Figuras básicas

Dibujar figuras básicas

Cualquier dibujo o figura que dibujemos con SVG debe tener su código dentro de la etiqueta `<svg> ... </svg>`, hecho que daremos por supuesto en algunos de los ejemplos que aquí pongamos. No hay que olvidar que debemos indicar también la altura y la anchura del contenedor que genera esta etiqueta.

Los dibujos que hagamos en esta página estatán todos incluídos dentro de una etiqueta "svg" como ésta:

> <svg width=200 height=200> ... código del dibujo ... </svg>

En la etiqueta indicamos la altura y anchura del contenedor del dibujo. Además para que encaje en nuestra página hemos puesto el siguiente código CSS en la hoja de estilos:

svg { margin: 1em 3em 1em 1em; float: right; border: 1px solid black; }

Dibujar un círculo

Para dibujar un círculo básico pondremos el siguiente código:

```
<svg width=200 height=200>
<circle cx=100 cy=100 r=90 stroke="blue" stroke-width=2 fill="orange">
</svg>
```

Explicamos a continuación el código. En él hay unos elementos que son imprescindibles:

•`circle`: Esta etiqueta con este nombre indica que se dibujará un círculo.

•`cx=n cy=n`: Los atributos `cx` y `cy` indican las coordenadas del centro del círculo. con `cx` se indica la distancia horizontal en pixeles desde el borde izquierdo del contenedor al centro del círculo. `cy` es la distancia en píxeles desde el borde superior al centro del círculo.

•`r=n`: Con este atributo indicamos la longitud del radio de la circunferencia medido en píxeles.

El resto de atributos definen el estilo y se pueden utilizar con otras figuras:

•`stroke="color"`: indica el color de la línea exterior que delimita a la figura. Como valor se pone un color. Este puede ir escrito en cualquier nomenclatura aceptada por HTML5

•`stroke-width=n`: indica el grosor de la línea exterior que delimita a la figura. El valor será un número que indica el número de pixeles.

•`fill="color"`: indica el color de relleno del interior de la figura. Como valor se pone un color que puede ir escrito en cualquier nomenclatura aceptada por HTML5.

Cabe destacar que en todo contenedor SVG tenemos un origen de coordenadas que en principio estará en la esquina superior derecha con el valor 0,0. Desde ahí se miden los pixeles para trazar los elementos del dibujo, tal como hemos hecho aquí para buscar el centro del círculo.

Todos los valores numéricos que indican distancia, de no indicar otra cosa se miden en píxeles. También podemos ponerlos en porcentaje siempre y cuando lo indiquemos. Esto es válido también para el resto de figuras que hagamos.

Cuadrados y rectángulos:

Para dibujar un rectángulo lo haremos con el siguiente código:

```
<svg width=200 height=200>
<rect x=10 y=30 width=180 height=140 stroke="green" stroke-width=2
fill="lime"/>
</svg>
```

Como en el dibujo anterior hemos utilizado los atributos `stroke`,`stroke-width` y `fill` para indicar el color del borde, el grosor del borde y el color de relleno respectivamente. Los demás elementos de la etiqueta son los siguientes:

•`rect`: con esta etiqueta indicamos que vamos a dibujar un rectángulo.

•`x=n y=n`: indicamos en estos atributos las coordenadas de la esquina superior izquierda del rectángulo. `x` es la coordenada horizontal e `y` es la coordenada vertical.

•`width=n height=n`: Indicamos aquí la anchura y altura del rectángulo. con `width` indicamos la anchura y con `height` la altura.

Podemos también dibujar el rectángulo con las esquinas redondeadas, para ello sólo tenemos que añadir los atributos `rx` y`ry` cuyo valor será el radio horizontal y vertical de la esquina.

El ejemplo que vemos a la derecha tiene el siguiente código:

```
<svg width=200 height=200>
<rect x=10 y=40 rx=20 ry=20 width=180 height=120 fill="orange" stroke="purple" stroke-
width=2 />
<svg>
```

Para dibujar un cuadrado no tenemos más que dibujar un rectángulo en el que la anchura y la altura sean iguales.

Dibujar una elipse

Para dibujar una elipse utilizaremos el siguiente código:

```
<svg height="200" width="200">
  <ellipse cx=100 cy=100 rx=95 ry=60
  stroke="red" stroke-width=2 fill="yellow" />
</svg>
```

Como en las figuras anteriores utilizamos los atributos `stroke`, `stroke-width` y `fill` para indicar el color del borde, el grosor del borde, y el color de relleno.

El resto de elementos de esta etiqueta son:

•`ellipse`: el nombre de la etiqueta indica que lo que dibujamos es una elipse.

•`cx=n cy=n`: coordenadas del centro de la elipse. `cx` es la coordenada horizontal y `cy` es la coordenada vertical.

•`rx=n`: radio de la elipse en el plano horizontal.

•`ry=n`: radio de la elipse en el plano vertical.

Polígonos

Para dibujar un polígono cualquiera utilizaremos un código como el siguiente:

```
<svg height="200" width="200">
  <polygon points="100,10 190,190 10,190"
  stroke="purple" stroke-width=2 fill="aqua"/>
</svg>
```

Como en las figuras anteriores indicamos el borde y el relleno con los atributos `stroke`, `stroke-width` y `fill`.

El nombre de la etiqueta `polygon` indica que lo que estamos dibujando es un polígono.

El atributo `points` indica los puntos de las esquinas del polígono. Escribimos aquí una serie de números por parejas. Los dos números de la pareja van separados por una coma, y cada pareja se separa de las demás por espacios en blanco. La primera pareja de números indica las coordenadas x e y respectivamente del primer punto. La segunda pareja indica las coordenadas x e y del segundo punto, y así sucesivamente, con cada pareja de números más que pongamos indicamos las coordenadas del siguiente punto. Al llegar al último punto el polígono busca el primer punto para cerrarse automáticamente.

Al ir poniendo los puntos se va trazando la línea que delimita el polígono, por lo que es importante el orden en el que éstos se ponen para trazar la línea correctamente. Por otra parte debemos poner un mínimo de tres puntos para que el polígono tenga una superficie.

Líneas rectas

Para dibujar una línea recta lo haremos con un código como el siguiente:

```
<svg height=200 width=200>
  <line x1=5 y1=5 x2=190 y2=190 stroke="red" stroke-width=2/>
</svg>
```

Los atributos `stroke` y `stroke-width` indican el color y grosor de la línea respectivamente.

El nombre de la etiqueta `line` indica que lo que dibujamos es una línea.

los atributos `x1` y `y1` indican respectivamente las coordenadas horizontal y vertical del principio de la línea.

los atributos `x2` y `y2` indican respectivamente las coordenadas horizontal y vertical del final de la línea.

La línea se traza desde el punto indicado como principio al punto indicado como final.

Estilo de líneas

Podemos poner un remate al final de la línea con el atributo `stroke-linecap`, el cual puede tener los siguientes valores:

•`stroke-linecap="butt"`: valor por defecto, la línea se queda como está.

•`stroke-linecap="round"`: añade un semicírculo al final de la línea, poniendo un remate redondeado.

•`stroke-linecap="square"`: añade medio cuadrado al final de la línea, da un remate cuadrado.

Para poder apreciar las diferencias las líneas deben ser bastante gruesas como para que se vea el remate.

El atributo `stroke-dasharray` permite hacer líneas discontinuas. Como valor se ponen una o varias parejas de números. En cada par de números el primero es la longitud de línea que se ve y el segundo la longitud que está oculta. El patron se repite a lo largo de toda la línea.

Por ejemplo la ultima línea del ejemplo de la derecha tiene el siguiente atributo `stroke-dasharray="12,4 2,4 2,4"`.

El contenedor SVG de la derecha muestra una serie de líneas con estos atributos. su código es el siguiente:

```
<svg width=200 height=200>
<line x1=20 y1=20 x2=180 y2=20 stroke="navy" stroke-width=10 stroke-
linecap="butt"/>
<line x1=20 y1=50 x2=180 y2=50 stroke="navy" stroke-width=10 stroke-
linecap="round"/>
<line x1=20 y1=80 x2=180 y2=80 stroke="navy" stroke-width=10 stroke-
linecap="square"/>
<line x1=20 y1=110 x2=180 y2=110 stroke= navy stroke-with=2 stroke-
dasharray="2,2"/>
<line x1=20 y1=140 x2=180 y2=140 stroke= navy stroke-with=2 stroke-
dasharray="6,6"/>
<line x1=20 y1=170 x2=180 y2=170 stroke= navy stroke-with=2 stroke-
dasharray="12,4 2,4 2,4"/>
</svg>
```

Polilíneas

Una polilínea es una sucesión de líneas rectas en la que cada una empieza donde acaba la anterior. También se le llama línea quebrada. Para dibujar una polilínea pondremos un código como el siguiente:

```
<svg height=200 width=200>
  <polyline points="10,50 50,50 50,90 90,90 90,130 130,130 130,170"
        stroke="red" stroke-width=4 fill="yellow" />
</svg>
```

El nombre de la etiqueta `polyline` indica que se traza una polilínea.

Los atributos `stroke` y `stroke-width` nos dan el color y el grosor de la línea.

El atributo `fill` crea un relleno en la figura. Para ello traza una línea recta del primer punto al último y rellena con el color indicado las zonas que quedan dentro.

El atributo `points` funciona como en el polígono, es decir, cada pareja de números indica las coordenadas de un punto. Al contrario que en el polígono, aquí el último punto no se cierra con el primero.

Dibujar rutas

definir una ruta

Una ruta o camino (path en inglés) es un trazado de una o varias líneas que se hacen una detrás de otra. Es como cuando dibujamos con un lápiz. Aquí vamos indicando el movimiento del lapiz de un punto a otro para trazar el dibujo.

Definir una ruta en SVG se hace mediante la etiqueta `path` y el atributo `d`.

<path d="..comandos de la ruta .." />

Además como en las figuras vistas anteriormente debemos poner también los atributos de estilo para poder verlo en pantalla, es decir `stroke` para el color de las líneas, `stroke width` para el grosor de las líneas y `fill` para el color de relleno.

Respecto al color de relleno (atributo `fill`) hay que indicar que la mayoría de los navegadores ponen por defecto el color negro, por lo que si queremos otro color hay que indicarlo.

Los elementos que pueden trazarse en una ruta no se limitan a simples líneas rectas, sino que podemos poner también varios tipos de curvas.

Al trazar la ruta el lapiz o puntero tiene unas coordenadas determinadas, las cuales vienen indicadas siempre por el último punto que hemos marcado.

Los comandos de la ruta, es decir lo que ponemos como valor del atributo `d` consiste en un serie de letras. Cada letra indica un tipo de trazado (línea recta, curva, mano alzada, etc) y va seguida de una serie de números que indican la posición y caracteríticas del trazado.

Los comandos que podemos poner en la ruta son las siguientes letras:

- **`M`** : El lápiz o puntero se sitúa en la posición indicada.
- **`L`** : Traza una línea recta desde la posición actual a la indicada.
- **`H`** : Traza una línea recta horizontal desde la posición actual a la indicada.

- **V** : Traza una línea recta vertical desde la posición actual a la indicada.
- **C** : Traza una línea curva de Bézier desde la posición actual a la indicada.
- **S** : Traza una línea curva de Bézier desde la posición actual a la indicada. La curvatura de la línea es aquí la misma que en la anterior línea de Bézier.
- **Q** : Traza una línea curva cuadrática de Bézier desde la posición actual a la indicada.
- **T** : Traza una línea curva cuadrática de Bézier desde la posición actual a la indicada. La curvatura de la línea es aquí la misma que en la anterior línea cuatrática de Bézier.
- **A** : Traza un arco elíptico desde la posición actual a la indicada.
- **Z** : Cierra la ruta actual.

todos estos comandos pueden escribirse en mayúsculas o en minúsculas. Si escribimos la letra en mayúsculas, se toma una posición absoluta y las coordenadas se miden desde su origen. Si escribimos la letra en minúsculas se toma una posición relativa y las coordenadas se miden desde el último punto que se ha puesto en la ruta.

Cada comando va seguido de una serie de números, llamados parámetros, que idican o bien unas coordenadas o las características del elemento trazado.

Vemos a continuación todo esto de una forma mucho más detallada, explicando estos comandos uno a uno.

Principio y fin de la ruta

Al iniciar una ruta lo normal es que ésta no empiece en el origen de coordenadas, por lo que debemos mover el lápiz a mano alzada hasta el punto de inicio. Para ello utilizamos el comando `M`.

Los dos números o parámetros que siguen a la letra M son las nuevas coordenadas x e y del lápiz o puntero. El próximo trazo que indiquemos mediante un comando empezará en este punto.

El comando Z cierra la ruta y no lleva parámetros. Es por tanto indiferente escribirlo en mayúscula o en minúscula.

líneas horizontales y verticales.

La línea horizontal se define mediante el comando H. Éste lleva un sólo parámetro que es la nueva posición de coordenada "x". Al ser línea horizontal la coordenada "y" sigue siendo la misma y no hace falta indicarla.

La línea vertical se define mediante el comando V. Éste lleva un sólo parámetro que es la nueva posición de coordenada "y". Al ser línea vertical la coordenada "x" sigue siendo la misma y no hace falta indicarla.

