CANVAS</>

Introducción al languaje CANVAS

Autor 1: Edward Mauricio Garcia Romero

*Pereira, Risaralda, colombia*

Correo-e: emgarcia36@utp.edu.co

***Resumen*<canvas> es un elemento HTML el cual puede ser usado para dibujar gráficos usando scripts (normalmente JavaScript). Este puede, por ejemplo, ser usado para dibujar gráficos, realizar composición de fotos o simples (y no tan simples) animaciones. Las imágenes a la derecha muestran algunos ejemplos de implementaciones <canvas> las cuales se verán en un futuro en este tutorial.**

**<canvas> fue introducido primero por Apple para el Mac OS X Dashboard y después implementado en Safari y Google Chrome. Navegadores basados en Gecko 1.8, tal como Firefox 1.5, que también soportan este elemento. El <canvas> es un elemento parte de las especificaciones de la WhatWG Web applications 1.0 mejor conocida como HTML5.**

**En este tutorial se describe cómo usar el elemento <canvas> para dibujar gráficos en 2D, empezando con lo básico. Los ejemplos le proveerán mayor claridad a las ideas que pueda tener referentes al canvas, así como los códigos que necesita para crear su propio contenido..**

***Palabras clave—*** [**break**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/break)**,**[**case**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/switch)**,**[**catch**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/try...catch)**,**[**class**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/class)**,**[**const**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/const)**,**[**continue**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/continue)**,**[**debugger**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/debugger)**,**[**default**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/default)**,**[**delete**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/delete)**,**[**do**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/do...while)**,**[**else**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/if...else)**,**[**export**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/export)**,**[**extends**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/class)**,**[**finally**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/try...catch)**,**[**for**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/for)**,**[**function**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/function)**,**[**if**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/if...else)**,**[**import**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/import)**,**[**in**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/in)**,**[**instanceof**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/instanceof)**,**[**new**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/new)**,**[**return**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/return)**,**[**super**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/super)**,**[**switch**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/switch)**,**[**this**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/this)**,**[**throw**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/throw)**,**[**try**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/try...catch)**,**[**typeof**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/typeof)**,**[**var**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/var)**,**[**void**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/void)**,**[**while**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/while)**,**[**with**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/with)**,**[**yield**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/yield)**.**

***Abstract*— JavaScript is the programming language of HTML and the web.**

**JavaScript is easy to learn.**

**This tutorial will teach you JavaScript from basic to advanced. JavaScript is one of the 3 languages that all web developers should learn:**

**1. HTML to define the content of web pages.**

**2. CSS to specify the design of web pages.**

**3. JavaScript to program the behavior of web pages.**

**Web pages are not the only place where JavaScript is used. Many desktop and server programs use JavaScript. Node.js is the best known. Some databases, such as MongoDB and CouchDB, also use JavaScript as their programming language.**

***Key Word* —**[**break**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/break)**,**[**case**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/switch)**,**[**catch**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/try...catch)**,**[**class**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/class)**,**[**const**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/const)**,**[**continue**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/continue)**,**[**debugger**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/debugger)**,**[**default**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/default)**,**[**delete**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/delete)**,**[**do**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/do...while)**,**[**else**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/if...else)**,**[**export**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/export)**,**[**extends**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/class)**,**[**finally**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/try...catch)**,**[**for**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/for)**,**[**function**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/function)**,**[**if**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/if...else)**,**[**import**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/import)**,**[**in**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/in)**,**[**instanceof**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/instanceof)**,**[**new**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/new)**,**[**return**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/return)**,**[**super**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/super)**,**[**switch**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/switch)**,**[**this**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/this)**,**[**throw**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/throw)**,**[**try**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/try...catch)**,**[**typeof**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/typeof)**,**[**var**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/var)**,**[**void**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/void)**,**[**while**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/while)**,**[**with**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/with)**,**[**yield**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/yield)**.**

1. INTRODUCCIÓN

Canvas (lienzo) es un elemento HTML que permite la creación de gráficos y animaciones de forma dinámica por medio de scripts. Sus aplicaciones son practicamente iniminaginables: crear juegos, interfaces, editores gráficos o efectos dinámicos, aplicaciones 3D…. Sólo la imaginación pone límites a Canvas. En este artículo, profundizamos en las posibilidades de Canvas y las ejemplificamos con un sencillo gráfico.

A modo anecdótico, sólo señalar Canvas fue creado por Apple, pero han liberado la propiedad intelectual para englobarlo dentro de los estándares de HTML. Para el usuario, todo son ventajas. No requiere ningún plugin adicional, sólo una un navegador que soporte HTML5 y hoy en día todos los navegadores importantes (Safari, Chrome, Firefox, Opera e Internet Explorer) soportan Canvas, desde hace ya unas cuantas versiones.

