Capítulo 3. Transformaciones y Animaciones

3.1.Transform.

La especificación oficial y el estado actual de desarrollo del módulo Transforms en CSS3 puede consultarse en http://www.w3.org/TR/css3-transforms/.

Al modificar el espacio de coordenadas, las transformaciones CSS permiten cambiar la posición del contenido afectado sin interrumpir el flujo normal del resto de cajas. Se llevan a cabo mediante un conjunto de propiedades CSS que permiten aplicar transformaciones lineales afines a los elementos HTML. Estas transformaciones incluyen la rotación, inclinación, escala y traslación tanto en el plano como en un espacio tridimensional.

Propiedades principales

Se utilizan dos **propiedades** principales para definir las transformaciones CSS: *transform y transform-origin.*

- **transform-origin**: especifica la posición de origen de la transformación. Por defecto se encuentra en el centro del elemento (como si definiéramos el punto en 50% 50%), pero se puede definir cualquier otro punto. Es utilizado por varias transformaciones, como rotación, escala o inclinación, que necesitan un punto inicial como parámetro.

- **transform**: especifica las transformaciones a aplicar al elemento. Se trata de una lista de funciones de transformación separadas por espacios, que se aplican una detrás de otra.

TRANSFORM

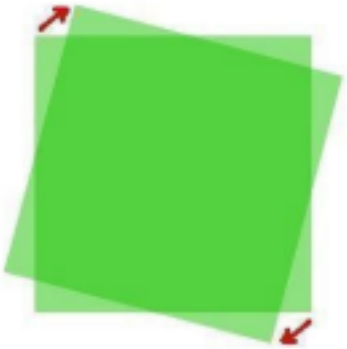
El atributo **transform** nos permite, como su propio nombre indica, transformar un elemento. Su sintaxis es la siguiente:

```
transform: tipo(cantidad);
```

El valor tipo puede tomar cuatro valores y cada uno de ellos realiza una función diferente:

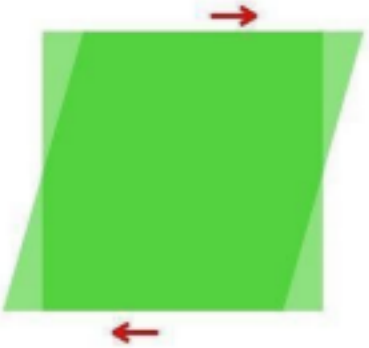
- ***Rotate****:* Nos permite girar los elementos un número de grados. La sintaxis es:

```
transform:rotate(25deg);
```



El valor que demos al giro se aplicará en sentido horario

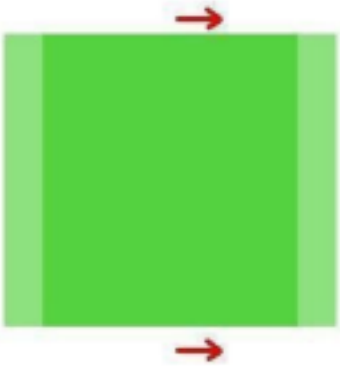
- ***Skew**:* Podemos inclinar un elemento tanto en coordenadas X como Y. El valor se expresa en grados y la sintaxis es la siguiente: `/*transform: skew(gradosX, gradosY);*/ transform: skew(15deg, 3deg);`



- ***Scale***: Con este tipo podremos escalar nuestro elemento tanto en X como en Y en una cantidad expresada en tantos por uno: `/*transform: scale(escalaX,escalaY);*/ transform: scale(1.5,0.6);*/`

- ***Translate**:* Podemos desplazar el elemento tanto en X como en Y.

```
 /*transform: translate(desplazamientoX, desplazamientoY);*/
transform:translate(12px, 19px);
```



NOTA: Se pueden aplicar diferentes transformaciones a un mismo elemento simplemente inscribiéndose de manera consecutiva: `transform: scale(1.6) skew(10deg) translate(5px) rotate(12deg);`

3.1.2 Transform-origin.

Especifica la posición de origen de la transformación, puede tomar los parámetros bottom, right ,top y left.

```
div { transform: rotate(90deg); transform

   origin: bottom left;

 }

 <div>

    <iframe src="http://www.google.com/" width="500" height="600"></iframe>
</div>
```

3.2 Propiedades 3D.

La realización de transformaciones CSS en el espacio es algo más complejo. Hay que empezar configurando el espacio 3D, dándole un punto de vista. Después, hay que configurar cómo se comportan los elementos 2D en ese espacio tridimensional.

Las siguientes propiedades, añaden control adicional a las transformaciones, permitiendo realizar **transformaciones 3D**:

- **perspective**: permite cambiar la perspectiva de los elementos y transmitir la sensación de encontrarse en un entorno en tres dimensiones.

- **perspective-origin**: especifica la posición de origen de la perspectiva.

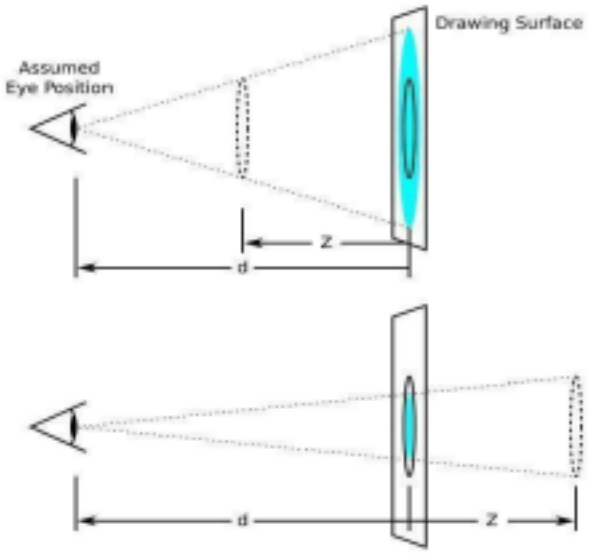
- **transform-style**: permite a los elementos transformados en 3D y a sus descendientes también

transformados en 3D, compartir un espacio 3D común.