Veamos un ejemplo en el que dibujamos varias líneas horizontales y verticales

```
<svg width=200 height=200>
<path d="M 25 25 h 50 v 50 h 50 v 50 h 50 v 50 m 0 0 z"
       stroke="navy" stroke-width=3 fill="white"/>
</svg>
```

Empezamos la ruta posicionando el puntero al principio (`M 25 25`).

Seguimos trazando la ruta mediante una serie de líneas horizontales y verticales (`h 50 v 50 h 50 v 50 h 50 v 50`). Al poner los comandos en minúsculas la posición es relativa, por lo que sólo tenemos que marcar la longitud de la línea.

Después volvemos a posicionar el puntero en el último punto (`m 0 0`). Esto es importante antes de cerrar la ruta, ya que si no lo hacemos el puntero volverá automáticamente al principio, trazando una línea de cierre. Fijate que también utilizamos la posición relativa, que con las coordenadas 0 0 deja el puntero en donde estaba.

Por último cerramos la ruta con el comando `z`.

líneas

Las líneas se dibujan con el comando `L`. Este lleva dos parámetros que son las coordenadas "x" e "y" del final de la línea respectivamente. El principio de línea viene marcado por la posición actual del puntero. Veamos un ejemplo.

```
<svg width=200 height=200>
<path d="M 20 25 l 40 150 l 40 -150 l 40 150 l 40 -150 m 0 0 z"
       stroke="navy" stroke-width=3 fill="white"/>
</svg>
```

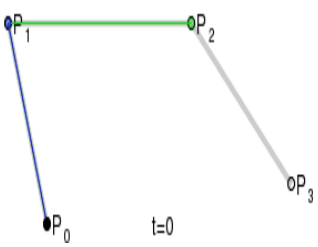
Como en el ejemplo anterior empezamos posicionando el puntero al principio de la ruta (`M 20 25`).

Seguimos indicando una serie de líneas con el comando l (ele minúscula) (`l 40 150 l 40 -150 l 40 150 l 40 -150`). En cada una de ellas se indican las coordenadas relativas "x" e "y" respecto de la anterior.

Después posicionamos el puntero en el último punto en el que está para que éste no vuelva al inicio al cerrar la ruta (`m 0 0`).

Y por último cerramos la ruta (`z`).

Curvas de Bézier



Las curvas de Bezier son un tipo de curvas que se calculan de forma matemática teniendo en cuenta la equidistancia de los puntos inicial y final a dos puntos externos a la misma. El esquema de la izquierda nos da una idea de cómo se forman. Para saber más sobre curvas de Bezier consultar la wikipedia en http://es.wikipedia.org/wiki/Curva_de_B%C3%A9zier.

Nosotros sólo necesitamos saber los puntos de principio y final y los dos puntos externos para trazarla, y ésto es lo que indicaremos en el comando `C` que traza la curva.

El comando C lleva 6 parámetros. Los dos primeros son las coordenadas "x" e "y" del primer punto

externo a la curva. Los dos siguientes son las coordenadas "x" e "y" del segundo punto externo a la curva. Los dos últimos son las coordenadas del punto final de la curva.

El punto de inicio de la curva no hay que marcarlo ya que será el punto actual del puntero o lápiz.

Visto esto vamos a ver un ejemplo de curva de Bezier.

```
<svg width=200 height=200>
<path d="M 10 150 C 30 1  170 50 190 150 M 190 150 z"
      stroke="navy" stroke-width=3 fill="white"  />
</svg>
```

En primer lugar marcamos el punto inicial posicionando el puntero al principio de la ruta (`M 10 150`). Es aquí donde empezará la curva.

Seguimos después con la curva (`C 30 1 170 50 190 150`). Los dos primeros números (30 1) marcan las coordenadas "x" e "y" del primer punto externo. Los dos números siguentes (170 50) son las coordenadas del segundo punto externo. Los dos últimos números (190 150) son el punto final donde acaba la curva.

Movemos el puntero dejándolo en el final de la curva, para que éste no vuelva al principio al cerrar la ruta (`M 190 150`).

Por último cerramos la ruta con el comando `z`.

Curva de Bézier con reflejo

El comando S se usa para trazar una segunda curva de Bezier con posicion relativa del primer punto externo análoga al de la curva anterior. Veamos un ejemplo.

```
<svg width=200 height=200>
<path d="M 10 100 C 1 1 199 1  100 100 S 199 199 190 100  M 190 100 z"
        stroke="navy" stroke-width=3 fill="white" />
</svg>
```

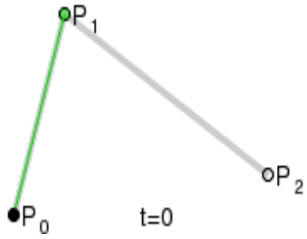
Como en el ejemplo anterior posicionamos primero el puntero (`M 10 100`) y después dibujamos una curva de Bezier (`C 1 1 199 1 100 100`).

Dibujamos después una segunda curva con el comando S (`S 199 199 190 100`). Los dos primeros números corresponden a las coordenadas del segundo punto externo de la curva y dos dos últimos a las coordenadas del punto final.

El principio de la curva viene dado por el final de la anterior, y el primer punto externo se calcula automáticamente en función de la primera curva para hacer que el paso de una curva a otra sea de una manera continua y sin esquinas.

Por último, como en los ejemplos anteriores posicionamos el puntero en el último punto (`M 190 100`) y cerramos la ruta (`z`).

Curvas cuadráticas de Bézier



Como una variante de las curvas de Bezier están las curvas cuadráticas de Bézier. Estas son más simples ya que sólo tienen un punto externo de referencia. Aquí la izquierda vemos un esquema de cómo se forman. Para más información sobre este tipo de curvas podemos consultar la página de wikipedia indicada anteriormente para las curvas de Bézier.

En SVG las curvas cuadráticas de Bézier se dibujan en la ruta con el comando `Q`. Este comando lleva 4 parámetros. Los dos primeros son el punto externo y los dos últimos el punto final de la curva. Vemos aquí un ejemplo.

```
<svg width=200 height=200>
<path d="M 10 100 Q 200 0 100 190 M 100 190 z"
        stroke="navy" stroke-width=3 fill="white"/>
</svg>
```

Como en ejemplos anteriores empezamos posicionando el puntero en el primer punto de la curva (`M 10 100`).

Seguimos trazando la curva cuadrática (`Q 200 0 100 190`) los dos primeros números son las coordenadas "x" e "y" del punto externo. Los otros dos números son las coordenadas "x" e "y" del punto final de la curva.

Para acabar posicionamos el puntero en el último punto (`M 100 190`) y cerramos la ruta (`z`).

Curvas cuadráticas con reflejo

El comando T se utiliza para dibujar una segunda curva cuadrática que será un reflejo de la anterior. No necesitamos poner el punto externo ya que éste se calcula automáticamente en función de la curva anterior. Veamos un ejemplo.

```
<svg width=200 height=200>
<path d="M 10 100 Q 50 -50  100 100 T 190 100  M 190 100 z"
      stroke="navy" stroke-width=3 fill="white" />
</svg>
```

En primer lugar posicionamos el puntero en el punto de inicio (`M 10 100`) y dibujamos una primera curva cuadrática (`Q 50 -50 100 100`).

Después dibujamos una segunda curva reflejo de la anterior (`T 190 100`). Los dos números que ponemos son las coordenadas "x" e "y" del punto final de la curva.

El punto inicial viene dado por la posición del puntero, y el punto exterior de la curva se calcula en función del punto anterior. Es por eso que sólo tenemos que poner el punto final.

Por último cerramos la ruta tal como hemos hecho en los ejemplos anteriores (`M 190 100 z`).

Arcos elípticos

El comando A dibuja un arco de una elipse que va desde el punto inicial, marcado por la posición del puntero hasta un punto final indicado.

Este comando tiene siete parámetros.

- 1º Indica la longitud del radio en el eje horizontal (x) de la elipse.
- 2º Indica la longitud del radio del eje vertical (y) de la elipse
- 3º Hace una rotación de la elipse respecto la posición que tendría originalmente. El número indica el número de grados que rotará en sentido antihoriario.
- 4º y 5 Con los datos aportados hay dos elipses que pueden pasar por los dos puntos que marcamos, por lo que con estos parámetros delimitamos cual es el arco que queremos que se vea. Estos dos parámetros sólo pueden tener los valores 1 ó 0. El cuarto parámetro indica si veremos el arco de mayor tamaño (valor 1) o el de menor tamaño (valor 0) y el quinto indica si e arco se dibuja en sentido horario (valor 1) o en sentido antihorario (valor 0). Con esto se define cuál de los cuatro arcos posibles es el que queremos ver.
- 6º y 7º Son las coordenadas "x" e "y" respectivamente del punto en el que se termina de dibujar el arco.

```
<svg width="300" height="300" >
<path d="M 150 75 a 80 100 45 1 1 0 150 m 0 -150 z"
      stroke="blue" stroke-width=2 fill="white"/>
<path d="M 150 75 a 80 100 45 0 1 0 150 m 0 -150 z"
      stroke="red" stroke-width=2 fill="white"/>
<path d="M 150 75 a 80 100 45 1 0 0 150 m 0 -150 z"
      stroke="green" stroke-width=2 fill="white"/>
<path d="M 150 75 a 80 100 45 0 0 0 150 m 0 -150 z"
      stroke="yellow" stroke-width=2 fill="white"/>
</svg>
```

El código de aquí arriba corresponde con la figura que vemos a la derecha.

En este ejemplo hemos dibujado los cuatro arcos elípticos que puede haber entre dos puntos, cada uno de ellos de un color, completando así las dos elipses.

Cada arco lo hemos puesto en una ruta. Las rutas son muy similares, lo único que las diferencia son los parámetros 4º y 5º de cada una y el color de línea.

Los demás parámetros son iguales en las cuatro rutas. El primero y el segundo (`80 100`) indican los radios de los ejes horizontal y vertical de las elipses.

El tercer parámetro (`45`) también es igual en todas las rutas e indica que la elipse se inclina 45 grados respecto a su posición original.

Los dos últimos parámetros (6º y 7º) son también iguales en todas las rutas e indican el punto final de los arcos dibujados.

Los parámetros 4º y 5º son los que varían de una ruta a otra. Sólo pueden tomar los valores 0 ó 1, y dependiendo de los valores que tomen se verá un arco u otro.

Incluir diferentes figuras.

En una misma ruta se pueden incluir diferentes trazados de los vistos anteriormente, sólo tenemos que ponerlos uno a continuación de otro.

```
<svg width=300 height=200>
<path d="M 10 10 l 50 50 c -70 0 -60 20 50 50 q 100 -100 0 70 h 120 m 0
-20 c 50 0 70 -70 0 -100 m 0 0 z" stroke="blue" stroke-width=2
fill="white"/>
</svg>
```

En este ejemplo hemos puesto varios trazados en una ruta. En primer lugar posicionamos el puntero en el punto inicio (`M 10 10`).

Dibujamos después con una línea recta (`l 50 50`).

Seguimos con un curva de Bézier (`c -70 0 -60 20 50 50`).

Ponemos a continuación una curva cuadrática (`q 100 -100 0 70`).

Seguimos con una línea horizontal (`h 120`).

Después movemos el puntero a mano alzada (`m 0 -20`).

Como último dibujo hacemos otra curva de Bézier (c 50 0 70 -70 0 -100).

Como último paso posicionamos el puntero en el último punto y cerramos la ruta (m 0 0 z).

Texto en SVG

Insertar texto

Para insertar texto dentro de un gráfico de SVG utilizamos la etiqueta `text`. Esta etiqueta tiene su correspondiente etiqueta de cierre, y entre la etiqueta de apertura y la de cierre incluimos el texto que queremos poner.

```
<text>mi texto </text>
```

Esta es la sintaxis básica de esta etiqueta, sin embargo para que funcione bien debemos poner algunos atributos que indiquen en qué lugar del contenedor queremos poner el texto. Estos son los atributos `x` e `y` que indican las coordenadas "x" e "y" respectivamente del lugar en donde queremos poner el texto.

```
<text x=50 y=70 >Mi texto</text>
```

Esto es suficiente para visualizar un texto en la mayoría de los navegadores. Estos nos dan el estilo por defecto del texto. Para insertar el texto a partir del punto dado éste traza una línea horizontal y el texto se coloca encima de la misma.

Primer texto de prueba Segundo texto de prueba

Para desplazar el texto respecto de esta línea en la que se coloca podemos añadir los atributos `dx` y `dy` que indican el desplazamiento horizontal y vertical respectivamente respecto a la línea que

marca su posición original.