Es un elemento introducido en el estandard HTML 5 para realizar dibujos y animaciones en las páginas web de manera más sencilla. Con esto se relega el uso de Flash y se termina de extinguir el uso de applets Java

¿Qué es el canvas? pues es como un lienzo donde se puede dibujar o bien situar imágenes predefinidas. Por dibujar se entiende: trazar lineas, figuras geométricas, texto, etc.

Se usa habitualmente con Javascript y facillita enormemente el diseño de animaciones. De hecho unido a Javascipt y CSS3 sustituye perfectamente a las aplicaciones Flash y applets Java.

Se incopora a la página web de la misma forma que un div, solo que tiene un tamaño por defecto de 150 px de alto por 300px de ancho, obviamente se le puede dar cualquier otra dimensión usando los atributos width and height, o mejor aún usando las definiciones adecuadas de estilo CSS.

Sus argumentos específicos son width para especificar el ancho, height para el alto. Para que los scripts puedan manejar un elemento canvas este identficarse mediante un atributo id. Y se le puede asignar una clase (class) o definición de estilo CSS (style). Si contiene texto este será mostrado por los navegadores que no soporten este elemento de HTML 5.

Es un simple contenedor, pero posee métodos y propiedades que se utilzan mediante un lenguaje de script (lo habitual es Javascript) para realizar los dibujos. Estos pueden ser en 2D y en 3D, a través del API WEbGL

El api Canvas combinado con CSS, HTML y JavaScript nos permite hacer todo aquello que se nos pueda ocurrir. Vamos a citar algunas de las aplicaciones habituales:

a) Dibujar formas (líneas, puntos, círculos, etc.)

b) Dibujar texto

c) Dibujar encima de imágenes o manipular imágenes

d) Dibujar gráficos

e) Crear animaciones de todo tipo (matemáticas, escolares, publicidad, etc.).

f) Crear espacios tridimensionales por donde el usuario puede moverse

g) Crear juegos

h) Manipular videos por ejemplo crear efectos gráficos sobre ellos

i) Hacer todo lo anterior en respuesta a eventos (por interacción con el usuario)

j) Hacer todo lo anterior ligado a datos variables

dibujar circulos canvas javascript

Las bolas de la imagen anterior se pueden dibujar con relativa facilidad con Canvas. También se pueden dotar de animación, hacer que salten, se dividan, etc.

Si escribes canvas examples en un buscador como google o bing podrás acceder a numerosos ejemplos de uso de canvas. Muchos de ellos facilitan el código fuente de forma ordenada para que te sea fácil reproducirlo en tu ordenador o incorporarlo a tus páginas web.

VENTAJAS E INCONVENIENTES DE CANVAS

La principal ventaja de Canvas es permitir la creacion y manipulación de todo tipo de formas e imágenes sin dependencias de plugins o tecnologías externas a los propios lenguajes de desarrollo web y navegadores. Igualmente es una gran ventaja la facilidad que brinda para la interacción gráfica con el usuario o para ligar el aspecto gráfico a datos que cambian dinámicamente.

Canvas es una herramienta que, al igual que todo, bien usado resulta muy útil y mal usado puede generarnos problemas. Un inconveniente ligado a canvas es pretender hacer páginas webs muy animadas y con esto sobrecargar el computador del usuario. Ten en cuenta que crear animaciones consume recursos y puede ralentizar la navegación por las páginas web o crear molestias al usuario.

1. CONTENIDO

Canvas: ¿Qué es?

Inicio: rectas

Formas de lineas

Esilos de lineas

Dibujar rectángulos

Dibujar arcos y círculos

Dibujar texto

Alinear texto

Poner sombras

Gradientes de color

Patrones

Imágenes externas

Mapas de bits

Superposiciones

Estado del canvas

Giro del canvas

Traslación

Escalados

Trasformación libre

Curvas Bézier

Cierres y Pertenencia

Máscaras y borrados

Animaciones

Canvas: las coordenadas del papel

Imagina que le tienes que indicar a alguien que dibuje una línea recta en un papel. Debes decirle que ponga el lápiz en el punto origen, donde debe iniciarse la línea, y lo arrastre por el papel hasta el punto final de la recta por el camino más corto. Fácil.

Para definir estos puntos origen y final se utilizan coordenadas: distancia a un punto de referencia conocido. El Canvas tiene ese punto de referencia (0,0) en la esquina superior izquierda. Para definir donde está cualquier punto dentro del canvas se usan dos números: La coordenada X u horizontal es la distancia hasta el lado izquierdo y la coordenada Y o vertical que es la distancia hasta el lado superior del canvas. Con esto podemos definir cualquier punto de nuestro lienzo de dibujo.

Por ejemplo si las coordenada de un punto son (20,80) ¿cómo se interpretan éstos números? Pues imagina el canvas como un papel rectangular, pues ese punto está situado a 20 pixeles del lado derecho y a 80 pixeles del lado superior de ese rectángulo.

