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

Kara Effector 3.2:

En este **Tomo XXVII** continuaremos viendo las funciones de esta interesante **librería text**. Esta librería contiene una serie de funciones interesantes que espero que con la ayuda de esta documentación, le puedan sacar el máximo provecho a la hora de llevar a cabo sus proyectos, no solo karaokes, sino también en la edición de los subtítulos.

Librería Text [KE]:

text.bord_to_shape(Text, Scale, Tags, Bord)

Esta función convierte el borde del texto ingresado y lo convierte en una **shape**.

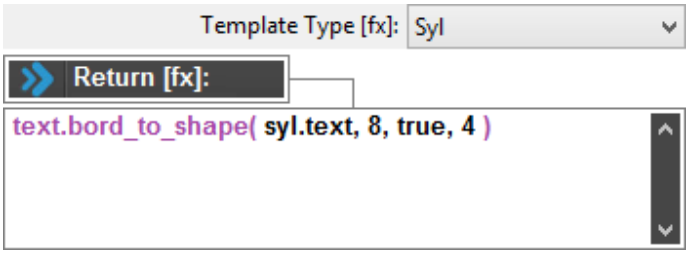
El parámetro **Text** es el string de texto al que la función le convertirá su borde en una **shape**, y su valor por default es el texto por default según el **Template Type**.

Los parámetros **Scale** y **Tags** cumplen con la misma tarea que en las dos funciones anteriores, darle proporción y agregar tags de posicionamiento. Sus valores por default son los mismos, es decir:

- **Scale** = 1
- **Tags** = nil

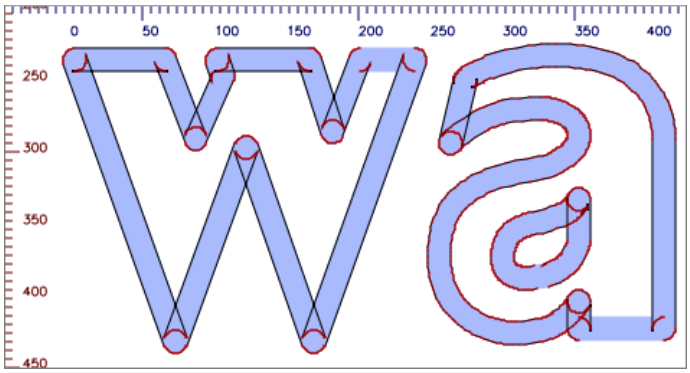
El parámetro **Bord** es un número que indica el grosor del borde hecho shape, su valor por default es 2.

Ejemplo:



Al aplicar veremos algo como esto:

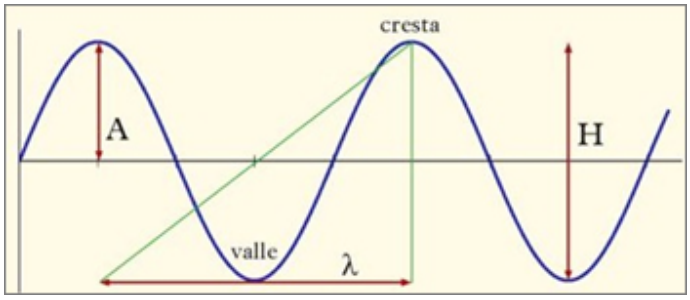
Una de las Sílabas vistas en el **ASSDraw3**:



Y el borde hecho **shape** ya queda apto para aplicarle la función de la librería **shape** que queramos.

text.deformed(Text, Deformed, Pixel, Axis)

Esta función convierte el texto ingresado en una **shape** y posteriormente lo deforma adaptando la forma de onda senoidal:



El parámetro **Deformed** es un número mayor que cero, que indica la cantidad de crestas y valles que tendrá la onda en la que se convertirá el texto. Su valor por default es 2.