En el contenedor de la derecha hemos puesto dos textos y también dos líneas horizontales que empiezan en las mismas coordenadas "x" e "y" que los textos. El primer texto lo hemos dejado en sus coordenadas originales, y el segundo lo hemos desplazado mediante los atributos `dx` y `dy`. el código que hemos empleado es el siguiente:

```
<svg width=300 height=150>
<text x=50 y=50>Primer texto de prueba</text>
<text x=50 y=120 dx=20 dy=20>Segundo texto de prueba</text>
<line x1=50 y1=50 x2=250 y2=50 stroke="black" stroke-width=1 />
<line x1=50 y1=120 x2=250 y2=120 stroke="black" stroke-width=1 />
</svg>
```

Estilo de texto

Para dar un estilo al texto emplearemos diferentes atributos:

- `fill="<color>"` : Cambia el color del texto al indicado. Se usa de igual manera que con las figuras y rutas.
- `stroke="<color>"` : Pone un borde del color indicado a las letras. Se usa de igual manera que con las figuras y rutas.
- `stroke-width="<medida>"` : Indica el grosor del borde del atributo `stroke`. Si no se indica el tipo de medida se entiende que son píxeles. Funciona igual que con las figuras y rutas.
- `font-family="<fuente>"` : Como valor podemos poner el nombre de una fuente o tipo de letra o una clase genérica o familia de fuente; tal como se hace con la propiedad `font-family` de CSS.
- `font-size="<medida>"` : Indica el tamaño de letra, igual que la propiedad `font-size` de CSS. Si ponemos sólo el número sin indicar medida, ésta se tomará en píxeles.
- `font-style="normal | italic | oblique"` : indica si se quiere poner la letra en cursiva. Los valores que podemos dar son los mismos que para la propiedad `font-style` de CSS.
- `font-weight="normal | bold | bolder | lighter"` : indica el grosor de la letra (letra en negrita). Los valores que podemos poner son los mismos que en la propiedad `font-weight` de CSS.
- `font-variant="normal | small-caps"` : Pone el texto en letra versal (mayúsculas pequeñas). Se emplea igual que la propiedad `font-variant` de CSS.
- `text-decoration=" underline | overline | line-through"` : Pone una raya en el texto que puede ser de subrayado (`overline`), de tachado (`line-througt`) o una línea por encima (`overline`). Funciona igual que la propiedad `text-decoration` de CSS.
- `letter-spacing="<medida>"` : indica la medida de espaciado entre caracteres. Funciona igual que la propiedad `letter-spacing` de CSS. Si se pone un número sin indicar la medida se entiende que son píxeles. Este atributo no funciona con el navegador

Firefox.

- `rotate="<número>"` : Hace rotar cada una de las letras el número de grados indicado. Podemos poner más de un número, en este caso cada número hará rotar una letra: 1º num. con 1ª letra, 2º num. con 2ª letra, y así sucesivamente. El último número hace rotar todas las letras que faltan hasta el final.
- `textLength="<medida>"` : indica la longitud que ocupará el texto. El espacio entre letras se adapta a la longitud indicada. Si se pone el número sin ninguna medida el espacio se medirá en píxeles.
- `lenghtAdjust="spacing | spacingAndGlyphs"` : Relacionado con el atributo anterior, una vez que hemos puesto un `textLength` ponemos este atributo para indicar si el espaciado se producirá entre letras (valor `spacing`) o sólo entre palabras (valor `spacingAndGlyphs`).

Primer texto svg Segundo texto svg rotación de texto uno dos tres cuatro uno dos tres cuatro uno dos tres cuatro

Además de estos atributos podemos poner también los atributos de estilo CSS `style=" ... "`, de clase `class="nombre_clase"` y de identidad `id= "nombre_id"` que funcionan igual que con el resto de elementos de HTML.

Hemos puesto aquí un ejemplo de un contenedor SVG en el que empleamos los atributos vistos anteriormente para texto. En los dos primeros empleamos los atributos de estilo para la fuente. En el tercero ponemos el atributo para rotación del texto. En los tres últimos comparamos los atributos para ajustar la longitud del texto y el espaciado entre letras.

El código fuente de este ejemplo es el siguiente:

```
<svg width=300 height=200 >
<text x=20 y=30 font-weight="bold" fill="aqua" stroke="blue" stroke-
width=2
      font-family="comic sans ms" font-size="2em" >
      Primer texto svg</text>
<text x=20 y=60 fill="green" font-family="arial" font-style="italic"
      font-variant="small-caps" font-size="1.5em">
      Segundo texto svg</text>
<text x=20 y=100 fill="red" textLength=260 rotate="10 20 30 40 50 40 30 20
10 0 -10 -20 -30">
     rotación de texto</text>
<text x=20 y=130 fill="blue" textLength=260
lengthAdjust='spacingAndGlyphs' >
     uno dos tres cuatro</text>
<text x=20 y=160 fill="blue" textLength=260 lengthAdjust='spacing' >
     uno dos tres cuatro</text>
<text x=20 y=190 fill="blue" letter-spacing="5px" text-
decoration="underline"  >
     uno dos tres cuatro</text>
</svg>
```

Estilo CSS en SVG

Gran parte de los atributos anteriores pueden ser sustituidos por propiedades CSS aplicadas al elemento de texto. Como con cualquier código CSS podemos ponerlo en el propio elemento con la etiqueta `style` o en una hoja de estilos aparte.

Los atributos relativos a la fuente (`font-family`, `font-size`, `font-weight`, `font-style` y `font-variant`) pueden sustituirse por sus correspondientes propiedades en CSS. También podemos usar la propiedad de tipo "shorthand" `font` que sustituye a todas ellas.

Los atributos `text-decoration` y `letter-spacing` también pueden sustituírse por sus correspondientes propiedades CSS.

Primer texto svg Segundo texto svg

Los atributos `fill`, `stroke` y `stroke-width` tienen también sus correspondientes propiedades CSS con el mismo nombre, por ejemplo:

text {fill: aqua; stroke: navy; stroke-width: 1px; }

Estas propiedades CSS pueden usarse también con otros elementos como figuras y rutas.

Los demás atributos no tienen correspondencia en CSS, y deben ponerse en el elemento correspondiente.

Siempre que se pueda es preferible utilizar las propiedades CSS en lugar de los atributos, ya que en caso de páginas con bastantes elementos, los atributos puede que no funcionen correctamente a causa de la herencia de los elementos padre en CSS.

El ejemplo que hemos puesto aquí simplifica bastante el código de los dos primeros textos del ejemplo anterior. Este es su código:

```
<svg width=300 height=120>
<text x=20 y=30
      style="font: bold 2em 'comic sans ms';fill: aqua; stroke: blue;
stroke-width: 2px;">
      Primer texto svg</text>
<text x=20 y=100
      style="font: italic small-caps 1.5em arial; fill: green;">
      Segundo texto svg</text>
</svg>
```

Al igual que con los demás elementos de HTML podemos usar también los atributos `class` o id para referirnos a uno o varios elementos en la hoja de estilos.

Agrupar elementos

Primer titulo svg Segundo titulo svg Sección primera de svg Sección segunda de svg

Supongamos que tenemos un contenedor con varios elementos, y queremos dar a todos ellos, o a una parte de ellos, el mismo estilo. Hay en todos ellos varios atributos que se repiten. Para evitar esto podemos poner una serie de etiquetas que son elementos contenedores, en las cuales podemos incluir estos atributos.

Utilizamos para ello la etiqueta `<g> ... </g>` por ejemplo si queremos poner dos textos con el mismo estilo podemos incluir el estilo en la etiqueta `g` que engloba las dos etiquetas de texto

El contenedor SVG de la derecha tiene dos tipos de texto diferente. Hemos utilizado la etiqueta `g` par agrupar los elementos con igual estilo. El código es el siguiente:

```
<svg width=320 height=200>
<g style="font: bolder 1.7em arial;fill: yellow; stroke: red; stroke-
width: 1px;">
<text x=20 y=40>Primer titulo svg</text>
<text x=20 y=140>Segundo titulo svg</text>
</g>
<g style="font: italic 1.2em verdana;fill: purple;">
<text x=20 y=80>Sección primera de svg</text>
<text x=20 y=180>Sección segunda de svg</text>
</g>
</svg>
```

Otras etiquetas de texto

La etiqueta tspan

texto en rojo destacado.

Esta etiqueta se incluye dentro de la etiqueta `text` para poner un estilo diferente a una parte del texto. Es como la etiqueta `span` de HTML pero dentro de un texto en SVG.

Vemos aquí a la derecha un ejemplo de utilización de esta etiqueta. Su código es el siguiente:

```
<svg width=300 height=100>
<text x=20 y=60 style="font: normal 1em arial; fill: navy;">
   texto en
     <tspan style="font: bold 1.2em 'courier new'; fill:
red;">rojo</tspan>
   destacado.
</text>
</svg>
```

incluir enlaces

Podemos incluir enlaces en los elementos de un contenedor SVG.

Usaremos para ello la misma etiqueta `<a>` que en HTML, pero con algunas modificaciones.

La ruta del enlace la pondremos dentro del atributo `xlink:href`. Aunque en realidad podríamos usar el atributo `href`, con `xlink:href` evitamos que el enlace tenga el estilo por defecto de los enlaces (color azul, subrayado, etc.).

Dentro del enlace podemos poner uno o varios elementos de SVG o podemos ponerlo dentro de una etiqueta de texto.

Los enlaces, tal como ocurre en HTML, no heredan algunas de las características de su elemento contenedor (sobre todo cuando éste es un texto), por lo que deberemos indicar éstas en el enlace.

Por lo demás admiten también los mismos atributos que el resto de enlaces, y también las pseudo-clases CSS.

Hemos hecho aquí un ejemplo en el que vemos cómo tanto figuras como texto pueden incluir enlaces. Para poder poner poner pseudo-clases hemos incluído también algo de código CSS en la hoja de estilos. Este es el código HTML-SVG:

```
<svg width=300 height=200>
<a xlink:href="http://www.google.es" target="_blank">
<circle cx=60 cy=60 r=50 class="c1"/>
</a>
<a xlink:href="http://yahoo.es" target="_blank" >
<rect x=160 y=20 width=130 height=80 class="r1"/>
</a>
<text x=10 y=130 style="font: normal 0.8em 'comic sans ms';" >
   Círculo: ir a Google ... Rectángulo: ir a yahoo</text>
<text x=30 y=190 style="fill: purple; font: italic bold 1.5em 'times new
roman';">
   Enlace a <a xlink:href="http://es.wikipedia.org/" target="_blank"
title="Wikipedia"
   class="e1" style="font: italic bold 1em 'times new
roman';">Wikipedia</a>
<text>
</svg>
```

Para completar el código hemos puesto unas líneas en la hoja de estilos CSS que se corresponden con los atributos de clase del código anterior:

```
.c1 { fill: yellow; stroke: blue; stroke-width: 2px; }
.c1:hover { fill: orange; }
.r1 { stroke: green; stroke-width: 2px; fill: aqua;}
.r1:hover { stroke: purple; stroke-width: 5px; }
.e1 { fill: red; }
.e1:hover {fill : green; }
```

Los enlaces en SVG pueden ser una opción para sustituir los mapas de imágenes con HTML.

Texto siguiendo una ruta

Podemos poner también el texto siguiendo la línea de una ruta o etiqueta `path`. para ello utilizamos la etiqueta `textPath`.

En primer lugar dibujaremos la ruta. Ésta tendrá también un atributo `id` par identificarla:

```
<path id="ruta1" d="M 20 200 Q 130 0 320 200 m 0 0 z" stroke="aqua" stroke-width=1
fill="white"/>
```

En este caso dibujamos una curva de Bezier cuadrática sobre la cual irá el texto.

Después ponemos la etiqueta `text` dentro de la cual incluimos la etiqueta `textPath`.

En la etiqueta `textPath` incluimos la referencia al atributo `id` de la ruta con el atributo `xlink:href`. Por ejemplo si en la ruta hemos puesto `<path id="ruta1">` en `textPath` podremos `<textPath xlink:href="#ruta1">`; es decir la misma referencia con el signo de almohadilla (#) delante.

Los atributos de estilo sobre el tipo de letra (fuente) es mejor ponerlos en la etiqueta `textPath` ya que no se heredan de la etiqueta `text` en la que está incluída.

Siguiendo con nuestro ejemplo, el código para incluir el texto será como el siguiente.

```
<text fill="blue" dy=-5>
<textPath xlink:href="#ruta1" style="font: bold 1em verdana" >
Este texto sigue una curva de Bézier</textPath>
</text>
```

Este ejemplo nos da la curva cuadrática de Bézier que aparece en el contenedor SVG que tenemos aquí. Además hemos añadido también un segundo texto siguiendo una ruta que nos crea un texto en vertical.

Para ello hemos dibujado una ruta que consiste en una línea vertical. El color de la línea es el mismo que el del fondo, para que no se vea. Hemos puesto un texto en la línea y lo hemos rotado para ver las letras en su posición correcta. El código del contenedor es el siguiente:

```
<svg width=330 height=250 style="border: 1px solid black;">
<path id="ruta1" d="M 20 200 Q 130 0 320 200  m 0 0 z"
      stroke="aqua" stroke-width=1 fill="white"/>
<text fill="blue" dy=-5>
<textPath xlink:href="#ruta1" style="font: bold 1em verdana" >
Este texto sigue una curva de Bézier</textPath>
</text>

<path id="ruta2" d="M 300 20 L 300 240 m 0 0 z" stroke="white" stroke-
width=1 fill="white"/>
<text fill="red" style="font: bold 1em arial;" rotate=270 textLength=220 >
<textPath xlink:href="#ruta2" style="font: bold 1em 'courier new';" >TEXTO
VERTICAL</textPath>
</text>
</svg>
```

Relleno de figuras

Introducción

En Páginas anteriores de este manual hemos visto la propiedad `fill` que rellena una figura con un color plano. No es la única opción para el relleno de una figura. Veremos aquí otras propiedades relacionadas con el relleno de figuras y con el tratamiento del borde de las mismas.