Si quiero que un programa dibuje una línea horizontal en el canvas le diría algo así como "recta desde 20,60 a 50,60". Ambos puntos están a la misma distancia del lado superior (60 pixeles) y la recta va desde un punto a 20 pixeles desde el lado izquierdo hasta 50 pixeles del lado izquierdo, o sea, tiene una longitud de 30 pixeles.

Formas de líneas: ancho y extremos

Una línea puede ser ancha, estrecha, de colores, con acabados redondeados, uniones... Existen varias propiedades aplicables a cualquier línea que sea trazada por un lápiz. En estos ejemplos irás viendo cuales son.

En el primer ejemplo tienes el grosor y los acabados. El grosor es la propiedad lineWidth, un valor numérico que indica el grosor de la línea en pixeles. Puedes consultarlo para usarlo en o escribirlo, como todas las propiedades.

El acabado de línea recta puede ser redondeada o plano. La diferencia se ve mejor usando líneas gruesas como en el ejemplo. Esta propiedad se denomina línea y los valores posibles son: butt, round, square.

Uniones de líneas

En el apartado anterior has visto cómo controlar el extremo de una línea, en este vas a ver cómo se pueden unir cuando se encuentran. La propiedad es lineJoin y permite tres valores: bevel, round, mitter. En el caso del mitter tienes una propiedad más la longitud de la unión, mitterLimit, si la unión es más ancha se convierte en biselada (bevel). No tiene demasiada utilidad.

Canvas: ¿Qué es?

Inicio: rectas

Formas de líneas

Estilos de líneas

Dibujar rectángulos

Dibujar arcos y círculos

Dibujar texto

Alinear texto

Poner sombras

Gradientes de color

Patrones

Imágenes externas

Mapas de bits

Superposiciones

Estado del canvas

Giro del canvas

Traslación

Escalados

Trasformación libre

Curvas Bézier

Cierres y Pertenencia

Máscaras y borrados

Animaciones

Colores de las líneas

Las líneas (trazos o pata) por defecto se dibujan en negro, pero el color también se puede cambiar mediante una propiedad strokeStyle, con ella podemos definir el color que se usará para dibujar.

Más que explicar cómo es cada una lo mejor es verlas en funcionamiento.

Líneas discontinuas

A veces se necesita destacar una línea como auxiliar por ejemplo o por dar efectos. Esto se puede hacer con líneas discontinuas. Se usa el método setLineDash(). Como argumento usa una lista (array) de valores que indican la longitud de los segmentos y el espaciado. Pueden ser dos o más valores. Si la lista está vacía se dibujan líneas continuas. Se puede especificar un offset o desplazamiento para el inicio del primer trazo: lineDashOffset.

Existe un método para determinar el valor usado es getLineDash()q ue lógicamente devuelve una lista de números (un array). .

Cómo dibujar rectángulos

En canvas podemos dibujar rectángulos vacíos, solo el contorno sin relleno, o con un relleno de color uniforme o degradado. En la unidad del canvas en detalle tienes el proceso genérico para dibujar en el canvas.

Realmente canvas tiene dos métodos para dibujar arcos. El que se muestra aquí se usa a partir de: el centro del arco, el radio y los ángulos de inicio y de final. Si este ángulo va de 0 a 360º tenemos una circunferencia. El ángulo se debe expresar en radianes (uf!!! matemáticas). Es fácil una circunferencia entera tiene 2\*pi (el famoso 3.1416) radianes. O sea 360 grados son 2pi radianes y 180 son pi radianes y 90º son pi/2 radianes. El 0 corresponde a la horizontal y va aumentando en el sentido horario. O sea, 0º son las 3 del reloj, 0.5 Pi son las 6, Pi son las 9, 1.5\*Pi son las 12 y 2·Pi vuelve las 3.

La otra forma de dibujar arcos la tienes en el apartado dedicado a las curvas Bézier. Lo explico a allí por su parecido con este método de dibujo de curvas.

Vamos a dibujar una circunferencia y esta vez vamos a usar una línea gruesa.

Poner texto en el canvas es también una operación de dibujar, solo que ahora no es una forma regular sino letras.

Texto: alineaciones

Ya habrás aprendido que al escribir texto en el canvas lo que haces es realmente un dibujo y que como tal admite efectos como sombras y también transformaciones tipo escalado, giros, traslaciones o usar las personalizadas. No es más que dibujar formas de letras.

Pero el texto aparte de escribirse con un tamaño o tipo de letra puede alinearse en vertical u horizontal. Esto puedes hacerlo en el canvas con dos sencillos propiedades:

textAlign

Permite alinear el texto en horizontal respecto al punto de referencia

textBaseLine.

Sería una alineación vertical respecto al punto de referencia.

Hemos usado un método para determinar el ancho de la frase de prueba: measureText(cadena de texto), esto devuelve un objeto avanzado del que por ahora solo nos interesa la propiedad width (soportada por todos los navegadores). La usamos para separar el texto de prueba que no se sobrescriba.

