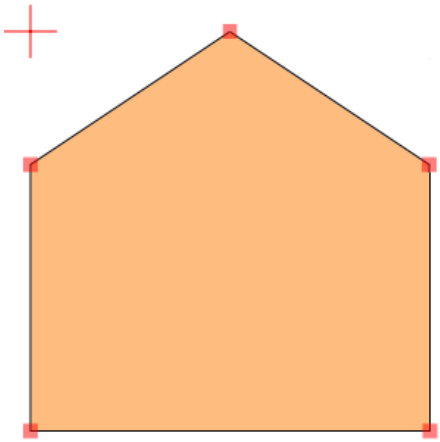
```
Return [fx]:
shape.multi3( "default", 8 )
```

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



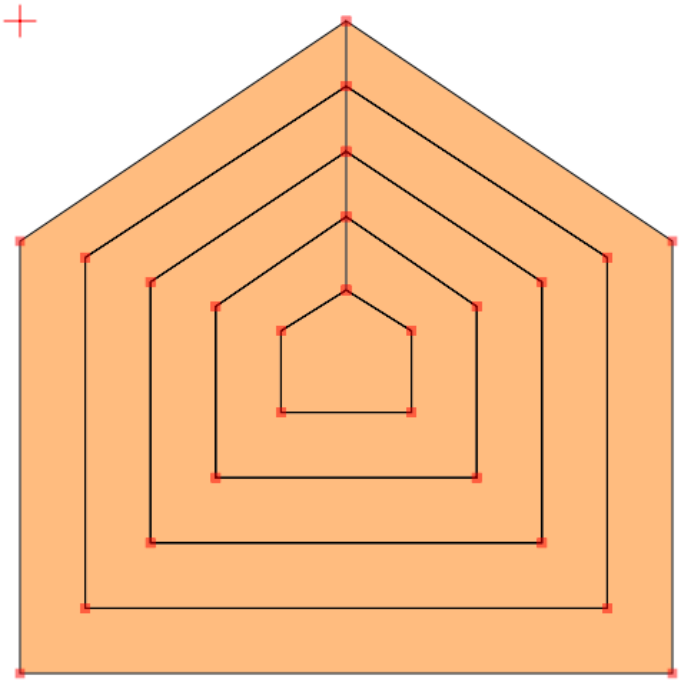
Ejemplo 2:

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



Con un espesor de 6 px:

```
Return [fx]:
shape.multi3( "default", 6, "m 15 0 l 0 10 l 0 30 l 30 30
                l 30 10 l 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**.

---

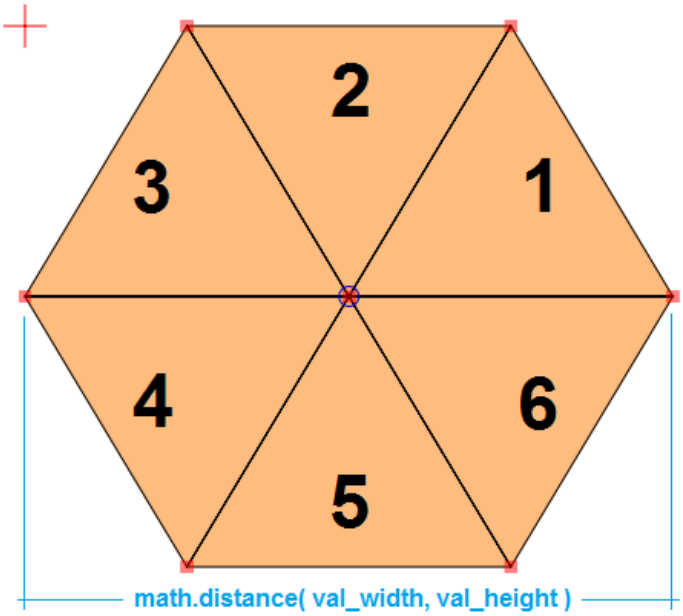
shape.multi4(Size, loop1, loop2): esta función retorna un **Arreglo Radial** de tamaño total **Size** y con una cantidad de repeticiones, en principio determinada por el parámetro **loop1**.
Esta función solo está disponible para la versión **3.2.7** o superior del **Kara Effector**.

Sus valores por default son:

- **Size: math.distance(val_width, val_height)**
- **loop1:** 6
- **loop2:** 1

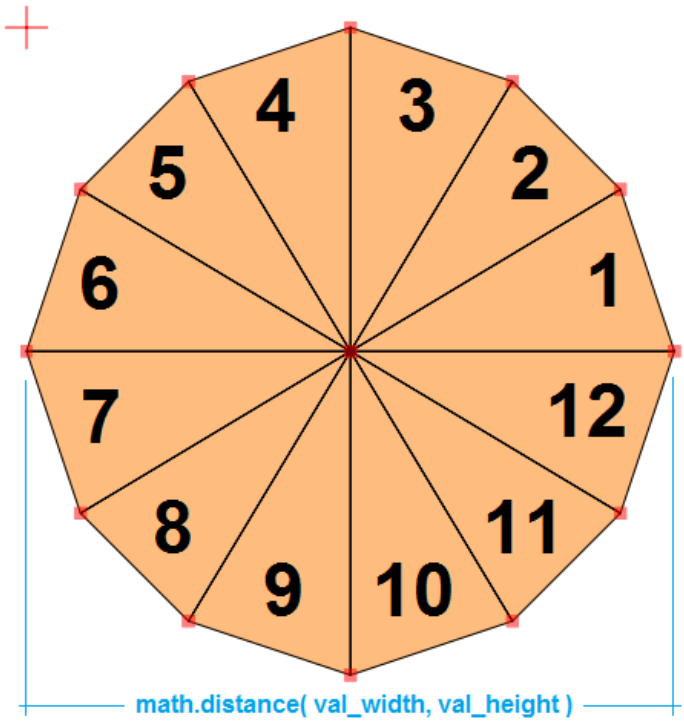
- **Ejemplo 1.** Todos los parámetros por default:

