

Librerías Color y Alpha [KE]:

Este artículo arrancará con las librerías “color” y “alpha” del **Kara Effector**, ya que sus funciones son muy similares entre sí.

Librerías Color y Alpha [KE]:

Son dos librerías con funciones dedicadas a los colores y a las transparencias (alpha). Algunas de ellas nos servirán para hacer Efectos directamente y otras como apoyo para los mismos.

color.to_RGB(color_or_table): retorna los valores de la cantidad de Rojo, Verde y Azul (Red, Green and Blue) en base decimal, de un color asignado. El formato del color ingresado puede ser .ass o **HTML**

La función retorna los valores en una **tabla**. Ejemplo:

color.to_RGB("&HF4E67A") = {122, 230, 244 }

- **122** (Rojo “7A”)
- **230** (Verde “E6”)
- **244** (Azul “F4”)

Y para el caso que le ingresemos una **tabla** de colores, la función retornará una **tabla**, en donde cada uno de sus elementos son a su vez otra **tabla**, que contienen los valores de Rojo, Verde y Azul de cada uno de los colores:

colores = { “&HF108B4&”, “&H30F304&” }

color.to_RGB(colores) = { {180, 8, 241}, {4, 243, 48} }

color.to_HVS(color_or_table): esta función es similar a **color.to_RGB**, pero con la diferencia que retorna los valores en base decimal del Tono, la Saturación y del Valor (Hue, Saturation and Value). La forma de usar esta función es la misma que la de **color.to_RGB**

color.vc(color_or_table): convierte a un color o a cada uno de los colores de una **tabla**, en formato “vc“, que es el formato de los colores en el **VSFilterMod**.

Ejemplo 1:

- **color.vc("&HFF00A3&")**
- **= (FF00A3, FF00A3, FF00A3, FF00A3)**

Ejemplo 2:

- **colores = { “&HFFFFFF&”, “H000000&” }**
- **color.vc(colores)**
- **= {(FFFFFF,FFFFFF,FFFFFF,FFFFFF), (000000,000000,000000,000000) }**

Es decir, que si ingresamos en la función un solo color, se retornará ese mismo color en formato “vc“. Si ingresamos una **tabla** de colores, retornará una **tabla** con cada uno de los colores en formato “vc“.

alpha.va(alpha_or_table): hace exactamente lo mismo que la función **color.vc**, pero con las transparencias (alpha). Ejemplo:

- **alpha.va("&HFF&") = (FF, FF, FF, FF)**

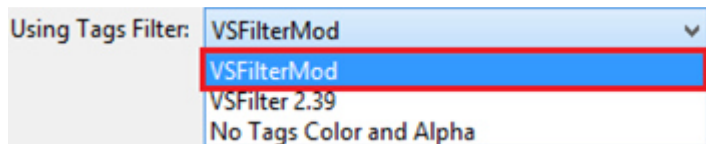
color.r(): retorna un color al azar en formato .ass entre todos los del espectro, incluidos el Blanco, el Negro y todos los matices entre ellos y los demás colores.

alpha.r(): retorna una transparencia (alpha) al azar en formato .ass, entre 0 (totalmente visible) y 255 (invisible).

color.rc(): retorna un color al azar en formato "vc" entre todos los del espectro. El random se ejecuta de manera diferente para cada uno de los cuatro colores que conformarán al color final. Ejemplo:

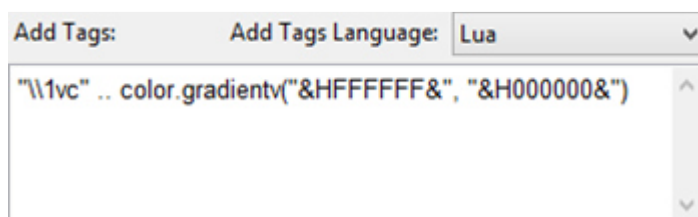


color.gradientv(color_top, color_bottom): esta función es de uso exclusivo del filtro **VSFilterMod**, ya que retorna un tag "vc" (vector color), y para ello es necesario que la opción de dicho filtro esté seleccionada en la primera ventana del **Kara Effector**:

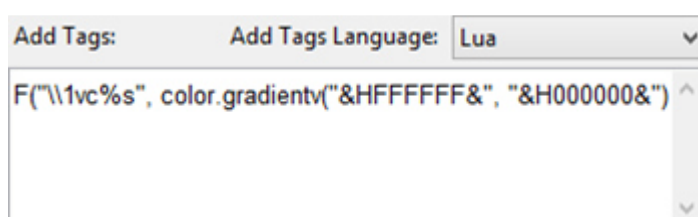


Esta función hace un **gradiente** (degradado) vertical entre dos colores asignados, **color_top** y **color_bottom**.