Texto: alineación vertical

Sería una alineación vertical respecto a la coordenada Y. Imagina una horizontal por la coordenada Y, el texto se escribirá en relación a esta línea según el valor que le demos: alphabetic (por defecto), top, botón, hanging, middle, ideográfica (casi igual al alphabetic). Más que una descripción mira esta imagen para entender el efecto de cada valor

Texto: alineación horizontal

Esta propiedad permite alinear el texto horizontalmente a izquierda, centro o derecha, tomando como referencia el punto donde comenzamos a escribir, es como si trazaos una vertical por la coordenada X y la usamos como referencia de alineamiento. Los valores posibles son Stuart (o left), end (o right), center.

Efecto de sombra

Cualquier forma que dibujemos en el canvas admite el efecto de sombra. Un texto, un rectángulo o un círculo pueden tener una sombra alrededor. Para ello el objeto de dibujo (al que estamos denominando lápiz en los ejemplos) posee una serie de propiedades como son:

shadowColor

El color que queremos darle a la sombra

shadowBlus

El grado de opacidad o desenfoque que muestra la sombra

shadowOffsetX, shadowOffsetY

Para la posición indicamos el desplazamiento en horizontal (shadowOffsetX) y en vertical (shadowOffsetY)

Gradientes de color

El gradiente de colores es un efecto de relleno que usa una transición entre varios colores en lugar de un color único.

El objeto gradiente se crea a partir del contexto de dibujo (lápiz en los ejemplos) con los métodos createLinearGradient(x0,y0,x1,y1) para los lineales y con el método createRAdialGradient(x0,y0,r0,x1,y1,r1) para los radiales o circulares.

Luego es necesario definir los colores que se usarán con el método addColorStop(pos, color). El primer parámetro, pos, tiene un valor ente 0.0 y 1.0 e indica la posición del color, que viene dado por el segundo parámetro. El color se puede expresar con nombre, código hexadecimal o las funciones RGB o RGBa, es decir, de la misma forma que se declara en CSS. De esta forma podemos crear gradientes multicolores

Gradiante radial

En el gradiente radial los colores van cambiando gradualmente desde un circulo central hacia el exterior. En este ejemplo se aplica a un círculo para que se vea más evidente y se usan tres colores para mostrar que puede ser un efecto multicolor.

El gradiente lineal se crea con createRadialGradient(x0,y0, r0, x1, y1, r1) que lleva seis parámetros para definir el centro y radio de los círculos donde empieza y termina el gradiente. Luego añadimos los colores como se explica en el apartado anterior. En este ejemplo los colores están descritos con tres métodos diferentes.

En el ejemplo el gradiente se centra en el mismo punto que el círculo, pero puede centrarse en cualquier otro punto dentro o fuera del círculo.

Gradiente lineal

En el gradiente lineal los colores cambian en un dirección (horizontal, vertical, inclinada). La fomra la eliges con los parámetros del punto de inicio y final. En este ejemplo ves como se usa para rellenar un rectángulo. Puedes hacer cambios para ver los efectos al cmabiar ls coordenadas de los puntos que definen el gradiente.

Patrones: el mosaico del canvas

En programas de dibujos hab´ras usado quizás los patrones: imágenes que se repiten para rellenar areas o para dibujar. Son como las piezas de un mosaico.

El contexto 2D del canvas puede crear y utilizar patrones se usa el método createPattern() que utiliza dos argumentos: uno especifica un objeto imagen y el otro es un valor que puede ser:

repeat: El elemento se repite tanto en horizontal como en vertical.

repeat-x, El elemento se repite solo en horizontal .

repeat-y, El elemento se repite solo en vertical.

no repeat: el elmento no se repite.

Imágenes externas

El canvas tamibén se puede usar como contenedor para imágenes, lo que te permite, entre otras cosas, dibujar sobre ella formas o gradientes como has visto en los otros capítulos dedicados al elemento canvas.

Lo primero que tienes que hacer es cargar la imagen. Eso lo puedes hacer con el elemento img de html5 o con javascript. En cualquier caso antes de dibujar la imagen en el canvas, hay que asegurarse que se ha cargado completamente, para eso se debe usar un evento onload, en los ejemplos tienes varias formas de hacerlo

Mapa de bits

En el apartado dedicado a imágenes externas viste como cargar un archivo de imagen y colocarlo en un canvas. En este caso las imágenes no se cargan sino que están definidas por un mapa de bits, o sea, una lista de números (0-255) que definen el color de cada pixel de la imagen (un pixel es un punto de una imagen).