3.2.1 La creación de un punto de vista (o perspectiva).

El primer elemento a configurar es la perspectiva. La perspectiva es lo que da la impresión de tres dimensiones. Cuánto más lejos del espectador se encuentran los elementos, más pequeños se nos muestran.

Cuán rápido estos elementos reducen su tamaño es definido por la propiedad perspective. Cuanto más pequeño es su valor, más profunda es la perspectiva.



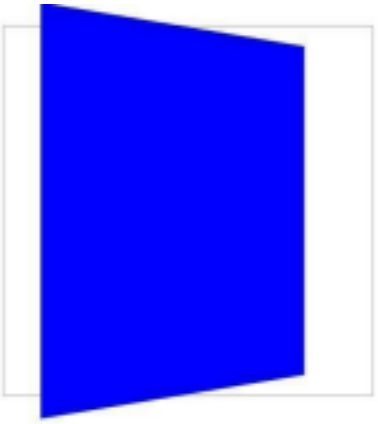
La perspectiva se podrá aplicar de dos maneras:

a) Mediante la propiedad **transform: perspective (valor);**

```
.box {
        background-color: red;
        transform: perspective( 600px ) rotateY( 45deg );
    }
```

b) Mediante la propiedad **perspective**: `#blue {`

```
perspective: 600px; }
#blue .box {
        background-color: blue; transform:
        rotateY( 45deg );
    }
```



El segundo elemento a configurar es la posición del espectador, con la propiedad perspective origin. Por defecto, la perspectiva se centra en el espectador, lo cual no siempre es lo adecuado. Esta propiedad es equivalente a la propiedad Transform-origin en 2d. Una vez realizado esto, se puede trabajar sobre el elemento en el espacio 3D.

3.3.2 Transform-style

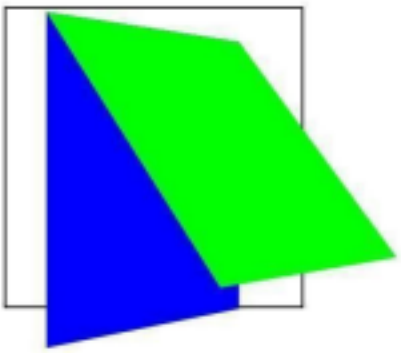
Esta es otra propiedad importante en el espacio 3D. Se necesitan dos valores*: **preserve-3D o flat*** .Cuando el valor de estilo de transformación es *preserve-3d* entonces le dice al navegador que el elemento de los hijos también se debe colocar en el espacio 3D. Por otro lado, cuando el valor de esta propiedad es flat, indica que los hijos están presentes en el plano del propio elemento. Ejemplo:

```
div { height: 150px;
      width: 150px;
      margin: 10px auto 50px; }
.contenedor {
      border: 1px solid black;
       -webkit-perspective: 500px; }
.transformar {
       background-color: blue;
       -webkit-transform-style: preserve-3d;

       -webkit-transform: rotateY(50deg); }
```

```
.hijo {
      -webkit-transform-origin: top left; -webkit
      transform: rotateX(40deg); background-color:
      lime; }
```

```
<div class="contenedor">
  <div class="transformar">
        <div class="hijo"></div>
 </div>
</div>
```



3.3.3 Funciones de transformación 3D

Funciones que se utilizan en la transformación 3D:

- **Translate3d** (<translation-value>, <translation-value>, <longitud>) : define una translación 3D.Lleva tres parámetros x, y, z valores. El valor Z especifica la translación en el eje Z.

- **TranslateZ** (<longitud>) : Funciona de manera similar a TranslateX () y TranslateY () . para el eje Z.

- **Scale3d** (<número>, <número>, <número>) : Esta función hace el escalamiento en las tres dimensiones. Se necesitan tres parámetros como sx, sy y sz. Cada valor define la escala en la dirección correspondiente.

- **ScaleZ** (<número>) : Al igual que el translate () función, igual scaleZ () define la escala en una sola dirección, es decir en la dirección Z. También tenemos scaleX ()y scaleY () funciones que también trabajan similar a scaleZ () , pero en sus respectivas direcciones.

- **Rotate3d** (<número>, <número>, <número>, <ángulo>) : Gira un elemento HTML en el ángulo especificado en el último parámetro de [tx, ty, tz] vector especificado por los primeros tres números.

- **RotateX** (<ángulo >) , **rotateY** (<angle>) y **RotateZ** (<ángulo >) tienen un único valor de

ángulo para rotar en el eje correspondiente.

Nota: Rotar3D (1,0,0,30 grados) es igual a RotateX (30deg) , Rotar3D (0,1,0,30 grados) es igual a rotateY (30deg) y Rotar3D (0,0,1,30 grados) es igual a RotateZ (30deg).

3.4. Animaciones

3.4.1 ¿Qué es una animación?. Tipos de animaciones

Las animaciones son pequeños cambios en el contenido o aspecto visual de una página, de modo que sea más amigable su visualización. Pueden ser sencillas, como cambiar el color o tamaño de una imagen, así como animaciones más complejas, como por ejemplo transformaciones o movimientos específicos.