Transparencias

La propiedad `opacity` permite dar transparencia a una figura. Esta propiedad podemos ponerla tanto como un atributo de html-svg, o como propiedad CSS y su valor es un número decimal entre el 0 (transparencia máxima) y el 1 (opacidad máxima). Como atributo lo escribimos así:

```
<circle cx=70 cy=70 r=60 stroke="yellow" stroke-width=5 fill="blue" opacity=0.5 />
```

Como propiedad CSS, lo escribiremos igual que la propiedad `opacity` de CSS3:

```
<circle cx=70 cy=70 r=60 style="stroke: yellow; stroke-width: 5px; fill: blue; opacity:
0.5" />
```

Hemos puesto aquí un ejemplo de un circulo semitransparente. Cualquiera de los dos códigos nos es válido.

La propiedad afecta a la figura entera, es decir, tanto el borde (stroke) como el relleno (fill) se vuelven transparentes en el grado que indica el valor de la propiedad.

En la figura de la derecha vemos cuatro círculos semitransparentes. El código que hemos utilizado para este gráfico SVG es el de cuatro círculos muy parecidos al del código de ejemplo, y es el siguiente:

```
<svg width=270 height=270>
<circle cx=70 cy=70 r=60 stroke="yellow" stroke-width=5 fill="blue"
opacity=0.5 />
<circle cx=110 cy=110 r=60 stroke="green" stroke-width=5 fill="red"
opacity=0.5 />
<circle cx=150 cy=150 r=60 stroke="blue" stroke-width=5 fill="yellow"
opacity=0.5 />
<circle cx=190 cy=190 r=60 stroke="red" stroke-width=5 fill="green"
opacity=0.5 />
</svg>
```

Podemos aplicar la transparencia sólo al relleno(fill) o sólo al borde (stroke) o aplicar transparencias distantas a cada uno de ellos.

La propiedad `fill-opacity` aplica la transparencia sólo al relleno(fill).

La propiedad `stroke-opacity` aplica la transparencia sólo al borde (stroke).

Ambas propiedades pueden escribirse como un atributo del elemento HTML-SVG o como una propiedad CSS aplicada al elemento. Sus valores son los mismos que para `opacity`, es decir un número decimal que va desde 0 (transparencia total) a 1 (opacidad total).

En el gráfico de la derecha hemos puesto cuatro cuadrados con transparencia total en el relleno y transparencia con valor 0.7 para el borde. Hemos separado los códigos HTML y CSS los cuales son los siguientes.

Código HTML del gráfico:

```
<svg width=270 height=270>
<rect x=10 y=10 width=110 height=110 id="r1"  />
<rect x=50 y=50 width=110 height=110 id="r2" />
<rect x=90 y=90 width=110 height=110 id="r3" />
<rect x=130 y=130 width=110 height=110 id="r4" />
</svg>
```

Código CSS del gráfico:

```
#r1, #r2, #r3, #r4 { stroke-width:15px; fill-opacity: 0; stroke-opacity:
0.7;  }
#r1 { stroke: green; }
#r2 { stroke: red; }
#r3 { stroke: yellow; }
#r4 { stroke: blue; }
```

Tipo de relleno

La propiedad `fill-rule` Permite elegir si el relleno se hará de forma total o parcial en figuras complejas que tienen varios compartimentos. Sus posibles valores son `non-zero` y `evenodd`.

En las figuras de la derecha, si nos fijamos en las dos de arriba, la de la izquierda tiene el atributo `fill-rule="non-zero"`, lo que hace que se rellene la figura entera. La figura de la derecha tiene el atributo `fill-rule="evernodd"`, lo que hace que sólo se rellenen los compartimentos exteriores, dejando el central sin rellenar.

Esta propiedad, al igual que las anteriores, puede escribirse como atributo del elemento o como propiedad CSS.

Esquina entre dos líneas

La propiedad `linejoin` permite dar un tratamiento diferenciado a la esquina en donde se juntan dos líneas. Esta propiedad tiene los siguientes valores:

- `"mitter"` : Es el valor por defecto y la esquina se muestra como tal como está: sin recortar sus aristas.
- `"round"` : Recorta la esquina dejándola redondeada.
- `"bevel"` : Recorta la esquina mediante una línea recta, dejándola como un chaflán.

En el gráfico anterior vemos en las dos estrellas de la parte de abajo los efectos de esta propiedad. La estrella de la izquierda tiene el atributo `linejoin="round"` y muestra sus esquinas redondeadas, mientras que la estrella de la derecha tiene el atributo `linejoin="bevel"` y muestra sus esquinas recortadas en línea recta.

Al igual que las anteriores, esta propiedad puede escribirse tanto como un atributo del elemento como en el código CSS.

El gráfico anterior muestra tanto esta propiedad como la propiedad `fil-rule` vista anteriormente. Su código es el siguiente.

```
<svg width=270 height=270>
<path id="p1" d="M 70 10 l -40 120 l 90 -70 l -110 0 l 90 70 z"
     stroke="blue" fill="yellow" stroke-width="3" fill-rule="non-zero"/>

<path id="p2" d="M 200 10 l -40 120 l 90 -70 l -110 0 l 90 70 z"
     stroke="blue" fill="yellow" stroke-width="3" fill-rule="evenodd"/>

<path d="M 70 140 l -40 120 l 90 -70 l -110 0 l 90 70 z"
     stroke="blue" fill="yellow" stroke-width="8" stroke-
linejoin="round"/>

<path d="M 200 140 l -40 120 l 90 -70 l -110 0 l 90 70 z"
     stroke="blue" fill="yellow" stroke-width="8" stroke-
linejoin="bevel" />
</svg>
```

Fíjate cómo en el código repetimos la misma figura al poner la misma ruta, en la que sólo cambia el punto de inicio. El resto de comandos están escritos de forma relativa, (en minúsculas), lo que permite referirse siempre al primer punto.

En las dos estrellas de abajo aumentamos el tamaño del borde (stroke-width="8") para apreciar mejor el efecto de la propiedad `linejoin`.

Gradiente de color

Definición

Un gradiente de color, también llamado degradado de color, es una transicion de manera suave desde un color a otro, pasando por todas las gamas intermedias entre ellos.

En SVG tenemos dos tipos de gradientes, los gradientes lineales y los gradientes radiales.

Los gradientes pueden aplicarse a cualquier elemento al que le hayamos dado color, tanto en su relleno (fill) como en el trazado del borde (stroke).

Gradiente lineal

Para crear un gradiente lineal utilizaremos la etiqueta `linearGradient`.

Este elemento y otros similares se suelen incluir dentro de la etiqueta `defs`:

```
<defs>
   <linearGradient ........ >
      .........
   </linearGradient>
</defs>
```

La etiqueta `<defs> ...</defs>` no incluye ningún tipo de elemento en la página, y sólo sirve para organizar el contenido. Normalmente incluimos en esta etiqueta una serie de etiquetas que sirven para definir algunos elementos que incluiremos más adelante en la página, como por ejemplo, los gradientes.

En el esquema anterior las lineas de puntos serán sustituidas por atributos y etiquetas que crean el gradiente.

Vemos aquí a la derecha un ejemplo de gradiente lineal en el que el fondo cambia gradualmente de unos colores a otros.

La etiqueta `defs` indica que lo que pondremos dentro serán definiciones y no parte del dibujo. En este caso definimos un degradado lineal con la etiqueta `linearGradient`. Una vez definido tendremos que hacer una referencia al degradado desde el dibujo para aplicarlo.

La etiqueta `linearGradient` puede llevar los siguientes atributos:

- `id="<nombre_id>"` : Como valor llevará un nombre identificador para poder referirse a este degradado en otro elemento. Es por tanto obligatorio.
- `x1="<medida>" y1="<medida>"` : Indican las coordenadas "x" e "y" del punto en donde se inicia el degradado de color. Si no indicamos la medida (y escribimos sólo el número), se entiende que las medidas son en píxeles. También podemos poner porcentajes.
- `x2="<medida>" y2="<medida>"` : Indican las coordenadas "x" e "y" del punto en donde termina el degradado de color. Al igual que en los anteriores podemos poner sólo el número (pixels) o porcentajes.
- `gradientUnits="userSpaceOnUse | objectBoundingBox"` : Indica desde dónde se miden las coordenadas. con el valor `"userSpaceOnUse"`Las coordenadas son

las mismas que para el elemento contenedor del SVG, mientras que `objectBoudingBox` indica que toma como origen de coordenadas el propio elemento al que se le aplica, en su esquina superior derecha de la caja (imaginaria) que lo contiene. Este último es el valor por defecto, y el que se aplica si el atributo no se pone.

- `spreadMethod ="pad | reflect | repeat"` : Indica cómo debe tratarse el resultado del degradado. El valor `pad` es el valor por defecto y hace que antes y después de las coordenadas indicadas, los colores se extiendan de manera uniforme (sin degradado). El valor `reflect` repite el degradado pero en cada repetición hace un reflejo del mismo como en un espejo. El valor `repeat` repite el degradado tal cual, sin reflejarlo en cada repetición.

Nos falta definir los colores del degradado. Para ello, dentro de la etiqueta `linearGradient` indicamos los colores mediante la etiqueta `stop`. Esta etiqueta se repite para cada cambio de color en el degradado y tiene la siguiente sintaxis:

<stop offset="<num>" stop-color="<color>"/>

El atributo `offset` tiene como valor un número decimal del 0 al 1. Coloca un marcador en la línea que va desde el punto de inicio (en el 0) al punto final (en el 1) del degradado. Es conveniente poner siempre como mínimo dos etiquetas `stop` una `stop offset="0"` para el color de inicio, y otra `stop offset=1` para el color final. entre medio podemos poner otros colores mediante decimales entre el 0 y el 1.

El atributo `color-stop` indica el color que habrá en ese punto. Entre un punto y otro se produce el cambio progresivo de color.

Hasta aquí el contenido de la etiqueta `linearGradient` con la que definimos el gradiente de color. Ahora queda incluir el gradiente de color en un elemento, como color de relleno del mismo, para ello en la propiedad o atributo `fill` haremos referencia al degradado, poniendo el siguiente valor:

fill="url(#nombre_id)"

Como valor ponemos la palabra "url" y después entre paréntesis y con un signo de almohadilla delante (#) el nombre del atributo `id` que le hemos dado a la gradiente.

Vemos un ejemplo que nos da como resultado el círculo que vemos arriba a la derecha. Este es su código.

```
<svg width=250 height=250>
<defs>
<linearGradient id="grad1"
    x1="50%" y1="0%" x2="50%" y2="100%">
    <stop offset="0" stop-color="silver"/>
    <stop offset="0.2" stop-color="blue"/>
    <stop offset="0.5" stop-color="lime"/>
    <stop offset="0.8" stop-color="blue"/>
    <stop offset="1" stop-color="silver"/>
</linearGradient>
</defs>
<circle cx=125 cy=125 r=120 fill="url(#grad1)"/>
</svg>
```

Hemos puesto otro ejemplo, un poco más complicado que nos da como resultado el dibujo de la derecha. Aquí vemos la diferencia entre el atributo `spreadMethod="reflect"` que hemos aplicado al borde (stroke), y `spreadMethod="repeat"`, aplicado al relleno (fill).

Tal como vemos aquí el degradado de color puede aplicarse también a los bordes o trazos (stroke), de la misma manera que al relleno.

Nota: (Agosto de 2014) Los navegadores Firefox y Safari no admiten el valor "reflect". Safari Tampoco admite el valor "repeat". Si estás usando alguno de estos navegadores, abre esta página en otro navegador para ver los efectos en el dibujo de la derecha.

También puede usarse un gradiente que hayamos definido anteriormente en otro contenedor SVG, siempre que éste esté en la misma página.

El código del ejemplo de la derecha es el siguiente:

```
<svg width=270 height=270>
<defs>
<linearGradient id="reflejo"
     x1="40%" y1="40%" x2="50%" y2="50%"
     spreadMethod="reflect">
     <stop offset="0" stop-color="green"/>
     <stop offset="1" stop-color="lime"/>
</linearGradient>

<linearGradient id="repite"
     x1="40%" y1="50%" x2="50%" y2="40%"
     spreadMethod="repeat">
     <stop offset="0" stop-color="yellow"/>
     <stop offset="1" stop-color="red"/>
</linearGradient>
</defs>
<rect x=10 y=10 width=250 height=125
    fill="url(#reflejo)" />
<rect x=10 y=140 width=250 height=125
    fill="url(#repite)" />
</svg>
```

Gradiente radial

En el Gradiente radial el efecto de cambio de color se produce mediante círculos concéntricos. El cambio de color va desde un punto central hacia el exterior formando círculos.

Utilizamos para ello la etiqueta `radialGradient`

La utilización de esta etiqueta es parecida a la etiqueta `linearGradient`.