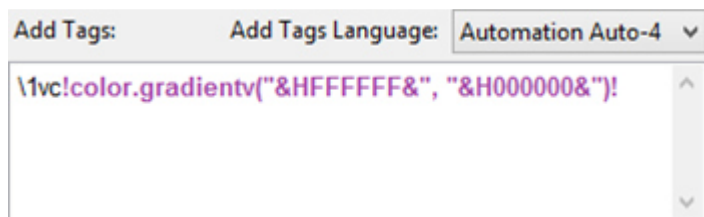
Un ejemplo simple en lenguaje **LUA** sería:



La anterior imagen muestra el método de concatenado en lenguaje **LUA**, pero recordemos que también lo podemos hacer usando la función **string.format** vista muchas veces anteriormente:



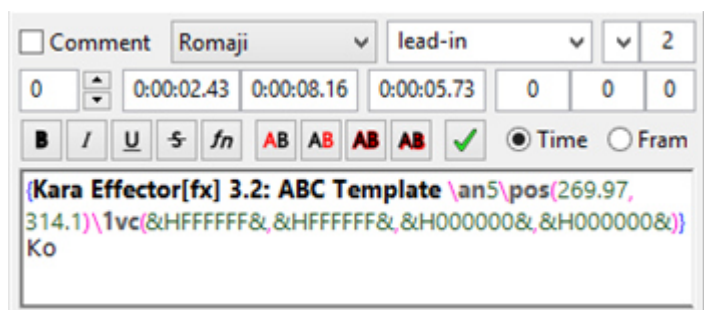
Y este mismo ejemplo en lenguaje **Automation-auto4** es:



Cualquiera de los tres anteriores métodos es válido para hacer Efectos en el **Kara Effector**. Del anterior ejemplo, en pantalla veremos el **gradiente** entre el Blanco y el Negro:



En la siguiente imagen vemos el tag “vc” que hace posible el **gradiente** (\1vc):



En el ejemplo anterior vimos cómo usar la función con ambos parámetros “**string**” (en este caso, colores), pero ambos parámetros también pueden ser **tablas** o en caso de ser necesario, también combinaciones entre un **string** y una **tabla**:

color.gradientv(color_top, color_bottom)			
Caso	color_top	color_bottom	return
1	string	string	string
2	string	tabla	tabla
3	tabla	string	tabla
4	tabla	tabla	tabla

Caso 1: al usar la función con string y string, entonces se retorna un **string** correspondiente al gradiente vertical de los dos strings ingresados.

Caso 2: retorna una **tabla** de gradientes entre el string del primer parámetro y cada uno de los elementos de la tabla ingresada.

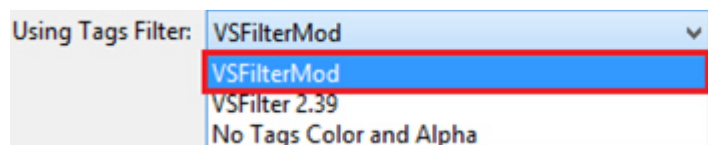
Caso 3: retorna una **tabla** de gradientes entre cada uno de los elementos de la tabla ingresada y el string del segundo parámetro.

Caso 4: retorna una **tabla** de gradientes entre las posibles combinaciones de los productos cartesianos entre los elementos de ambas tablas ingresadas.

alpha.gradientv(alpha_top, alpha_bottom): hace exactamente lo mismo que la función **color.gradientv**, pero con las transparencias (alpha). Ejemplo:

- **alpha.gradientv("&HFF&", "&H44&") = (&HFF&, &HFF&, &H44&, &H44&)**

color.gradienth(color_left, color_right): esta función es de uso exclusivo del filtro **VSFilterMod**, ya que retorna un tag "vc" (vector color), y para ello es necesario que la opción de dicho filtro esté seleccionada en la ventana de inicio del **Kara Effector**:



Esta función hace un **gradiente** (degradado) horizontal entre dos colores asignados, **color_left** y **color_right** (color izquierdo y color derecho).