Tipos de animaciones

Descripción
Una animación simple, desde un estado inicial a un estado final.
Una animación más compleja, con 2 o más estados.
Movimiento de un elemento a lo largo de un trayecto o ruta.
Animación controlada por desplazamiento de ratón.
Animación de vistas, al cambiar de una página a otra.
Animaciones nativas complejas, creadas desde Javascript.

Tipo de animación
Transiciones
Animaciones
Trayectos animados
Animaciones de scroll
View Transition
WebAnimations

3.4.2 Módulo Animations

Será un conjunto de transiciones entre estados, de modo que es posible cambiar entre dos o más estados.

3.4.2.1 Fotogramas claves de las animaciones

Un fotograma clave no es más que un punto destacado en el tiempo de nuestra animación. Cualquier animación consta al menos de dos fotogramas claves: el punto inicial y el punto final. Imaginad que nuestra animación es como una carretera:

El primer semáforo actuaría como fotograma clave inicial y el segundo como el final. Entre uno y otro se produciría nuestra animación, que no es más que el desplazamiento del coche hacia la derecha.

3.4.2.2 @keyframes

En CSS3 creamos animaciones completas mediante @keyframes, que son un conjunto de fotogramas clave. Su sintaxis es la siguiente:

```
@keyframes nombreAnimacion{
                puntoDelKeyframe{ atributosIniciales; }
```

```
                puntoDelKeyframe{ nuevosAtributos; }

                .......

                puntoDelKeyframe{ últimosAtributos;}
    }
```

Veamos que tendríamos que hacer para desplazar nuestro coche a la derecha:

```
    @keyframes animacionCoche{

8
                /*Indicamos que salimos de la posición 0*/

                from{ left:0px; }

                /*Indicamos que al final la posición debe ser 350*/

                to{ left:350px; }
    }
```

Podemos crear animaciones más complejas estableciendo fotogramas claves intermedios mediante porcentajes:

```
    @keyframes animacionCoche{

                from{ left:0px;}

          /*Hasta el 65% de la reproducción sólo queremos que se desplace 10 pixels*/

                65%{ left:10px;}

                to{ left:350px;}
    }
```

3.4.2.3 Aplicando la animación a un elemento

Hemos creado nuestra animación. Para aplicarla sobre un elemento de nuestra página, deberemos acudir al mismo y agregarle el atributo de animación:

```
    #coche{
          animation-name: animacionCoche; animation
            duration: 3s; animation-iteration-count: 1;
            position:relative;
     }
```

Hay tres nuevos atributos que no habíamos visto antes. Estos y otros tantos conforman los atributos destinados a la animación que describo a continuación:

- **Animation-name.** Indica a qué animación corresponde nuestro elemento. Es posible definir más de una animación usando comas como separador.

• **Animation-duration.** Define la duración en segundos de nuestra animación.

• **Animation-iteration-count**. Define cuántas veces se reproduce la animación. Podemos darle el valor *infinite* para que se reproduzca hasta el fin de los tiempos.

• **Animation-direction.** Por defecto nuestra animación se reproducirá hacia delante, como es lógico. Sin embargo si le damos el valor *alternate*, al reproducirse completamente la animación, esta volverá a reproducirse en sentido opuesto, es decir, como si se rebobinara.

• **Animation-delay**. Indica si se produce un retardo en el inicio de la animación. •

Animation. El shorthand de todos los anteriores:

9

```
name duration timing-function delay iteration-count direction;
```

• **Animation-timing-function.** Define cómo progresa la animación entre los keyframes mediante ecuaciones matemáticas.

• **Animation-fill-mode**: Esta propiedad especifica un estilo para el elemento cuando la animación no se está reproduciendo (antes de que comience, después de que termine o ambos). *forwards*, mantendrá los estilos una vez termine la animación.

3.4.2.4 animation-timing-function

Este atributo sirve para aplicar un efecto suavizado a la animación.

Por defecto responde al valor *ease*, pero puede responder a diferentes valores:

- **Ease**
- **Linear**
- **Ease-in**
- **Ease-out**
- **Ease-in-out**

3.4.2.5 Anidar y encadenar animaciones

Como en muchas propiedades de CSS, es posible separar sus valores mediante comas para indicar múltiples valores. En este caso, esos múltiples valores indicarán que queremos realizar varias animaciones a la vez.

```
animation: move-car 5s, change-color-car 5s;
```

Es posible encadenar animaciones utilizando animaciones múltiples combinadas con la propiedad *animation-delay*.

```
move-right 5s 0s, /* Comienza a los 0s (no hay anterior) */
```

```
change-color-car 2.5s 5s, /* Comienza a los 5s (5 de la anterior) */
```

3.4.2.6 Animar sprites

Un sprite es una imagen o animación bidimensional que se utiliza para representar personajes, objetos o efectos dentro de un entorno digital. Pero los sprites nos permiten hacer aún más cosas,

como animaciones. En las que iremos recorriendo el sprite, a modo de animación stop-motion, consiguiendo un efecto similar al de un GIF animado.

```
.zombi-sprite {
width: 155px;
height: 194px;
background: url('zombi-sprite.png');
animation: andarzombi 0.7s steps(5) infinite; }

@keyframes andarzombi{
from{background-position:10px;
}
to{background-position:-800px;}
}

<div class="zombi-sprite"></div>
```

