Kara Effector 3.2:

El **Tomo XIII** es la continuación de las librerías "**color**" y "**alpha**" del **Kara Effector** que se iniciaron en el **Tomo** anterior, en el cual ya vimos numerosas funciones y aún restan muchas más por ver.

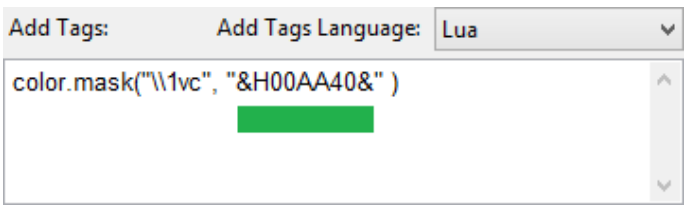
Librerías Color y Alpha [KE]:

color.mask(mode, color, color_mask): esta función retorna un color para cada uno de los elementos de una línea de texto (word, syl, char y demás) a manera de "**máscara**".

El parámetro **mode** hace referencia al único tag de color en formato "**vc**" en el que se realizará la "**máscara**", es decir:

- "\\1vc"
- "\\3vc"
- "\\4vc"

El parámetro **color** puede ser, tanto un **string** de color como una **tabla** de colores. Ejemplo:



En este ejemplo, el parámetro **color_mask** no está ya que es opcional, y el parámetro **color** es un **string**, un color en tono de verde.

El texto se vería de la siguiente forma:

Este ejemplo está hecho con un **Template Type: Syl**, y podemos notar cómo cada una de las sílabas tiene el mismo todo de verde, pero mezclado en ciertas partes con tonos entre el blanco y el negro.

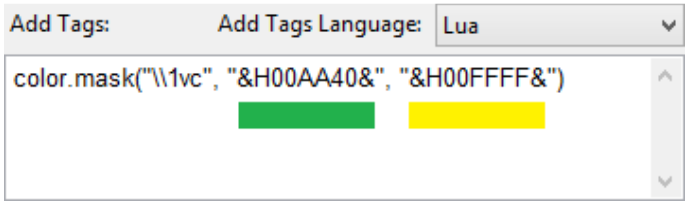
Si remplazamos a cada sílaba por un rectángulo con sus mismas dimensiones, se podrá apreciar más claramente el efecto de la "**máscara**":

La función **color.mask** hace por default la "**mascara**" con el color ingresado en el parámetro **color** y los tonos al azar entre el blanco y el negro.

En el caso en que el parámetro **color** sea una **tabla** de colores, la función retornará una **tabla** con la "**máscara**" de cada uno de los colores de dicha tabla.

El parámetro **color_mask**, al igual que el parámetro **color**, también puede ser tanto un color **string** como una **tabla** de colores. Ejemplos:

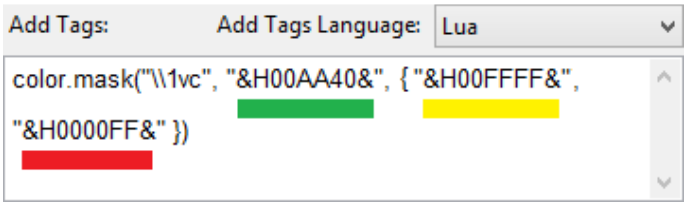
- **color_mask: string**



De este modo, la función ya no hace la "**máscara**" con los tonos por default entre el blanco y el negro, sino que la hace entre el parámetro **color** y el color ingresado en el parámetro **color_mask** (verde y amarillo):

Así que de esta forma uno decide un solo color con el que el parámetro **color** hará la "**máscara**".

- **color_mask: table**



Si empleamos al parámetro **mask_color** como una **tabla** (en este ejemplo, una de dos colores: amarillo y rojo), la función hará la "**máscara**" entre el color principal que es el parámetro **color**, y tonos al azar entre todos los colores de la tabla de colores **mask_color**:

Resumiendo los modos, tenemos:

color.mask (mode, color, color_mask)			
Caso	**color**	**color_mask**	**return**
1	string	string	string
2	string	tabla	string
3	tabla	string	tabla
4	tabla	tabla	tabla

---

alpha.mask(mode, alpha): es similar a la función **color.mask**, con la diferencia que esta función carece del tercer parámetro, y que trabaja con las transparencias (alpha) en lugar de colores.