El parámetro **Pixel** es la altura en pixeles de cada una de las crestas y valles que tendrá la onda, lo que en la imagen anterior corresponde a la letra "**A**". Su valor por default es **line.height**

El parámetro **Axis** tiene tres opciones:

- "**x**": dibuja la curva sobre el eje "x"
- "**y**": dibuja la curva sobre el eje "y"
- { **Deforme2**, **Pixel2** }: asume a **Deformed** y **Pixel** como valores en la deformación respecto al eje "x" y a **Deformed2** y **Pixel2** como valores para la deformación respecto al eje "y". O sea que de este modo se deforma el texto respecto a ambos ejes.

El valor por default del **Axis** es "x".

Para los siguiente ejemplos usaremos un **Template Type**: **Line**, aunque se puede usar esta función con cualquiera de los demás modos. Es solo que para fines ilustrativos es más visible ver la deformación sobre la línea de texto completa que en una letra o sílaba.

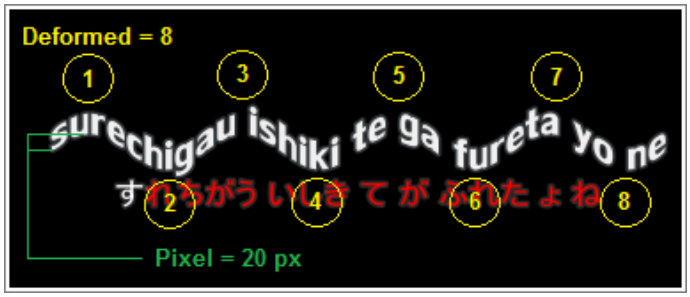
Ejemplo:

Template Type [fx]: Line

Return [fx]:

```
text.deformed( line.text_stripped, 8, 20, "x" )
```

Y al aplicar. Vemos cómo hay 8, entre crestas y valles, y la altura de cada una de ellas es de 20 px:



Ejemplo:

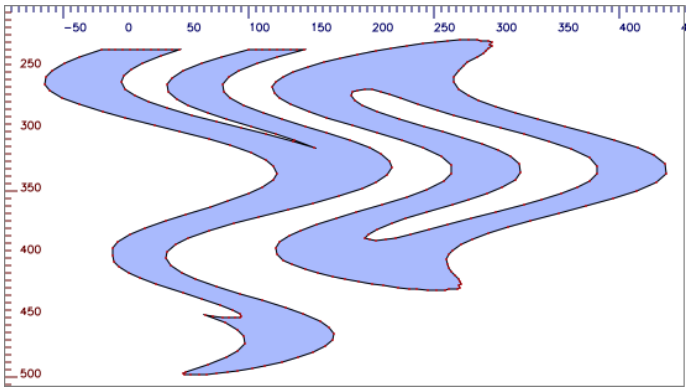
Template Type [fx]: Line

Return [fx]:

```
text.deformed( line.text_stripped, 5, 10, "y" )
```

Ahora el texto se deforma respecto al eje "y" con cinco, entre crestas y valles, y 10 px como su altura:

El poder controlar los valores en que se deformará el texto nos da muchas opciones y resultados. En la siguiente imagen vemos en el **ASSDraw3** la Sílaba "yo" deformada:



Ejemplo:

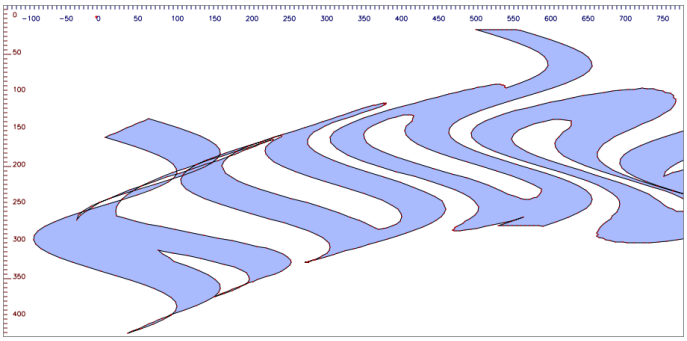
Template Type [fx]: Line

Return [fx]:

```
text.deformed( line.text_stripped, 6, 18, {3, 12} )
```

De este modo, se deforma el texto respecto a ambos ejes:

Visto en **ASSDraw3**, vemos un ejemplo de la deformación, en este caso pone: “kodo”



text.deformed2(Text, Mode)

Esta función retorna tres ejemplos más de cómo deformar un texto convertido a **shape**.