3.5 Módulo Transition

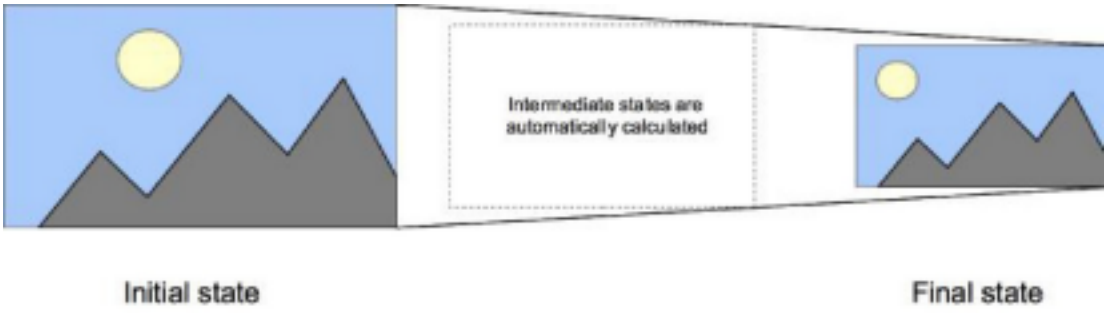
La especificación oficial y el estado actual de desarrollo del módulo Transitions en CSS3 puede consultarse en http://www.w3.org/TR/css3-transitions/.

Lista de propiedades que se pueden animar: enlace

Las transiciones CSS, que forman parte del conjunto de especificaciones de CSS3, proporcionan una forma de controlar la velocidad de la animación al cambiar las propiedades CSS. Los cambios en las propiedades no tienen efecto inmediato, sino que se puede establecer un periodo de tiempo para que éstos se ejecuten. Por ejemplo, al cambiar el color de un elemento de blanco a negro, este cambio normalmente se ejecuta de manera inmediata. Sin embargo, con las transiciones CSS, los cambios se

producen con intervalos de tiempo que siguen una curva de aceleración, y pueden ser personalizados.

Las animaciones que implican transición entre dos estados se llaman a menudo transiciones implícitas, ya que los estados intermedios entre el inicial y el final son implícitamente definidos por el navegador.



Las transiciones CSS permiten decidir qué propiedades animar (mediante su inclusión explícita), cuándo comenzará esta animación (estableciendo un retraso o delay), cuánto durará (estableciendo una duración), y cómo se ejecutará (definiendo una función de tiempo).

3.5.1 Propiedades "animables"

Se puede definir qué propiedad debe ser animada y en qué manera. Esto permite la creación de transiciones complejas. Como la animación de algunas propiedades no tiene sentido, la lista de propiedades "animables" se limita a un conjunto finito.

- Lista de propiedades animables.

También el valor auto es un caso complejo. La especificación nos dice que no debemos animar desde y hacia dicho valor. Algunos agentes de usuario, como los basados en Gecko o WebKit, implementan este requisito, que al usar animaciones con auto nos puede dar lugar a resultados impredecibles, dependiendo del navegador y su versión; por lo que debemos evitarlo.

3.5.2 Propiedad transition-property

El primer paso al crear una transición, es especificar la propiedad o propiedades sobre las que se va a aplicar los efectos de la transición. Podemos decidir que sean todas las propiedades, ninguna o un listado de ellas.

```
transition-property: none | all | [ <property> ] [, <property> ]*
transition-property: all;
transition-property: none;
```

```
transition-property: background-color;
transition-property: background-color, height, width;
```

Si se indica la palabra reservada all, todas las propiedades que sean capaces de ser animadas y para las que se ha definido un cambio, serán animadas. Si se especifica none, ninguna propiedad será animada.

3.5.3 Propiedad transition-duration

La propiedad transition-duration acepta una lista separada por comas de tiempos, especificadas en segundos o milisegundos, que determinan cuánto tiempo tarda cada propiedad, en completar la transición.

```
transition-duration: <time> [, <time>]*
transition-duration: 2s;
transition-duration: 4000ms;
transition-duration: 4000ms, 8000ms;
```

3.5.4 Propiedad transition-timing-function

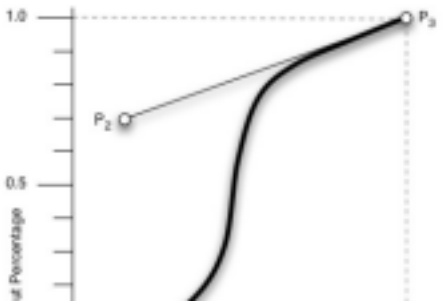
La propiedad transition-timing-function es utilizada para especificar el ritmo en el que se producen los cambios durante la transición. Esto puede realizarse de dos maneras: indicando el nombre de una función de tiempo (ease, linear, ease-in, ease-out o ease-in-out) o definiendo una función de tiempo personalizada (especificando cuatro coordenadas para definir una curva bezier).

```
transition-timing-function: <timing-function> [, <timing-function>]*
transition-timing-function: ease;
transition-timing-function: ease, linear;
```

Además de estas funciones de tiempo predefinidas, tenemos la posibilidad de declarar nuestra propia función de tiempo, utilizando una función cubic-bezier.

```
transition-timing-function: cubic-bezier(0.6, 0.1, 0.15, 0.8);
```

Los valores de la curva Bezier



serán para los puntos P1 y P2, (x1,y1,x2,y2) donde X debe de estar en el rango [0,1] e Y puede salir de este.

Figura Puntos de control en una curva bezier

Si no se especifica ninguna función de tiempo, por defecto se aplica ease.

Ver: **wikipedia.org y cubic-bezier.com**

13

3.5.5 Propiedad transition-delay

El último paso para crear una transición, es especificar un retraso (opcional) en el inicio de la transición. Aquí también podemos especificar una lista de tiempos, en segundos o milisegundos, que determinan el inicio de la transición desde que esta se lanza. El valor por defecto es 0, esto es, se inicia de inmediato.