Debemos poner el atributo `id` para dar un nombre que identifique al degradado.

Una serie de atributos definen la posición del degradado y sus puntos principales. Estos son:

- `cx="<medida>" cy="<medida>"` : Coordendas "x" e "y" del centro del círculo de degradado. En estos atributos y los siguientes podemos indicar una medida, un número (píxeles) o un porcentaje.
- `r="<medida>"` : Indica el radio dentro del cual tendrá alcance el degradado.

- `fx="<medida>" fy="<medida>"` : indica el foco del degradado. El foco es el punto desde el cual se inicia el degradado. Éste se extiende desde el foco hacia el resto del círculo marcado. El foco no tiene porqué ser el mismo punto que el centro del círculo. En caso de serlo podemos omitir estos atributos.

El resto de atributos son los mismos que para `linearGradient` y con los mismos valores, es decir `gradientUnits` y `spreadMethod`.

Al igual que con `linearGradient` definimos aquí también los colores y transparencias con las etiquetas `<stop offset ...>` que se emplean exactamente igual. En el atributo `offset` el número 0 será el punto del foco y el número 1 será el exterior del radio.

La inclusión del degradado en un elemento se hace de igual manera a lo visto anteriormente, es decir mediante `fill="url(#nombre_id)"` para incluirlo en el relleno, o `stroke="url(#nombre_id)"`

`para incluirlo en el borde o trazo.`

`La figura que hemos puesto aquí a la derecha es un círculo con un degradado radial. su código es el siguiente.`

```
<svg width=270 height=270 >
<defs>
<radialGradient id="rad1"
   cx="50%" cy="50%" r="75%" fx="40%" fy="20%">
   <stop offset="0" stop-color="aqua"/>
   <stop offset="1" stop-color="navy"/>
</radialGradient>
</defs>
<circle cx=135 cy=135 r=120 fill="url(#rad1)"/>
</svg>
```

Incluir elementos

Incluir imágenes

Para incluir imágenes en un contenedor SVG utilizaremos la etiqueta `image` la cual tiene varios atributos:

- `xlink:href="<url>"` : Incluimos aquí la ruta de la imagen que queremos insertar.
- `x="<medida>" y="<medida>"` : Las coordenadas "x" e "y" del sitio en el que se inserta la imagen (punto superior izquierdo de la imagen).
- `width="<medida>" height="<medida>"` : El ancho y el alto de la imagen que se va a insertar.

El ancho y el alto (`width` y `height`) no son automáticos, por lo que hay que ponerlos siempre y hay que tener en cuenta las proporciones, para que no haya distorsión de la imagen.

Como ejemplo hemos hecho un SVG con tres imágenes. El código es el siguiente:

```
<svg width=270 height=220>
<rect x=0 y=120 width=270 height=100 fill="#ffb6b7"/>
<image xlink:href="objetos/estrella.gif" x=40 y=10 width=150 height=50 />
<image xlink:href="objetos/bobesponja.gif" x=10 y=50 width=150
height=135 />
<image xlink:href="objetos/garfield.gif" x=120 y=80 width=130 height=90 />
<svg>
```

El resultado es el dibujo que se ve aquí a la derecha. En él hemos incluido tres imágenes. La imagen de la estrella es un gif animado y las otras son gif transparentes.

Patrones

Un patrón es un elemento que se repite como una imagen de fondo que podemos poner para rellenar figuras. El elemento que podemos poner para rellenar puede ser una imagen o unas figuras en SVG creadas por nosotros.

Para crear un patrón utilizaremos la etiqueta `pattern`. Esta se usa de forma parecida a las etiquetas para degradados de color. Por lo tanto la etiqueta `pattern` estará incluida dentro de una etiqueta `defs`.

Pondremos una atributo `id` para identificar el patrón con un nombre, de la misma manera que hacemos con los degradados de color.

Pattern crea un subespacio rectangular dentro del contenedor, para ello pondremos los atributos `x`, `y`, `width` y `height`:

```
<pattern id="nombre_id" x="0" y="0" width="50" height="50">
```

Para que la etiqueta funcione debemos poner también el atributo `patternUnits = "userSpaceOnUse"`. Este atributo es parecido al atributo `gradientUnits` de los degradados de color. Tiene también el valor `objectBoundingBox`, sin embrgo este valor no funciona bien en todos los navegadoes, por lo que usaremos `userSpacingOnUse`.

El atributo `viewBox` Crea un contenedor virtual en el que se verá la imagen. Como valor tiene cuatro números, los dos primeros son la esquina superior izquierda del contenedor, y los otros dos el ancho y el alto del mismo. La imagen se adapta al espacio que indicamos, de manera que a mayor espacio indicado ésta se vera más pequeña. Veremos esto en temas posteriores.

Vemos aquí un ejemplo:

```
<pattern id="nombre_id" x="0" y="0" width="30" height="30"
patternUnits="userSpaceOnUse" viewBox="0 0 50 50">
```

Con esto completamos la etiqueta `pattern`. Después, dentro de la etiqueta "pattern", incluimos otra etiqueta con el contenido que queremos que se repita , y luego cerramos la etiqueta "pattern" (`</pattern>`).

Para ver el contenido como imagen de fondo en una figura o elemento, en la etiqueta haremos

referencia al patrón en la etiqueta fill `fill="url(#nombre_id)` donde `nombre_id` es el valor de la etiqueta `id` del patrón (`pattern`).

En el ejemplo de la derecha hemos puesto una imagen que se repite. El código de este SVG es el siguiente:

```
<svg width="270" height="200" >
<defs>
<pattern id="patron1" x="0" y="0" width="30" height="30"
   patternUnits="userSpaceOnUse" viewBox="0 0 50 50" >
   <image xlink:href="europa.gif" x="0" y="0" width="50" height="50" >
</pattern>
</defs>
<ellipse cx="135" cy="100" rx="110" ry="90"
    stroke="red" stroke-width="3" fill="url(#patron1)"/>
</svg>
```

La imagen empleada para el SVG anterior es la que veis aquí a la derecha. La he puesto ahí por si quereis copiarla para practicar con este ejemplo.

Mediante el atributo `viewBox`, aunque el espacio real que ocupa la imagen es 30 x 30 píxeles (atributos `width` y `height`), esta se ve en un espacio virtual de 50 x 50. Esto es lo que permite verla entera en un tamaño más pequeño.

Podemos crear nuestras propias figuras para incluirlas como imagen de fondo. Vemos aquí un ejemplo en el que hemos creado una figura como imagen de fondo.

El código para crear la figura se pone dentro de la etiqueta `pattern`.

En la figura de relleno hemos puesto primero un rectángulo del mismo tamaño de la imagen para rellenar el fondo. Seguimos después con un círculo que colocamos en el centro, y ponemos luego mediante una ruta (`path`) las líneas que unen los círculos.

El código para la figura que vemos aquí a la derecha es el siguiente.

```
<svg width="270" height="200">
<defs>
<pattern id="patron2" x="0" y="0" width="30" height="30"
     patternUnits="userSpaceOnUse" viewBox="0 0 100 100">
     <rect x="0" y="0" width="100" height="100" fill="#ceedff"/>
     <circle cx="50" cy="50" r="38"
       stroke="#990099" stroke-width="2" fill="#ffb6b7"/>
     <path d="M 50,0 L 50,10 M 50,90 L 50,100 M 0,50 L 10,50 M 90,50 L
100,50 z"
       stroke="#990099" stroke-width="2" fill-opacity="0"/>
</pattern>
</defs>
<ellipse cx="135" cy="100" rx="120" ry="90"
     stroke="#663366" stroke-width="5" fill="url(#patron2)"
</svg>
```

Marcadores

Un marcador es un pequeño dibujo o símbolo que puede introducirse en los vértices de algunas figuras, concretamente en líneas, polilíneas, polígonos y rutas, y también al principio y final de las mismas.

Para incluir un marcador en una de estas figuras lo haremos de manera parecida a los patrones. Primero dentro de un elemento `defs` creamos la figura con la etiqueta `marker` y después la incluimos en el elemento mediante una serie de atributos.

Uno de los usos más corrientes de los marcadores es hacer puntas de flecha para líneas y otros elementos. Haremos aquí un ejemplo de una línea con una punta de flecha a la vez que vamos explicando su funcionamiento.

Utilizaremos la etiqueta `<marker> ... </marker>`, dentro de esta etiqueta dibujaremos mediante una ruta (`path`) el triángulo de la punta de flecha. En primer lugar ponemos la etiqueta SVG y dentro de ella una etiqueta `defs` donde irá el marcador:

```
<svg width="250" height="100" >
<defs>
```

Seguimos con la etiqueta `marker`, la cual tiene los siguientes atributos:

```
<marker id="flecha" markerWidth="10" markerHeight="6">
```

Los atributos `markerWidth` y `markerHeight` indican el ancho y el alto de la caja que contiene el dibujo a insertar (en este caso el triángulo de punta de flecha). Debemos poner también el atributo `id` para poder referirnos luego a este marcador desde el elemento en que lo insertamos.

Debemos indicar también el punto, dentro de esta caja que coincidirá con el vértice, o final de línea en este caso, para ello ponemos los atributos `refX` y `refY` (coordenadas "x" e "y" respecto a la caja creada anteriormente)

```
<marker id="flecha" markerWidth="10" markerHeight="6" refX="5" refY="3">
```

El atributo `orient` indica el ángulo que tomará el marcador. Como valor podemos poner un número, que será el número de grados que gire, o también la palabra `"auto"` que lo alineará con el principio o final de línea, o si el marcador está en un vértice lo pone en un ángulo que será la media entre las dos lineas.

Por último el atributo `markerUnits` puede tomar los valores `UserSpaceOnUse` que toma como unidad de medida para la caja del marcador la que tiene el SVG (normalmente el pixel), o `strokeWidth`, que toma como unidad de medida el grosor de la línea (por ejemplo si la línea mide 3px `-stroke-width="3"-`, el tamaño de la caja se multiplica por 3).

```
<marker id="flecha" markerWidth="10" markerHeight="6" refX="5" refY="3 orient="auto"
markerUnits="strokeWidth" >
```

Después dentro de la etiqueta `marker` insertamos el elemento que hará de punta de flecha, para ello hacemos una ruta que dibuja un triángulo

<path d="M 0 3 L 10 0 L 10 6 z" fill="blue">

Cerramos la etiqueta "marker" y también la etiqueta `defs`:

</marker>
</defs>

Nos falta dibujar la línea en la que insertaremos el marcador. Para insertar un marcador en un elemento (linea, polígono, ruta, etc) usaremos en ese elemento los siguientes atributos:

- `marker-start` : inserta el marcador al principio del elemento
- `marker-mid` : inserta el marcador en todas las esquinas que hay en medio del elemento
- `marker-end` : inserta el marcador al final del elemento

Como valor de estos atributos pondremos una referencia al elemento de la forma `"url(#id_marcador)"` donde `id marcador` es el valor que le hemos dado al atributo `id` de la etiqueta marker.

Dibujamos la línea con el marcador con el siguiente código

<line x1="10" y1="90" x2="220" y2="10"
stroke="blue" stroke-width="3" marker-end="url(#flecha)" />

Por último sólo nos queda cerrar la etiqueta `svg` para completar el dibujo.

</svg>

La figura nos quedará como hemos visto en el dibujo de arriba a la derecha.

Hemos hecho otro ejemplo, en el que añadimos una imagen en los puntos de intersección de una polilínea.

Aprovechamos el marcador hecho en el SVG anterior para ponerlo al final de la línea.

Para poner todas las imágenes en horizontal en el marcador hemos puesto el atributo `orient="0"`, y para controlar que la imagen tenga el tamaño que se indica en `markerWidth` y `markerHeight` hemos puesto un `viewBox` con el tamaño de la imagen.

El resultado es el que ves aquí a la derecha, y el código que hemos empleado es el siguiente:

```
<svg width="270" height="200">
<defs>
<marker id="stop" markerWidth="10" markerHeight="10"
    refX="25" refY="25"
    orient="0" markerUnits="strokeWidth"
    viewBox="0 0 50 50">
<image xlink:href="../objetos/stop.gif" x="0" y="0" width="50" height="50"
</marker>
</defs>
<polyline points="0,5 70,30 50,120 160,30 110,110 230,180 200,50"
    stroke="blue" stroke-width="3" fill-opacity="0"
    marker-mid="url(#stop)" marker-end="url(#flecha)" />
</svg>
```

Aquí hemos utilizado también el marcador que hemos hecho en el ejemplo anterior. Esto es posible porque están los dos en la misma página. De no ser así deberíamos haber incluido dentro de la etiqueta `defs` los dos marcadores.

Transformaciones

Definición

La transformación es un efecto que permite a un elemento o conjunto de elementos cambiar de tamaño, forma o posición.

El elemento cambia respecto a su estado original, lo que nos permite entre otras cosas realizar animaciones, pero eso lo veremos más adelante.

Si conoces canvas de HTML5 o la animación en CSS3 seguramente sabrás lo que son las transformaciones. En SVG se realizan de forma parecida a CSS3, de hecho los métodos usados para las transformaciones son los mismos que en CSS3.