Para logar esto en el canvas existen tres métodos:

createImageData(): crea un objeto imageData, una lista de pixels

getImageData(): copia los pixels de una región del canvas

putImageData(): dibuja una imagen a partir de sus pixels

Estos métodos usan tres propiedades

width, hegith: ancho y alto de la imagen

data: lista con los datos de los pixels de la imagen

Cada pixel viene definidos por cuatro valores: Rojo, Verde, Azul, Opcidad. Cada uno va de 0 a 255 así el primer pixel de una imagen va en rojo y opaco sería imagen.data[0] = 255, imagen.data[1] = 0, imagen.data[1] = 0, imagen.data[3] = 255, Como ves el pixel primero va de data[0] a data[3], el segundo pixel va de data[4] a data[7]. En general el pixel n del objeto imagen (imageData)va desde imagen.data[4\*n] a imagen.data[4\*n+3), ojo que n comienza en 0.

Superposiciones

Cuando se dibujan varias formas en un canvas de forma superpuesta estas se muestran en el orden en que se dibujan, de manera que cada forma se dibuja encima de la anterior. Por defecto se superponen sin más: la de arriba oculta a la anterior, pero este comportamiento puede cambiarse.

De entrada podemos hacer que las figuras sean más o menos opacas, con lo que una imagen no oculta del todo a la de abajo. Para ello usaremos la propiedad globalAlpha, que toma valores entre 0 (totalmente trasnparente) y 1 (totalmente opaco)

Estado del canvas

Hasta ahora solo has dibujado las formas usando el sistema de coordenadas por defecto y solo has modificado el el tamaño del canvas según se necesitara. Pero tambien dispones de las transformaciones: formas más potentes para modificar como se realizan los dibujos como: el origen de coordenadas, girar, cambiar tamaño o realziar transformaciones personalizadas.

Una vez aplicas una transformación (o asignas un valor a una propiedad del lápiz) se seguirá dibujando con esa transformación o cambio de propiedades hasta que la deshagas. Para facilitar esta tarea dispnes de un par de métodos fundamentales:

save()

Guarda el estado actual de canvas en una pila LIFO, puedes verla como una lista en la que se guarda el estado del canvas cada vez que usas este método. Cada vez que llamas a este método se crea un nuevo elemento al final de esta lista.

restore()

Recupera el último estado del canvas almacenada y lo elimina de la lista. Importante: recupera el último estado guardado. Es decir lee la lista de estados guardados desde el final y cuando lee un elemento de esta lista lo elimina de la misma. Es como una pila de platos: el último plato lo dejas encima de la pila y cuando necesitas cojes ese último plato. De ahí el nombre pila LIFO: Las In First Out

Al guardar un estado del canvas almacenas las propiedades siguientes:

strokeStyle, fillStyle, globalAlpha, lineWidth, lineCap, lineJoin, miterLimit, lineDashOffset, shadowOffsetX, shadowOffsetY, shadowBlur, shadowColor, globalCompositeOperation, font, textAlign, textBaseline, direction, imageSmoothingEnabled.

Además de las transformaciones que se hayan aplicado y el cuadro de recorte (máscara) que se haya aplicado si se ha aplicado alguno

Como ves se almacena el estado completo del canvas para poderlo reproducir sin más que hacer un restore(). Esto facilita bastante la tarea pues permite deshacer operaciones sin tener que deshacer comandos. Por ejemplo si para un dibujo puedes cambiar color, relleno, grosor de linea, orientación y luego recuperar los valores previos con una sola instrucción.

Rotate: Girar el canvas

Este método gira el canvas un ángulo dado por el argumento del método en radianes. El centro de giro es el origen de coordenadas, la esqina superior izquierda si no se ha aplicado ninguna vez translate()

Si aplicas un rotate(0.75) a partir de ahora todos los dibujos que se hagan aparecerán rotados a la derecha un ángulo de 0.75 radianes. Si usas un argumento negativo el giro se hace hacia la izquierda, contrario a las agujs del reloj (cuando los relojes eran de agujas)

Recuerda que el argumento es el ángulo en radianes, esa unidad que nunca terminaste de entender en el colegio. REcuerda que para pasar de grados a radianes la fórmula es muy simple:

radianes = grados\*Math.PI/180

Así que π radianes es media vuelta y 2·π es una vuelta completa. Fácil. Siquieres cambiar el centro de giro debes usar translate()

Translate: Cambiar de origen de coordenadas

Este método modifica el origen de coordenadas del canvas. Recuerda que las coordenadas de cualquier punto del canvas eran distancias en horizontal y vertical a la esquina superior izquierda, ese es el punto (0, 0). Al aplicar este método ese punto cambia de posición.

Si tienes el punto (20,15) y aplicas un translate(10, 5) ahora ese punto se habrá movido qo a la izquierda y 5 hacia abajo, ahora será el (30, 20). A paritr de ahora todas las coordenadas se medirán desde el punto (10, 5)

Este método se llama con dos valores como argumentos; primero es el valor que se suma a la coordenada horizontal y el segundo el que se suma a la vertical. Digo suma, pero si el argumento es negativo, realmente se resta.

translate(sumarX, sumarY)

En este ejemplo se dibuja un rectángulo en el punto (20, 15) luego hacemos una traslación de 25 en horizontal y vertical y volvemos a dibujar el mismo rectángulo con las mismas coordenadsa de origen. Verás que aparece desplazado respecto al primero. Aparece en el punto X = 35 (10+25 ) , Y = 40 (15+25 ).