En este caso los valores negativos sí son aceptados. En este caso, la transición se iniciará tan pronto sea posible, pero dará la impresión que ya lleva tiempo ejecutándose.

```
transition-delay: <time> [, <time>]*
transition-delay: 5s;
transition-delay: 4000ms, 8000ms;
```

3.5.6 Propiedad transition

Como de costumbre, disponemos de la propiedad shorthand que nos permite definir todas las propiedades de una sola vez.

```
transition: <transition> [, <transition>]*
<transition> = <transition-property> <transition-duration>
<transition-timing-function> <transition-delay>
transition: background-color 3s linear 1s;
transition: 4s ease-in-out;
transition: 5s;
```

El único valor requerido en esta propiedad es transition-duration.

3.5.7 Listas de valores

Si la lista de valores de cualquier propiedad es más corta que otras, sus valores son repetidos hasta hacer que coincidan. Por ejemplo:

```
div {
transition-property: opacity, left, top, height;
transition-duration: 3s, 5s;
}
```

Esto es tratado como si fuese:

```
div {
       transition-property: opacity, left, top, height;
       transition-duration: 3s, 5s, 3s, 5s; }
```

De manera similar, si la lista de valores de cualquier propiedad es más larga que la detransition property, es acortado, por lo que si tienes este código CSS:

```
div {
       transition-property: opacity, left;
```



```
       transition-duration: 3s, 5s, 2s, 1s;
}
```

Se interpreta como

```
div {
       transition-property: opacity, left;
       transition-duration: 3s, 5s;
}
```

3.5.8 Finalización de una transición

Existe un sólo evento que se dispara cuando las transiciones se completan. En todos los navegadores que cumplen el estándar, el evento es transitionend, en WebKit es *webkitTransitionEnd*. El evento ***transitionend*** ofrece dos propiedades:

- **propertyName**: string que indica el nombre de la propiedad CSS cuya transición está completada.

- **elapsedTime**: un float que indica el número de segundos que la transición ha estado ejecutándose en el momento en el que se dispara el evento. Este valor no está afectado por el valor de transition-delay.

Se puede utilizar el método element.addEventListener() para supervisar este evento:

```
el.addEventListener("transitionend", updateTransition, true);
```

3.5.9 Transiciones y JavaScript

Las transiciones son una buena herramienta para crear una apariencia mucho más equilibrada sin tener que modificar la funcionalidad JavaScript. Por ejemplo:

```
<p>Hacer click para mover la bola</p>
<div id="mover"></div>
```

Utilizando JavaScript:

```
var f = document.getElementById('mover');
document.addEventListener('click',
function(ev){
	f.style.left = (ev.clientX-25)+'px';
	f.style.top = (ev.clientY-25)+'px';
},false);
```


Con CSS simplemente es necesario añadir una transición al elemento:

```
p {
	padding-left: 60px; }
#bola {
	border-radius: 50px;
	width: 50px;
	height: 50px;
	background: #c00;
	position: absolute;
	top: 0;
	left: 0;
	transition: all 1s; }
```

3.5.10 Optimizar animaciones

Las transiciones y animaciones CSS son un mecanismo super potente para crear animaciones fluidas de forma sencilla en nuestras páginas. Sin embargo, es probable que realizándose, de repente descubras que no funcionan como esperabas o que al navegador le cuesta moverlas de forma suave y fluida.

Dependiendo del código escrito (y otros factores) **puede que nuestras animaciones sean costosas para el navegador** y no se vean lo bien que esperamos. Vamos a repasar una serie de consejos que mejorarán el rendimiento de nuestras animaciones.

Como ya vimos, al interpretar código CSS, los navegadores pasan por varias fases:

- **Javascript**: Esta fase se activa cuando Javascript está realizando tareas visuales que, aunque se podrían hacer con CSS, se están realizando con Javascript.
- **Styles**: En esta fase, se revisan los selectores y se analiza qué reglas CSS se aplican a qué elementos del documento HTML.
- **Layout**: Esta fase calcula los tamaños, posiciones y dimensiones que los elementos ocupan en una página. Además, el cambio de un elemento puede afectar a otros como hermanos o hijos. ● **Paint**: Es el proceso de renderización en varias capas, relacionado con colores, imágenes, bordes, sombras, textos, etc...
- **Composite**: La composición es la fase donde se organizan o reordenan las diferentes capas para mostrar visualmente de la forma correcta.

No todas las propiedades CSS afectan a la misma fase. Ciertas propiedades CSS como *transform, opacity, z-index o cursor*, sólo repercuten en la fase de Composite, por lo que son muy eficientes.

Otras propiedades como *color, background, outline, border-radius, box-shadow* (u otras) afectan a la fase Paint, la cuál luego necesita revisar la fase Composite, por lo que serán algo más costosas.

Propiedades como *width, height, margin, padding, display, font-size, text-align* (u otras) son de las propiedades CSS más costosas, ya que causan un reflow (proceso en el que cambia la geometría) y hay que recalcular esta fase de layout, la fase de pintado y la fase de composición.

NOTA: A la hora de animar estas propiedades, es importante analizar si es posible darle prioridad o buscar formas alternativas de hacer animaciones mediante las propiedades menos costosas.

- Evita usar **all** en las transiciones. Ej: *transition: all 1s linear;*
- Evita usar Javascript. Realizar animaciones con CSS es mucho más eficiente y adecuado.