```
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:

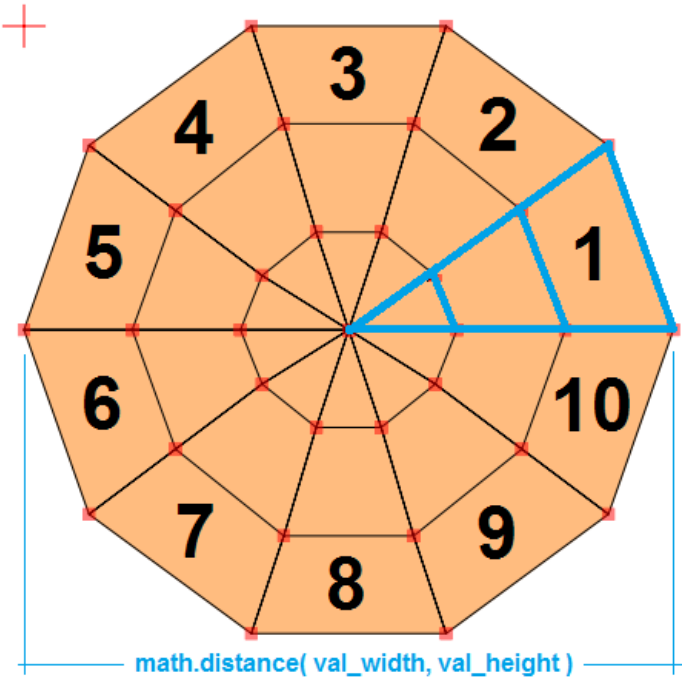
math.distance(val_width, val_height)

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 X 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:

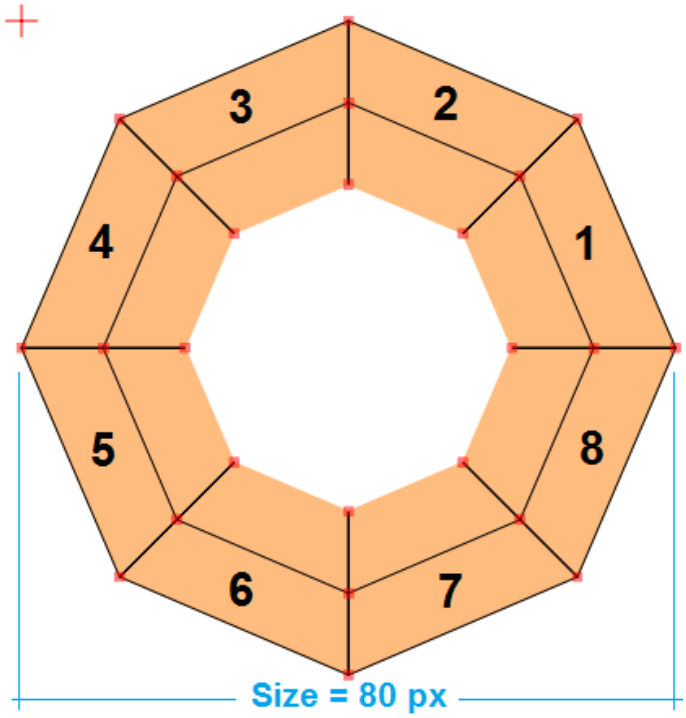


Como **loop1** es 10 en este ejemplo, entonces cada **Arreglo Radial** está compuesto por 10 Shapes individuales, que multiplicados por 3, de **loop2**, hace que el **loop** total sea de 10 X 3 = 30.

- **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 cuanta: 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.

Es todo por ahora. En el **Tomo XXI** 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