El parámetro **Text** es el string de texto a convertir a **shape** para deformar y su valor por default es el texto por default dependiendo del **Template Type**.

El parámetro **Mode** es un entero entre 1 y 3 que hará que la función retorne uno de los tres ejemplos de deformación. Su valor por default es 1.

Ejemplo:

- **Mode** = 1

Template Type [fx]: Syl

Return [fx]:

```
text.deformed2( syl.text, 1 )
```

Los puntos de la Sílaba convertida en **shape** se desplazan hasta el perímetro de un círculo de 80 px de diámetro:

- **Mode** = 2
  Los puntos de desplazan al perímetro de 3 círculos con diferentes diámetros:

- **Mode** = 3
  Los puntos de la **shape** se desplazan al perímetro de dos hexágonos concéntricos:

Las tres anteriores deformaciones son logradas gracias a las distintas funciones filtros que contienen cada una de ellos, es por eso que anteriormente les mencionaba que las posibilidades son infinitas a la hora de deformar una **shape** por medio de un filtro como función.

El **Mode** = 3 está basado en el siguiente filtro:

```
mi_filtro = function(x, y)
    local center_dx = minx + w_shape/2
    local center_dy = miny + h_shape/2
    local def_angle = math.angle(center_dx, center_dy, x, y)
    local def_angRE = (def_angle - 1)%60 + 1
    local def_ang3A = 180 - 60 - def_angRE
    local des_radiu = 200
    local des_dista = des_radiu*sin(rad(60))/sin(rad(def_ang3A))
    local des_radDE = (math.distance(center_dx,center_dy,x,y)<=80)
                      and des_dista/2 or des_dista
    x = center_dx + math.polar(def_angle, des_radDE, "x")
    y = center_dy + math.polar(def_angle, des_radDE, "y")
    return x, y
end
```

Este filtro nos ayudará a definir a una de las Shapes del siguiente ejemplo, también como variable. Y usaremos la función **shape.morphism** para ver a cada una de las Shapes, desde que el texto está normal hasta que se convierte en uno de los hexágonos del ejemplo anterior:

Ejemplo:

shape1 = shape.filter2(text.to_shape(“DE”, 8), nil, 6)

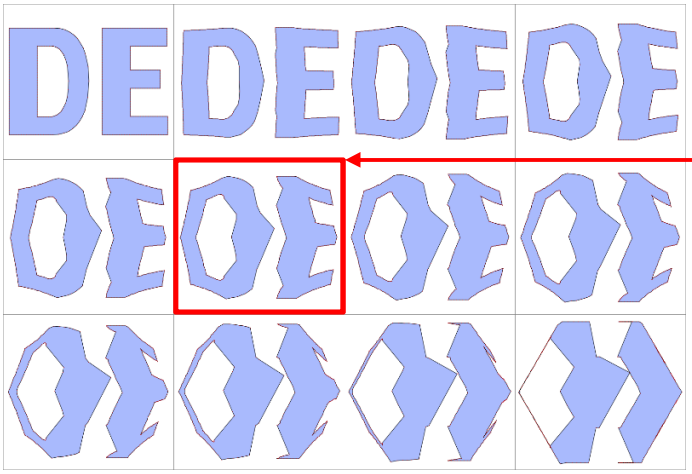
shape2 = shape.filter2(text.to_shape(“DE”, 8), mi_filtro, 6)

En la **shape1** el texto “DE” está normal, ya que en el filtro de la función pusimos “**nil**”. Y la **shape2**, como le pusimos el filtro del ejemplo anterior, entonces el texto “DE” ya está convertido en unos de los hexágonos vistos.