Las opciones de uso de esta función son los mismos que los de la función **color.gradientv**:

color.gradienth(color_left, color_right)			
Caso	color_left	color_right	return
1	string	string	string
2	string	tabla	tabla
3	tabla	string	tabla
4	tabla	tabla	tabla

Veamos un ejemplo de cómo usar esta función:



alpha.gradienth(alpha_left, alpha_right): hace exactamente lo mismo que la función **color.gradienth**, pero con las transparencias (alpha). Ejemplo:

- **alpha.gradienth("&HAA&", "&H00&") = (&HAA&, &H00&, &HAA&, &H00&)**

Las anteriores cuatro funciones guardan relación cercana con los gradientes, las dos primeras con los verticales y las dos siguientes con los horizontales.

color.vc_to_c(color_vc): esta función convierte al color ingresado, de formato "vc" (**VSFilterMod**) a formato "c" (**VSFilter 2.39**). El parámetro **color_vc** puede ser un **string** o una **tabla** de **string**, de manera que si es un **string**, la función retornará a ese color cambiado de un formato al otro, y si es una **tabla**, retornará una **tabla** con cada uno de los colores de la misma, convertidos en formato "c".

color.vc_to_c(color_vc)		
Caso	color_vc	return
1	string	string
2	tabla	tabla

alpha.va_to_a(alpha_va): hace lo mismo que la función **color.vc_to_c**, pero con las transparencias (alpha).

color.c_to_vc(color_c): es la función opuesta a **color_c_to_vc**. Convierte al color ingresado, de formato “c” (VSFilter 2.39) a formato “vc” (VSFilterMod).

Los modos son los mismos que los de su función opuesta:

color.c_to_vc(color_c)		
Caso	color_c	return
1	string	string
2	tabla	tabla

alpha.a_to_va(alpha_a): hace lo mismo que la función **color.c_to_vc**, pero con las transparencias (alpha).

color.interpolate(color1, color2, index_ipol): es similar a la función **_G.interpolate_color** del **Automation**

Auto-4, pero con la diferencia que tanto **color1** como **color2** pueden ser o un **string** o una **tabla**.

El parámetro **index_ipol** es opcional y es un número real entre 0 y 1 inclusive. Su valor por default es 0.5 con el fin de que la función retorne el color intermedio entre **color1** y **color2**. Si **index_ipol** es cero o cercano a él, la función retorna un color cercano a **color1**, y si es 1 o cercano a 1, retornará un color cercano al de parámetro **color2**, o sea que el color que se retornará dependerá únicamente de este valor (**index_ipol**).

Los modos de usar esta función son los siguientes:

color.interpolate(color1, color2, index_ipol)			
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 string y string, entonces se retorna un **string** correspondiente a la interpolación entre los dos colores, dependiendo del valor de **index_ipol**, o justo el color intermedio, en el caso de que no exista dicho parámetro.

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.

alpha.interpolate(alpha1, alpha2, index_ipol): similar a la función **_G.interpolate_alpha** del **Automation**

Auto-4. Hace lo mismo que la función **color.interpolate**, pero con las transparencias (alpha).

color.delay(time_i, delay, color_i, color_f, ...): es una función que convierte progresivamente al parámetro **color_i** (color inicial) al parámetro **color_f** (color final). Es exclusiva del filtro **VSFilterMod** ya que retorna un color en formato “**vc**” seguido de cuatro transformaciones que contienen colores también en formato “**vc**”. Las tres primeras transformaciones son colores al azar entre **color_i** y **color_f** y la última contiene a **color_f**.

El parámetro **time_i** es el tiempo relativo al tiempo de inicio de la línea fx y corresponde al momento en que iniciarán las transformaciones.

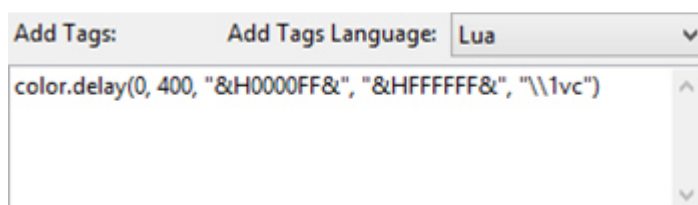
El parámetro **delay** es la duración total de las cuatro transformaciones que retorna la función.