---

color.movemask(dur, delay, mode, color): esta función es similar a **color.mask**, ya que también genera una "**máscara**", pero ésta da la sensación de movimiento gracias a una serie de transformaciones. Dicho efecto de movimiento es realizado de derecha a izquierda.

El parámetro **dur** es la duración total de la función, o sea la duración total de todas las transformaciones. Puede ser un **número** o una **tabla** con dos números, el primero de ellos indicaría el inicio de la función y el segundo el tiempo final, ambos tiempos son relativos al tiempo de inicio de cada una de las líneas fx generadas.

Las opciones del parámetro **dur** son:

1. dur
2. {time1, time2}

El parámetro **delay** cumple con la misma tarea que su parámetro homónimo en la función **tag.oscill**, y estas son sus opciones:

1. delay
2. { { delay, dur_delay } }
3. { delay, accel }
4. { {delay, dur_delay }, accel }
5. { delay, accel, dilat }
6. { { delay, dur_delay }, accel, dilat }

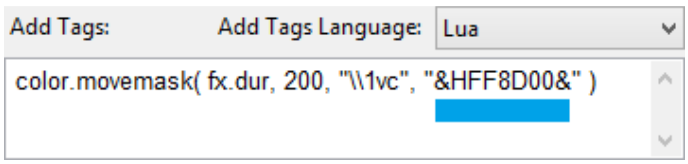
Todas las anteriores variables mencionadas para este parámetro hacen referencia a valores numéricos, es decir, a números.

- **delay**: valor numérico que indica cada cuánto suceden las transformaciones generadas.
- **dur_delay:** valor numérico que indica la duración de cada una de las transformaciones generadas. Su valor por default es **delay**.
- **accel:** valor numérico que indica la aceleración de las transformaciones generadas. Su valor por default es 1.
- **dilat:** valor numérico que indica el tiempo que se va extendiendo cada una de las transformaciones con respecto a la anterior, o sea que extiende la duración. Su valor por default es 0.

El parámetro **mode** hace referencia al tag de color en formato "**vc**" que se retornará en las transformaciones generadas por la función:

1. "\\1vc"
2. "\\3vc"
3. "\\4vc"

El parámetro **color** hace referencia a un **string** de color. A diferencia de la función **color.mask**, acá este parámetro ya no puede ser una **tabla**. Ejemplo:



Del anterior ejemplo tenemos:

- **dur** = fx.dur
- **delay** = 200
- **mode** = "\\1vc"
- **color** = "&HFF8D00"

Las "**máscara**" se genera igual que con **color.mask**, pero ahora se mueve de derecha a izquierda con la duración de cada transformación en 200 ms.

---

alpha.movemask(dur, delay, mode, alpha): es similar a la función **color.movemask**, con la diferencia que trabaja con las transparencias (alpha) en lugar de colores.

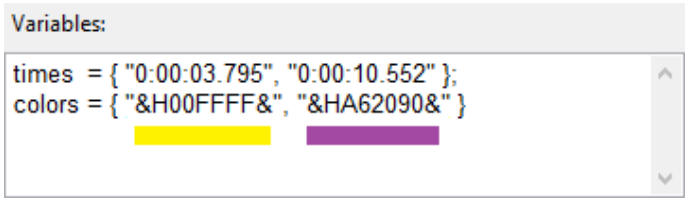
---

color.setmovemask(delay, mode, times, colors): esta función es la combinación de la función **color.set** y la función **color.movemask**, sus parámetros son:

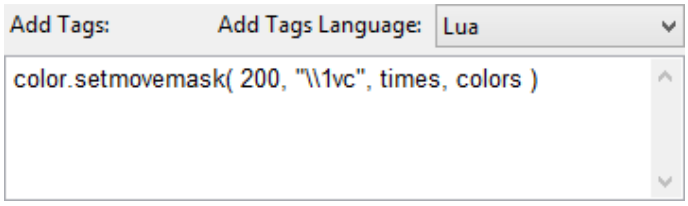
- **delay**: es un valor numérico que hace referencia a la duración de cada una de las transformaciones, al igual, también indica cada cuánto suceden las mismas.
- **mode**: hace referencia al tag de color en formato "**vc**" al que afectará la función:
  1. "\\1vc"
  2. "\\3vc"
  3. "\\4vc"
- **times:** hace referencia a la **tabla** con los tiempos copiados de la parte inferior izquierda del vídeo en el **Aegisub**, pero todos los elementos de la tabla deben ser un string, ya no pueden ser una tabla para modificar el tiempo del cambio de un color a otro.
- **colors:** hace referencia a la **tabla** de colores que hacen pareja con cada uno de los tiempos de la tabla **times**, pero esta vez solo pueden un **string** de color, ya no pueden ser tablas.