Scale: Cambiar escala

Este método modifica la escala de dibujo usada en el canvas. Así cualquier dibujo que hagas verá modificadas su anchura y altura así como las coordenadas del punto donde se dibuje.

Si imaginas que el canvas es un papel cuadrículado, el método scale() aumenta o disminuye el tamaño de las cuadrículas. Este cambio afecta por separado al ancho y al alto, con lo cual se puede deformar el dibujo y, por ejemplo, convertir una circunferencia en una elipse.

Este método posee dos argumentos: el primero multiplica al ancho y el segundo al alto.

scale(factorAncho, factorAlto)

En este ejemplo creamos un circulo rojo de radio 80 y en la posición X=300, Y=150. Al aplicar el método scale(1,0.5) multipliamo el ancho por 1 (no cambia) y el alto por 0.5 (altura a la mitad) y a continuación se dibuja. Verás que el centro estará en X=300 Y=75 (no en 150), pues las coordenadas verticales se han multiplicaod po 0.5, la escala también afecta a las coordenadas.

Transform: Transformaciones personalizadas

Ya has visto algunos métodos de transformación. tales como la el escalado, traslación o la rotación. Pero aún existe otro método para transformar el contexto de dibujo de forma totalmente personalizada. Es un sistema que incluye a los tres anteriores y además permite distorsionar la imagen dibujada, lo que le denomida sesgado, que no es más que un desplazamiento de algunos puntos clave del dibujo inclinando las figuras en vertical u horizontal

Son dos los métodos usados: transfrom() y setTransfrom(). La única diferencia es que el primero es acumulativo como el escalado, traslación y rotación, mientras que el segundo es absoluto.

Ambos usan seis argumentos: ex, sy, sx, ey, tx, ty.

ex, ey: son factores de escala, ex para la escala horizontal y ey opra la vertical. funcionan igual que en scale()

sx, sy: deforman la malla del canvas en horizontal (sx) y en vertical, sy. Aplicados a un rectángulo sx inclinaría los lados verticales y sy inclinaría los lados superior e inferior. Funcionan como un desplazamiento

tx, ty: trasladan el origen en horizontal y vertical, de forma similar al método translate(tx, ty).

El método setTransform() hace dos cosas: primero resetea los seis valores de la transformación a sus valores por defecto y luego aplica transform() con los valores que se le pasan como argumento.

El resetero de estos valores también se puede hacer mediante el método resetTransform(), con ello los seis valores recuperan sus valores por defectro: ex: 1, sy: 0, sx: 0, ey: 1 , tx: 0, ty: 0

**Dibujo libre: Curvas Bézier**

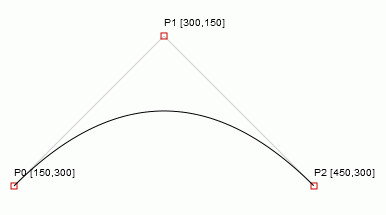
Además de las formas básicas (rectas, rectánculos y arcos) html 5 permite dibujar en el canvas formas libres o personalizadas mediante las denominadas **curvas Bézier**.

Resumiendo mucho muchísimo una cruva Bézier podríamos definirla como una curva donde cada punto es calculado a partir de unos puntos de control. Existen diferentes formas de realizar estos trazados, pero para el canvas tenemos dos métodos: **quadraticCurveTo()**(curva Bézier cuadrática) y**bezierCurveTo()** (curva Bzier cúbica)

El uso de este método puede parecer algo complejo, pero con él es posible lograr formas complejas. Ojo: debes situar el lápiz en el punto donde comienza a dibujarse la curva, para ello puedes usar el método **moveTo()**

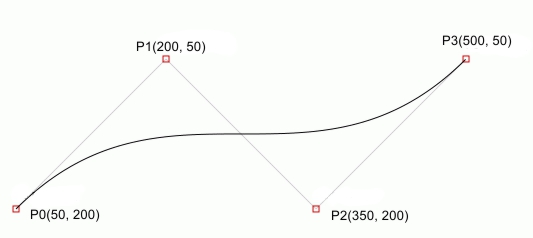
**Bézier quadrática**

Usa dos puntos para definir la curva: un **punto de control**y el **punto donde finaliza** la curva. Si el punto donde comienza la curva coincide con el punto de control obtienes una**linea recta**. En la imagen **P0 es el origen**, **P1 es el punto de control**, y **P2 es el punto final** de la curva. El método para dibujarla se llamaría como **lapiz.quadraticCurveTo(300,150, 450, 300)**. Como ves los argumentos son el **Punto de control** y el **Punto final,**donde termina la curva. Para poner el lugar donde se inicia la curva usamos el método **moveTo(150, 300)**