Para realizar una transformación utilizaremos el atributo `transform` al cual le aplicaremos como valor los distintos métodos.

Traslación

el método `translate` permite realizar un desplazamiento del elemento respecto a su posición original.

Para ello utilizaremos el atributo `transform` en el elemento que queramos trasladar.

transform="translate(x,y)

Original Transalte

donde "x" e "y" indicará el desplazamiento horizontal y vertical respectivamente que tendrá el elemento.

Si queremos trasladar un conjunto de elementos, basta con ponerlos todos dentro de una etiqueta `<g> ... </g>`, a la cual le aplicaremos la transformación.

En el ejemplo de la derecha hemos hecho un primer cuadrado con un texto dentro. Repetimos este cuadrado con el texto (que aquí lo hemos cambiado), y hemos hecho una traslación del cuadrado y su texto (Usamos la etiqueta "g" que engloba los dos elementos).

El código de este ejemplo muestra cómo la copia se ha trasladado al incluir la traslación:

```
<svg width="270" height="200">
<g>
  <rect x="10" y="10" width="80" height="80" fill="orange"/>
  <text x="20" y="50" style="font: bold 0.8em arial; fill: purple;">
    Original</text>
</g>
<g transform="translate(150,80)">
  <rect x="10" y="10" width="80" height="80" fill="orange"/>
  <text x="20" y="50" style="font: bold 0.8em arial; fill: purple;">
    Transalte</text>
</g>
```

```
</svg>
```

Escalado

Este método de transformación consiste en reducir o ampliar el elemento al que se le aplica. para ello se utiliza el método `scale`

Como en el método anterior utilizamos el atributo `transform` con el siguiente valor:

transform="scale(sx,xy)"(

Original Scale 1 Scale 2

en donde `sx` y `sy` es el valor que indica las veces que se amplia o reduce el elemento en la coordenada "x" e "y". El 1 indica el tamaño del elemento, por debajo de 1 el elemento se reduce (por ejemplo 0.5 reduce el tamaño a la mitad, y por encima de 1 el elemento se agranda (por ejemplo 1.5 amplia el elemento en un 50%).

También podemos ampliar o reducir en los dos ejes de coordenadas al mismo tiempo si dentro del paréntesis ponemos un sólo valor:

translate="scale(0.5)"

El ejemplo puesto en la línea anterior reduce el tamaño a la mitad tanto para la coordenada "x" como para la coordenada "y", conservado así las proporciones originales.

El cuadro de ejemplo de la derecha tiene tres cuadrados que en principio tienen el mismo tamaño, pero los hemos modificado con un escalado. Su código es el siguiente:

```
<svg width="270" height="200">
<g>
<rect x="10" y="10" width="80" height="80" fill="orange"/>
<text x="20" y="50" style="font: bold 0.8em arial; fill: purple;">
   Original</text>
</g>
<g transform="scale(0.8,0.6)">
  <rect x="180" y="30" width="80" height="80" fill="orange"  />
  <text x="200" y="80" style="font: bold 0.8em arial; fill: purple;">
    Scale 1</text>
</g>
<g transform="scale(1.2)">
  <rect x="100" y="80" width="80" height="80" fill="orange"  />
  <text x="120" y="130" style="font: bold 0.8em arial; fill: purple;">
    Scale 2</text>
</g>
</svg>
```

El escalado se produce también en las medidas que sitúan las figuras en el contenedor, lo cual lo debemos de tener en cuenta para que se vean en el sitio que queremos.

Cambiar el origen de coordenadas

A veces puede interesarnos cambiar el origen de coordenadas para algunos elementos, como en el método de rotación que veremos a continuacion, ya que el elemento gira tomando como eje el origen de coordenadas, para ello englobamos todos los elementos en una etiqueta `g` a la que le

aplicamos el método `translate`:

```
<g transform="translate(100,100)">
... elementos con origen de coordenadas en 100,100 ...
</g>
```

El punto 100,100 será el nuevo origen de coordenadas para todos los elementos que estén dentro de esta etiqueta, esto hay que tenerlo en cuenta al posicionarlos.

Podemos usar también otros métodos para variar otros aspectos de las coordenadas, por ejemplo con `scale` variamos la distancia entre puntos, o con el método de rotación rotamos los ejes "x" e "y".

Rotación

Este método de transformación consiste en rotar el elemento respecto a su posición original. Utilizamos para ello el método `rotate`, en el cual indicamos el número de grados que girará el elemento:

transform="rotate(num)"

Tal como se ha indicado anteriormente el elemento gira tomando como eje el origen de coordenadas, por lo que para que para que gire sobre sí mismo debemos cambiarlo. Como hemo visto en el apartado anterior esto se hace con el método `translate`.

Original Rotate

podemos aplicar los dos métodos (`translate` y `rotate`) sobre un mismo elemento. Para aplicar varios métodos sobre un mismo elemento basta con ponerlos separados por comas:

transform="translate(190,75),rotate(60)"

Con `translate` cambiamos el origen de coordenadas, y con `rotate` hacemos girar el elemento sobre el nuevo origen de coordenadas.

El giro se produce en el sentido horario. Podemos poner también números negativos en `rotate`, que producirán un giro en sentido antihorario.

Tenemos aquí a la derecha un ejemplo en el que hemos aplicado lo visto anteriormente para rotar una figura. Su código es el siguiente:

```
<svg width="270" height="150">
<g >
   <rect x="20" y="25" width="80" height="80" fill="orange"/>
   <text x="30" y="75" style="font: bold 0.8em arial; fill: purple;">
     Original</text>
</g>
<g transform="translate(190,75),rotate(60)">
   <rect x="-40" y="-40" width="80" height="80" fill="orange"/>
   <text x="-20" y="0" style="font: bold 0.8em arial; fill: purple;">
     Rotate</text>
</g>
</svg>
```

Sesgado

El sesgado se produce con el método `skew`. Consiste en deformar la figura variando el ángulo de sus coordenadas. El resultado es una inclinación de la misma en el sentido indicado.

Original Skew

En realidad tenemos dos métodos: `skewX(num)`, que indica la inclinación de la figura para el eje "x"; y `skewY(num)`, que indica la inclinación del eje "y". En `num` escribiremos el número de grados que el eje se inclina.

translate="skewX(15),skewY(30)"

En el ejemplo de la derecha además de los métodos `skewX` y `skewY` hemos puesto también los métodos `translate` y `rotate`. El código es el siguiente:

```
<svg width="270" height="150">
<g >
   <rect x="20" y="25" width="80" height="80" fill="orange"/>
   <text x="30" y="75" style="font: bold 0.8em arial; fill: purple;">
     Original</text>
</g>
<g transform="translate(190,75),rotate(45),skewX(15),skewY(15)">
   <rect x="-40" y="-40" width="80" height="80" fill="orange"/>
   <text x="-20" y="0" style="font: bold 0.8em arial; fill: purple;">
     Skew</text>
</g>
</svg>
```

El método matrix

El método `matrix` es una combinación de los métodos de traslación, de escalado y de sesgado. Tiene seis parámetros o números dentro del paréntesis.

transform="matrix(a,b,c,d,e,f)"

Original Matrix

Los dos últimos parámetros actúan igual que el método `translate` de forma que "e" y "f" indican el desplazamiento en los ejes "x" e "y" respectivamente.

Los parámetros primero y cuarto actúan igual que el método `scale` de forma que el parámetro "a" hace un escalado en el eje "x", y el parámetro "d" hace un escalado en el eje "y".

Los parámetros segundo y tercero actúan como los métodos `skew`. Aquí el número que debemos indicar no es el número de grados del ángulo sino su tangente. El parámetro "b" inclina el eje "x" un angulo cuya tangente es la indicada. El parámetro "c" hace lo mismo con el eje "y".

El código del ejemplo que se ve a la derecha es el siguiente:

```
<svg width="270" height="150">
<g >
   <rect x="20" y="25" width="80" height="80" fill="orange"/>
   <text x="30" y="75" style="font: bold 0.8em arial; fill: purple;">
     Original</text>
</g>
```

```
<g  transform="matrix(1.2 0.3 -0.9 0.8 190 75)">
   <rect x="-40" y="-40" width="80" height="80" fill="yellow"/>
   <text x="-30" y="0" style="font: bold 0.8em arial; fill: purple;">
     Matrix</text>
</g>
</svg>
```

Máscaras

Definición

Una máscara es una ventana que sólo deja ver los elementos enmascarados en ciertos lugares del contenedor svg.

Por ejemplo, podemos crear una máscara con forma de círculo, entonces los elementos que haya dentro de esa máscara sólo se verán dentro del círculo, quedando ocultos los elementos o la parte de los elementos que no estén dentro del círculo.

La forma de la máscara se delimita mediante una o varias figuras o rutas. Los elementos que pongamos dentro de la máscara sólo se verán si asoman por el espacio que ocupa la máscara. La máscara en sí misma no es visible, a no ser que pongamos una figura de fondo de otro color (por ejemplo un rectángulo) que sea más grande que la máscara.

Tenemos dos formas de hacer máscaras, mediante el elemento `clipPath` y el elemento `mask`.

El elemento clipPath

Para crear una máscara utilizamos la etiqueta `clipPath`. Pondremos siempre el atributo `id` de identificación. También podemos poner el atributo `clipPathUnits`.

```
<clipPath id="mascara1" clipPathUnits="useSpaceOnUse">
... figuras que forman la máscara ...
</clipPath>
```

El atributo `clipPathUnits` indica el origen de coordenadas de las figuras que estarán dentro de la máscara. Como valores podemos poner `userSpaceOnUse` (valor por defecto), que conserva el origen de coordenadas de la caja principal, o `objectBoundingBox` que crea un origen de coordenadas a partir del espacio que ocupa la máscara.

Dentro de la etiqueta `clipPath` pondremos las figuras que forman la máscara. Como esta es una etiqueta de definición, la incluiremos dentro de la etiqueta `defs`

Después para que uno o varios elementos estén dentro de la máscara, lo indicaremos en esos elementos con el atributo `clip-path` cuyo valor será la url hacia ese elemento:

clip-path="url(#id_mascara)"

El ejemplo que hemos puesto aquí a la derecha tiene una máscara que consiste en un círculo a través del cual se muestra una imagen y varios cuadrados semitransprentes. el código es el siguiente.

```
<svg width="270" height="270">
<defs>
```

```
<clipPath id="mascara1" clipPathUnits="useSpaceOnUse">
  <circle cx="135" cy="135" r="130" />
</clipPath>
</defs>

<g clip-path="url(#mascara1)" >
  <image xlink:href="objetos/nieve.jpg" x="0" y="0" width="350"
height="270" />
  <g fill="lime" stroke="maroon" stroke-width="5" fill-opacity="0.25">
    <rect x="5" y="5" width="130" height="130" />
    <rect x="135" y="5" width="130" height="130" />
    <rect x="5" y="135" width="130" height="130" />
    <rect x="135" y="135" width="130" height="130" />
  </g>
</g>
</svg>
```

Ponemos aquí a la derecha la imagen que hemos usado por si quieres copiarla para seguir el ejemplo.

Observa que los elementos que forman la máscara, en este caso el círculo, no necesitan tener definido el color de fondo ni los bordes ("fill" o "stroke").

También puedes observar que hemos puesto los cuatro cuadrados que forman las ventanas dentro de un elemento `g` que permite darles el mismo estilo a todos sin necesidad de repetirlo en cada uno de ellos.

El elemento mask

Podemos usar también el elemento `mask` para crear una máscara. Su uso es parecido al del elemento `clipPath`. Sin embargo tiene algunas diferencias.

Para cear la máscara utilizaremos la etiqueta `mask` que llevará el atributo `id` para identificarla:

<mask id="id_mascara"> ..... </mask>

Además del atributo `id` podemos poner los atributos `maskUnits` y `maskContentUnits`, los valores que admiten estos atributos son `userSpaceOnUse` y `objectBouldingBox`, que indican el origen de coordenadas para los elementos, tal como ocurre con `clipPathUnits`. La diferencia entre ambos atributos es que `maskUnits` se refiere a la máscara en sí, mientras que `maskContentUnits` se refiere a los elementos que se incluyen dentro de la máscara.

Para incluir elementos dentro de la máscara, a estos les pondremos el aributo `mask` cuyo valor será la url de la máscara:

mask="url(#id_mascara)"

Otra diferencia con `clipPath` es que los elementos que forman la máscara (dentro de la etiqueta mask) tienen que tener un color de fondo (fill), el componente de la escala de grises de este color se "mezcla" con el color de los elementos de la máscara, formando una transparencia, de manera que si ponemos el color blanco el elemento se vera totalmente, y si ponemos el negro el elemento no se verá.