Entonces es fácil deducir lo que esta función hace.

Ejemplo:



Y en **Add Tags**:



Entonces, justo antes de la primera transformación se verá algo como esto:

Luego la primera transformación:

---

color.movemaskv(dur, delay, mode, color, color_mask): similar a la función **color.movemask**, pero ahora la "**máscara**" no se moverá de derecha a izquierda, sino de forma vertical, de abajo a arriba. La otra diferencia está en que el parámetro **color** ya no puede ser una **tabla**, solo está habilitado para ser un **string** de color.

El resto de las cualidades y características son todas las mismas, y el uso entre una función y otra, solo depende de los resultados que queremos obtener.

---

alpha.movemaskv(dur, delay, mode, alpha): similar a la función **color.movemaskv**, pero carece del quinto parámetro y que trabaja con las transparencias (alpha) en vez de colores.

color.masktable(color): esta función crea una **tabla** con los colores necesarios para hacer una "**máscara**" dependiendo del **Template Type**, ya que éste determina el tamaño de la **tabla**.

El parámetro **color** puede ser tanto un **string** de color, como una **tabla** de colores. Para el primer caso, retorna una **tabla** con la "**máscara**" de dicho color y para el caso en que el parámetro **color** sea una **tabla**, la función retorna una tabla de tablas, es decir, una **tabla** en donde cada uno de sus elementos es otra tabla, y cada una de ellas contiene la "**máscara**" correspondiente a cada color de la **tabla** ingresada.

---

alpha.masktable(alpha): similar a la función **color.masktable**, pero con la diferencia que trabaja con las transparencias (alpha) en vez de colores.

---

color.module(color1, color2): esta función retorna la interpolación completa entre los colores de los parámetros **color1** y **color2**, con respecto al **loop**.

Los parámetros color1 y color2 pueden ser tanto strings de colores, como tablas de colores:

color.moduke (color1, color2)			
Caso	**color1**	**color2**	**return**
1	string	string	string
2	string	tabla	tabla
3	tabla	string	tabla
4	tabla	tabla	tabla

Caso 1: al usar la función con ambos parámetros **strings**, entonces se retorna un **string** de color correspondiente a la interpolación entre los dos colores. Se puede decir que es la forma más convencional de usar esta función.

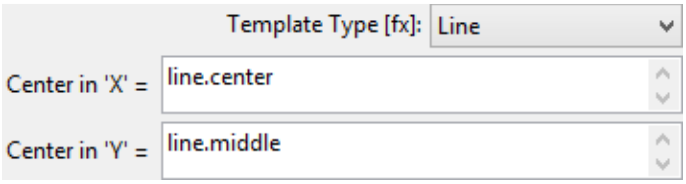
Caso 2: retorna una **tabla** con las interpolaciones entre el color del primer parámetro y cada uno de los colores de la tabla ingresada en el segundo parámetro.

Caso 3: retorna una **tabla** con las interpolaciones entre cada uno de los colores de la tabla ingresada en el primer parámetro y el color del segundo parámetro.

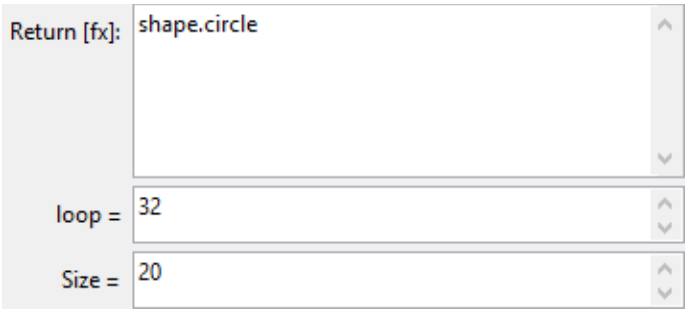
Caso 4: retorna una **tabla** con las interpolaciones entre las posibles combinaciones de los productos cartesianos entre los colores de ambas tablas ingresadas.