El parámetro (...) hace referencia a uno o más tags de colores “vc” que eventualmente retornará la función. Las posibilidades serían:

- “\\1vc”
- “\\3vc”
- “\\4vc”
- “\\1vc”, “\\3vc”
- “\\1vc”, “\\4vc”
- “\\3vc”, “\\4vc”
- “\\1vc”, “\\3vc”, “\\4vc”

Ejemplos:

- **color.delay**(0, 600, “&HFFFFFF&”, “&H000000&”, “\\1vc”)
- **color.delay**(1200, 2000, “&HFF00FF&”, “&H00FF00&”, “\\1vc”, “\\3vc”)
- **color.delay**(500, 1000, “&HFF0000&”, “&H00FFFF&”, “\\3vc”)



alpha.delay(time_i, delay, alpha_i, alpha_f, ...): hace lo mismo que la función **color.delay**, pero con las transparencias (alpha).

color.movedelay(dur, delay, mode, ...): es similar a la función **color.delay**, pero con la diferencia que la cantidad de transformaciones que retorna depende del parámetro **dur**, que es la duración total de la función. La otra diferencia es que en esta función se pueden ingresar la cantidad de colores que deseemos o también podemos poner como cuarto parámetro a una tabla de colores.

El parámetro **dur** es la duración total de todas las transformaciones que retorna la función.

El parámetro **delay** es la duración que tendrá cada una de las transformaciones retornadas por la función.

El parámetro **mode** es un número entero que indica a cuál o a cuáles tags de colores “vc” se aplicará la función. Pueden ser:

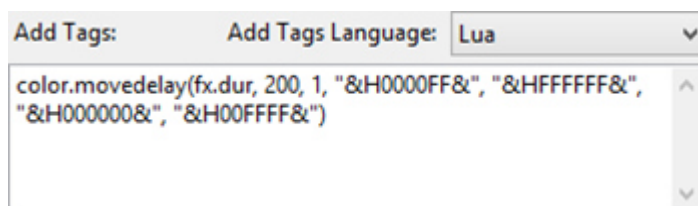
- 1 = \\1vc
- 3 = \\3vc
- 4 = \\4vc
- 13 = \\1vc\\3vc
- 14 = \\1vc\\4vc
- 34 = \\3vc\\4vc
- 134 = \\1vc\\3vc\\4vc

Si se quiere, el parámetro **mode** también se puede usar como un **string** que indique el o los tags a los que queremos que se aplique la función, ejemplo:

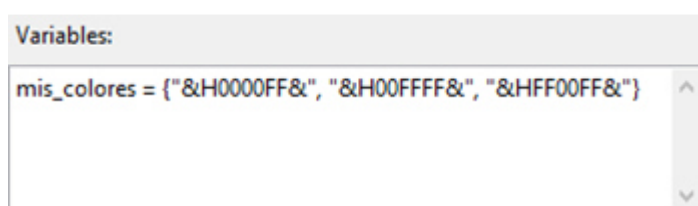
mode = “\\1vc\\3vc”

El parámetro (...) hace referencia a los colores ingresados. Puede ser:

- Un solo color
- Una serie de colores (separados por comas):



- Una tabla de colores:



Add Tags: Add Tags Language: Lua ▼

`color.movedelay(fx.dur, 500, 3, mis_colores)`

color.set(times, colors, ...): esta función retorna una o más transformaciones de colores de la tabla **colors** con respecto a los tiempos de la tabla **times**.

Todos los tiempos de la tabla **times** están medidos con referencia al cero absoluto del vídeo y a cada uno de los tiempos de esta tabla le debe corresponder un color de la tabla **colors**.

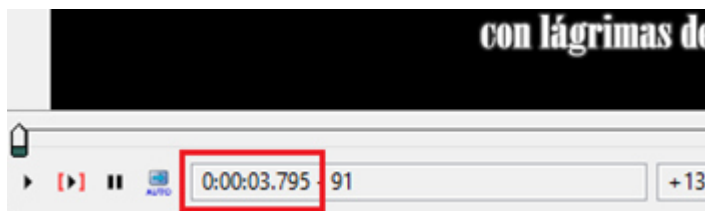
El parámetro (...) hace referencia al tag o tags de colores a los que afectarán las transformaciones que retornará la función. Las posibilidades serían:

- “\\1vc”
- “\\3vc”
- “\\4vc”
- “\\1vc”, “\\3vc”
- “\\1vc”, “\\4vc”
- “\\3vc”, “\\4vc”
- “\\1vc”, “\\3vc”, “\\4vc”

Dado que esta función no es exclusiva de ninguno de los filtros que se pueden usar en el **Aegisub**, entonces las posibilidades de extenderían a:

- “\\1c”
- “\\3c”
- “\\4c”
- “\\1c”, “\\3c”
- “\\1c”, “\\4c”
- “\\3c”, “\\4c”
- “\\1c”, “\\3c”, “\\4c”

Cada uno de los tiempos de la tabla del parámetro **times** deben ser copiados de la parte inferior izquierda del vídeo en el **Aegisub**. Dichos tiempos están en formato **HMSms** (Horas, Minutos, Segundo y milisegundos). Ejemplo:



Los tiempos deben ser pegados en la tabla **times** usando comillas simples o dobles para que el **Kara Effector** pueda reconocerlos a cada uno de ellos como un **string**.