Ahora creamos la **tabla** con las interpolaciones entre una shape y la otra:

mi_tabla = **shape.morphism**(12, shape1, shape2)

Entonces veremos 12 Shapes de interpolación, de cómo el texto “DE” se transforma en uno de los hexágonos del ejemplo anterior:



Aplicando el anterior ejemplo, en un **Template Type: Syl**, cambiamos el texto “DE” por **syl.text**, así:

```
mi_filtro = function(x, y)
  local center_dx=minx + w_shape/2
  local center_dy=miny + h_shape/2
  local def_angle=math.angle(center_dx, center_dy, x, y)
  local def_angRE=(def_angle - 1)%60 + 1
  local def_ang3A=180 - 60 - def_angRE
  local des_radiu=200
  local des_dista=des_radiu*sin(rad(60))/sin(rad(def_ang3A))
  local des_radDE=(math.distance(center_dx,center_dy,x,y)<=80)
                  and des_dista/2 or des_dista
  x = center_dx + math.polar(def_angle, des_radDE, "x")
  y = center_dy + math.polar(def_angle, des_radDE, "y")
  return x, y
end;
shape_txt = text.to_shape( syl.text, 8 )
```

Y como vimos en las 12 Shapes de la transformación del texto “DE”, éste es legible como hasta la sexta o séptima **shape**. Sabido esto, adicionalmente hacemos lo siguiente:

Ejemplo:

Template Type [fx]: Syl

Center in "X" = syl.left

Center in "Y" = syl.top

Align [\an] = 7

Finalmente, ponemos esto en **Return [fx]:**

Return [fx]:

```
"{\\p4}" .. shape.morphism( 12,
    shape.filter2( shape_txt, nil, 6),
    shape.filter2( shape_txt, mi_filtro, 6)
)[6]
```

Y al aplicar veremos algo como esto:

Lo que hará que cada una de las Sílabas quede deformada de forma similar a la sexta **shape** del ejemplo anterior por haber puesto: **mi_tabla[6]**

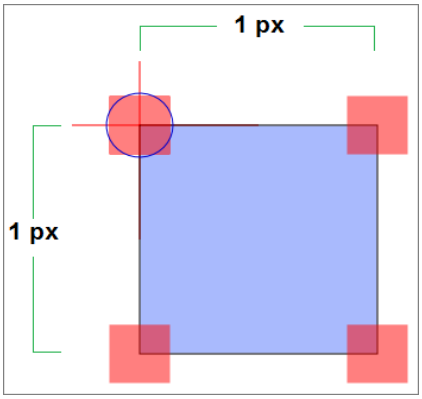
text.to_pixels(Text, Mode, Shape)

Esta función convierte al texto ingresado en pixeles. Esta función genera un loop equivalente a la cantidad de pixeles que tenga el texto ingresado, según los valores de escala y tamaño que tenga en el estilo del mismo.

El parámetro **Text** es el texto a convertir en pixeles y su valor por default es el texto por default según el **Template Type**.

El parámetro **Mode** es un número entero entre 1 y 5 que hace referencia a las distintas posiciones y movimientos de cada uno de los pixeles que conforman el texto. Su valor por default es 1.