Recordemos que todas las interpolaciones en esta función dependen únicamente del **Loop** (**maxj**). Ejemplo:

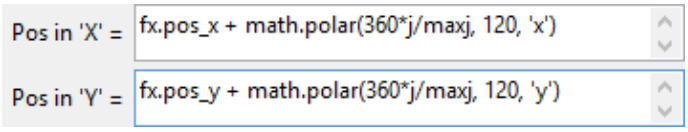
- **Template Type: Line**



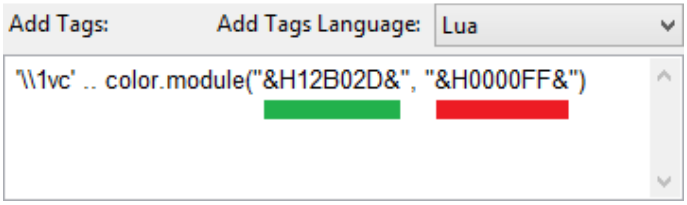
- Configuramos a **Return [fx], loop** y **Size**



- En **Pos in "X"** y **Pos in "Y"** usaremos la función **math.polar** para que los 32 círculos se ubiquen equidistantemente respecto al centro de cada línea karaoke, con un radio de 120 pixeles:

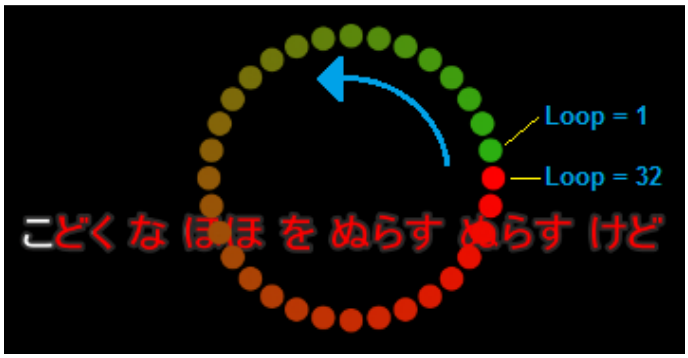


- Por último, en **Add Tags** llamamos a la función de la siguiente forma (a esta altura ya deben estar familiarizados con los métodos de concatenación entre dos **string**):



Entonces la función asignará a cada uno de los **loops** un único color correspondiente a la interpolación entre los dos colores asignados en los parámetros **color1** y **color2**.

Se pueden notar los círculos y el color primario ("\\1vc") asignado a cada uno de ellos:



La función **color.module** se puede usar con los tags de colores de ambos filtros, o sea:

1. \\1c
2. \\3c
3. \\4c
4. \\1vc
5. \\3vc
6. \\4vc

La función **color.module** recibe su nombre gracias a la variable **module** del **Kara Effector**, que recordemos hace referencia a la interpolación de todos los números entre cero y uno, con respecto al **Loop**.

---

alpha.module(alpha1, alpha2): similar a la función **color.module**, pero con la diferencia que trabaja con las transparencias (alpha) en vez de colores.

---

color.module1(color1, color2): esta función es similar a **color.module**, pero retorna la interpolación completa entre los colores de los parámetros **color1** y **color2**, con respecto al **Temaplate Type**.

Como su nombre lo indica, genera la interpolación por medio de la variable **module1**, que hace referencia a la interpolación entre cero y uno sin importar el Loop y teniendo en cuenta al **Template Type** (ejemplo: desde 1 hasta **syl.n**, o desde 1 hasta **word.n**).

Se podría decir que **color.module** interpola a todos los elementos de cada una de las partes de línea karaoke, por

otra parte, la función **color.module1** interpola a la línea de karaoke de principio a fin sin importar el **Loop** que tenga cada una de las parte de la misma.

---

alpha.module1(alpha1, alpha2): similar a la función **color.module1**, pero con la diferencia que trabaja con las transparencias (alpha) en vez de colores.

---

color.module2(color1, color2): esta función es similar a **color.module** y a **color.module1**, pero retorna la interpolación completa entre los colores **color1** y **color2**, con respecto a todas las líneas a las que le se aplica un efecto. Es decir que genera la interpolación desde el primer elemento (char, syl, word y demás) de la primera línea a la que se le aplique un efecto, hasta el último elemento de la última línea a la que se le haya aplicado un efecto.

Esta función hereda su nombre de la variable **module2** del **Kara Effector**, que hace una interpolación de los números entre cero y uno con respecto a la cantidad de líneas a las que se les aplique un efecto.

---

alpha.module2(alpha1, alpha2): similar a la función **color.module2**, pero con la diferencia que trabaja con las transparencias (alpha) en vez de colores.

Con el final de este **Tomo XIII** se dan por concluidas las librerías **color** y **alpha**. 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