Esta sería una opción para definir los dos parámetros en la celda de texto “**Variables**”:

```
Variables:
times = {"0:00:03.795", "0:00:10.552"};
colors = {"&H00FFFF&", "&HFF00FF"}
```

Notamos cómo ambas tablas tienen la misma cantidad de elementos. La cantidad total de transformaciones que la función retornará depende de cada uno de nosotros, puede ser mínimo un elemento y máximo hasta donde lleguemos a necesitar.

Al definir las tablas **times** y **colors** en “**Variables**” (el nombre con que se definen las tablas es decisión de cada uno), usaremos la función en **Add Tags**, de alguna de las siguientes dos maneras:

1. Indicando el tipo de variable que será cada una de las dos tablas:

```
Add Tags:      Add Tags Language: Lua
color.set( var.line.times, var.line.colors, "\\1c" )
```

1. Como usamos punto y coma (;) para separar las tablas al definir las en “**Variables**”, entonces las podemos llamar con tan solo escribir el nombre con que las definimos:

```
Add Tags:      Add Tags Language: Lua
color.set( times, colors, "\\1c" )
```

Otra forma de usar la función es ingresando directamente las tablas dentro de ella, sin necesidad de definirla:

```
Add Tags:      Add Tags Language: Lua
color.set( {"0:00:03.795", "0:00:10.552"}, {"&H00FFFF&", "&HFF00FF"}, "\\3c")
```

Cuando algún tiempo de la tabla **times** está definido como un **string** (como en los ejemplos anteriores), entonces la transformación correspondiente a ese tiempo tendrá su duración por default que es de 1 ms. Es decir que el cambio de un color a otro será instantáneo.

Si queremos que alguna transformación tenga una duración superior a la que tiene por default (1 ms), hay dos opciones posibles para poderlo hacer.

1. Convertir a dicho tiempo en otra **tabla** con dos elementos, y dentro de ella poner los tiempos de inicio y final de la transformación, ejemplo:

```
Variables:
times = {"0:00:03.795", {"0:00:10.552", "0:00:11.052"} };
colors = {"&H00FFFF&", "&HFF00FF&"}
```

En este caso, la transformación tendrá una duración de 500 ms que es la diferencia entre los dos tiempos de la tabla del segundo elemento de la tabla **times**. Esta opción tiene la ventaja de poner los dos tiempos sin la necesidad de calcular previamente la duración de la transformación.

1. Convertir a dicho tiempo en otra **tabla** con dos elementos, y dentro de ella poner el tiempo de inicio de la transformación y la duración en ms de la misma, ejemplo:

```
Variables:
times = {"0:00:03.795", {"0:00:10.552", 360} };
colors = {"&H00FFFF&", "&HFF00FF&"}
```

Haciéndolo de este modo, la transformación tendrá una duración de 360 ms, que es el valor que pone el segundo elemento de la tabla correspondiente al segundo tiempo de la tabla **times**. La ventaja de esta opción es poner la duración de la transformación sin poner el tiempo final.

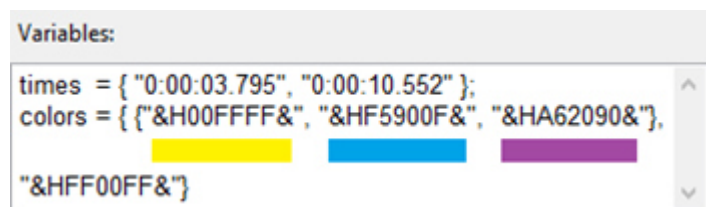
Los anteriores dos métodos son válidos para cualquiera o para todos los elementos de la tabla **times**, y como ya lo había mencionado, son usados para modificar la duración por default de las transformaciones que se retornarán al usar la función.