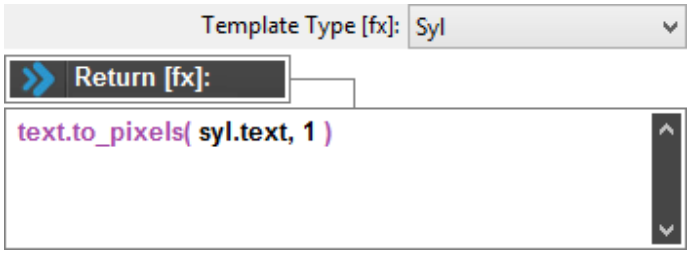
El parámetro **Shape** nos da la opción de que el texto no se convierta en pixeles (**shape.pixel**), sino en la **shape** que asignemos en este parámetro de la función. Su valor por default es **shape.pixel**:



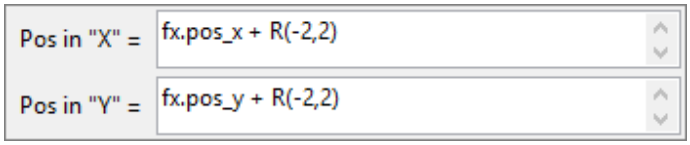
Lo que hace la función es, que por medio de Shapes de 1 x 1 px, remplazar el texto en su tamaño, forma y posición, por todos los pixeles que lo componen. Ejemplo:

Ejemplo:

- **Mode = 1**



En **Modo = 1**, todos los pixeles que componen al texto salen en su posición exacta de tal manera que no se notará la diferencia entre el texto normal y el texto conformado por lo pixeles. Para notar a los pixeles que conforma al texto, añadí cierta cantidad de variación en la posición de los mismos, así:



Y obtendremos algo similar a esto:

Ejemplo:

- **Mode = 2**

Los pixeles se empiezan a desplazar desde el centro del texto, en todas direcciones, y conforman un círculo:

El tiempo total de este desplazamiento está determinado por **fx.dur**

Ejemplo:

- **Mode = 3**

Los pixeles se desplazan de forma muy similar a la anterior, pero ya no tienden a dibujar un círculo, sino una elipse vertical:

Ejemplo:

- **Mode = 4**

Los pixeles ahora se desplazan desde el centro del texto, describiendo tres elipses horizontales alineadas una arriba de la otra:

Ejemplo:

- **Mode = 5**

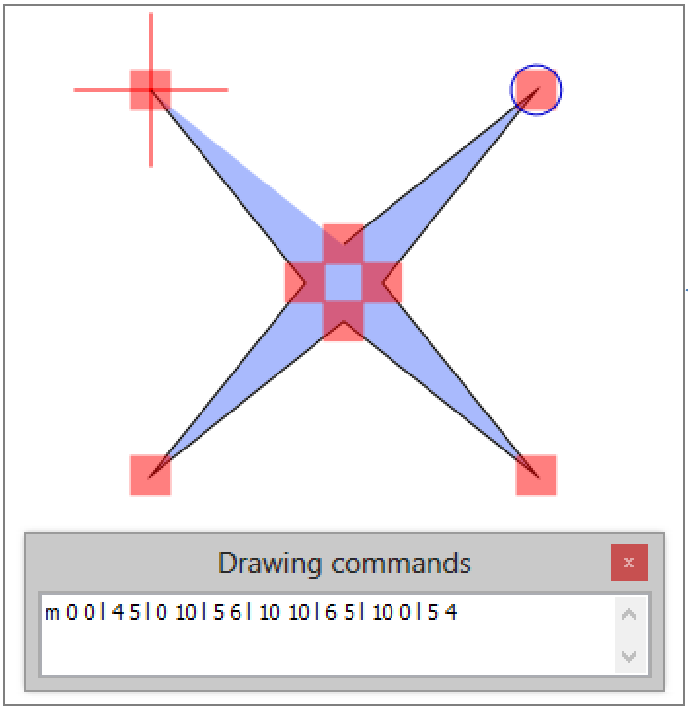
Los pixeles de desplazan describiendo una curva bezier seleccionada al azar:

A partir del modo 1, y con algo de imaginación, se pueden hacer muchas cosas con los pixeles de un texto que genera esta función. Recordemos que con solo variar el tiempo de inicio o el final de una línea fx, se puede modificar el **fx.dur**, lo que le dará una duración diferente a cada uno de los pixeles en el desplazamiento. Es cuestión de experimentar con todo aquello que hasta acá han aprendido.