En el ejemplo que hemos puesto aquí a la derecha hemos puesto cuatro círculos dentro de una máscara, cada uno con un color. Como elementos incluidos en la máscara hemos puesto una foto (la misma que en el ejemplo anterior) y un rectángulo semitransprente. Tanto la foto como el rectángulo cubren toda la máscara. Los diferentes tonos en cada circulo se deben a que cada uno de ellos es de diferente color.

el código empleado para hacer el ejemplo es el siguiente:

```
<svg width="270" height="270" viewBox="0 0 400 400">
<defs>
<mask id="mascara2" >
   <circle cx="120" cy="120" r="110" fill="white" />
   <circle cx="280" cy="120" r="110" fill="lime" />
   <circle cx="120" cy="280" r="110" fill="aqua" />
   <circle cx="280" cy="280" r="110" fill="yellow" />
</mask>
</defs>
<g  mask="url(#mascara2)" >
   <image xlink:href="objetos/nieve.jpg" x="-50" y="0" width="600"
height="400" />
   <rect x="0" y="0" width="400" height="400" fill="lime" opacity="0.2"/>
</g>
</svg>
```

Animación

Definición

Una animación es un cambio gradual de una propiedad o atributo de un elemento. Por gradual se entiende que es un cambio progresivo en el tiempo y que puede apreciarse a simple vista.

Las animaciones más habituales consisten en un cambio de posición, tamaño o color, aunque puede ponerse cualquier otra propiedad que sea controlada de forma numérica.

Tenemos varios métodos para hacer animaciones. Podemos en principio aplicar un código javascript para hacer una animación sobre un elemento en SVG de la misma manera que para cualquier otro elemento de HTML. También podemos usar el método de animación con CSS3, para animar las propiedades CSS que pueden aplicarse al elemento SVG.

Sin embargo SVG tiene también su propio método de animación, que consiste en una serie de etiquetas que se incluyen dentro del lenguaje SVG. Este es el que vamos a ver aquí.

La etiqueta "animate"

Esta etiqueta incluida dentro de cualquier elemento SVG, hace que podamos cambiar de forma progresiva alguno de sus atributos.

Para controlar la animación la etiqueta `animate` tiene varios atributos. por ejemplo queremos mover un cuadrado haciendo que se desplace a lo largo del eje "x". En principio pondremos:

```
<rect x="10" y="10" width="80" height="80" fill="#0000ff">
<animate attributeType="XML" attributeName="x" />
<rect>
```

Dentro de la etiqueta que define el cuadrado (`<rect ...> </rect>`) hemos puesto la etiqueta `animate`, la cual tiene los siguientes atributos:

- `attributeType` : Indica el tipo de atributo al que se le aplicará la animacion, como valores podemos poner `"XML"` (valor por defecto) o `"CSS"`. Emplearemos éste último cuando la animación se aplique a una propiedad definida mediante CSS.
- `attributeName` : Tendrá como valor el nombre del atributo al que se aplica la animación. En este caso indicamos el atributo `"x"` (eje x del rectángulo).

Debemos poner unos cuantos atributos más en `animate` para indicar el recorrido y el tiempo de la animación.

- `from` : indica el valor que tendrá la propiedad al principio de la animación.
- `to` : indica el valor que tendrá la propiedad al final de la animación.
- `dur` : indica el tiempo de duración de la animación. Éste puede expresarse en segundos (`"3s"`) o en milisegundos (`"3000ms"`).
- `begin` : se indica aquí el tiempo que transcurre desde que se da la orden de ejecutar la animación (en principio cuando se carga la página), hasta que la animación se pone en marcha.

Visto esto, modificamos el código anterior para incluir estos atributos:

```
<rect x="10" y="10" width="80" height="80" fill="#0000ff">
<animate attributeType="XML" attributeName="x"
from="10" to="180" dur="3s" begin="0s" />
</rect>
```

Aunque la animación ya funciona con este código hemos añadido dos atributos más a `animate` para que ésta sea más completa. Estos son:

- `fill` : indica lo que debe hacer el elemento al acabar la animación. Sus posibles valores son `"freeze"`: el elemento se queda tal como está al final de la animación; y `"restore"` : el elemento vuelve a su posicion o valor original.
- `repeatCount` : indica el número de veces que se repetirá la animación (1 por defecto). Como valor podemos poner un número o la palabra clave `"indefinite"`, con la cual se repetirá indefinidamente.

Anadimos estos dos atributos al código que tenemos de antes, la animación que conseguimos es la que se ve en el recuadro de la derecha, y su código es el siguiente:

```
<svg width="270" height="110">
<rect x="10" y="10" width="80" height="80" fill="#0000ff">
   <animate attributeType="XML" attributeName="x"
      from="10" to="180" dur="3s" begin="0s"
      fill="restore" repeatCount="indefinite" />
</rect>
</svg>
```

El elemento `animate` puede llevar también otros atributos. Estos son:

- `min` : Indica la duración mínima de la animación. Como valor se pondrá una unidad de tiempo (segundos o milisegundos).
- `max` : indica la duración máxima de la animación. Como valor se pondrá una unidad de tiempo.
- `restart` : Indica si la animación puede reiniciarse (por ejemplo con javascript). Los posibles valores son `"always"` (siempre), `"wenNotActive"` (cuando la animación no se está ejecutando) o `"never"` (nunca).
- `repeatDur` : Es como `repeatCount` pero aquí no se indica el número de repeticiones sino el tiempo. Se pone como valor una unidad de tiempo y la animación se repite el durante el tiempo marcado. También se puede poner como valor `"indefinite"` para repetir indefinidamente.
- `end` : se indica aquí cuando acabará la animación. Al igual que en `begin` (inicio de animación) podemos indicarlo en medidas de tiempo, pero en estos dos atributos podemos indicar el comienzo o el fin de otras formas, tal como veremos más adelante.
- `calcMode` : indica la manera en que avanza la animación en el tiempo. Sus posibles valores son `"discrete" | "linear" | "paced" | "spline"` . Cada uno de ellos imprime una velocidad diferente para cada punto de la animación. El valor por defecto es "linear" que imprime una velocidad constante. El valor "spline" requiere además de otros atributos (`keyTimes` y `keySpliners` para trazar la velocidad según una curva cuadática de Bézier. Veremos esto más adelante.
- `additive` : indica lo que debe hacerse en la siguiente repetición de la animación. Sus valores son `"replace"` en el que la siguiente animación será igual que la anterior (valor por defecto), y `"sum"` en el que en cada repetición se van sumando los valores finales a los obtenidos en el final de la repetición anterior.

Veremos más detenidamente algunos de estos atributos más adelante.

La etiqueta "set"

Esta etiqueta no produce una animación sino un cambio de estado instantáneo de un elemento

Como atributos se usan los mismos que para `animate`, sin embargo algunos de ellos no tiene sentido ponerlos al no haber un cambio gradual.

El ejemplo que vemos aquí de la derecha cambia el color de la elipse a los 7 segundos de cargar la página, para volver a su color original 10 segundos después.

El código que hemos usado para este ejemplo es el siguiente:

```
<svg width="250" height="150">
<ellipse cx="125" cy="75" rx="120" ry="70" fill="#0000ff">
   <set attributeType="XML" attributeName="fill"
      to="#ff0000" dur="10s" begin="7s"
      fill="restore" />
```

```
</ellipse>
</svg>
```

No es dificil saber qué función cumple cada atributo. Con `to` indicamos el color al que cambia el elemento. Con `begin` indicamos el tiempo de espera para que empiece la animación. `dur` marca el tiempo que dura la animación, y `fill="restore"` indica que al acabar la animación el elemento debe volver a su color original.

La etiqueta "animateMotion"

Esta etiqueta permite hacer que un elemento se mueva siguiendo una ruta determinada. Tenemos dos formas de hacerlo. La primera es hacer que el elemento siga una ruta ya existente, la segunda es crear una ruta específica para el elemento junto con la etiqueta.

Seguir una ruta ya existente

En primer lugar creamos una ruta mediante la etiqueta `path`, la única particularidad es que tenemos que ponerle el atributo `id` para identificarlo.

Después creamos el elemento que queremos que se mueva. En este caso una elipse. Las coordenadas de la figura estarán en referencia con el principio de la ruta (0,0 = principio de ruta)

Dentro de la etiqueta de la elipse pondremos la etiqueta `animateMotion`. Esta etiqueta tendrá como mínimo el atributo `dur` que indica la duración de la animación. Aquí hemos puesto también los atributos `repeatCount="indefinite"` y `rotate="auto"`, éste último indica que la figura debe rotar para estar perpendicular a la ruta.

Dentro de la etiqueta `animateMotion` pondremos otra etiqueta que hará referencia a la ruta que se debe seguir. Esta etiqueta es:

```
<mpath xlink:href="#id_ruta"/>
```

El ejemplo que mostramos aquí sigue lo explicado anteriormente, y tiene el siguiente código:

```
<svg width="270" height="270">
<path id="ruta1" d="M50,180 c0,-200 100,-150 80,-50 s30,150 100,-70
m180,120 z"
    fill="none" stroke="blue" stroke-width="3"/>
<ellipse cx="0" cy="-18" rx="25" ry="15" fill="orange" stroke="maroon"
stroke-width="3">
  <animateMotion dur="6s" rotate="auto" repeatCount="indefinite">
     <mpath xlink:href="#ruta1" />
  </animateMotion>
</ellipse>
</svg>
```

La etiqueta `animateMotion` usa los mismos atributos que `animate` para controlar la animación. Sin embargo aquí no se usan los atributos `attributeName` y `attributeType` ya que no se modifica ningún atributo.

La referencia a la ruta se hace mediante la etiqueta `<path xlink:href="id_ruta">` la cual

se coloca anidada dentro de la etiqueta `animateMotion`.

El atributo `rotate` hace que el elemento se alinee de forma perpendicular a la ruta, girando para estar siempre mostrando la misma cara delante de la ruta. Sus valores son `"auto"`: se mantiene por encima de la ruta, siguiéndola; `"auto-reverse"`: como el anterior pero se mantiene por debajo de la ruta; o podemos poner un número que indica los grados que el elemento gira respecto de su posición inicial. En este último caso el elemento mantiene siempre el mismo ángulo y no gira con la ruta.

Seguir una nueva ruta

La nueva ruta se define con el atributo `path` dentro de la etiqueta `animateMotion` este atributo tiene como valor los elementos que conforman la ruta, de la misma manera que lo haría el atributo `d` dentro de una etiqueta `path`.

El resto de atributos de `animateMotion` son iguales que para el caso anterior. La diferencia es que aquí no tenemos que hacer referencia a ninguna ruta externa ni poner la etiqueta `mpath`.

Al igual que en el caso anterior las coordenadas del elemento animado hay que ponerlas en referencia a la ruta.

La ruta no es visible más que por la trayectoria que sigue el elemento animado, ya que no está definida como tal.

El ejemplo que hemos puesto aquí a la derecha tiene el siguiente código:

```
<svg width="270" height="270" >
<rect x="-40" y="-40" width="80" height="80" fill="lime"
     stroke="green" stroke-width="2">
        <animateMotion path="M50,50 l170,0 l0,85 a85,85,0,0,1,-170,0 l0,-
85 z"
              dur="6s" rotate="auto" repeatCount="indefinite" />
</rect>
</svg>
```

La etiqueta "animateTransform"

La etiqueta `animateTransform` permite hacer animaciones en las cuales el cambio que se produce en el elemento es debido a una transformacion de las ya vistas con el atributo `transform`.

El uso de esta etiqueta es similar a las anteriores, sin embargo debemos poner siempre los atributos `attributeType="XML"` y `attributeName="transform"`.

además debemos poner siempre el atributo `type` donde indicamos el tipo de transformación:

<animateTransform attributeType="XML" attributeMode="transform" type="scale"/>

Los demás atributos son los normales de una animación, en los atributos `from` y `to` incluiremos los valores inicial y final de la transformación, es decir los valores que se ponen entre paréntesis, por ejemplo para un `type="scale"` puede ser `from="0.5 0.5" to="1.5 1.5"`, o para un `type="rotate"` podría ser `from="0deg" to="90deg"`.

La animación de la derecha está hecha con una transformación de escalado, el código de este ejemplo es el siguiente.

```
<svg width="270" height="120">
<g transform="translate(10 20)">
<rect x="0" y="0" width="80" height="80"
   fill="aqua" sroke="blue" stroke-width="3">
   <animateTransform attributeType="XML" attributeName="transform"
     type="scale" from="1 0.5"  to="2.7 1" dur="4s"
     begin="0s" fill="freeze" repeatCount="indefinite"/>
</rect>
</g>
</svg>
```

Puntos de control

Hemos visto en la página anterior cómo para controlar la animación indicamos siempre el estado de la misma al principio y al final mediante los aributos `from` (principio) y `to` final. Tenemos por tanto dos puntos de control.

Podemos también poner uno o más puntos intermedios si sustituímos los atributos `from` y to por el atributo `values`

El valor del atributo `values` será la lista de los puntos de control separados entre sí por el signo de punto y coma:

<animate values="a;b;c"... >

El tiempo marcado para la animación se distribuye uniformemente para ir desde el punto marcado al principio de la lista hasta el punto final.

Este atributo puede aplicarse tanto a la etiqueta `animate` como a la etiqueta `animateTransfrom`.

Los demás atributos para la animación son los normales vistos ya en la página anterior.