Los colores de la tabla **colors** pueden estar todos en formato del filtro **VSFilter 2.39**, del **VSFilterMod** o una combinación de ambos, si así se quiere. Ejemplo:

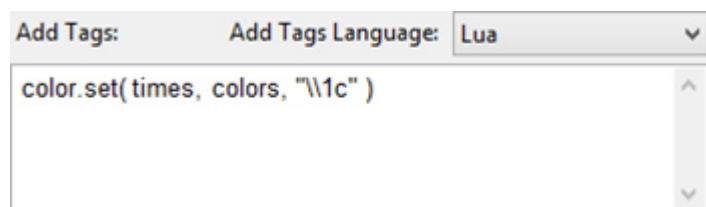
```
Variables:
times = {"0:00:03.795", "0:00:10.552"};
colors = { "(&H00FFFF&,&H00FFFF&,&HAA00D8&,&HAA00D8&)", "&HFF00FF&" };
```

Del ejemplo anterior vemos cómo el primer color está en formato “**vc**” y el segundo está en formato normal. Es obvio que para que los colores salgan tal cual como los pusimos en formato “**vc**”, la opción del filtro **VSTFilterMod** debe estar seleccionada en la ventana de inicio del **Kara Effector**.

Un tercer formato que puede tener uno o más colores de la tabla **colors** es el de una **tabla** de colores. Ejemplo:



En el anterior ejemplo, el primer elemento de la tabla **colors** es una tabla de 3 colores (amarillo, azul y morado), pero la cantidad puede ser la que nosotros convengamos. La función la usaremos en **Add Tags** del mismo modo ya aprendido hasta este momento:



En la siguiente imagen vemos el texto justo antes de que ocurra la primera transformación:



Y lo que sucederá en la primera transformación, dado que el primer elemento de la tabla **colors** es una tabla de colores y que el ejemplo se está haciendo en el modo **Template Type: Syl**, es que a cada sílaba le corresponderá un color de esa tabla; pero como dicha tabla solo tiene 3 colores, entonces a la cuarta sílaba se le volverá a asignar el color 1 de la tabla, lo mismo que a las séptima sílaba y así sucesivamente hasta completar todas las sílabas:



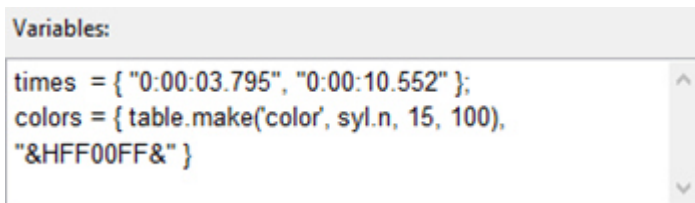
En la próxima imagen veremos el texto justo antes de la tercera transformación:



Y así se verá el texto justo después del cambio de color, ya que el segundo elemento de la tabla **colors** era el color magenta (&HFF00FF&):



Esta tercera habilidad que tienen los elementos de la tabla de **colors** de poder ser una tabla de colores nos da muchas posibilidades, como hacer “**máscaras**” o **degradaciones** en una o más de las transformaciones que retorna la función. Ejemplo:



El primer elemento de la tabla **colors** está determinado por la función **table.make** así:

- El modo “**color**” creará una tabla de colores.
- **syl.n** determinará el tamaño de dicha tabla.
- 15 y 100 serán los límites inferior y superior de la degradación con respecto al círculo cromático de la teoría del color **HSV**. Los 15° corresponden un tono naranja rojizo de dicho círculo, y los 100° a un tono verde limón.

De este modo, la función hará que el texto cambie a la degradación anteriormente descrita:



Y para la segunda transformación el texto pasará al color que se haya asignado como segundo elemento en la tabla **colors** (magenta = &HFF00FF&):



Todo es cuestión de poner a volar un poco la imaginación y empezar a experimentar con las distintas opciones que nos ofrece esta función, y las posibilidades son casi que infinitas al igual que los resultados.

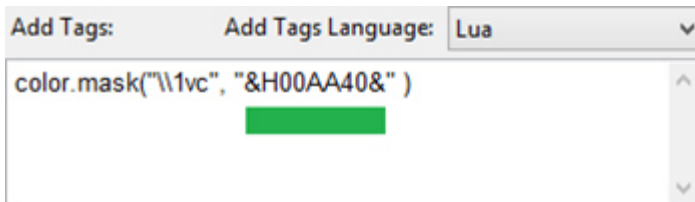
alpha.set(times, alphas, ...): esta función hace lo exactamente lo mismo que la función **color.set**, pero usa las transparencias (alpha) en vez de colores.