Hasta este punto, el parámetro **Shape** siempre se usó por default, lo que hacía que la función siempre "pixelará" el texto con la shape cuadrada de 1 x 1 (**shape.pixel**):

Romaji	lead-in	Effector [Fx]	m 0 0 l 0 1 l 1 1 l 1 0 l 0 0
Romaji	lead-in	Effector [Fx]	m 0 0 l 0 1 l 1 1 l 1 0 l 0 0
Romaji	lead-in	Effector [Fx]	m 0 0 l 0 1 l 1 1 l 1 0 l 0 0
Romaji	lead-in	Effector [Fx]	m 0 0 l 0 1 l 1 1 l 1 0 l 0 0
Romaji	lead-in	Effector [Fx]	m 0 0 l 0 1 l 1 1 l 1 0 l 0 0
Romaji	lead-in	Effector [Fx]	m 0 0 l 0 1 l 1 1 l 1 0 l 0 0
Romaji	lead-in	Effector [Fx]	m 0 0 l 0 1 l 1 1 l 1 0 l 0 0

Para el siguiente ejemplo, usaremos esta **shape** en el tercer parámetro de la función, con el fin que el texto ingresado en ella se descomponga por medio de esta **shape**:



Ejemplo:

Y lo seguiremos haciendo en **Template Type: Syl**, aunque esta función puede aplicarse en cualquiera de los modos de fx a aplicar.

Elegí la función en **Mode = 2**, pero ustedes pueden practicar con cualquiera de los ya vistos hasta acá, y pegamos el código de la shape en el tercer parámetro de la función:

Template Type [fx]: Syl

Return [fx]:

```
text.to_pilxels( syl.text, 2, "m 0 0 l 4 5 l 0 10 l 5 6
                 l 10 10 l 6 5 l 10 0 l 5 4 " )
```

Y al aplicar, el texto se descompuso en esta nueva **shape**, lo que da un efecto similar a un brillo:

lead-in	Effector [Fx]	m 0 0 l 4 5 l 0 10 l 5 6 l 10 10 l 6 5 l 10 0 l 5 4
lead-in	Effector [Fx]	m 0 0 l 4 5 l 0 10 l 5 6 l 10 10 l 6 5 l 10 0 l 5 4
lead-in	Effector [Fx]	m 0 0 l 4 5 l 0 10 l 5 6 l 10 10 l 6 5 l 10 0 l 5 4
lead-in	Effector [Fx]	m 0 0 l 4 5 l 0 10 l 5 6 l 10 10 l 6 5 l 10 0 l 5 4
lead-in	Effector [Fx]	m 0 0 l 4 5 l 0 10 l 5 6 l 10 10 l 6 5 l 10 0 l 5 4
lead-in	Effector [Fx]	m 0 0 l 4 5 l 0 10 l 5 6 l 10 10 l 6 5 l 10 0 l 5 4
lead-in	Effector [Fx]	m 0 0 l 4 5 l 0 10 l 5 6 l 10 10 l 6 5 l 10 0 l 5 4
lead-in	Effector [Fx]	m 0 0 l 4 5 l 0 10 l 5 6 l 10 10 l 6 5 l 10 0 l 5 4
lead-in	Effector [Fx]	m 0 0 l 4 5 l 0 10 l 5 6 l 10 10 l 6 5 l 10 0 l 5 4

Este hecho de poder elegir a nuestro antojo a la **shape** en la que "pixelará" el texto ingresado en la función, aumenta en forma exponencial nuestras posibilidades de nuevos e ingeniosos efectos. Tengan en cuenta que esta función genera un loop aproximado entre 7000 y 10000 líneas de fx por cada línea de karaoke o de traducción, a la que se le aplique estos efectos.

Es todo por ahora para el **Tomo XXVII**. Intenten poner en práctica todos los ejemplos vistos 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. 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/NatsuoKE
- www.youtube.com/user/karalaura2012