En el ejemplo de la derecha hemos puesto una animación con varios puntos de control sobre una transformación de tipo "translate". Su código es el siguiente:

```
<svg width="270" height="270">
<rect x="10" y="10" width="70" height="70"
     fill="blue" stroke="red" stroke-width="2">
                  <animateTransform attributeType="XML"
attributeName="transform"
          type="translate" values="10 10;170 10; 170 170; 10 170; 10 10"
dur="6s"
    begin="0s"  repeatCount="indefinite"/>
</rect>
</svg>
```

Encadenar animaciones

Consiste en que si tenemos varias animaciones, se vean una detrás de otra, de manera que cuando acabe una empiece la siguiente.

Los atributos `begin` (comienzo) y `(end)` (final) no sólo admiten como valor medidas de tiempo, sino que pueden marcar cuando empieza o acaba la animación respecto al comienzo o final de otras animaciones. También sirven, como veremos más adelante, para incluir eventos en las animaciones.

Para marcar el comienzo de una animación (a1) cuando termina otra (a2), en la segunda animación (a2) debemos poner el atributo `id` de identificación.

en la primera animación (a1) pondremos el atributo `begin` donde indicamos el inicio de la animación de la siguiente manera:

<animate begin="id_a2.end" ...>

aquí id_a2 es el valor del atributo `id` de la animación de referencia, y `.end` indica que la animación empezará cuando acabe la anterior. Si hubieramos puesto `.begin` indicaría que la animación empieza al mismo tiempo que la indicada.

podemos añadir también una medida de tiempo (con el signo mas + o menos -) para indicar el comienzo con retardo o con anterioridad al final de la animación anterior:

<animate begin="id_a2.end+4s" ...>

En este caso la animación empezará a los cuatro segundos de acabar la anterior.

Por último podemos poner también varias condiciones o valores para el atributo `begin`. Para ello debemos separarlos por punto y coma. Por ejemplo podemos hacer que una animación se produzca al cargarse la página y también cuando acabe otra.

<animate begin="0s;id_a2.end+4s" ...>

Lo explicado aquí para el atributo `begin` es exactamente igual para el atributo `end` con la unica diferencia de que no indicamos el principio de la animación, sino el final, siempre que ésta esté en marcha.

El ejemplo mostrado a la derecha encadena cuatro animaciones una tras otra de manera que formen un bucle continuo. Su código es el siguiente:

```
<svg width="270" height="290">
<rect x=10 y=10 width=80 height=60
    fill="yellow" stroke="navy" stroke-width="3">
    <animate  id="r1" attributeType="XML" attributeName="x"
       values="10;170;10" dur="2s" begin="0s;r4.end+2s"/>
</rect>
<rect  x=10 y=80 width=80 height=60
    fill="red" stroke="navy" stroke-width="3">
    <animate id="r2" attributeType="XML" attributeName="x"
       values="10;170;10" dur="2s" begin="r1.end"/>
</rect>
<rect  x=10 y=150 width=80 height=60
    fill="green" stroke="navy" stroke-width="3">
    <animate id="r3" attributeType="XML" attributeName="x"
       values="10;170;10" dur="2s" begin="r2.end"/>
</rect>
<rect  x=10 y=220 width=80 height=60
    fill="blue" stroke="navy" stroke-width="3">
    <animate id="r4" attributeType="XML" attributeName="x"
```

```
       values="10;170;10" dur="2s" begin="r3.end"/>
</rect>
</svg>
```

Observa cómo lo que hace que las animaciones vayan encadenadas es el atributo `begin`.

Eventos de ratón

Podemos hacer también que la animación empiece cuando el usuario haga alguna acción con el ratón, es lo que se entiende como eventos de ratón.

Puede haber los siguientes eventos de ratón:

- `click` : Se produce cuando se hace clic con el ratón sobre un elemento.
- `mousedown` : Se produce mientras se pulsa el botón (sin soltarlo) del ratón sobre un elemento.
- `mouseup` : Se produce al soltar el botón del ratón, que estaba pulsado, sobre un elemento.
- `mouseover` : Se produce al entrar el ratón dentro de un elemento (ponerse encima).
- `mouseout` : Se produce al salir el ratón de un elemento.
- `mousemove` : Se produce al moverse el ratón dentro de un elemento.

ROTAR INCLINAR PARAR

Para que el evento funcione el elemento sobre el que actúa el raton debe llevar un atributo identificador `id`.

<text id="rotar" ...>ROTAR</text>

Después en la animación lo indicaremos en el atributo `begin`.

begin="rotar.click"

Donde `rotar` es el "id" del elemento de control, y `click` es el evento que queremos aplicar.

Si aplicamos el evento de la misma manera sobre el atributo `end` conseguiremos que la animación se pare:

end="parar.click"

Al pulsar sobre el elemento que lleva el `id="parar"` la animación se detiene.

El ejemplo de la derecha tiene tres animaciones controladas por eventos, La rotación y el sesgado se activan pulsando en los textos superiores y se desactivan pulsando en el texto inferior. Además el cuadrado va cambiando de color mientras el ratón este encima. El código de este ejemplo es el siguiente:

```
<svg width="270" height="270">
<!--recuadros y textos de control -->
<rect x="5" y="5" width="260" height="30px"m fill="yellow" stroke="navy"/>
<rect x="5" y="235" width="260" height="30px"m fill="yellow"
stroke="navy"/>
<text id="rotar" x="15" y="30"
    style="fill: navy; font: bold 20px 'comic sans ms'; cursor: pointer">
```

```
    ROTAR</text>
<text id="inclinar" x="120" y="30"
    style="fill: navy; font: bold 20px 'comic sans ms'; cursor: pointer">
    INCLINAR</text>
<text id="parar" x="90" y="260"
    style="fill: navy; font: bold 20px 'comic sans ms'; cursor: pointer">
    PARAR</text>
<g transform="translate(135 135)">
        <!-- elemento al que se le aplican las animaciones -->
<rect id="rect1" x="-50" y="-50" width="100" height="100"
    fill="#ff0000" stroke="black" stroke-width="2">
        <!-- animaciones: 1ª rotación, 2ª inclinacion, 3ª color -->
    <animateTransform attributeType="XML" attributeName="transform"
        type="rotate" values="0;360" dur="3s"
        repeatCount="indefinite" begin="rotar.click" end="parar.click"/>
    <animateTransform attributeType="XML" attributeName="transform"
        type="skewX" values="-30;45;-30" dur="3s"
        repeatCount="indefinite" begin="inclinar.click"
end="parar.click" />
    <animate atributeType="XML" attributeName="fill"
        values="#ff0000;#00ff00;#0000ff;#ff0000" dur="4s"
        repeatCount="indefinite" begin="rect1.mouseover"
end="rect1.mouseout" />
</rect>
</g>
</svg>
```

Eventos de teclado

Podemos tambien controlar la animación mediante el teclado, de forma que al pulsar una tecla determinada se active o desactive la animación.

Los eventos de teclado se marcan también en los atributos `begin` y `end` con el siguiente valor:

begin="accessKey(caracter)"

Ponemos primero la palabra clave `accessKey` y después entre paréntesis el caracter de la tecla que activa la animación. Por ejemplo `begin="accessKey(c)"` activará la animación al pulsar la tecla "c".

Teclas: "c" "1" empezar ; "f" "0" acabar

De igual manera `end="accessKey(f)"` parará la animación al pulsar la tecla "f".

Podemos poner varios comandos de eventos en un mismo atributo siempre que vayan separados por punto y coma. Además los botones para eventos de ratón podemos ponerlos fuera del contenedor SVG siempre que lleven su correspondiente "id".

El ejemplo de la derecha lleva el sigiente atributo:

begin="accessKey(1);accessKey(c);on.click"

...

Esto permite empezar la animación tanto al pulsar en la tecla "1", como en la tecla "c", como en el botón de empezar" de aquí arriba.

Para acabarla podemos usar la tecla "0", la tecla "f" o el botón de "acabar", para ello hemos puesto

este código:

end="accessKey(0);accessKey(f);off.click"

El código completo de la animación es el siguiente:

```
<svg width="270" height="270">
<path id="ruta1" d="M50,180 c0,-200 100,-150 80,-50 s30,150 100,-70 l0,0
z"
    fill="none" stroke="blue" stroke-width="3"/>
<ellipse cx="0" cy="-18" rx="25" ry="15"
    fill="orange" stroke="maroon" stroke-width="3">
    <animateMotion dur="6s" rotate="auto" repeatCount="indefinite"
        begin="accessKey(1);accessKey(c);on.click"
        end="accessKey(0);accessKey(f);off.click">
        <mpath xlink:href="#ruta1" />
    </animateMotion>
</ellipse>
<rect x="5" y="245" width="260" height="20"
    fill="lime" stroke="maroon" stroke-width="2"/>
<text x="10" y="260"
    style="fill: maroon; font: normal 13px 'comic sans ms';">
    Teclas: "c" "1" empezar ; "f" "0" acabar</text>
</svg>
```

Más sobre SVG

Presentación

Con lo visto hasta ahora ya podemos manejarnos de una forma básica en SVG para hacer algunos dibujos y animaciones. Todo depende de la habilidad que tengamos para realizar nuestros gráficos.

Nos hemos dejado algunas cosas por explicar, las cuales dan otras posibilidades a SVG, las diremos en esta página, aunque sea por encima.

Filtros

Los filtros son elementos que modifican el aspecto de uno o varios elementos del gráfico. Como su nombre indica "filtran" los elementos de forma que sabiéndolos utilizar bien podemos obtener algunos efectos.

Hay 16 filtros diferentes, unos inciden sobre la luz, otros sobre la dispersión de los colores, otros deforman la figura, etc.

No vamos a explicar aquí el funcionamiento de todos ellos, ya que esto requeriría otro manual completo, sin embargo vemos aquí un ejemplo para hacernos una idea de lo que éstos pueden hacer sobre las figuras.

En el recuadro de la derecha hemos puesto una figura repetida varias veces. El cuadrado de arriba a la izquierda es el original, sin aplicarle ningún filtro. A los otros cuadrados les hemos aplicado varios filtros, modificando su estado original.

De forma general un filtro se aplica mediante la etiqueta `filter` la cual lleva un "id" de identificación para poder ponerlo luego en el sitio deseado. Dentro de esta etiqueta se incluye la

etiqueta específica del filtro, la cual llevará también sus propios atributos, unos generales, y otros específicos de cada filtro.

Por si teneis curiosidad incluyo aquí el código del ejemplo aquí visto.

```
<svg width="270" height="270" id="svg1"
    xmlns="http://www.w3.org/2000/svg" version="1.1">
<defs>
<filter id="filtro1" x="0" y="0" width="100" height="100"
filterUnits="useSpaceOnUse">
    <feGaussianBlur stdDeviation="3,2"  />
</filter>
<filter id="filtro2" x="0" y="0" width="100" height="100"
filterUnits="useSpaceOnUse">
    <feColorMatrix type="hueRotate" values="180" />
</filter>
<filter id="filtro3" x="0" y="0" width="100" height="100"
filterUnits="useSpaceOnUse"  >
    <feSpecularLighting in="SourceAlpha" >
    <feSpotLight x="150" y="150" z="50" pointsAtX="200" pointsAtY="200"
pointsAtZ="0"
        limitingConeAngle="80" specularExponent="3" />
    </feSpecularLighting>
</filter>
</defs>
<rect x=0 y=0 width=270 height=270 fill="aqua" />
<rect x=20 y=20 width=100 height=100
   fill="lime" stroke="red" stroke-width="10"/>
<rect x=150 y=20 width=100 height=100
   fill="lime" stroke="red" stroke-width="10"
         filter="url(#filtro1)"/>
<rect x="20" y="150" width="100" height="100"
   fill="lime" stroke="red" stroke-width="10"
         filter="url(#filtro2)"/>
<rect x="150" y="150" width="100" height="100"
   fill="lime" stroke="red" stroke-width="10"
         />
<rect x="150" y="150" width="100" height="100"
         filter="url(#filtro3)"/>
</svg>
```

Para saber más sobre los filtros podeis consultar la página oficial del W3C donde lo explica. La página está en ingles, y puede que resute un poco complicado de entenderlo. Es la siguiente:

http://www.w3.org/TR/2011/REC-SVG11-20110816/filters.html

La aplicación Inkscape

Inkscape es un programa o aplicación que puede ayudarnos a realizar gráficos de SVG, tanto si éstos son simples, como más complicados.

Esta es una aplicación gratuíta que crea archivos SVG de una manera gráfica. Sin duda es una buena alternativa a los clásicos gráficos hechos con "flash", ya que además el usuario no necesita tener instalado en su ordenador ningún plugin.

Podemos descargar Inkscape desde su página oficial:

http://www.inkscape.org/es/descargas/

Desde esta página podemos elegir descargar la versión que mejor se adapte a nuestro sistema operativo. Si tenemos *Windows* debemos descargarnos la de "Windows / instalador", para instalarlo como cualquier otro programa.

Una vez instalado, vemos que se parece bastante a cualquier otro programa de dibujo, pero con una mayor cantidad de opciones. Como toda aplicación un poco compleja debemos aprender a utilizarlo. Para ello Inscape tiene varios tutoriales que podemos seguir y nos indican paso a paso cómo utilizarlo.

Los gráficos creados con Inscape son archivos en SVG que podemos incorporar directamente a la página como si fuera una imagen un "iframe" o un objeto incrustado con la etiqueta "object" o "embed".