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:

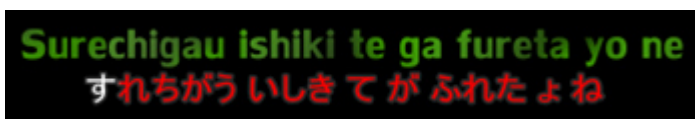
- “\\1vc”
- “\\3vc”
- “\\4vc”

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**”:

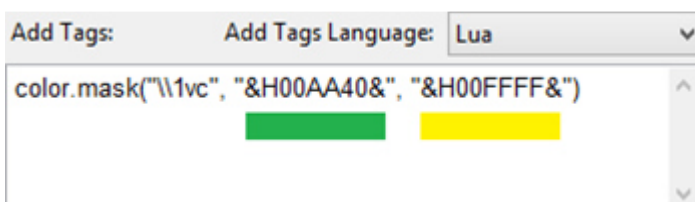


La función **color.mask** hace por default la “**mascara**” con el color ingresado en el parámetro **color** y los tonos al azar entre el blanco y el negro.

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**

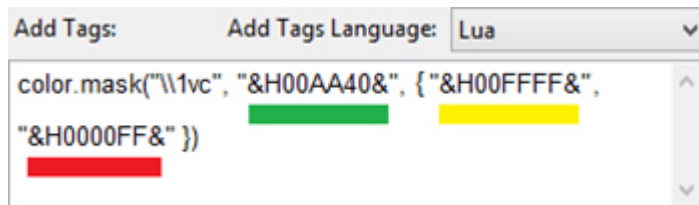


De este modo, la función ya no hace la “**máscara**” con los tonos por default entre el blanco y el negro, sino que la hace entre el parámetro **color** y el color ingresado en el parámetro **color_mask** (verde y amarillo):



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 }

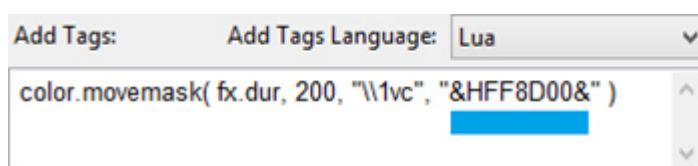
Todas las anteriores variables mencionadas para este parámetro hacen referencia a valores numéricos, es decir, a números.

- **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:

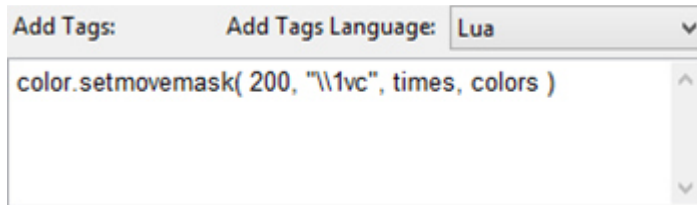
- **delay**: es un valor numérico que hace referencia a la duración de cada una de las transformaciones, al igual, también indica cada cuánto suceden las mismas.
- **mode**: hace referencia al tag de color en formato “vc” al que afectará la función:
 1. “\\1vc”
 2. “\\3vc”
 3. “\\4vc”
- **times**: hace 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.module (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**

Template Type [fx]:

Line

Center in 'X' =

line.center

Center in 'Y' =

line.middle

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

Return [fx]:

shape.circle

loop =

32

Size =

20

- 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:

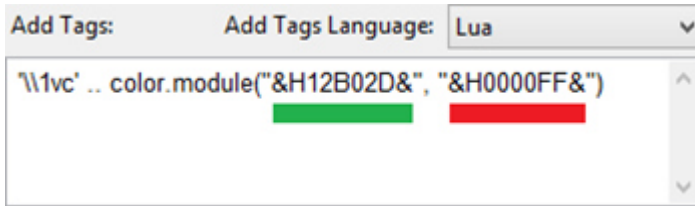
Pos in 'X' =

fx.pos_x + math.polar(360*j/maxj, 120, 'x')

Pos in 'Y' =

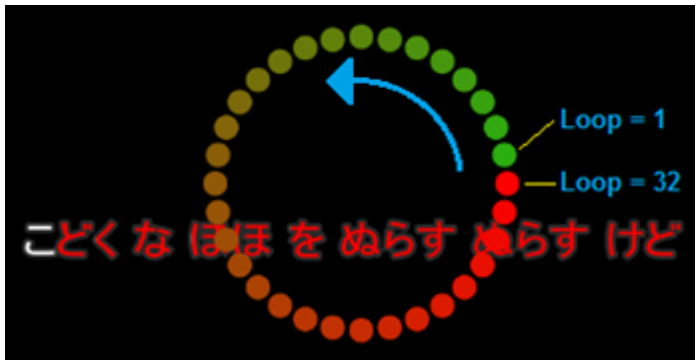
fx.pos_y + math.polar(360*j/maxj, 120, 'y')