3.5.11 Trayectos Animados

En CSS existen los trayectos animados, también denominados como **Motion Paths**. Se trata de una característica mediante la cuál podemos crear de forma muy sencilla trayectos, rutas o caminos por los que animamos un elemento.

Propiedades de trayectos animables

Para crear estos trayectos debemos realizar dos tareas. En primer lugar, **definir el trayecto**. En segundo lugar, **crear la animación para mover un elemento** a través de dicho trayecto, definiendo su punto de partida y su punto final.

Para hacer esto, se utilizan una serie de propiedades CSS prefijadas por offset- :

- **offset-path**: Define un trayecto, ruta o camino.
- **offset-distance:** Indica la posición actual en el trayecto.
- **offset-position:** Define el punto de inicio del trayecto.
- **offset-rotate:** Define la orientación del elemento del trayecto.
- **offset-anchor:** Define el punto de anclaje (punto de origen) del elemento que se mueve por el trayecto.
- **offset:** Propiedad de atajo de las anteriores.

Definir un trayectoria

La propiedad **offset-path** creará el camino por el que se puede mover el elemento que animaremos más adelante.

Hay varias formas de crear trayectos con offset-path: link

17

- **none** No establece ningún trayecto. Es el valor por defecto.
- **Forma básica CSS** Una forma básica, mediante funciones como xywh(), circle(), polygon() o similares.
- **Función ray()** Permite definir trayectos como porciones de un círculo (ángulo cónico). ● **Función path()** Permite definir trayectos mediante rutas de SVG.
- **Función url()** Permite definir trayectos mediante ficheros externos SVG.

Ejemplo forma básica:

```
.element {
 border-radius: 50%;
 width: var(--size);
 height: var(--size);
 background-color: aquamarine;
 offset-path: polygon(50px 50px, 350px 0, 350px 100px, 0 100px);
animation: move 5s infinite linear;
```

```
}
@keyframes move {
0% { offset-distance: 0%; }
100% { offset-distance: 100%; }
}
```

Ejemplo función **url():**

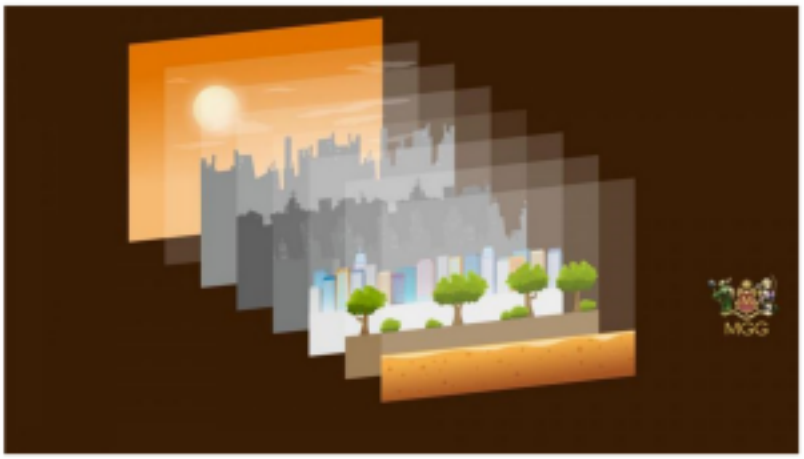
```
.box {
position: absolute;
width: 50px;
height: 50px;
animation: move 3s alternate infinite;
background: indigo;
offset-path: url("#triangle");
}
@keyframes move {
0% {offset-distance: 0%; }
```



```
100% { offset-distance: 100%; }
}
////////////////////////////////////////////////////////////////////////////
/ </style>
<svg width="450" height="200" viewBox="0 0 450 200">
<path id="triangle" fill="none" stroke="#000" stroke-width="2" d="M0,0 L200,0
L200,200 L0,0 Z" />
</svg>
```

3.6 Control del Scroll

3.6.1 Parallax



El desplazamiento de Parallax es una tendencia del sitio web donde el contenido de fondo (es decir, una imagen) se mueve a una velocidad diferente que el contenido de primer plano durante el desplazamiento.

Cómo crear un efecto de desplazamiento de paralaje

Use un elemento contenedor y agregue una imagen de fondo al contenedor con una altura específica. Luego use **background-attachment: fixed** para crear el efecto de paralaje real. Las otras propiedades de fondo se utilizan para centrar y escalar la imagen perfectamente:

```
.parallax {
 background-image: url("img_parallax.jpg");
 height: 500px;
 background-attachment: fixed;
```



```
 background-position: center;
```

```
background-repeat: no-repeat;
background-size: cover;
}
```

```
<div class="parallax"></div>
```

Algunos dispositivos móviles tienen un problema con background-attachment: fixed, la solución será desactivarlo para dispositivos móviles:

```
@media only screen and (max-device-width: 1366px){
        .parallax{ background-attachment:scroll; }
 }
```

3.6.2 Scroll Snap

Con la propiedad **scroll-snap-type** y **scroll-snap-align** podemos controlar el funcionamiento del desplazamiento del scroll.

El Scroll Snap nos va a permitir que, al hacer scroll, ya sea con el ratón desde un ordenador o portátil, o desde la pantalla de un smartphone, este desplazamiento se detenga en un punto en concreto de nuestra web, normalmente el comienzo de una nueva sección.