**Bézier cúbica**

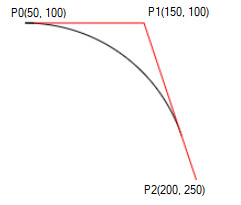
En este caso la curva se dibuja usando tres argumentos: **dos puntos de control**y el**punto donde finaliza**la curva. Para ver como funcionan los puntos de control mira esta imagen



Al igual que en la cuadrática, el punto origen no se pasa como argumento al método, sino que el lápiza se pone en él usando **movetTo()**. Esta curva se conseguiría dibujando a partir de 50, 200 y usando el método de esta forma

**lapiz.bezierCurveTo(200, 50, 350, 200, 500, 50)**

**Arco entre lineas**



Es una forma de dibujar arcos que susa punto de controy y un parámetro exrtra, el radio de la curva. Con este método se dibuja **un arco de radio dado tangente a dos rectas**, o sea, que toca a cada una en un punto. Las rectas van desde el punto donde comienza el dibujo hasta el primer punto de control, y la segunda va desde éste hasta el segundo punto de control. Los argumentos son coordenadas: X1, Y1 del primer punto, X2, Y2 para el segundo punto y R para el radio del arco:

**arcTo( X1, Y1, X2, Y2, R)**

El **punto 0** se usa con **moveTo(X0, Y0)** para colocar el lápiz antes de empezar el dibujo

Cerrar trazados en el canvas

A partir de dos líneas podemos consturir una figura cerrada con tan solo unir sus extremos con una recta, esto se puede hacer con el método closePath(): traza una linea recta desde el punto donde comienza el dibujo hasta el punto donde termina, formando así un trazado o camino cerrado. En este ejemplo lo usamos para cerrar un arco de 180º. Se dibuja sin cerrar y cerrándolo, para que veas la diferencia.

Pertenencia a tramos y areas

Existen dos métodos para determinar donde se encuentra un punto en relación a una figura o trazado que se haya creado inmediatamente antes en el canvas. No es necesario dibujarla, solo crearla.

isPointInStroke(x, y)

Permite saber si el punto dado por las coordenadas X, Y forma parte de un trazado

isPointInPath(x, y)

En este caso el método me dice si el punto está en el área encerrado por la linea, que se entiende está formando una figura cerrada (rectángulos, circunferencia, polígono)

Máscaras de dibujo

En el canvas se puede recortar regiones de cualquier forma y tamaño, de manera que los dibujos posteriores no se salgan de esa zona. De esta manera podemos por ejemplo borrar una parte de una imagen o dibujar sin modificar otras zonas del canvas. Hemos creado una máscara de dibujo.

La forma de usar este método sigue el siguiente patrón: creamos una figura cualquiera (en el ejemplo un triángulo), luego aplicamos el método clip(), que no usa argumentos, y a partir de ese momento cualquier acción sobre el canvas queda reducida a ese área, al triángulo en este ejemplo.

En el ejemplo: creo el triángulo (altura 100 y base 100), luego uso el método clip() para enmascarar el canvas, luego dibujo un rectángulo (de alto 50 y ancho 200), pero solo se verá la parte del rectángulo dentro del area recortada, dentro de la máscara.

Borrado en el canvas

El canvas puede borrarse total o parcialmente usando el método clearRect(). Este método borra un área rectangular del canvas, como argumentos usa las coordenadas de la esquina superior izquierda del retángulo, el ancho y el alto. Igual que si se dibuja un rectángulo.

Si se quiere borrar un área con forma no rectangular puedes usar el enmascaramiento con el método clip(). Para poder seguir dibujando en todo el canvas debes eliminar la máscara creada con clip() para lo cual usa los métodos save() y restore(). Con el primero guardas el contexto del dibujo antes de crear la máscara y con el segundo lo recuperas después de haber borrado.

Animaciones

El <canvas> introducido por las especificaciones HTML 5 es la base para los juegos web online que sustituyen a los juegos Flash, en vias de esxtinción. También sustituyen perfectamente a los desaperecidos applets, aplicaciones java para web muy utilizados en demos y aplicaciones educativas como simulaciones en física o matemática.

La animación exige un conocimiento básico de Javascript, pues este es el lenguaje que habitualmente se utiliza en las páginas web interactivas. Básicamente una animación es tan solo una sucesión de imágenes que se muestran una tras otra cada cierto intervalo de tiempo. Así de fácil. Trasladado a lo que sabes del canvas sería como redibujar un canvas cada cierto intervalo de tiempo.

En Javascript se venía usando la función setInterval() que permite ejecutar un código cada cierto tiempo. Este código se encarga de redibujar el canvas para ir mostrando los diferentes fotogramas de la animación.