- 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 **Template 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**

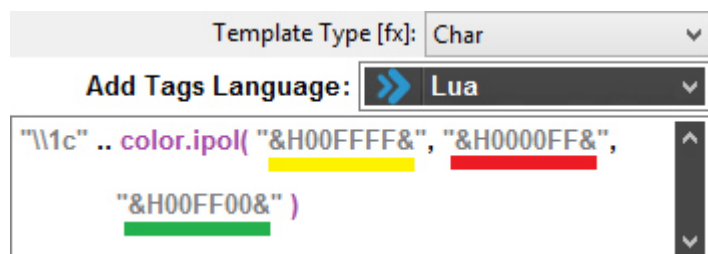
Effector, que hace una interpolación de los números entre cero y uno con respecto a la cantidad de líneas a las que se les aplique un efecto.

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

color.ipol(...):

Esta función crea un gradiente (**degradación**) entre dos o más colores ingresados en ella, para posteriormente asignarle un único color a cada uno de los elementos de la línea.

Ejemplo:

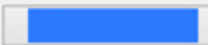

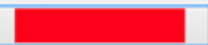


Al aplicar, notamos cómo cada una de las letras, dado que el **Template Type** es **Char**, hace el gradiente ente los tres colores, empezando por el amarillo, pasando por el rojo y llegando hasta el verde:



Ejemplo:

Podemos sacar provecho de uno o más de los colores que en principio son para las Shapes:

Shape Primary Color	Shape Border Color	Shape Shadow Color
		
0	0	0

Template Type [fx]:	Char
Add Tags Language:	Lua
<pre>"\1c" .. color.ipol("&H00FF00&", shape.color1, shape.color3, shape.color4)</pre>	

Usamos 4 colores, primero el verde, y luego los tres colores que modificamos en la segunda Ventana del **KE**, y luego de aplicar veremos algo muy similar a esto:



color.loop(...) :

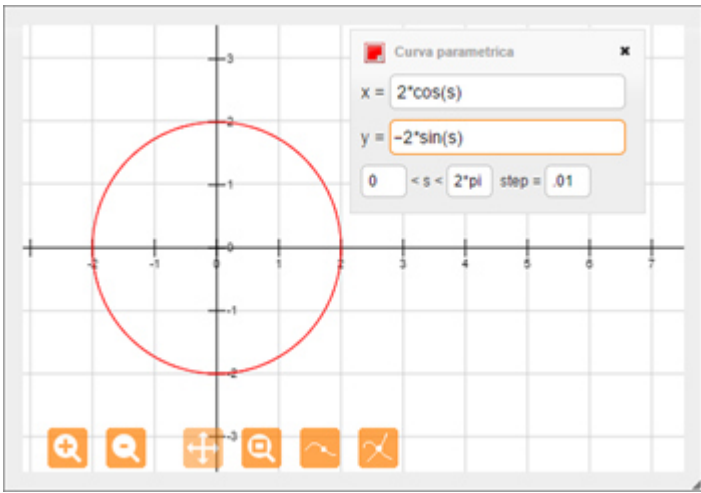
Esta función es similar a la anteriormente vista, pero con la diferencia de que crea el gradiente (**degradación**) entre todos los colores ingresados respecto al **loop** y no a los componentes karaokes de la línea.

Ejemplo:

Seleccionamos un **Template Type: Line**

Template Type [fx]:	Line
Center in "X" =	line.center
Center in "Y" =	line.middle

Ingresaremos la siguiente función paramétrica:



Aumentamos un poco la escala, ya que un círculo de 2 px de radio sería muy pequeño:

x(s) =	132*cos(s)
y(s) =	-132*sin(s)
s =	0 to 2*pi

Aumentamos el **loop**, modificamos el tamaño de la shape y ponemos **shape.circle** en **Return [fx]:**

loop =	42
Size =	16
Return [fx]:	shape.circle

Y usando los mismos cuatro colores del ejemplo anterior, llamamos a la función **color.loop**:

Shape Primary Color	Shape Border Color	Shape Shadow Color
0	0	0

Add Tags Language: > Lua

```
"\1c" .. color.loop( "&H00FF00&", shape.color1,
shape.color3, shape.color4 )
```

Los resultados son:



Entonces la función genera el gradiente a través de los 42 **loops** que habíamos puesto de la **shape** **shape.circle**, empezando por el verde, pasando por el azul y el amarillo y terminando en el rojo.