Esto es muy conveniente en sitios web que tengan una disposición en diferentes bloques horizontales, y queremos que nuestros usuarios salten de uno a otro de la forma más directa posible.

```
<div class="snap-container">
 <div class="snap-section">
 <h3>Section 01</h3>
 <p> Lorem ipsum...</p>
 </div>
 <div class="snap-section">
 <h3>Section 02</h3>
 <p> Lorem ipsum...</p>
 </div>
 <div class="snap-section">
 <h3>Section 03</h3>
 <p> Lorem ipsum...</p>
 </div>
```

```
</div>
```

CSS que le daremos al div padre:

```
.snap-container {
height: 100vh;
            overflow: auto;
 scroll-snap-type: y mandatory; }
```

Con el valor ***y*** estamos diciendo que el scroll que estamos marcando es el vertical y con el valor ***mandatory***, que la página se mueva hasta el principio de la sección siguiente o anterior, según hagamos scroll hacia abajo o hacia arriba. Podemos usar el **proximity**, que hará que nos desplazamos hasta la sección siguiente o anterior hasta que no estemos muy próximos al principio de dicha sección.

```
.snap-section {
 scroll-snap-align: start;
 min-height: 60vh; }
```

Con **scroll-snap-align** designaremos en qué punto el desplazamiento se parará. En nuestro caso, le hemos dado el valor **start**, por lo que se parará justo al inicio de la sección. Aunque también podemos usar los valores **center** para quedarnos a mitad de la sección, o **end** para quedarnos al final. La altura de la sección la hemos marcado en vh o altura del viewport.

También podemos hacerlo en horizontal. Esto nos puede venir si la exploración de nuestra web se hace en esta dirección, o si queremos hacer un slider. Para conseguirlo, solo hay que cambiar un valor en el contenedor padre: `scroll-snap-type: x proximity;`

Referencia: MDN Web Doc

Ejemplo: Apple

3.6.3 scroll-behavior

Si queremos hacer el mismo efecto scroll-snap pero con enlaces, podemos usar **scroll-behavior**.

Por defecto no se hereda y tendremos que aplicarlo al elemento contenedor que tendrá el scroll. En el caso del viewport del navegador, deberemos aplicarlo al elemento raíz, html.

```
html {
 scroll-behavior: smooth;
}
```

3.6.4 Control del scroll por JavaScript.

Podemos utilizar Javascript para controlar la posición del scroll. Cuando se produce el movimiento de la barra de desplazamiento, se produce un evento **onscroll** que podemos utilizar con el manejador de eventos.

Ejemplos:

- Pico4
- atelierdesign.be
- muskegonartmuseum.org
- firewatchgame.com

Por otro lado con **window.scrollY** podemos saber la posición exacta del desplazamiento del scroll.

```
window.onscroll = function () {
 let imagen = document.getElementById("imagen");
 imagen.style.width= parseInt(window.scrollY)/4 + "px" ;
};
```

Existe otro evento llamado **wheel** que detecta el movimiento de la rueda del ratón y permite capturar desplazamientos de este tipo. Es útil para hacer efectos específicos cuando el usuario usa la rueda del ratón para desplazarse.

Por otro lado, el evento contiene una propiedad **deltaY** que que indica la dirección y magnitud del desplazamiento:

- event.deltaY < 0 significa desplazamiento hacia arriba.
- event.deltaY > 0 significa desplazamiento hacia abajo.

```
window.onwheel = function (event) {
 const message = document.getElementById('message');
```

```
if (event.deltaY < 0) {
```



```
 message.textContent = "Desplazamiento hacia arriba "; ⬜⬜ }
else if (event.deltaY > 0) {
 message.textContent = "Desplazamiento hacia abajo "; ⬜⬜ }
 };
```

3.6.5 Parallax con el movimiento del ratón por JavaScript.

Para crear un efecto parallax controlado por el movimiento del ratón, puedes hacer que los elementos en la pantalla se muevan en direcciones opuestas al movimiento del ratón, simulando una sensación de profundidad. Esto se logra ajustando las posiciones de los elementos en función de la posición del ratón. Usaremos el evento **mousemove** para capturar la posición del ratón y ajustar la posición de cada capa en función de esa posición.

Utilizaremos las propiedades **clientX** y **clientY** para obtener la posición del cursor del ratón en el viewport (área visible de la ventana) cuando ocurre un evento, como mousemove o click.

```
document.addEventListener("mousemove", (event) => {
 // Obtiene las coordenadas del ratón en el viewport
 const posX = event.clientX;
 const posY = event.clientY;
 document.getElementById("coordinates").textContent = ${posX}, ${posY};
});
```

JavaScript contiene las propiedades **pageX** y **pageY** que representan las coordenadas del cursor del ratón en la página completa (documento) cuando ocurre un evento, como mousemove o click. A diferencia de clientX y clientY, que miden la posición relativa al viewport, pageX y pageY tienen en cuenta el desplazamiento de scroll.

```
document.addEventListener('mousemove', function(e) {
 if (arrastrando) {
 elemento.style.left = e.pageX + 'px';
 elemento.style.top = e.pageY + 'px';
 }
});
```

Ejemplo: Link

3.6.6 Scroll-driven animations

Este tipo de animaciones, donde se incluyen efectos como parallax, zoom, movimientos, aparición/desaparición, o en definitiva, animaciones específicas dirigidas por scroll, requerían Javascript y podían ser altamente costosas o ineficientes, dependiendo de cómo se realizarán.

CSS ha añadido una serie de propiedades relacionadas que permiten generar este tipo de animaciones de forma muy sencilla, utilizando CSS.

Nota: Pueden existir limitaciones en algunos navegadores como Firefox.

Las líneas de tiempo (timelines) se trata de un concepto para medir el tiempo de una animación, que parte desde un punto inicial, hasta un momento actual o final.

Tipos de líneas de tiempo:

- **Duración**: Son aquellas que permiten crear una animación a lo largo del tiempo o una duración específica.