Pero existe otro método más eficiente y con resultados de más calidad, pues deja al explorador gestionar los tiempos de procesador dedicados a ejecutar la animación, entre otras cosas. Se trata del método requestAnimationFrame(), este método del objeto window, que lanza al explorador una petición para redibujar la ventana y el explorador lo hará cuando sea posible (este refresco de pantalla suele ser de 60 veces por minuto). Solo lleva un argumento: el nombre de la función que deba ejecutarse para actualizar la animación.

Cuando se ejecuta window.requestAnimationFrame(funcionLlamada) se devuelve un identificador para identificar la llamada, o sea, la animación. Cada vez que el explorador refresca la pantalla (unas 60 veces por segundo) se ejecuta la funcionLlamada con un argumento: el instante en que se ejecuta (una medida de tiempo en milisegundos). Dentro de esa función se modifica el canvas (se cambia de fotograma) y se muestra, volviendo a ejecutar la llamada a requestAnimationFrame() si quieres seguir la animación o a window.cancelAnimationFrame(idAnim) si deseas finalizar la animación.

1. CONCLUSIONES

Canvas (lienzo) es un elemento HTML que permite la creación de gráficos y animaciones de forma dinámica por medio de scripts. Sus aplicaciones son practicamente iniminaginables: crear juegos, interfaces, editores gráficos o efectos dinámicos, aplicaciones 3D…. Sólo la imaginación pone límites a Canvas. En este artículo, profundizamos en las posibilidades de Canvas y las ejemplificamos con un sencillo gráfico.

A modo anecdótico, sólo señalar Canvas fue creado por Apple, pero han liberado la propiedad intelectual para englobarlo dentro de los estándares de HTML. Para el usuario, todo son ventajas. No requiere ningún plugin adicional, sólo una un navegador que soporte HTML5 y hoy en día todos los navegadores importantes (Safari, Chrome, Firefox, Opera e Internet Explorer) soportan Canvas, desde hace ya unas cuantas versiones.

<canvas> es un elemento HTML el cual puede ser usado para dibujar gráficos usando scripts (normalmente JavaScript). Este puede, por ejemplo, ser usado para dibujar gráficos, realizar composición de fotos o simples (y no tan simples) animaciones. Las imágenes a la derecha muestran algunos ejemplos de implementaciones <canvas> las cuales se verán en un futuro en este tutorial.

<canvas> fue introducido primero por Apple para el Mac OS X Dashboard y después implementado en Safari y Google Chrome. Navegadores basados en Gecko 1.8, tal como Firefox 1.5, que también soportan este elemento. El <canvas> es un elemento parte de las especificaciones de la WhatWG Web applications 1.0 mejor conocida como HTML5.

En este tutorial se describe cómo usar el elemento <canvas> para dibujar gráficos en 2D, empezando con lo básico. Los ejemplos le proveerán mayor claridad a las ideas que pueda tener referentes al canvas, así como los códigos que necesita para crear su propio contenido.

Antes de EmpezarSección

Usar el elemento <canvas> no es algo muy díficil pero necesita saber y entender los aspectos básicos del HTML y JavaScript. El elemento <canvas> no está soportado en navegadores viejos, pero están soportado en la mayoría de las versiones más recientes de los navegadores. El tamaño por defecto del canvas es 300px \* 150px [ancho (width) \* alto (height)]. Pero se puede personalizar el tamaño usando las propiedades height y width de CSS. Con el fin de dibujar gráficos en el lienzo <canvas> se utiliza un objeto de contexto de JavaScript que crea gráficos sobre la marcha.

RECOMENDACIONES

Esta sección sigue el formato regular del resto del documento. La única observación es notar que el título no está numerado.

En esta sección se agregan agradecimientos a personas que colaboraron en el proyecto pero que no figuran como autores del paper.

REFERENCIAS

https://www.aprenderaprogramar.com/index.php?option=com\_content&view=article&id=885:api-canvas-html-y-javascript-ejemplos-dibujar-circulos-formas-graficos-animaciones-juegos-cu01195e&catid=78&Itemid=206

Por mayor información diríjase al siguiente enlace donde encontrará una lista actualizada de todas las librerías disponibles: www.github.com/Modernizr/Modernizr/wiki/

HTML5-Cross-browser-Polyfills

Referencias de libros:

 http://librosweb.es/javascript/

 http://www.w3schools.com/js/default.asp

 Una buena colección de ejemplos:

http://www.w3schools.com/js/js\_ex\_dom.asp

 Artículos varios: <http://javascript.about.com/>

Documentos presentados en conferencias (No publicadas aún):

1. D. Ebehard and E. Voges, "Digital single sideband detection for interferometric sensors," presented at the 2nd Int. Conf. Optical Fiber Sensors, Stuttgart, Germany, 1984.
2. Process Corp., Framingham, MA. Intranets: Internet technologies deployed behind the firewall for corporate productivity. Presented at INET96 Annu. Meeting. [Online]. Available: http://home.process.com/ Intranets/wp2.htp