- **Scroll**: Son aquellas que permiten realizar una animación cuando el usuario realiza desplazamiento en una página o región de la página.
- **Vistas**: Son aquellas que permiten realizar una animación cuando los elementos son visibles en la región visible del navegador (viewport).

3.6.6.1 La propiedad animation-timeline

La forma de crear animaciones asociadas a otras líneas de tiempo (o timelines) es utilizando esta propiedad. Nos permite asociar una animación a una línea de tiempo determinada, en lugar de asociarla al clásico @keyframes donde definimos la animación.

Los valores posibles:

- **scroll():** Usa una línea de tiempo de scroll anónima, basada en el desplazamiento de scroll.
- **view():** Usa una línea de tiempo de vista anónima, basada en la visibilidad del viewport.
- **--nombre:** Usa una línea de tiempo con el nombre indicado (variable).

La función scroll():

La función scroll() recibe dos parámetros. El primero es el elemento contenedor que usaremos de referencia para el desplazamiento, y el segundo es el eje en el que vamos a desplazarnos. La sintaxis es la siguiente:

scroll(contenedor eje) Ej :scroll(nearest block).

Opciones de scroll():

- **Contenedor**:
  - **nearest:** Utiliza el elemento contenedor padre más cercano. Valor por defecto.
  - **root**: Utiliza el documento entero como contenedor.
  - **self**: Utiliza el elemento actual como contenedor específico.
- **Eje:**
  - **block | x** Indica el eje horizontal como eje de progreso

○ **inline | y** Indica el eje vertical como eje de progreso

```
.bar {
 width: 0%;
 height: 100%;
 box-shadow: 0 0 5px darkred;
 background: red;
25
 animation: resize linear forwards;
 animation-timeline: scroll(root block);
 }
 @keyframes resize {
 to { width: 100% }
 }
```

La función view():

La función **view**() nos viene muy bien para situaciones en las que necesitamos que las animaciones de scroll se realicen justo cuando aparecen en la región visible del navegador (**viewport**). Sintaxis:

view(eje ajuste) Ej: view(block auto).

Los parámetros disponibles para la función view() son los siguientes:

- **Axis**:
  - **block | x:** Indica el eje horizontal como eje de progreso.
  - **inline | y:** Indica el eje vertical como eje de progreso.
- **Ajuste**:
  - **auto:** Utiliza el valor de la propiedad scroll-padding.
  - **tamaño:** Ajusta la posición donde se dispara la animación: inicio y final. Ej: view(block 150px)
  - **start end:** Ajusta la posición, pero especificando inicio y final. Ej:view(block 80% auto);

```
.box {
 width: 250px;
```

```
  height: 250px;
  margin: auto;
  border-radius: 5%;
  box-shadow: 5px 5px 10px #888;
  background: indigo;
  animation: change linear ;
  animation-timeline: view();
26
}
@keyframes change {
from {scale: 0; opacity: 0 }
to { scale: 1; opacity: 1; }
}
```

3.7 View transitions

Si pensamos en la animación o transición de una página a otra página, se trata de animar unos elementos del DOM de una página hasta unos elementos del DOM de otra página que aún no está cargada.

¿Cómo se hace?, cuando se navega entre páginas, el navegador hace una «foto» de la página antes de abandonarla. El navegador carga la nueva página de forma transparente al usuario (mantiene la foto anterior). Cuando la nueva página ha cargado, hace una nueva «foto» de la página de destino y aplica la animación o transición CSS de una «foto» a otra.

La regla @view-transition

Utilizaremos la regla @view-transition, dentro, usaremos la propiedad **navigation** con el valor auto:

Ejemplo: https://astro-zerojs-transitions.vercel.app/playlist/2a

```
@view-transition{
  navigation: auto;
  }
```

```
 .container {
 view-transition-name: page;
 }
::view-transition-old(page) {
 animation: fade 0.5s linear ;
 }
::view-transition-new(page) {
 animation: fade 0.5s linear ;
```

27

```
 }
 @keyframes fade { 0% { opacity: 1; transform: translateY(0); }
100% { opacity: 0; transform: translateY(50px);}  }
```

3.8 Web Animations

Es una tecnología que incorpora **Javascript**, mediante la cuál podemos crear animaciones CSS, ayudándonos de una API de Javascript. De esta forma, las animaciones siguen siendo eficientes, pero podemos aprovechar la potencia y flexibilidad de un lenguaje de programación como Javascript.

La idea básica de WebAnimations es que podemos animar elementos del DOM, muy fácilmente, simplemente utilizando el **método .animate()**, existente en cualquier elemento HTML.

```
const element = document.querySelector(".element");
 const keyframes = [
 { transform: "translate(0, 0)" },
 { transform: "translate(200px, 0)" },
 { transform: "translate(0, 0)" }
 ];
 const animation = element.animate(keyframes, 4000);
 console.log(animation);
```

.animate() recibe como primer parámetro un array de objetos keyframes en el cuál tenemos definido, de forma similar a como haríamos en CSS, los estados por los que pasa la animación. Como segundo parámetro, en este caso recibe la duración, en milisegundos.

Ventajas de Web Animations

- Las animaciones CSS son rápidas y eficientes, pero no se pueden controlar al 100%. ● Las Web Animations tienen mayor control, pero también mayor complejidad. ● Las Web Animations nos permiten animar fácilmente elementos del DOM. ● Las Web Animations son prácticamente igual de eficientes que las animaciones CSS y mucho más eficientes que las librerías de terceros.

Librerías de terceros: