



EBook Gratis

APRENDIZAJE

html5-canvas

Free unaffiliated eBook created from
Stack Overflow contributors.

#html5-
canvas

Tabla de contenido

Acerca de.....	1
Capítulo 1: Empezando con html5-canvas.....	2
Examples.....	2
Cómo agregar el elemento de lienzo de HTML5 a una página web.....	2
Tamaño y resolución del lienzo.....	2
Lienzo fuera de pantalla.....	3
Detectando la posición del ratón en el lienzo.....	4
Hola Mundo.....	4
Un índice de capacidades y usos de lienzo de HTML5.....	5
Capacidades del lienzo.....	5
Usos del lienzo.....	5
Girar.....	6
Guardar lienzo en archivo de imagen.....	7
Capítulo 2: Animación.....	9
Examples.....	9
Animación simple con contexto 2D y requestAnimationFrame.....	9
Animar en un intervalo específico (agregar un nuevo rectángulo cada 1 segundo).....	9
Animar a una hora determinada (un reloj animado).....	10
Use requestAnimationFrame () NOT setInterval () para los bucles de animación.....	12
Animar una imagen a través del lienzo.....	13
No dibujes animaciones en tus controladores de eventos (una aplicación de boceto simple).....	14
Facilitando el uso de las ecuaciones de Robert Penners.....	16
Establecer la velocidad de fotogramas utilizando requestAnimationFrame.....	19
Animar de [x0, y0] a [x1, y1].....	20
Capítulo 3: Arrastrando formas de ruta e imágenes sobre lienzo.....	22
Examples.....	22
Cómo las formas e imágenes REALMENTE (!) Se "mueven" en el lienzo.....	22
Arrastrando círculos y rectángulos alrededor del lienzo.....	23
¿Qué es una "forma"?	23
Usando eventos del mouse para hacer arrastrar.....	24

Demostración: arrastrando círculos y rectángulos en el lienzo	24
Arrastrando formas irregulares alrededor del lienzo.....	27
Arrastrando imágenes alrededor del lienzo.....	31
Capítulo 4: Borrar la pantalla	34
Sintaxis.....	34
Observaciones.....	34
Examples.....	34
Rectángulos.....	34
Datos de imagen en bruto.....	34
Formas complejas.....	35
Lienzo claro con degradado.....	35
Lienzo transparente utilizando operación compuesta.....	35
Capítulo 5: Caminos	36
Examples.....	36
Elipse.....	36
Línea sin borrosidad.....	37
Capítulo 6: Colisiones e Intersecciones	39
Examples.....	39
¿Chocan 2 círculos?.....	39
¿Chocan 2 rectángulos?.....	39
¿Están colisionando un círculo y un rectángulo?.....	39
¿Están interceptando los segmentos de 2 líneas?.....	40
¿Un segmento de línea y un círculo colisionan?.....	41
¿Están el segmento de línea y el rectángulo colisionando?.....	41
¿Chocan 2 polígonos convexos?.....	42
¿Chocan 2 polígonos? (Se permiten polis tanto cóncavas como convexas).....	44
¿Un punto X, Y está dentro de un arco?.....	45
¿Un punto X, Y está dentro de una cuña?.....	45
¿Un punto X, Y está dentro de un círculo?.....	46
¿Un punto X, Y está dentro de un rectángulo?.....	46
Capítulo 7: Compositing	48
Examples.....	48

Dibuja detrás de las formas existentes con "destino sobre"	48
Borre las formas existentes con "destination-out"	48
Composición predeterminada: las nuevas formas se dibujan sobre las formas existentes	49
Clip imágenes dentro de formas con "destino-en"	49
Clip imágenes dentro de formas con "fuente-en"	50
Sombras internas con "fuente-encima"	50
Invertir o negar imagen con "diferencia"	51
Blanco y negro con "color"	51
Incrementa el contraste de color con "saturación"	52
Sepia FX con "luminosidad"	53
Cambia la opacidad con "globalAlpha"	53
Capítulo 8: Diseño de respuesta	55
Examples	55
Creación de un lienzo de página completa sensible	55
Coordenadas del ratón después de cambiar el tamaño (o desplazamiento)	55
Animaciones de lienzo sensibles sin eventos de cambio de tamaño	56
Evento de cambio de tamaño anunciado	57
Simple y el mejor tamaño	57
Capítulo 9: Gráficos y diagramas	59
Examples	59
Línea con puntas de flecha	59
Curva cúbica y cuadrada de Bezier con puntas de flecha	59
Cuña	61
Arco con relleno y trazo	62
Gráfico circular con demo	62
Capítulo 10: Imágenes	65
Examples	65
Recorte de imágenes utilizando lienzo	65
El lienzo retenido	65
¿"Context.drawImage" no muestra la imagen en el lienzo?	66
Escala de imagen para ajustar o rellenar	66
Escala de ejemplo para ajustar	67

Escala de ejemplo para llenar.....	68
Capítulo 11: Los tipos de medios y el lienzo.....	69
Observaciones.....	69
Examples.....	70
Cargando y mostrando una imagen.....	70
Dibujando una imagen svg.....	71
Carga básica y reproducción de un video en el lienzo.....	72
Solo una imagen.....	72
Consigue lienzo y configuración básica.....	73
Creando y cargando el video.....	73
El evento puede jugar (equivalente a la carga de imágenes).....	73
Mostrando.....	74
Control de pausa de juego básico.....	74
Ahora el evento de pausa de juego.....	75
Resumen.....	75
Capturar lienzo y guardar como video webM.....	75
Ejemplo de captura y reproducción de lienzo.....	75
Capítulo 12: Manipulación de píxeles con "getImageData" y "putImageData".....	82
Examples.....	82
Introducción a "context.getImageData".....	82
Una ilustración que muestra cómo se estructura la matriz de datos de píxeles.....	83
Capítulo 13: Navegando por un sendero.....	85
Examples.....	85
Encontrando puntos a lo largo de una curva Bezier cúbica.....	85
Encontrar puntos a lo largo de una curva cuadrática.....	86
Encontrar puntos a lo largo de una línea.....	86
Encontrar puntos a lo largo de todo un Sendero que contiene curvas y líneas.....	87
Longitud de una curva cuadrática.....	94
Dividir curvas de bezier en la posición.....	94
Ejemplo de uso.....	94
La funcion dividida.....	95
Recorte la curva de bezier.....	97

Ejemplo de uso.....	97
Ejemplo de función.....	98
Longitud de una curva de Bezier cúbica (una aproximación cercana).....	100
Encontrar el punto en la curva.....	101
Ejemplo de uso.....	101
La función.....	101
Encontrar la extensión de la curva cuadrática.....	102
Capítulo 14: Oscuridad.....	104
Examples.....	104
Efecto adhesivo utilizando sombras.....	104
¿Cómo parar más sombras?.....	105
El sombreado es computacionalmente costoso: ¡caché esa sombra!.....	105
Añade profundidad visual con sombras.....	106
Sombras interiores.....	107
Trazos con una sombra interior.....	107
Trazos rellenos con una sombra interior.....	108
Rellenos sin trazos con una sombra interior.....	109
Capítulo 15: Poligonos.....	111
Examples.....	111
Estrellas.....	111
Polígono regular.....	112
Renderiza un polígono redondeado.....	112
Capítulo 16: Ruta (solo sintaxis).....	115
Sintaxis.....	115
Examples.....	115
Resumen de los comandos básicos de trazado de trayectos: líneas y curvas.....	115
Descripción de los comandos básicos de dibujo:.....	116
lineTo (un comando de ruta).....	118
arco (un comando de ruta).....	120
quadraticCurveTo (un comando de ruta).....	121
bezierCurveTo (un comando de ruta).....	122

arcTo (un comando de ruta).....	123
rect (un comando de ruta).....	124
closePath (un comando de ruta).....	126
beginPath (un comando de ruta).....	127
lineCap (un atributo de estilo de ruta).....	130
lineJoin (un atributo de estilo de ruta).....	131
strokeStyle (un atributo de estilo de ruta).....	132
fillStyle (un atributo de estilo de ruta).....	134
lineWidth (un atributo de estilo de ruta).....	136
shadowColor, shadowBlur, shadowOffsetX, shadowOffsetY (atributos de estilo de ruta).....	137
createLinearGradient (crea un objeto de estilo de ruta).....	140
createRadialGradient (crea un objeto de estilo de ruta).....	143
Los detalles oficiales de miedo.....	146
createPattern (crea un objeto de estilo de ruta).....	147
trazo (un comando de ruta).....	149
La inusual forma en que se dibujan los trazos.....	150
Rellenar (un comando de ruta).....	153
clip (un comando de ruta).....	154
Capítulo 17: Texto.....	156
Examples.....	156
Dibujo de texto.....	156
Formato de texto.....	157
Envolver el texto en párrafos.....	158
Dibuja párrafos de texto en formas irregulares.....	158
Rellena texto con una imagen.....	161
Representación de texto a lo largo de un arco.....	161
Representación de ejemplo.....	162
Código de ejemplo.....	162
Descripciones de funciones.....	165
CanvasRenderingContext2D.fillCircleText (texto, x, y, radio, inicio, [final, [adelante]]);.....	165
CanvasRenderingContext2D.strokeCircleText (texto, x, y, radio, inicio, [final, [adelante]].....	165

CanvasRenderingContext2D.measureCircleText (texto, radio);	165
Ejemplos de uso.....	166
Texto en curva, beziers cúbicos y cuadráticos.....	167
Ejemplo de uso:.....	167
Texto justificado.....	170
Ejemplo de renderizado.....	170
El ejemplo.....	171
Cómo utilizar.....	173
Argumentos de función.....	173
Ejemplos de uso.....	174
Párrafos justificados.....	175
Ejemplo de render.....	175
Código de ejemplo.....	176
Cómo utilizar.....	179
Devolver objeto.....	179
Ejemplo de uso.....	180
Capítulo 18: Transformaciones	182
Examples.....	182
Dibujar rápidamente muchas imágenes traducidas, escaladas y rotadas.....	182
Girar una imagen o camino alrededor de su punto central.....	183
Introducción a las transformaciones.....	184
Una matriz de transformación para realizar un seguimiento de las formas traducidas, rotada.....	186
¿Por qué usar una matriz de transformación?	186
Una matriz de transformación "clase"	187
Creditos	193

Acerca de

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: [html5-canvas](#)

It is an unofficial and free html5-canvas ebook created for educational purposes. All the content is extracted from [Stack Overflow Documentation](#), which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official html5-canvas.

The content is released under Creative Commons BY-SA, and the list of contributors to each chapter are provided in the credits section at the end of this book. Images may be copyright of their respective owners unless otherwise specified. All trademarks and registered trademarks are the property of their respective company owners.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Capítulo 1: Empezando con html5-canvas

Examples

Cómo agregar el elemento de lienzo de HTML5 a una página web

Html5-Canvas ...

- Es un elemento html5.
- Es compatible con la mayoría de los navegadores modernos (Internet Explorer 9+).
- Es un elemento visible que es transparente por defecto.
- Tiene un ancho predeterminado de 300 px y una altura predeterminada de 150 px.
- Requiere JavaScript porque todo el contenido debe agregarse programáticamente al Lienzo.

Ejemplo: cree un elemento Html5-Canvas utilizando tanto el marcado Html5 como JavaScript:

```
<!doctype html>
<html>
<head>
<style>
  body{ background-color:white; }
  #canvasHtml5{border:1px solid red; }
  #canvasJavascript{border:1px solid blue; }
</style>
<script>
window.onload=(function(){

  // add a canvas element using javascript
  var canvas=document.createElement('canvas');
  canvas.id='canvasJavascript'
  document.body.appendChild(canvas);

}); // end $(function){}
</script>
</head>
<body>

  <!-- add a canvas element using html -->
  <canvas id='canvasHtml5'></canvas>

</body>
</html>
```

Tamaño y resolución del lienzo.

El tamaño de un lienzo es el área que ocupa en la página y está definido por las propiedades de ancho y alto de CSS.

```
canvas {
  width : 1000px;
  height : 1000px;
}
```

La resolución del lienzo define el número de píxeles que contiene. La resolución se establece configurando las propiedades de ancho y alto del elemento del lienzo. Si no se especifica el lienzo por defecto a 300 por 150 píxeles.

El siguiente lienzo utilizará el tamaño de CSS anterior, pero como no se especifica el `width` y el `height` la resolución será de 300 por 150.

```
<canvas id="my-canvas"></canvas>
```

Esto resultará en que cada píxel se estire de manera desigual. El aspecto del píxel es 1: 2. Cuando el lienzo se estire, el navegador utilizará filtrado bilineal. Esto tiene un efecto de desenfoque de píxeles que se estiran.

Para obtener los mejores resultados al utilizar el lienzo, asegúrese de que la resolución del lienzo coincida con el tamaño de visualización.

Siguiendo el estilo CSS anterior para que coincida con el tamaño de visualización, agregue el lienzo con el `width` y la `height` establecidos en el mismo número de píxeles que define el estilo.

```
<canvas id = "my-canvas" width = "1000" height = "1000"></canvas>
```

Lienzo fuera de pantalla

Muchas veces, cuando trabaje con el lienzo, necesitará tener un lienzo para contener algunos datos de píxeles internos. Es fácil crear un lienzo fuera de la pantalla, obtener un contexto 2D. Un lienzo fuera de la pantalla también utilizará el hardware de gráficos disponible para renderizar.

El siguiente código simplemente crea un lienzo y lo llena con píxeles azules.

```
function createCanvas(width, height){
    var canvas = document.createElement("canvas"); // create a canvas element
    canvas.width = width;
    canvas.height = height;
    return canvas;
}

var myCanvas = createCanvas(256,256); // create a small canvas 256 by 256 pixels
var ctx = myCanvas.getContext("2d");
ctx.fillStyle = "blue";
ctx.fillRect(0,0,256,256);
```

Muchas veces, el lienzo fuera de la pantalla se utilizará para muchas tareas, y es posible que tenga muchos lienzos. Para simplificar el uso del lienzo, puede adjuntar el contexto del lienzo al lienzo.

```
function createCanvasCTX(width, height){
    var canvas = document.createElement("canvas"); // create a canvas element
    canvas.width = width;
    canvas.height = height;
    canvas.ctx = canvas.getContext("2d");
    return canvas;
}
```

```
}  
var myCanvas = createCanvasCTX(256,256); // create a small canvas 256 by 256 pixels  
myCanvas.ctx.fillStyle = "blue";  
myCanvas.ctx.fillRect(0,0,256,256);
```

Detectando la posición del ratón en el lienzo.

Este ejemplo mostrará cómo obtener la posición del mouse en relación con el lienzo, de modo que (0,0) será la esquina superior izquierda del HTML5 Canvas. `e.clientX` y `e.clientY` obtendrán las posiciones del mouse en relación con la parte superior del documento. Para cambiar esto y basarse en la parte superior del lienzo, restamos las posiciones `left` y `right` del canvas del cliente X e Y.

```
var canvas = document.getElementById("myCanvas");  
var ctx = canvas.getContext("2d");  
ctx.font = "16px Arial";  
  
canvas.addEventListener("mousemove", function(e) {  
    var cRect = canvas.getBoundingClientRect(); // Gets CSS pos, and width/height  
    var canvasX = Math.round(e.clientX - cRect.left); // Subtract the 'left' of the canvas  
    var canvasY = Math.round(e.clientY - cRect.top); // from the X/Y positions to make  
    ctx.clearRect(0, 0, canvas.width, canvas.height); // (0,0) the top left of the canvas  
    ctx.fillText("X: "+canvasX+", Y: "+canvasY, 10, 20);  
});
```

Ejemplo ejecutable

El uso de `Math.round` se debe a garantizar que las posiciones `x,y` sean enteros, ya que el rectángulo delimitador del lienzo puede no tener posiciones enteras.

Hola Mundo

HTML

```
<canvas id="canvas" width=300 height=100 style="background-color:#808080;">  
</canvas>
```

Javascript

```
var canvas = document.getElementById("canvas");  
var ctx = canvas.getContext("2d");  
ctx.font = "34px serif";  
ctx.textAlign = "center";  
ctx.textBaseline="middle";  
ctx.fillStyle = "#FFF";  
ctx.fillText("Hello World",150,50);
```

Resultado



Hello World

Un índice de capacidades y usos de lienzo de HTML5

Capacidades del lienzo

Canvas le permite dibujar programáticamente en su página web:

- [Imágenes](#) ,
- [Textos](#) ,
- [Líneas y curvas](#) .

Los dibujos en lienzo pueden ser estilizados extensamente:

- [ancho de trazo](#) ,
- [color de trazo](#) ,
- [color de relleno de forma](#) ,
- [opacidad](#)
- [a la sombra](#)
- [degradados lineales](#) y [degradados radiales](#) ,
- [cara de la fuente](#) ,
- [tamaño de letra](#) ,
- [alineación del texto](#) ,
- [el texto se puede trazar, rellenar o tanto el trazo como el relleno](#) ,
- [redimensionamiento de imagen](#) ,
- [recorte de imágenes](#) ,
- [compositing](#)

Usos del lienzo

Los dibujos se pueden combinar y ubicar en cualquier lugar del lienzo para que se puedan usar para crear:

- Aplicaciones de pintura / bosquejo,
- Juegos interactivos de ritmo rápido,
- Análisis de datos como tablas, gráficos,
- Imágenes tipo Photoshop,
- Flash-como publicidad y contenido web Flashy.

Canvas le permite manipular los colores de las imágenes de los componentes Rojo, Verde, Azul y Alfa. Esto permite que el lienzo manipule imágenes con resultados similares a Photoshop.

- Vuelva a colorear cualquier parte de una imagen a nivel de píxeles (si usa HSL, puede incluso cambiar el color de una imagen mientras conserva la Iluminación y la Saturación importantes para que el resultado no se vea como si alguien hubiera pintado la imagen),
- "Knockout" el fondo alrededor de una persona / elemento en una imagen,
- Detecte y llene una parte de una imagen de la inundación (por ejemplo, cambie el color de un pétalo de flor con clic en el usuario de verde a amarillo, ¡solo ese pétalo con clic!),
- Hacer distorsión de perspectiva (por ejemplo, envolver una imagen alrededor de la curva de una taza),
- Examinar una imagen para el contenido (por ejemplo, reconocimiento facial),
- Responda preguntas sobre una imagen: ¿hay un automóvil estacionado en esta imagen de mi lugar de estacionamiento?
- Aplicar filtros de imagen estándar (escala de grises, sepia, etc.)
- Aplique cualquier filtro de imagen exótico que pueda imaginar (Sobel Edge Detection),
- Combina imágenes. Si la querida abuela Sue no pudo asistir a la reunión familiar, simplemente "photoshop" ella en la imagen de la reunión. No me gusta el primo Phil, solo "le saca el photoshop,
- Reproduce un video / Toma un fotograma de un video,
- Exportar el contenido del lienzo como .jpg | .png imagen (incluso puede recortar o anotar opcionalmente la imagen y exportar el resultado como una nueva imagen),

Acerca de mover y editar dibujos de lienzo (por ejemplo, para crear un juego de acción):

- Después de dibujar algo en el lienzo, ese dibujo existente no se puede mover ni editar. Vale la pena aclarar esta idea errónea de que los dibujos de lienzo son móviles: ¡ *los dibujos de lienzo existentes no se pueden editar ni mover!*
- Lienzo dibuja muy, muy rápido. Canvas puede dibujar cientos de imágenes, textos, líneas y curvas en una fracción de segundo. Utiliza la GPU cuando está disponible para acelerar el dibujo.
- Canvas crea la ilusión de movimiento dibujando algo rápida y repetidamente y luego volviéndolo a dibujar en una nueva posición. Al igual que la televisión, este dibujo constante le da al ojo la ilusión de movimiento.

Girar

El método `rotate(r)` del contexto 2D rota el lienzo en la cantidad `r` especificada de *radianes* alrededor del origen.

HTML

```
<canvas id="canvas" width=240 height=240 style="background-color:#808080;">
</canvas>

<button type="button" onclick="rotate_ctx();">Rotate context</button>
```

Javascript

```
var canvas = document.getElementById("canvas");
var ctx = canvas.getContext("2d");
```

```

var ox = canvas.width / 2;
var oy = canvas.height / 2;
ctx.font = "42px serif";
ctx.textAlign = "center";
ctx.textBaseline = "middle";
ctx.fillStyle = "#FFF";
ctx.fillText("Hello World", ox, oy);

rotate_ctx = function() {
  // translate so that the origin is now (ox, oy) the center of the canvas
  ctx.translate(ox, oy);
  // convert degrees to radians with radians = (Math.PI/180)*degrees.
  ctx.rotate((Math.PI / 180) * 15);
  ctx.fillText("Hello World", 0, 0);
  // translate back
  ctx.translate(-ox, -oy);
};

```

[Demo en vivo en JSfiddle](#)

Guardar lienzo en archivo de imagen

Puede guardar un lienzo en un archivo de imagen utilizando el método `canvas.toDataURL()`, que devuelve el *URI de datos* para los datos de imagen del lienzo.

El método puede tomar dos parámetros opcionales `canvas.toDataURL(type, encoderOptions)`: `type` es el formato de la imagen (si se omite, el valor predeterminado es `image/png`); `encoderOptions` es un número entre 0 y 1 que indica la calidad de la imagen (el valor predeterminado es 0.92).

Aquí dibujamos un lienzo y adjuntamos el URI de datos del lienzo al enlace "Descargar en myImage.jpg".

HTML

```

<canvas id="canvas" width=240 height=240 style="background-color:#808080;">
</canvas>
<p></p>
<a id="download" download="myImage.jpg" href="" onclick="download_img(this);">Download to
myImage.jpg</a>

```

Javascript

```

var canvas = document.getElementById("canvas");
var ctx = canvas.getContext("2d");
var ox = canvas.width / 2;
var oy = canvas.height / 2;
ctx.font = "42px serif";
ctx.textAlign = "center";
ctx.textBaseline = "middle";
ctx.fillStyle = "#800";
ctx.fillRect(ox / 2, oy / 2, ox, oy);

download_img = function(el) {
  // get image URI from canvas object
  var imageURI = canvas.toDataURL("image/jpeg");

```

```
el.href = imageURI;  
};
```

Demo en vivo en JSfiddle.

Lea Empezando con html5-canvas en línea: <https://riptutorial.com/es/html5-canvas/topic/1892/empezando-con-html5-canvas>

Capítulo 2: Animación

Examples

Animación simple con contexto 2D y requestAnimationFrame

Este ejemplo le mostrará cómo crear una animación simple utilizando el lienzo y el contexto 2D. Se supone que sabe cómo crear y agregar un lienzo al DOM y obtener el contexto

```
// this example assumes ctx and canvas have been created
const textToDisplay = "This is an example that uses the canvas to animate some text.";
const textStyle     = "white";
const BGStyle       = "black"; // background style
const textSpeed     = 0.2;     // in pixels per millisecond
const textHorMargin = 8;      // have the text a little outside the canvas

ctx.font = Math.floor(canvas.height * 0.8) + "px arial"; // size the font to 80% of canvas
height
var textWidth      = ctx.measureText(textToDisplay).width; // get the text width
var totalTextSize = (canvas.width + textHorMargin * 2 + textWidth);
ctx.textBaseline  = "middle"; // not put the text in the vertical center
ctx.textAlign     = "left";   // align to the left
var textX         = canvas.width + 8; // start with the text off screen to the right
var textOffset    = 0;        // how far the text has moved

var startTime;
// this function is call once a frame which is approx 16.66 ms (60fps)
function update(time){ // time is passed by requestAnimationFrame
  if(startTime === undefined){ // get a reference for the start time if this is the first
frame
    startTime = time;
  }
  ctx.fillStyle = BGStyle;
  ctx.fillRect(0, 0, canvas.width, canvas.height); // clear the canvas by
drawing over it
  textOffset    = ((time - startTime) * textSpeed) % (totalTextSize); // move the text left
  ctx.fillStyle = textStyle; // set the text style
  ctx.fillText(textToDisplay, textX - textOffset, canvas.height / 2); // render the text

  requestAnimationFrame(update); // all done request the next frame
}
requestAnimationFrame(update); // to start request the first frame
```

[Una demostración de este ejemplo en jsfiddle](#)

Animar en un intervalo específico (agregar un nuevo rectángulo cada 1 segundo)

Este ejemplo agrega un nuevo rectángulo al lienzo cada 1 segundo (== un intervalo de 1 segundo)

Código anotado:

```

<!doctype html>
<html>
<head>
<style>
  body{ background-color:white; }
  #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

  // canvas related variables
  var canvas=document.getElementById("canvas");
  var ctx=canvas.getContext("2d");
  var cw=canvas.width;
  var ch=canvas.height;

  // animation interval variables
  var nextTime=0;      // the next animation begins at "nextTime"
  var duration=1000;  // run animation every 1000ms

  var x=20;           // the X where the next rect is drawn

  // start the animation
  requestAnimationFrame(animate);

  function animate(currentTime){

    // wait for nextTime to occur
    if(currentTime<nextTime){
      // request another loop of animation
      requestAnimationFrame(animate);
      // time hasn't elapsed so just return
      return;
    }
    // set nextTime
    nextTime=currentTime+duration;

    // add another rectangle every 1000ms
    ctx.fillStyle='#'+Math.floor(Math.random()*16777215).toString(16);
    ctx.fillRect(x,30,30,30);

    // update X position for next rectangle
    x+=30;

    // request another loop of animation
    requestAnimationFrame(animate);
  }

}); // end $(function(){});
</script>
</head>
<body>
  <canvas id="canvas" width=512 height=512></canvas>
</body>
</html>

```

Animar a una hora determinada (un reloj animado)

Este ejemplo anima un reloj que muestra los segundos como una cuña llena.

Código anotado:

```
<!doctype html>
<html>
<head>
<style>
  body{ background-color:white; }
  #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

  // canvas related variables
  var canvas=document.getElementById("canvas");
  var ctx=canvas.getContext("2d");
  var cw=canvas.width;
  var ch=canvas.height;
  // canvas styling for the clock
  ctx.strokeStyle='lightgray';
  ctx.fillStyle='skyblue';
  ctx.lineWidth=5;

  // cache often used values
  var PI=Math.PI;
  var fullCircle=PI*2;
  var sa=-PI/2; // == the 12 o'clock angle in context.arc

  // start the animation
  requestAnimationFrame(animate);

  function animate(currentTime){

    // get the current seconds value from the system clock
    var date=new Date();
    var seconds=date.getSeconds();

    // clear the canvas
    ctx.clearRect(0,0,cw,ch);

    // draw a full circle (== the clock face);
    ctx.beginPath();
    ctx.moveTo(100,100);
    ctx.arc(100,100,75,0,fullCircle);
    ctx.stroke();
    // draw a wedge representing the current seconds value
    ctx.beginPath();
    ctx.moveTo(100,100);
    ctx.arc(100,100,75,sa,sa+fullCircle*seconds/60);
    ctx.fill();

    // request another loop of animation
    requestAnimationFrame(animate);
  }

}); // end $(function){}
</script>
</head>
<body>
  <canvas id="canvas" width=512 height=512></canvas>
</body>
</html>
```

Use requestAnimationFrame () NOT setInterval () para los bucles de animación

`requestAnimationFrame` es similar a `setInterval`, pero tiene estas importantes mejoras:

- El código de animación se sincroniza con las actualizaciones de pantalla para mejorar la eficiencia. El código de borrado + redibujado está programado, pero no se ejecuta de inmediato. El navegador ejecutará el código de borrado + redibujado solo cuando la pantalla esté lista para actualizarse. Esta sincronización con el ciclo de actualización aumenta el rendimiento de su animación al darle a su código el tiempo más disponible para completar.
- Cada bucle siempre se completa antes de que se pueda iniciar otro bucle. Esto evita "rasgar", donde el usuario ve una versión incompleta del dibujo. El ojo nota particularmente lagrimeo y se distrae cuando se produce lagrimeo. Por lo tanto, prevenir el desgarro hace que su animación se vea más suave y más consistente.
- La animación se detiene automáticamente cuando el usuario cambia a una pestaña diferente del navegador. Esto ahorra energía en los dispositivos móviles porque el dispositivo no desperdicia energía en una animación que el usuario no puede ver actualmente.

Las pantallas del dispositivo se actualizarán aproximadamente 60 veces por segundo, por lo que `requestAnimationFrame` puede volver a dibujar continuamente a aproximadamente 60 "cuadros" por segundo. El ojo ve movimiento a 20-30 cuadros por segundo, por lo que `requestAnimationFrame` puede crear fácilmente la ilusión de movimiento.

Observe que `requestAnimationFrame` se recupera al final de cada `animateCircle`. Esto se debe a que cada `requestAnimatonFrameonly` solicita una ejecución única de la función de animación.

Ejemplo: simple `requestAnimationFrame`

```
<!doctype html>
<html>
<head>
<style>
  body{ background-color:white; }
  #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

  // canvas related variables
  var canvas=document.getElementById("canvas");
  var ctx=canvas.getContext("2d");
  var cw=canvas.width;
  var ch=canvas.height;

  // start the animation
  requestAnimationFrame(animate);

  function animate(currentTime){

    // draw a full randomly circle
```

```

var x=Math.random()*canvas.width;
var y=Math.random()*canvas.height;
var radius=10+Math.random()*15;
ctx.beginPath();
ctx.arc(x,y,radius,0,Math.PI*2);
ctx.fillStyle='#'+Math.floor(Math.random()*16777215).toString(16);
ctx.fill();

// request another loop of animation
requestAnimationFrame(animate);
}

}); // end $(function(){});
</script>
</head>
<body>
  <canvas id="canvas" width=512 height=512></canvas>
</body>
</html>

```

Para ilustrar las ventajas de requestAnimationFrame, esta [pregunta de stackoverflow](#) tiene una [demostración en vivo](#)

Animar una imagen a través del lienzo.

Este ejemplo carga y anima una imagen a través del lienzo.

Sugerencia importante! Asegúrese de darle tiempo a su imagen para que se cargue completamente usando `image.onload`.

Código anotado

```

<!doctype html>
<html>
<head>
<style>
  body{ background-color:white; }
  #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

  // canvas related variables
  var canvas=document.getElementById("canvas");
  var ctx=canvas.getContext("2d");
  var cw=canvas.width;
  var ch=canvas.height;

  // animation related variables
  var minX=20;           // Keep the image animating
  var maxX=250;         // between minX & maxX
  var x=minX;           // The current X-coordinate
  var speedX=1;         // The image will move at 1px per loop
  var direction=1;      // The image direction: 1==rightward, -1==leftward
  var y=20;             // The Y-coordinate

  // Load a new image

```

```

// IMPORTANT!!! You must give the image time to load by using img.onload!
var img=new Image();
img.onload=start;
img.src="https://dl.dropboxusercontent.com/u/139992952/stackoverflow/sun.png";
function start(){
    // the image is fully loaded so start animating
    requestAnimationFrame(animate);
}

function animate(time){

    // clear the canvas
    ctx.clearRect(0,0,cw,ch);

    // draw
    ctx.drawImage(img,x,y);

    // update
    x += speedX * direction;
    // keep "x" inside min & max
    if(x<minX){ x=minX; direction*=-1; }
    if(x>maxX){ x=maxX; direction*=-1; }

    // request another loop of animation
    requestAnimationFrame(animate);
}

}); // end $(function){}
</script>
</head>
<body>
    <canvas id="canvas" width=512 height=512></canvas>
</body>
</html>

```

No dibujes animaciones en tus controladores de eventos (una aplicación de boceto simple)

Durante `mousemove` te `mousemove` con 30 eventos de ratón por segundo. Es posible que no pueda volver a dibujar sus dibujos a 30 veces por segundo. Incluso si puede, probablemente esté desperdiciando poder de cómputo al dibujar cuando el navegador no está listo para dibujar (desperdiciado == a través de los ciclos de actualización de la pantalla).

Por lo tanto, tiene sentido separar los eventos de entrada de sus usuarios (como `mousemove`) del dibujo de sus animaciones.

- En los controladores de eventos, guarde todas las variables de eventos que controlan dónde se ubican los dibujos en el Lienzo. Pero en realidad no dibujar nada.
- En un bucle `requestAnimationFrame`, renderice todos los dibujos al lienzo utilizando la información guardada.

Al no dibujar en los controladores de eventos, no está obligando a Canvas a intentar actualizar dibujos complejos a velocidades de eventos del mouse.

Al hacer todos los dibujos en `requestAnimationFrame` , obtiene todos los beneficios descritos aquí. Use 'requestAnimationFrame' no 'setInterval' para los bucles de animación .

Código anotado:

```
<!doctype html>
<html>
<head>
<style>
  body{ background-color: ivory; }
  #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

  function log(){console.log.apply(console,arguments);}

  // canvas variables
  var canvas=document.getElementById("canvas");
  var ctx=canvas.getContext("2d");
  var cw=canvas.width;
  var ch=canvas.height;
  // set canvas styling
  ctx.strokeStyle='skyblue';
  ctx.lineJoin='round';
  ctx.lineCap='round';
  ctx.lineWidth=6;

  // handle windows scrolling & resizing
  function reOffset(){
    var BB=canvas.getBoundingClientRect();
    offsetX=BB.left;
    offsetY=BB.top;
  }
  var offsetX,offsetY;
  reOffset();
  window.onscroll=function(e){ reOffset(); }
  window.onresize=function(e){ reOffset(); }

  // vars to save points created during mousemove handling
  var points=[];
  var lastLength=0;

  // start the animation loop
  requestAnimationFrame(draw);

  canvas.onmousemove=function(e){handleMouseMove(e);}

  function handleMouseMove(e){
    // tell the browser we're handling this event
    e.preventDefault();
    e.stopPropagation();

    // get the mouse position
    mouseX=parseInt(e.clientX-offsetX);
    mouseY=parseInt(e.clientY-offsetY);

    // save the mouse position in the points[] array
    // but don't draw anything
```

```

        points.push({x:mouseX,y:mouseY});
    }

    function draw(){
        // No additional points? Request another frame an return
        var length=points.length;
        if(length==lastLength){requestAnimationFrame(draw);return;}

        // draw the additional points
        var point=points[lastLength];
        ctx.beginPath();
        ctx.moveTo(point.x,point.y)
        for(var i=lastLength;i<length;i++){
            point=points[i];
            ctx.lineTo(point.x,point.y);
        }
        ctx.stroke();

        // request another animation loop
        requestAnimationFrame(draw);
    }

}); // end window.onload
</script>
</head>
<body>
    <h4>Move mouse over Canvas to sketch</h4>
    <canvas id="canvas" width=512 height=512></canvas>
</body>
</html>

```

Facilitando el uso de las ecuaciones de Robert Penners.

Una flexibilización hace que alguna **variable** cambie de **manera desigual a lo** largo de una **duración** .

"**variable**" debe poder expresarse como un número y puede representar una variedad notable de cosas:

- una coordenada X,
- el ancho de un rectángulo,
- un ángulo de rotación,
- El componente rojo de un color R, G, B.
- Cualquier cosa que se pueda expresar como un número.

La "**duración**" debe poder expresarse como un número y también puede ser una variedad de cosas:

- un período de tiempo,
- una distancia a recorrer,
- una cantidad de bucles de animación para ser ejecutados,
- cualquier cosa que pueda ser expresada como

"**desigual**" significa que la variable avanza de manera desigual desde el principio hasta el final:

- más rápido al principio y más lento al final, o viceversa,
- sobrepasa el final pero retrocede hasta el final a medida que termina la duración,
- Avanza / retrocede repetidamente elásticamente durante la duración,
- "rebota" en el final mientras se detiene a medida que la duración termina.

Atribución: Robert Penner ha creado el "estándar de oro" de las funciones de aceleración.

Cite: <https://github.com/danro/jquery-easing/blob/master/jquery.easing.js>

```
// t: elapsed time inside duration (currentTime-startTime),
// b: beginning value,
// c: total change from beginning value (endingValue-startingValue),
// d: total duration
var Easings={
  easeInQuad: function (t, b, c, d) {
    return c*(t/=d)*t + b;
  },
  easeOutQuad: function (t, b, c, d) {
    return -c *(t/=d)*(t-2) + b;
  },
  easeInOutQuad: function (t, b, c, d) {
    if ((t/=d/2) < 1) return c/2*t*t + b;
    return -c/2 * ((--t)*(t-2) - 1) + b;
  },
  easeInCubic: function (t, b, c, d) {
    return c*(t/=d)*t*t + b;
  },
  easeOutCubic: function (t, b, c, d) {
    return c*((t=t/d-1)*t*t + 1) + b;
  },
  easeInOutCubic: function (t, b, c, d) {
    if ((t/=d/2) < 1) return c/2*t*t*t + b;
    return c/2*((t-=2)*t*t + 2) + b;
  },
  easeInQuart: function (t, b, c, d) {
    return c*(t/=d)*t*t*t + b;
  },
  easeOutQuart: function (t, b, c, d) {
    return -c * ((t=t/d-1)*t*t*t - 1) + b;
  },
  easeInOutQuart: function (t, b, c, d) {
    if ((t/=d/2) < 1) return c/2*t*t*t*t + b;
    return -c/2 * ((t-=2)*t*t*t - 2) + b;
  },
  easeInQuint: function (t, b, c, d) {
    return c*(t/=d)*t*t*t*t + b;
  },
  easeOutQuint: function (t, b, c, d) {
    return c*((t=t/d-1)*t*t*t*t + 1) + b;
  },
  easeInOutQuint: function (t, b, c, d) {
    if ((t/=d/2) < 1) return c/2*t*t*t*t*t + b;
    return c/2*((t-=2)*t*t*t*t + 2) + b;
  },
  easeInSine: function (t, b, c, d) {
    return -c * Math.cos(t/d * (Math.PI/2)) + c + b;
  },
  easeOutSine: function (t, b, c, d) {
    return c * Math.sin(t/d * (Math.PI/2)) + b;
  }
}
```

```

},
easeInOutSine: function (t, b, c, d) {
    return -c/2 * (Math.cos(Math.PI*t/d) - 1) + b;
},
easeInExpo: function (t, b, c, d) {
    return (t==0) ? b : c * Math.pow(2, 10 * (t/d - 1)) + b;
},
easeOutExpo: function (t, b, c, d) {
    return (t==d) ? b+c : c * (-Math.pow(2, -10 * t/d) + 1) + b;
},
easeInOutExpo: function (t, b, c, d) {
    if (t==0) return b;
    if (t==d) return b+c;
    if ((t/=d/2) < 1) return c/2 * Math.pow(2, 10 * (t - 1)) + b;
    return c/2 * (-Math.pow(2, -10 * --t) + 2) + b;
},
easeInCirc: function (t, b, c, d) {
    return -c * (Math.sqrt(1 - (t/=d)*t) - 1) + b;
},
easeOutCirc: function (t, b, c, d) {
    return c * Math.sqrt(1 - (t=t/d-1)*t) + b;
},
easeInOutCirc: function (t, b, c, d) {
    if ((t/=d/2) < 1) return -c/2 * (Math.sqrt(1 - t*t) - 1) + b;
    return c/2 * (Math.sqrt(1 - (t-=2)*t) + 1) + b;
},
easeInElastic: function (t, b, c, d) {
    var s=1.70158;var p=0;var a=c;
    if (t==0) return b; if ((t/=d)==1) return b+c; if (!p) p=d*.3;
    if (a < Math.abs(c)) { a=c; var s=p/4; }
    else var s = p/(2*Math.PI) * Math.asin (c/a);
    return -(a*Math.pow(2,10*(t-=1)) * Math.sin( (t*d-s)*(2*Math.PI)/p )) + b;
},
easeOutElastic: function (t, b, c, d) {
    var s=1.70158;var p=0;var a=c;
    if (t==0) return b; if ((t/=d)==1) return b+c; if (!p) p=d*.3;
    if (a < Math.abs(c)) { a=c; var s=p/4; }
    else var s = p/(2*Math.PI) * Math.asin (c/a);
    return a*Math.pow(2,-10*t) * Math.sin( (t*d-s)*(2*Math.PI)/p ) + c + b;
},
easeInOutElastic: function (t, b, c, d) {
    var s=1.70158;var p=0;var a=c;
    if (t==0) return b; if ((t/=d/2)==2) return b+c; if (!p) p=d*(.3*1.5);
    if (a < Math.abs(c)) { a=c; var s=p/4; }
    else var s = p/(2*Math.PI) * Math.asin (c/a);
    if (t < 1) return -.5*(a*Math.pow(2,10*(t-=1)) * Math.sin( (t*d-s)*(2*Math.PI)/p )) + b;
    return a*Math.pow(2,-10*(t-=1)) * Math.sin( (t*d-s)*(2*Math.PI)/p )*.5 + c + b;
},
easeInBack: function (t, b, c, d, s) {
    if (s == undefined) s = 1.70158;
    return c*(t/=d)*t*((s+1)*t - s) + b;
},
easeOutBack: function (t, b, c, d, s) {
    if (s == undefined) s = 1.70158;
    return c*((t=t/d-1)*t*((s+1)*t + s) + 1) + b;
},
easeInOutBack: function (t, b, c, d, s) {
    if (s == undefined) s = 1.70158;
    if ((t/=d/2) < 1) return c/2*(t*t*(((s*(1.525))+1)*t - s)) + b;
    return c/2*((t-=2)*t*(((s*(1.525))+1)*t + s) + 2) + b;
},

```

```

easeInBounce: function (t, b, c, d) {
    return c - Easings.easeOutBounce (d-t, 0, c, d) + b;
},
easeOutBounce: function (t, b, c, d) {
    if ((t/=d) < (1/2.75)) {
        return c*(7.5625*t*t) + b;
    } else if (t < (2/2.75)) {
        return c*(7.5625*(t-=(1.5/2.75))*t + .75) + b;
    } else if (t < (2.5/2.75)) {
        return c*(7.5625*(t-=(2.25/2.75))*t + .9375) + b;
    } else {
        return c*(7.5625*(t-=(2.625/2.75))*t + .984375) + b;
    }
},
easeInOutBounce: function (t, b, c, d) {
    if (t < d/2) return Easings.easeInBounce (t*2, 0, c, d) * .5 + b;
    return Easings.easeOutBounce (t*2-d, 0, c, d) * .5 + c*.5 + b;
},
};

```

Ejemplo de uso:

```

// include the Easings object from above
var Easings = ...

// Demo
var startTime;
var beginningValue=50; // beginning x-coordinate
var endingValue=450; // ending x-coordinate
var totalChange=endingValue-beginningValue;
var totalDuration=3000; // ms

var keys=Object.keys(Easings);
ctx.textBaseline='middle';
requestAnimationFrame(animate);

function animate(time){
    var PI2=Math.PI*2;
    if(!startTime){startTime=time;}
    var elapsedTime=Math.min(time-startTime,totalDuration);
    ctx.clearRect(0,0,cw,ch);
    ctx.beginPath();
    for(var y=0;y<keys.length;y++){
        var key=keys[y];
        var easing=Easings[key];
        var easedX=easing(
            elapsedTime,beginningValue,totalChange,totalDuration);
        if(easedX>endingValue){easedX=endingValue;}
        ctx.moveTo(easedX,y*15);
        ctx.arc(easedX,y*15+10,5,0,PI2);
        ctx.fillText(key,460,y*15+10-1);
    }
    ctx.fill();
    if(time<startTime+totalDuration){
        requestAnimationFrame(animate);
    }
}

```

Establecer la velocidad de fotogramas utilizando requestAnimationFrame

El uso de `requestAnimationFrame` puede actualizarse en algunos sistemas a más cuadros por segundo que los 60 fps. 60fps es la tasa predeterminada si la representación puede continuar. Algunos sistemas se ejecutarán a 120 fps tal vez más.

Si usa el siguiente método, solo debe usar velocidades de cuadro que sean divisiones enteras de 60, de modo que `(60 / FRAMES_PER_SECOND) % 1 === 0` sea `true` o obtendrá velocidades de cuadro inconsistentes.

```
const FRAMES_PER_SECOND = 30; // Valid values are 60,30,20,15,10...
// set the min time to render the next frame
const FRAME_MIN_TIME = (1000/60) * (60 / FRAMES_PER_SECOND) - (1000/60) * 0.5;
var lastFrameTime = 0; // the last frame time
function update(time){
  if(time-lastFrameTime < FRAME_MIN_TIME){ //skip the frame if the call is too early
    requestAnimationFrame(update);
    return; // return as there is nothing to do
  }
  lastFrameTime = time; // remember the time of the rendered frame
  // render the frame
  requestAnimationFrame(update); // get next frame
}
requestAnimationFrame(update); // start animation
```

Animar de [x0, y0] a [x1, y1]

Usa vectores para calcular [x, y] incrementales de [startX, startY] a [endX, endY]

```
// dx is the total distance to move in the X direction
var dx = endX - startX;

// dy is the total distance to move in the Y direction
var dy = endY - startY;

// use a pct (percentage) to travel the total distances
// start at 0% which == the starting point
// end at 100% which == then ending point
var pct=0;

// use dx & dy to calculate where the current [x,y] is at a given pct
var x = startX + dx * pct/100;
var y = startY + dx * pct/100;
```

Código de ejemplo:

```
// canvas vars
var canvas=document.createElement("canvas");
document.body.appendChild(canvas);
canvas.style.border='1px solid red';
var ctx=canvas.getContext("2d");
var cw=canvas.width;
var ch=canvas.height;
// canvas styles
ctx.strokeStyle='skyblue';
ctx.fillStyle='blue';
```

```

// animating vars
var pct=101;
var startX=20;
var startY=50;
var endX=225;
var endY=100;
var dx=endX-startX;
var dy=endY-startY;

// start animation loop running
requestAnimationFrame(animate);

// listen for mouse events
window.onmousedown=(function(e){handleMouseDown(e)});
window.onmouseup=(function(e){handleMouseUp(e)});

// constantly running loop
// will animate dot from startX,startY to endX,endY
function animate(time){
    // demo: rerun animation
    if(++pct>100){pct=0;}
    // update
    x=startX+dx*pct/100;
    y=startY+dy*pct/100;
    // draw
    ctx.clearRect(0,0,cw,ch);
    ctx.beginPath();
    ctx.moveTo(startX,startY);
    ctx.lineTo(endX,endY);
    ctx.stroke();
    ctx.beginPath();
    ctx.arc(x,y,5,0,Math.PI*2);
    ctx.fill()
    // request another animation loop
    requestAnimationFrame(animate);
}

```

Lea Animación en línea: <https://riptutorial.com/es/html5-canvas/topic/4822/animacion>

Capítulo 3: Arrastrando formas de ruta e imágenes sobre lienzo

Examples

Cómo las formas e imágenes REALMENTE (!) Se "mueven" en el lienzo

Un problema: el lienzo solo recuerda píxeles, no formas o imágenes

Esta es una imagen de una pelota de playa circular y, por supuesto, no puede arrastrar la pelota alrededor de la imagen.



Puede que le sorprenda que al igual que una imagen, si dibuja un círculo en un Lienzo no puede arrastrar ese círculo alrededor del lienzo. Eso es porque el lienzo no recordará donde dibujó el círculo.

```
// this arc (==circle) is not draggable!!
context.beginPath();
context.arc(20, 30, 15, 0, Math.PI*2);
context.fillStyle='blue';
context.fill();
```

Lo que el lienzo NO sabe ...

- ... donde dibujaste el círculo (no sabe $x, y = [20,30]$).
- ... el tamaño del círculo (no sabe $radio = 15$).
- ... el color del círculo. (No se sabe que el círculo es azul).

Lo que el lienzo sabe ...

Canvas sabe el color de cada píxel en su superficie de dibujo.

El lienzo puede decirle que en $x, y == [20,30]$ hay un píxel azul, pero no sabe si este píxel azul es parte de un círculo.

Lo que esto significa...

Esto significa que todo lo dibujado en el lienzo es permanente: inmóvil e inmutable.

- El lienzo no puede mover el círculo o cambiar el tamaño del círculo.
- El lienzo no puede recolorar el círculo o borrar el círculo.
- Canvas no puede decir si el mouse está flotando sobre el círculo.
- Canvas no puede decir si el círculo está chocando con otro círculo.
- Canvas no puede permitir que un usuario arrastre el círculo alrededor de Canvas.

Pero Canvas puede dar la ILUSIÓN de movimiento.

El lienzo puede dar la **ilusión de movimiento** al borrar continuamente el círculo y volver a dibujarlo en una posición diferente. Al volver a dibujar el Lienzo varias veces por segundo, se engaña al ojo para que vea que el círculo se mueve a través del Lienzo.

- Borrar el lienzo
- Actualizar la posición del círculo.
- Redibuje el círculo en su nueva posición.
- Repetir, repetir, repetir ...

Este código da la **ilusión de movimiento** redibujando continuamente un círculo en nuevas posiciones.

```
// create a canvas
var canvas=document.createElement("canvas");
var ctx=canvas.getContext("2d");
ctx.fillStyle='red';
document.body.appendChild(canvas);

// a variable indicating a circle's X position
var circleX=20;

// start animating the circle across the canvas
// by continuously erasing & redrawing the circle
// in new positions
requestAnimationFrame(animate);

function animate(){
  // update the X position of the circle
  circleX++;
  // redraw the circle in it's new position
  ctx.clearRect(0,0,canvas.width,canvas.height);
  ctx.beginPath();
  ctx.arc( circleX, 30,15,0,Math.PI*2 );
  ctx.fill();
  // request another animate() loop
  requestAnimationFrame(animate);
}
```

Arrastrando círculos y rectángulos alrededor del lienzo

¿Qué es una "forma"?

Por lo general, guarda sus formas creando un objeto de "forma" de JavaScript que representa cada forma.

```
var myCircle = { x:30, y:20, radius:15 };
```

Por supuesto, no estás realmente guardando formas. En cambio, estás guardando la definición de cómo dibujar las formas.

Luego ponga cada objeto de forma en una matriz para una referencia fácil.

```
// save relevant information about shapes drawn on the canvas
var shapes=[];

// define one circle and save it in the shapes[] array
shapes.push( {x:10, y:20, radius:15, fillcolor:'blue'} );

// define one rectangle and save it in the shapes[] array
shapes.push( {x:10, y:100, width:50, height:35, fillcolor:'red'} );
```

Usando eventos del mouse para hacer arrastrar.

Arrastrar una forma o imagen requiere responder a estos eventos del mouse:

En mousedown:

Probar si alguna forma está debajo del ratón. Si una forma está debajo del mouse, el usuario tiene la intención de arrastrar esa forma. Por lo tanto mantener una referencia a la forma y establecer un verdadero / falso `isDragging` bandera que indica que un lastre está en proceso.

En mousemove:

Calcule la distancia que ha arrastrado el mouse desde el último evento de `mousemove` y cambie la posición de la forma arrastrada por esa distancia. Para cambiar la posición de la forma, cambia las propiedades de posición `x,y` en el objeto de esa forma.

En mouseup o mouseout:

El usuario tiene la intención de detener la operación de arrastre, así que borre el indicador "isDragging". Se completa el arrastre.

Demostración: arrastrando círculos y

rectángulos en el lienzo

Esta demostración arrastra círculos y rectángulos en el lienzo respondiendo a los eventos del mouse y dando la ilusión de movimiento al borrar y volver a dibujar.

```
// canvas related vars
var canvas=document.createElement("canvas");
var ctx=canvas.getContext("2d");
var cw=canvas.width;
var ch=canvas.height;
document.body.appendChild(canvas);
canvas.style.border='1px solid red';

// used to calc canvas position relative to window
function reOffset(){
    var BB=canvas.getBoundingClientRect();
    offsetX=BB.left;
    offsetY=BB.top;
}
var offsetX,offsetY;
reOffset();
window.onscroll=function(e){ reOffset(); }
window.onresize=function(e){ reOffset(); }
canvas.onresize=function(e){ reOffset(); }

// save relevant information about shapes drawn on the canvas
var shapes=[];
// define one circle and save it in the shapes[] array
shapes.push( {x:30, y:30, radius:15, color:'blue'} );
// define one rectangle and save it in the shapes[] array
shapes.push( {x:100, y:-1, width:75, height:35, color:'red'} );

// drag related vars
var isDragging=false;
var startX,startY;

// hold the index of the shape being dragged (if any)
var selectedShapeIndex;

// draw the shapes on the canvas
drawAll();

// listen for mouse events
canvas.onmousedown=handleMouseDown;
canvas.onmousemove=handleMouseMove;
canvas.onmouseup=handleMouseUp;
canvas.onmouseout=handleMouseOut;

// given mouse X & Y (mx & my) and shape object
// return true/false whether mouse is inside the shape
function isMouseInShape(mx,my,shape){
    if(shape.radius){
        // this is a circle
        var dx=mx-shape.x;
        var dy=my-shape.y;
        // math test to see if mouse is inside circle
        if(dx*dx+dy*dy<shape.radius*shape.radius){
            // yes, mouse is inside this circle
```

```

        return(true);
    }
} else if(shape.width){
    // this is a rectangle
    var rLeft=shape.x;
    var rRight=shape.x+shape.width;
    var rTop=shape.y;
    var rBott=shape.y+shape.height;
    // math test to see if mouse is inside rectangle
    if( mx>rLeft && mx<rRight && my>rTop && my<rBott){
        return(true);
    }
}
// the mouse isn't in any of the shapes
return(false);
}

function handleMouseDown(e){
    // tell the browser we're handling this event
    e.preventDefault();
    e.stopPropagation();
    // calculate the current mouse position
    startX=parseInt(e.clientX-offsetX);
    startY=parseInt(e.clientY-offsetY);
    // test mouse position against all shapes
    // post result if mouse is in a shape
    for(var i=0;i<shapes.length;i++){
        if(isMouseInShape(startX,startY,shapes[i])){
            // the mouse is inside this shape
            // select this shape
            selectedShapeIndex=i;
            // set the isDragging flag
            isDragging=true;
            // and return (==stop looking for
            // further shapes under the mouse)
            return;
        }
    }
}

function handleMouseUp(e){
    // return if we're not dragging
    if(!isDragging){return;}
    // tell the browser we're handling this event
    e.preventDefault();
    e.stopPropagation();
    // the drag is over -- clear the isDragging flag
    isDragging=false;
}

function handleMouseOut(e){
    // return if we're not dragging
    if(!isDragging){return;}
    // tell the browser we're handling this event
    e.preventDefault();
    e.stopPropagation();
    // the drag is over -- clear the isDragging flag
    isDragging=false;
}

function handleMouseMove(e){

```

```

// return if we're not dragging
if(!isDragging){return;}
// tell the browser we're handling this event
e.preventDefault();
e.stopPropagation();
// calculate the current mouse position
mouseX=parseInt(e.clientX-offsetX);
mouseY=parseInt(e.clientY-offsetY);
// how far has the mouse dragged from its previous mousemove position?
var dx=mouseX-startX;
var dy=mouseY-startY;
// move the selected shape by the drag distance
var selectedShape=shapes[selectedShapeIndex];
selectedShape.x+=dx;
selectedShape.y+=dy;
// clear the canvas and redraw all shapes
drawAll();
// update the starting drag position (== the current mouse position)
startX=mouseX;
startY=mouseY;
}

// clear the canvas and
// redraw all shapes in their current positions
function drawAll(){
    ctx.clearRect(0,0,cw,ch);
    for(var i=0;i<shapes.length;i++){
        var shape=shapes[i];
        if(shape.radius){
            // it's a circle
            ctx.beginPath();
            ctx.arc(shape.x,shape.y,shape.radius,0,Math.PI*2);
            ctx.fillStyle=shape.color;
            ctx.fill();
        }else if(shape.width){
            // it's a rectangle
            ctx.fillStyle=shape.color;
            ctx.fillRect(shape.x,shape.y,shape.width,shape.height);
        }
    }
}
}
}

```

Arrastrando formas irregulares alrededor del lienzo.

La mayoría de los dibujos de lienzo son rectangulares (rectángulos, imágenes, bloques de texto) o circulares (círculos).

Los círculos y rectángulos tienen pruebas matemáticas para verificar si el mouse está dentro de ellos. Esto hace que las pruebas de círculos y rectángulos sean fáciles, rápidas y eficientes. Puedes "probar" cientos de círculos o rectángulos en una fracción de segundo.

También puedes arrastrar formas irregulares. Pero las formas irregulares no tienen una prueba de golpe matemática rápida. Afortunadamente, las formas irregulares tienen una prueba de impacto integrada para determinar si un punto (mouse) está dentro de la forma: `context.isPointInPath`. Si bien `isPointInPath` funciona bien, no es tan eficiente como las pruebas de resultados puramente matemáticas; a menudo es hasta 10 veces más lento que las pruebas de resultados puramente

matemáticas.

Un requisito al usar `isPointInPath` es que debe "redefinir" la ruta que se está probando inmediatamente antes de llamar a `isPointInPath`. "Redefinir" significa que debe emitir los comandos de dibujo de ruta (como anteriormente), pero no necesita trazar () o completar () la ruta antes de probarla con `isPointInPath`. De esta manera, puede probar rutas previamente dibujadas sin tener que sobrescribir (trazar / rellenar) las rutas anteriores en el lienzo.

La forma irregular no necesita ser tan común como el triángulo cotidiano. También puedes hacer una prueba de golpe en cualquier camino irregularmente salvaje.

Este ejemplo anotado muestra cómo arrastrar formas de ruta irregular, así como círculos y rectángulos:

```
// canvas related vars
var canvas=document.createElement("canvas");
var ctx=canvas.getContext("2d");
var cw=canvas.width;
var ch=canvas.height;
document.body.appendChild(canvas);
canvas.style.border='1px solid red';

// used to calc canvas position relative to window
function reOffset(){
    var BB=canvas.getBoundingClientRect();
    offsetX=BB.left;
    offsetY=BB.top;
}
var offsetX,offsetY;
reOffset();
window.onscroll=function(e){ reOffset(); }
window.onresize=function(e){ reOffset(); }
canvas.onresize=function(e){ reOffset(); }

// save relevant information about shapes drawn on the canvas
var shapes=[];
// define one circle and save it in the shapes[] array
shapes.push( {x:20, y:20, radius:15, color:'blue'} );
// define one rectangle and save it in the shapes[] array
shapes.push( {x:100, y:-1, width:75, height:35, color:'red'} );
// define one triangle path and save it in the shapes[] array
shapes.push( {x:0, y:0, points:[{x:50,y:30},{x:75,y:60},{x:25,y:60}],color:'green'} );

// drag related vars
var isDragging=false;
var startX,startY;

// hold the index of the shape being dragged (if any)
var selectedShapeIndex;

// draw the shapes on the canvas
drawAll();

// listen for mouse events
canvas.onmousedown=handleMouseDown;
canvas.onmousemove=handleMouseMove;
canvas.onmouseup=handleMouseUp;
canvas.onmouseout=handleMouseOut;
```

```

// given mouse X & Y (mx & my) and shape object
// return true/false whether mouse is inside the shape
function isMouseInShape(mx,my,shape){
  if(shape.radius){
    // this is a circle
    var dx=mx-shape.x;
    var dy=my-shape.y;
    // math test to see if mouse is inside circle
    if(dx*dx+dy*dy<shape.radius*shape.radius){
      // yes, mouse is inside this circle
      return(true);
    }
  }else if(shape.width){
    // this is a rectangle
    var rLeft=shape.x;
    var rRight=shape.x+shape.width;
    var rTop=shape.y;
    var rBott=shape.y+shape.height;
    // math test to see if mouse is inside rectangle
    if( mx>rLeft && mx<rRight && my>rTop && my<rBott){
      return(true);
    }
  }else if(shape.points){
    // this is a polyline path
    // First redefine the path again (no need to stroke/fill!)
    defineIrregularPath(shape);
    // Then hit-test with isPointInPath
    if(ctx.isPointInPath(mx,my)){
      return(true);
    }
  }
  // the mouse isn't in any of the shapes
  return(false);
}

function handleMouseDown(e){
  // tell the browser we're handling this event
  e.preventDefault();
  e.stopPropagation();
  // calculate the current mouse position
  startX=parseInt(e.clientX-offsetX);
  startY=parseInt(e.clientY-offsetY);
  // test mouse position against all shapes
  // post result if mouse is in a shape
  for(var i=0;i<shapes.length;i++){
    if(isMouseInShape(startX,startY,shapes[i])){
      // the mouse is inside this shape
      // select this shape
      selectedShapeIndex=i;
      // set the isDragging flag
      isDragging=true;
      // and return (==stop looking for
      // further shapes under the mouse)
      return;
    }
  }
}

function handleMouseUp(e){
  // return if we're not dragging

```

```

    if(!isDragging){return;}
    // tell the browser we're handling this event
    e.preventDefault();
    e.stopPropagation();
    // the drag is over -- clear the isDragging flag
    isDragging=false;
}

function handleMouseOut(e){
    // return if we're not dragging
    if(!isDragging){return;}
    // tell the browser we're handling this event
    e.preventDefault();
    e.stopPropagation();
    // the drag is over -- clear the isDragging flag
    isDragging=false;
}

function handleMouseMove(e){
    // return if we're not dragging
    if(!isDragging){return;}
    // tell the browser we're handling this event
    e.preventDefault();
    e.stopPropagation();
    // calculate the current mouse position
    mouseX=parseInt(e.clientX-offsetX);
    mouseY=parseInt(e.clientY-offsetY);
    // how far has the mouse dragged from its previous mousemove position?
    var dx=mouseX-startX;
    var dy=mouseY-startY;
    // move the selected shape by the drag distance
    var selectedShape=shapes[selectedShapeIndex];
    selectedShape.x+=dx;
    selectedShape.y+=dy;
    // clear the canvas and redraw all shapes
    drawAll();
    // update the starting drag position (== the current mouse position)
    startX=mouseX;
    startY=mouseY;
}

// clear the canvas and
// redraw all shapes in their current positions
function drawAll(){
    ctx.clearRect(0,0,cw,ch);
    for(var i=0;i<shapes.length;i++){
        var shape=shapes[i];
        if(shape.radius){
            // it's a circle
            ctx.beginPath();
            ctx.arc(shape.x,shape.y,shape.radius,0,Math.PI*2);
            ctx.fillStyle=shape.color;
            ctx.fill();
        }else if(shape.width){
            // it's a rectangle
            ctx.fillStyle=shape.color;
            ctx.fillRect(shape.x,shape.y,shape.width,shape.height);
        }else if(shape.points){
            // its a polyline path
            defineIrregularPath(shape);
            ctx.fillStyle=shape.color;

```

```

        ctx.fill();
    }
}

function defineIrregularPath(shape) {
    var points=shape.points;
    ctx.beginPath();
    ctx.moveTo(shape.x+points[0].x, shape.y+points[0].y);
    for(var i=1; i<points.length; i++) {
        ctx.lineTo(shape.x+points[i].x, shape.y+points[i].y);
    }
    ctx.closePath();
}

```

Arrastrando imágenes alrededor del lienzo.

Consulte este [Ejemplo](#) para obtener una explicación general del arrastre de formas alrededor del lienzo.

Este ejemplo anotado muestra cómo arrastrar imágenes alrededor del lienzo

```

// canvas related vars
var canvas=document.createElement("canvas");
var ctx=canvas.getContext("2d");
canvas.width=378;
canvas.height=378;
var cw=canvas.width;
var ch=canvas.height;
document.body.appendChild(canvas);
canvas.style.border='1px solid red';

// used to calc canvas position relative to window
function reOffset() {
    var BB=canvas.getBoundingClientRect();
    offsetX=BB.left;
    offsetY=BB.top;
}
var offsetX,offsetY;
reOffset();
window.onscroll=function(e){ reOffset(); }
window.onresize=function(e){ reOffset(); }
canvas.onresize=function(e){ reOffset(); }

// save relevant information about shapes drawn on the canvas
var shapes=[];

// drag related vars
var isDragging=false;
var startX,startY;

// hold the index of the shape being dragged (if any)
var selectedShapeIndex;

// load the image
var card=new Image();
card.onload=function(){
    // define one image and save it in the shapes[] array

```

```

shapes.push( {x:30, y:10, width:127, height:150, image:card} );
// draw the shapes on the canvas
drawAll();
// listen for mouse events
canvas.onmousedown=handleMouseDown;
canvas.onmousemove=handleMouseMove;
canvas.onmouseup=handleMouseUp;
canvas.onmouseout=handleMouseOut;
};
// put your image src here!
card.src='https://dl.dropboxusercontent.com/u/139992952/stackoverflow/card.png';

// given mouse X & Y (mx & my) and shape object
// return true/false whether mouse is inside the shape
function isMouseInShape(mx,my,shape){
  // is this shape an image?
  if(shape.image){
    // this is a rectangle
    var rLeft=shape.x;
    var rRight=shape.x+shape.width;
    var rTop=shape.y;
    var rBott=shape.y+shape.height;
    // math test to see if mouse is inside image
    if( mx>rLeft && mx<rRight && my>rTop && my<rBott){
      return(true);
    }
  }
  // the mouse isn't in any of this shapes
  return(false);
}

function handleMouseDown(e){
  // tell the browser we're handling this event
  e.preventDefault();
  e.stopPropagation();
  // calculate the current mouse position
  startX=parseInt(e.clientX-offsetX);
  startY=parseInt(e.clientY-offsetY);
  // test mouse position against all shapes
  // post result if mouse is in a shape
  for(var i=0;i<shapes.length;i++){
    if(isMouseInShape(startX,startY,shapes[i])){
      // the mouse is inside this shape
      // select this shape
      selectedShapeIndex=i;
      // set the isDragging flag
      isDragging=true;
      // and return (==stop looking for
      // further shapes under the mouse)
      return;
    }
  }
}

function handleMouseUp(e){
  // return if we're not dragging
  if(!isDragging){return;}
  // tell the browser we're handling this event
  e.preventDefault();
  e.stopPropagation();
}

```



```

    // the drag is over -- clear the isDragging flag
    isDragging=false;
}

function handleMouseOut(e){
    // return if we're not dragging
    if(!isDragging){return;}
    // tell the browser we're handling this event
    e.preventDefault();
    e.stopPropagation();
    // the drag is over -- clear the isDragging flag
    isDragging=false;
}

function handleMouseMove(e){
    // return if we're not dragging
    if(!isDragging){return;}
    // tell the browser we're handling this event
    e.preventDefault();
    e.stopPropagation();
    // calculate the current mouse position
    mouseX=parseInt(e.clientX-offsetX);
    mouseY=parseInt(e.clientY-offsetY);
    // how far has the mouse dragged from its previous mousemove position?
    var dx=mouseX-startX;
    var dy=mouseY-startY;
    // move the selected shape by the drag distance
    var selectedShape=shapes[selectedShapeIndex];
    selectedShape.x+=dx;
    selectedShape.y+=dy;
    // clear the canvas and redraw all shapes
    drawAll();
    // update the starting drag position (== the current mouse position)
    startX=mouseX;
    startY=mouseY;
}

// clear the canvas and
// redraw all shapes in their current positions
function drawAll(){
    ctx.clearRect(0,0,cw,ch);
    for(var i=0;i<shapes.length;i++){
        var shape=shapes[i];
        if(shape.image){
            // it's an image
            ctx.drawImage(shape.image,shape.x,shape.y);
        }
    }
}
}

```

Lea Arrastrando formas de ruta e imágenes sobre lienzo en línea: <https://riptutorial.com/es/html5-canvas/topic/5318/arrastrando-formas-de-ruta-e-imagenes-sobre-lienzo>

Capítulo 4: Borrar la pantalla

Sintaxis

- `void clearRect (x, y, ancho, alto)`
- `ImageData createlImageData (ancho, alto)`

Observaciones

Ninguno de estos métodos producirá píxeles transparentes si el contexto se creó con el parámetro `alpha: false`.

Examples

Rectángulos

Puede utilizar el método `clearRect` para borrar cualquier sección rectangular del lienzo.

```
// Clear the entire canvas
ctx.clearRect(0, 0, canvas.width, canvas.height);
```

Nota: `clearRect` depende de la matriz de transformación.

Para solucionar esto, es posible restablecer la matriz de transformación antes de borrar el lienzo.

```
ctx.save(); // Save the current context state
ctx.setTransform(1, 0, 0, 1, 0, 0); // Reset the transformation matrix
ctx.clearRect(0, 0, canvas.width, canvas.height); // Clear the canvas
ctx.restore(); // Revert context state including
// transformation matrix
```

Nota: `ctx.save` y `ctx.restore` solo se requieren si desea mantener el estado de contexto 2D del lienzo. En algunas situaciones, guardar y restaurar puede ser lento y, por lo general, debe evitarse si no es necesario.

Datos de imagen en bruto

Es posible escribir directamente en los datos de la imagen renderizada usando `putImageData`. Al crear nuevos datos de imagen y luego asignarlos al lienzo, se borrará toda la pantalla.

```
var imageData = ctx.createImageData(canvas.width, canvas.height);
ctx.putImageData(imageData, 0, 0);
```

Nota: `putImageData` no se ve afectado por ninguna transformación aplicada al contexto. Escribirá los datos directamente en la región de píxeles representados.

Formas complejas

Es posible borrar regiones con formas complejas cambiando la propiedad

`globalCompositeOperation`.

```
// All pixels being drawn will be transparent
ctx.globalCompositeOperation = 'destination-out';

// Clear a triangular section
ctx.globalAlpha = 1; // ensure alpha is 1
ctx.fillStyle = '#000'; // ensure the current fillStyle does not have any transparency
ctx.beginPath();
ctx.moveTo(10, 0);
ctx.lineTo(0, 10);
ctx.lineTo(20, 10);
ctx.fill();

// Begin drawing normally again
ctx.globalCompositeOperation = 'source-over';
```

Lienzo claro con degradado.

En lugar de utilizar `clearRect` que hace que todos los píxeles sean transparentes, es posible que desee un fondo.

Para borrar con un gradiente

```
// create the background gradient once
var bgGrad = ctx.createLinearGradient(0,0,0,canvas.height);
bgGrad.addColorStop(0,"#0FF");
bgGrad.addColorStop(1,"#08F");

// Every time you need to clear the canvas
ctx.fillStyle = bgGrad;
ctx.fillRect(0,0,canvas.width,canvas.height);
```

Esto es aproximadamente la mitad de `0.008ms` rápidos que los `0.008ms` ms `0.004ms` pero los 4 millones de segundos no deberían afectar negativamente a ninguna animación en tiempo real. (Los tiempos variarán considerablemente según el dispositivo, la resolución, el navegador y la configuración del navegador. Los tiempos son solo para comparación)

Lienzo transparente utilizando operación compuesta

Borrar el lienzo utilizando la operación de composición. Esto borrará el lienzo independientemente de las transformaciones, pero no es tan rápido como `clearRect()`.

```
ctx.globalCompositeOperation = 'copy';
```

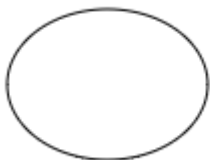
Todo lo que se dibuje a continuación borrará el contenido anterior.

Lea Borrar la pantalla en línea: <https://riptutorial.com/es/html5-canvas/topic/5245/borrar-la-pantalla>

Capítulo 5: Caminos

Examples

Elipse



Nota: los navegadores están en el proceso de agregar un comando de dibujo incorporado en `context.ellipse`, pero este comando no se adopta universalmente (en particular no en IE). Los siguientes métodos funcionan en todos los navegadores.

Dibuja una elipse dado que se desea la coordenada superior izquierda:

```
// draws an ellipse based on x,y being top-left coordinate
function drawEllipse(x,y,width,height){
    var PI2=Math.PI*2;
    var ratio=height/width;
    var radius=Math.max(width,height)/2;
    var increment = 1 / radius;
    var cx=x+width/2;
    var cy=y+height/2;

    ctx.beginPath();
    var x = cx + radius * Math.cos(0);
    var y = cy - ratio * radius * Math.sin(0);
    ctx.lineTo(x,y);

    for(var radians=increment; radians<PI2; radians+=increment){
        var x = cx + radius * Math.cos(radians);
        var y = cy - ratio * radius * Math.sin(radians);
        ctx.lineTo(x,y);
    }

    ctx.closePath();
    ctx.stroke();
}
```

Dibuja una elipse dado que es la coordenada del punto central deseada:

```
// draws an ellipse based on cx,cy being ellipse's centerpoint coordinate
function drawEllipse2(cx,cy,width,height){
    var PI2=Math.PI*2;
    var ratio=height/width;
    var radius=Math.max(width,height)/2;
    var increment = 1 / radius;

    ctx.beginPath();
```

```

var x = cx + radius * Math.cos(0);
var y = cy - ratio * radius * Math.sin(0);
ctx.lineTo(x,y);

for(var radians=increment; radians<PI2; radians+=increment){
    var x = cx + radius * Math.cos(radians);
    var y = cy - ratio * radius * Math.sin(radians);
    ctx.lineTo(x,y);
}

ctx.closePath();
ctx.stroke();
}

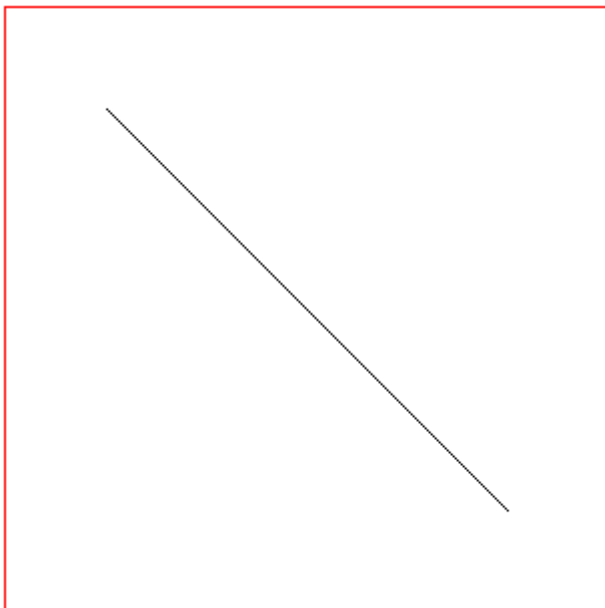
```

Línea sin borrosidad

Cuando Canvas dibuja una línea, agrega automáticamente suavizado para curar visualmente el "jaggedness". El resultado es una línea que es menos irregular pero más borrosa.

Esta función dibuja una línea entre 2 puntos sin suavizado utilizando el [algoritmo de línea de Bresenham](#) . El resultado es una línea nítida sin la irregularidad.

Nota importante: este método píxel por píxel es un método de dibujo mucho más lento que `context.lineTo` .



```

// Usage:
bresenhamLine(50,50,250,250);

// x,y line start
// xx,yy line end
// the pixel at line start and line end are drawn
function bresenhamLine(x, y, xx, yy){
    var oldFill = ctx.fillStyle; // save old fill style
    ctx.fillStyle = ctx.strokeStyle; // move stroke style to fill
    xx = Math.floor(xx);
    yy = Math.floor(yy);
    x = Math.floor(x);

```

```

y = Math.floor(y);
// BRENSHAM
var dx = Math.abs(xx-x);
var sx = x < xx ? 1 : -1;
var dy = -Math.abs(yy-y);
var sy = y < yy ? 1 : -1;
var err = dx+dy;
var errC; // error value
var end = false;
var x1 = x;
var y1 = y;

while(!end){
  ctx.fillRect(x1, y1, 1, 1); // draw each pixel as a rect
  if (x1 === xx && y1 === yy) {
    end = true;
  }else{
    errC = 2*err;
    if (errC >= dy) {
      err += dy;
      x1 += sx;
    }
    if (errC <= dx) {
      err += dx;
      y1 += sy;
    }
  }
}
ctx.fillStyle = oldFill; // restore old fill style
}

```

Lea Caminos en línea: <https://riptutorial.com/es/html5-canvas/topic/5133/caminos>

Capítulo 6: Colisiones e Intersecciones

Examples

¿Chocan 2 círculos?

```
// circle objects: { x:, y:, radius: }
// return true if the 2 circles are colliding
// c1 and c2 are circles as defined above

function CirclesColliding(c1,c2){
    var dx=c2.x-c1.x;
    var dy=c2.y-c1.y;
    var rSum=c1.radius+c2.radius;
    return(dx*dx+dy*dy<=rSum*rSum);
}
```

¿Chocan 2 rectángulos?

```
// rectangle objects { x:, y:, width:, height: }
// return true if the 2 rectangles are colliding
// r1 and r2 are rectangles as defined above

function RectsColliding(r1,r2){
    return !(
        r1.x>r2.x+r2.width ||
        r1.x+r1.width<r2.x ||
        r1.y>r2.y+r2.height ||
        r1.y+r1.height<r2.y
    );
}
```

¿Están colisionando un círculo y un rectángulo?

```
// rectangle object: { x:, y:, width:, height: }
// circle object: { x:, y:, radius: }
// return true if the rectangle and circle are colliding

function RectCircleColliding(rect,circle){
    var dx=Math.abs(circle.x-(rect.x+rect.width/2));
    var dy=Math.abs(circle.y-(rect.y+rect.height/2));

    if( dx > circle.radius+rect.width/2 ){ return(false); }
    if( dy > circle.radius+rect.height/2 ){ return(false); }

    if( dx <= rect.width ){ return(true); }
    if( dy <= rect.height ){ return(true); }

    var dx=dx-rect.width;
    var dy=dy-rect.height
    return(dx*dx+dy*dy<=circle.radius*circle.radius);
}
```

¿Están interceptando los segmentos de 2 líneas?

La función en este ejemplo devuelve `true` si dos segmentos de línea se intersecan y `false` si no.

El ejemplo está diseñado para el rendimiento y utiliza el cierre para mantener las variables de trabajo

```
// point object: {x:, y:}
// p0 & p1 form one segment, p2 & p3 form the second segment
// Returns true if lines segments are intercepting
var lineSegmentsIntercept = (function(){ // function as singleton so that closure can be
used

    var v1, v2, v3, cross, u1, u2; // working variable are closed over so they do not
need creation

    // each time the function is called. This gives a
significant performance boost.
    v1 = {x : null, y : null}; // line p0, p1 as vector
    v2 = {x : null, y : null}; // line p2, p3 as vector
    v3 = {x : null, y : null}; // the line from p0 to p2 as vector

    function lineSegmentsIntercept (p0, p1, p2, p3) {
        v1.x = p1.x - p0.x; // line p0, p1 as vector
        v1.y = p1.y - p0.y;
        v2.x = p3.x - p2.x; // line p2, p3 as vector
        v2.y = p3.y - p2.y;
        if((cross = v1.x * v2.y - v1.y * v2.x) === 0){ // cross prod 0 if lines parallel
            return false; // no intercept
        }
        v3 = {x : p0.x - p2.x, y : p0.y - p2.y}; // the line from p0 to p2 as vector
        u2 = (v1.x * v3.y - v1.y * v3.x) / cross; // get unit distance along line p2 p3
        // code point B
        if (u2 >= 0 && u2 <= 1){ // is intercept on line p2, p3
            u1 = (v2.x * v3.y - v2.y * v3.x) / cross; // get unit distance on line p0, p1;
            // code point A
            return (u1 >= 0 && u1 <= 1); // return true if on line else false.
            // code point A end
        }
        return false; // no intercept;
        // code point B end
    }
    return lineSegmentsIntercept; // return function with closure for optimisation.
})();
```

Ejemplo de uso

```
var p1 = {x: 100, y: 0}; // line 1
var p2 = {x: 120, y: 200};
var p3 = {x: 0, y: 100}; // line 2
var p4 = {x: 100, y: 120};
var areIntersecting = lineSegmentsIntercept (p1, p2, p3, p4); // true
```

El ejemplo se modifica fácilmente para devolver el punto de intersección. Reemplace el código entre el `code point A` y el `A end` con

```
if(u1 >= 0 && u1 <= 1){
```



```

return {
    x : p0.x + v1.x * u1,
    y : p0.y + v1.y * u1,
};
}

```

O si desea obtener el punto de intercepción en las líneas, ignorando el inicio y el final de los segmentos de línea, reemplace el código entre el `code point B` y el `B end` con

```

return {
    x : p2.x + v2.x * u2,
    y : p2.y + v2.y * u2,
};

```

Ambas modificaciones devolverán `false` si no hay una intercepción o el punto de intercepción como `{x : xCoord, y : yCoord}`

¿Un segmento de línea y un círculo colisionan?

```

// [x0,y0] to [x1,y1] define a line segment
// [cx,cy] is circle centerpoint, cr is circle radius
function isCircleSegmentColliding(x0,y0,x1,y1,cx,cy,cr){

    // calc delta distance: source point to line start
    var dx=cx-x0;
    var dy=cy-y0;

    // calc delta distance: line start to end
    var dxx=x1-x0;
    var dyy=y1-y0;

    // Calc position on line normalized between 0.00 & 1.00
    // == dot product divided by delta line distances squared
    var t=(dx*dxx+dy*dyy)/(dxx*dxx+dyy*dyy);

    // calc nearest pt on line
    var x=x0+dxx*t;
    var y=y0+dyy*t;

    // clamp results to being on the segment
    if(t<0){x=x0;y=y0;}
    if(t>1){x=x1;y=y1;}

    return( (cx-x)*(cx-x)+(cy-y)*(cy-y) < cr*cr );
}

```

¿Están el segmento de línea y el rectángulo colisionando?

```

// var rect={x:,y:,width:,height:};
// var line={x1:,y1:,x2:,y2:};
// Get interesing point of line segment & rectangle (if any)
function lineRectCollide(line,rect){

    // p=line startpoint, p2=line endpoint
    var p={x:line.x1,y:line.y1};

```

```

var p2={x:line.x2,y:line.y2};

// top rect line
var q={x:rect.x,y:rect.y};
var q2={x:rect.x+rect.width,y:rect.y};
if(lineSegmentsCollide(p,p2,q,q2){ return true; }
// right rect line
var q=q2;
var q2={x:rect.x+rect.width,y:rect.y+rect.height};
if(lineSegmentsCollide(p,p2,q,q2){ return true; }
// bottom rect line
var q=q2;
var q2={x:rect.x,y:rect.y+rect.height};
if(lineSegmentsCollide(p,p2,q,q2){ return true; }
// left rect line
var q=q2;
var q2={x:rect.x,y:rect.y};
if(lineSegmentsCollide(p,p2,q,q2){ return true; }

// not intersecting with any of the 4 rect sides
return(false);
}

// point object: {x:, y:}
// p0 & p1 form one segment, p2 & p3 form the second segment
// Get intersecting point of 2 line segments (if any)
// Attribution: http://paulbourke.net/geometry/pointlineplane/
function lineSegmentsCollide(p0,p1,p2,p3) {

var unknownA = (p3.x-p2.x) * (p0.y-p2.y) - (p3.y-p2.y) * (p0.x-p2.x);
var unknownB = (p1.x-p0.x) * (p0.y-p2.y) - (p1.y-p0.y) * (p0.x-p2.x);
var denominator = (p3.y-p2.y) * (p1.x-p0.x) - (p3.x-p2.x) * (p1.y-p0.y);

// Test if Coincident
// If the denominator and numerator for the ua and ub are 0
// then the two lines are coincident.
if(unknownA==0 && unknownB==0 && denominator==0){return(null);}

// Test if Parallel
// If the denominator for the equations for ua and ub is 0
// then the two lines are parallel.
if (denominator == 0) return null;

// test if line segments are colliding
unknownA /= denominator;
unknownB /= denominator;
var isIntersecting=(unknownA>=0 && unknownA<=1 && unknownB>=0 && unknownB<=1)

return(isIntersecting);
}

```

¿Chocan 2 polígonos convexos?

Use el teorema del eje de separación para determinar si dos polígonos convexos se intersecan

LOS POLÍGONOS DEBEN SER CONVEXOS

Atribución: Markus Jarrot @ [¿Cómo verificar la intersección entre 2 rectángulos girados?](#)

```

// polygon objects are an array of vertices forming the polygon
//     var polygon1=[{x:100,y:100},{x:150,y:150},{x:50,y:150},...];
// THE POLYGONS MUST BE CONVEX
// return true if the 2 polygons are colliding

function convexPolygonsCollide(a, b){
  var polygons = [a, b];
  var minA, maxA, projected, i, i1, j, minB, maxB;

  for (i = 0; i < polygons.length; i++) {

    // for each polygon, look at each edge of the polygon, and determine if it separates
    // the two shapes
    var polygon = polygons[i];
    for (i1 = 0; i1 < polygon.length; i1++) {

      // grab 2 vertices to create an edge
      var i2 = (i1 + 1) % polygon.length;
      var p1 = polygon[i1];
      var p2 = polygon[i2];

      // find the line perpendicular to this edge
      var normal = { x: p2.y - p1.y, y: p1.x - p2.x };

      minA = maxA = undefined;
      // for each vertex in the first shape, project it onto the line perpendicular to
the edge
      // and keep track of the min and max of these values
      for (j = 0; j < a.length; j++) {
        projected = normal.x * a[j].x + normal.y * a[j].y;
        if (minA==undefined || projected < minA) {
          minA = projected;
        }
        if (maxA==undefined || projected > maxA) {
          maxA = projected;
        }
      }

      // for each vertex in the second shape, project it onto the line perpendicular to
the edge
      // and keep track of the min and max of these values
      minB = maxB = undefined;
      for (j = 0; j < b.length; j++) {
        projected = normal.x * b[j].x + normal.y * b[j].y;
        if (minB==undefined || projected < minB) {
          minB = projected;
        }
        if (maxB==undefined || projected > maxB) {
          maxB = projected;
        }
      }

      // if there is no overlap between the projects, the edge we are looking at
separates the two
      // polygons, and we know there is no overlap
      if (maxA < minB || maxB < minA) {
        return false;
      }
    }
  }
  return true;
}

```

```
};
```

¿Chocan 2 polígonos? (Se permiten polis tanto cóncavas como convexas)

Prueba todos los lados de los polígonos en busca de intersecciones para determinar si 2 polígonos están colisionando.

```
// polygon objects are an array of vertices forming the polygon
//     var polygon1=[{x:100,y:100},{x:150,y:150},{x:50,y:150},...];
// The polygons can be both concave and convex
// return true if the 2 polygons are colliding

function polygonsCollide(p1,p2){
  // turn vertices into line points
  var lines1=verticesToLinePoints(p1);
  var lines2=verticesToLinePoints(p2);
  // test each poly1 side vs each poly2 side for intersections
  for(i=0; i<lines1.length; i++){
    for(j=0; j<lines2.length; j++){
      // test if sides intersect
      var p0=lines1[i][0];
      var p1=lines1[i][1];
      var p2=lines2[j][0];
      var p3=lines2[j][1];
      // found an intersection -- polys do collide
      if(lineSegmentsCollide(p0,p1,p2,p3)){return(true);}
    }
  }
  // none of the sides intersect
  return(false);
}

// helper: turn vertices into line points
function verticesToLinePoints(p){
  // make sure polys are self-closing
  if(!(p[0].x==p[p.length-1].x && p[0].y==p[p.length-1].y)){
    p.push({x:p[0].x,y:p[0].y});
  }
  var lines=[];
  for(var i=1;i<p.length;i++){
    var p1=p[i-1];
    var p2=p[i];
    lines.push([
      {x:p1.x, y:p1.y},
      {x:p2.x, y:p2.y}
    ]);
  }
  return(lines);
}

// helper: test line intersections
// point object: {x:, y:}
// p0 & p1 form one segment, p2 & p3 form the second segment
// Get intersecting point of 2 line segments (if any)
// Attribution: http://paulbourke.net/geometry/pointlineplane/
function lineSegmentsCollide(p0,p1,p2,p3) {
  var unknownA = (p3.x-p2.x) * (p0.y-p2.y) - (p3.y-p2.y) * (p0.x-p2.x);
  var unknownB = (p1.x-p0.x) * (p0.y-p2.y) - (p1.y-p0.y) * (p0.x-p2.x);
  var denominator = (p3.y-p2.y) * (p1.x-p0.x) - (p3.x-p2.x) * (p1.y-p0.y);

  // Test if Coincident
  // If the denominator and numerator for the ua and ub are 0
```

```

// then the two lines are coincident.
if(unknownA==0 && unknownB==0 && denominator==0){return(null);}

// Test if Parallel
// If the denominator for the equations for ua and ub is 0
// then the two lines are parallel.
if (denominator == 0) return null;

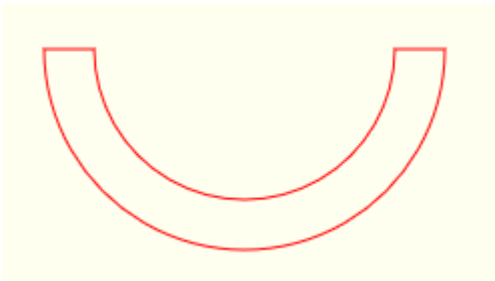
// test if line segments are colliding
unknownA /= denominator;
unknownB /= denominator;
var isIntersecting=(unknownA>=0 && unknownA<=1 && unknownB>=0 && unknownB<=1)

return(isIntersecting);
}

```

¿Un punto X, Y está dentro de un arco?

Comprueba si el punto [x, y] está dentro de un arco cerrado.



```

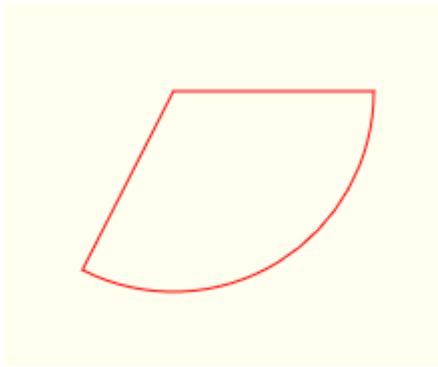
var arc={
  cx:150, cy:150,
  innerRadius:75, outerRadius:100,
  startAngle:0, endAngle:Math.PI
}

function isPointInArc(x,y,arc){
  var dx=x-arc.cx;
  var dy=y-arc.cy;
  var dxy=dx*dx+dy*dy;
  var rrOuter=arc.outerRadius*arc.outerRadius;
  var rrInner=arc.innerRadius*arc.innerRadius;
  if(dxy<rrInner || dxy>rrOuter){return(false);}
  var angle=(Math.atan2(dy,dx)+PI2)%PI2;
  return(angle>=arc.startAngle && angle<=arc.endAngle);
}

```

¿Un punto X, Y está dentro de una cuña?

Comprueba si el punto [x, y] está dentro de una cuña.



```
// wedge objects: {cx:,cy:,radius:,startAngle:,endAngle:}
// var wedge={
//     cx:150, cy:150, // centerpoint
//     radius:100,
//     startAngle:0, endAngle:Math.PI
// }
// Return true if the x,y point is inside the closed wedge

function isPointInWedge(x,y,wedge) {
    var PI2=Math.PI*2;
    var dx=x-wedge.cx;
    var dy=y-wedge.cy;
    var rr=wedge.radius*wedge.radius;
    if(dx*dx+dy*dy>rr){return(false);}
    var angle=(Math.atan2(dy,dx)+PI2)%PI2;
    return(angle>=wedge.startAngle && angle<=wedge.endAngle);
}
```

¿Un punto X, Y está dentro de un círculo?

Comprueba si un punto [x, y] está dentro de un círculo.

```
// circle objects: {cx:,cy:,radius:,startAngle:,endAngle:}
// var circle={
//     cx:150, cy:150, // centerpoint
//     radius:100,
// }
// Return true if the x,y point is inside the circle

function isPointInCircle(x,y,circle) {
    var dx=x-circle.cx;
    var dy=y-circle.cy;
    return(dx*dx+dy*dy<circle.radius*circle.radius);
}
```

¿Un punto X, Y está dentro de un rectángulo?

Comprueba si un punto [x, y] está dentro de un rectángulo.

```
// rectangle objects: {x:, y:, width:, height: }
// var rect={x:10, y:15, width:25, height:20}
// Return true if the x,y point is inside the rectangle

function isPointInRectangle(x,y,rect) {
```

```
return(x>rect.x && x<rect.x+rect.width && y>rect.y && y<rect.y+rect.height);  
}
```

Lea Colisiones e Intersecciones en línea: <https://riptutorial.com/es/html5-canvas/topic/5017/colisiones-e-intersecciones>

Capítulo 7: Compositing

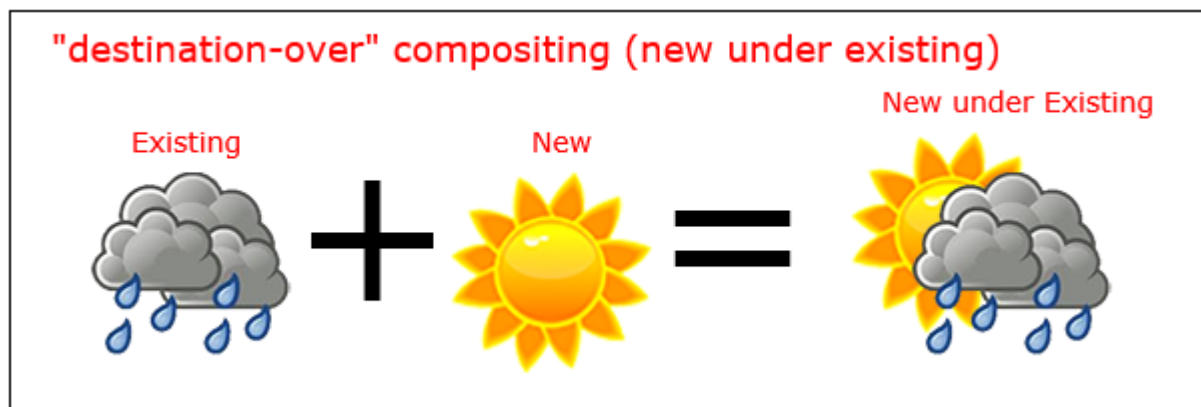
Examples

Dibuja detrás de las formas existentes con "destino sobre"

```
context.globalCompositeOperation = "destination-over"
```

La composición "destino sobre" coloca el nuevo dibujo *debajo de los dibujos existentes*.

```
context.drawImage(rainy, 0, 0);  
context.globalCompositeOperation='destination-over'; // sunny UNDER rainy  
context.drawImage(sunny, 0, 0);
```



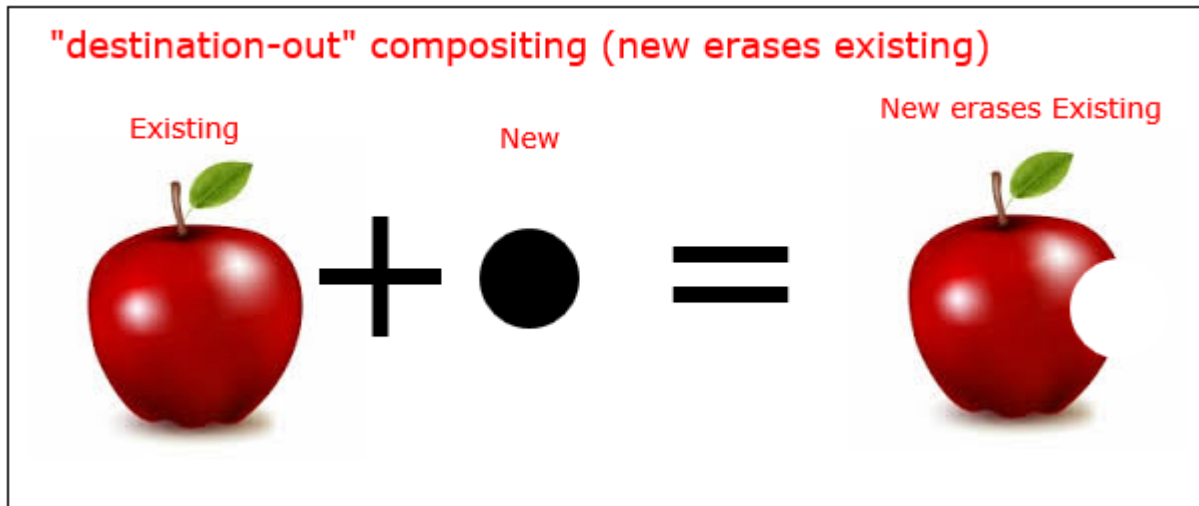
Borre las formas existentes con "destination-out"

```
context.globalCompositeOperation = "destination-out"
```

La composición "destination-out" utiliza nuevas formas para borrar los dibujos existentes.

La nueva forma no está realmente dibujada, solo se utiliza como "cortador de galletas" para borrar los píxeles existentes.

```
context.drawImage(apple, 0, 0);  
context.globalCompositeOperation = 'destination-out'; // bitmap erases  
context.drawImage(bitmap, 100, 40);
```

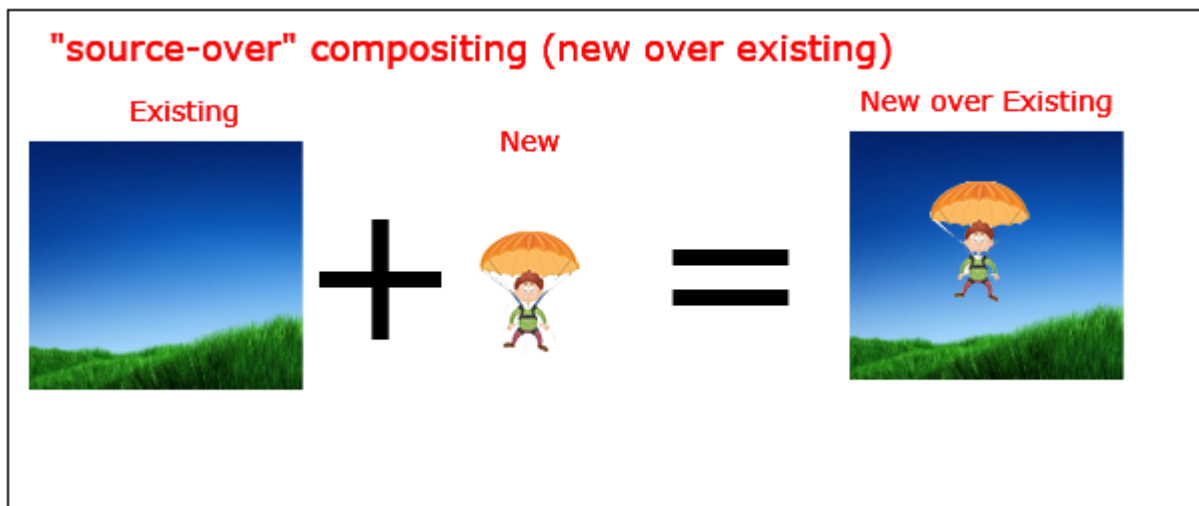



Composición predeterminada: las nuevas formas se dibujan sobre las formas existentes

```
context.globalCompositeOperation = "source-over"
```

La composición de "fuente sobre" [**predeterminada**] , coloca todos los dibujos nuevos sobre los dibujos existentes.

```
context.globalCompositeOperation='source-over'; // the default
context.drawImage(background,0,0);
context.drawImage(parachuter,0,0);
```



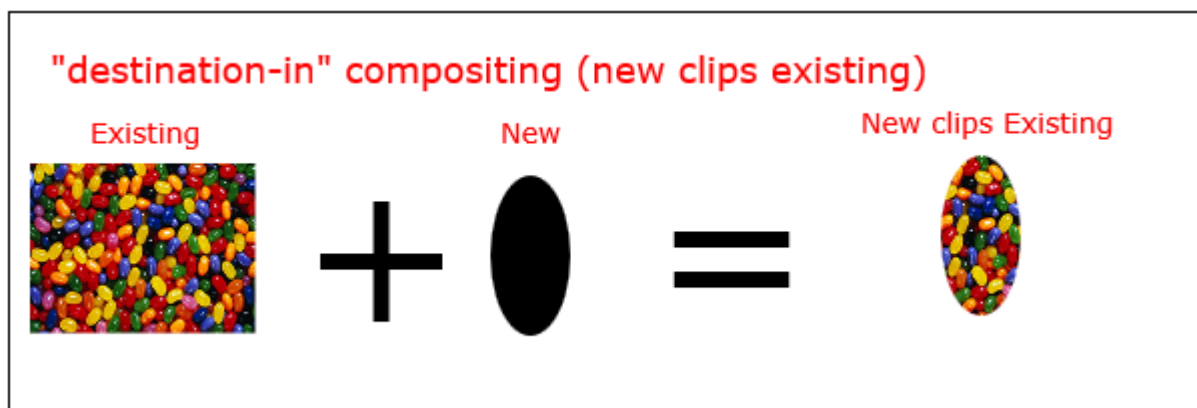
Clip imágenes dentro de formas con "destino-en"

```
context.globalCompositeOperation = "destination-in"
```

La composición de "destino en" recorta los dibujos existentes dentro de una nueva forma.

Nota: se borra cualquier parte del dibujo existente que se encuentre fuera del dibujo nuevo.

```
context.drawImage (picture,0,0);
context.globalCompositeOperation='destination-in'; // picture clipped inside oval
context.drawImage (oval,0,0);
```



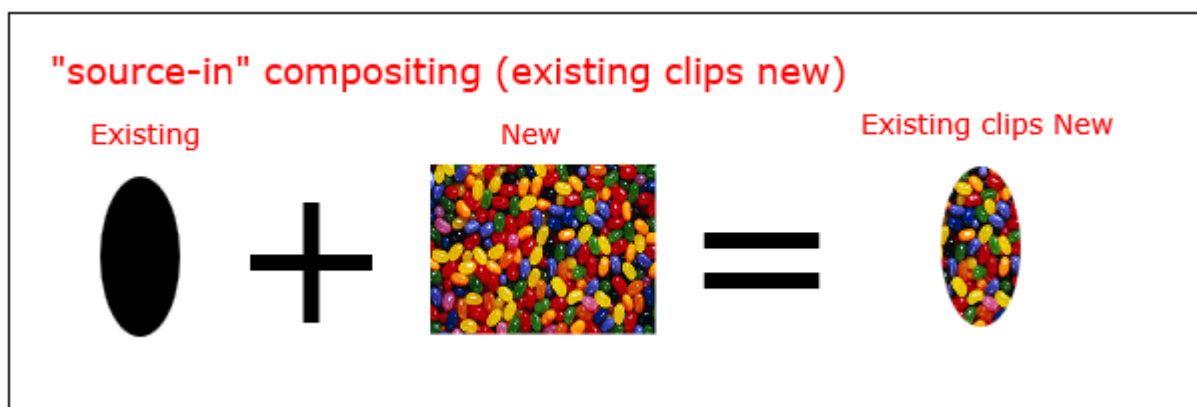
Clip imágenes dentro de formas con "fuente-en"

```
context.globalCompositeOperation = "source-in";
```

`source-in` composición de `source-in` clips nuevos dibujos dentro de una forma existente.

Nota: se borra cualquier parte del nuevo dibujo que se encuentre fuera del dibujo existente.

```
context.drawImage (oval,0,0);
context.globalCompositeOperation='source-in'; // picture clipped inside oval
context.drawImage (picture,0,0);
```



Sombras internas con "fuente-encima"

```
context.globalCompositeOperation = 'source-atop'
```

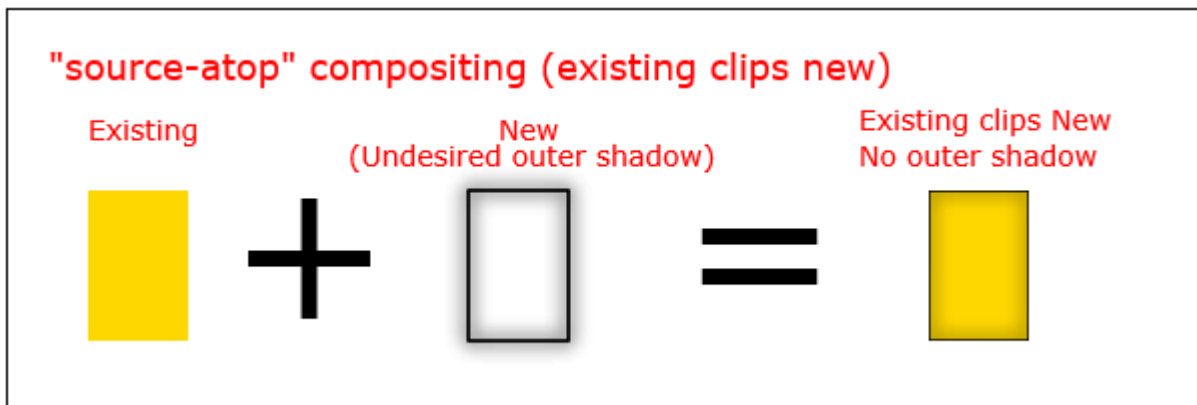
`source-atop` composición de `source-atop` sujeta una nueva imagen dentro de una forma existente.

```
// gold filled rect
ctx.fillStyle='gold';
ctx.fillRect (100,100,100,75);
// shadow
```

```

ctx.shadowColor='black';
ctx.shadowBlur=10;
// restrict new draw to cover existing pixels
ctx.globalCompositeOperation='source-atop';
// shadowed stroke
// "source-atop" clips off the undesired outer shadow
ctx.strokeRect(100,100,100,75);
ctx.strokeRect(100,100,100,75);

```



Invertir o negar imagen con "diferencia"

Renderizar un rectángulo blanco sobre una imagen con la operación compuesta

```

ctx.globalCompositeOperation = 'difference';

```

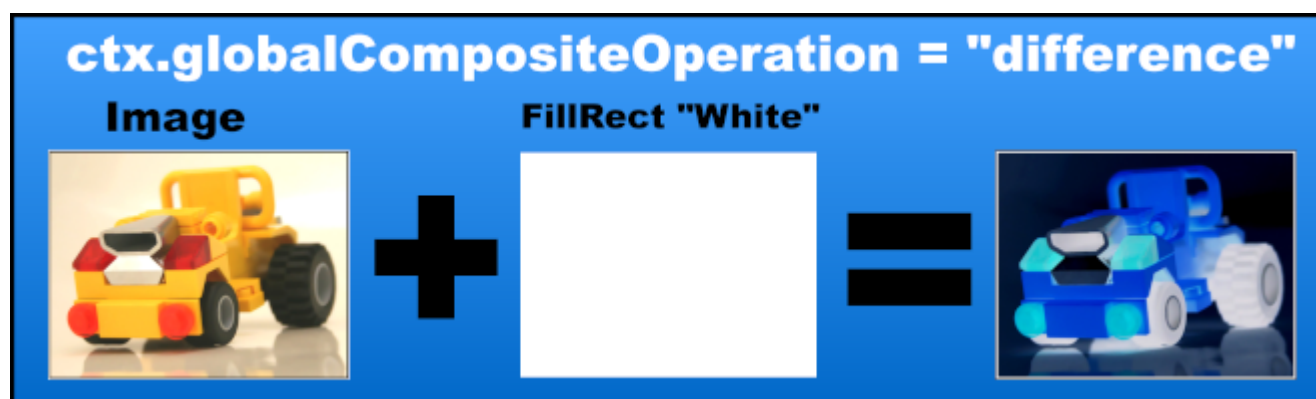
La cantidad del efecto se puede controlar con el ajuste alfa

```

// Render the image
ctx.globalCompositeOperation='source-atop';
ctx.drawImage(image, 0, 0);

// set the composite operation
ctx.globalCompositeOperation='difference';
ctx.fillStyle = "white";
ctx.globalAlpha = alpha; // alpha 0 = no effect 1 = full effect
ctx.fillRect(0, 0, image.width, image.height);

```



Blanco y negro con "color"

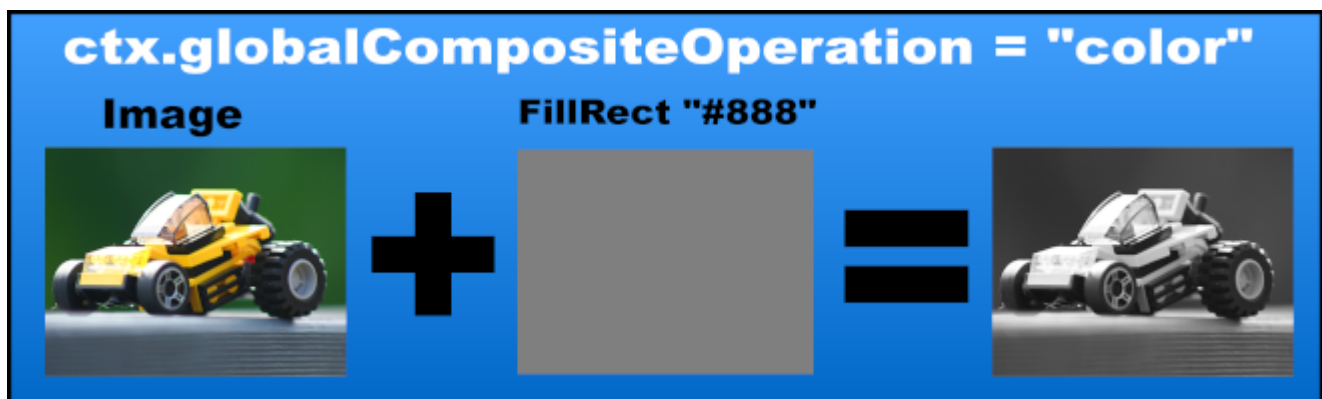
Eliminar color de una imagen a través de

```
ctx.globalCompositeOperation = 'color';
```

La cantidad del efecto se puede controlar con el ajuste alfa

```
// Render the image
ctx.globalCompositeOperation='source-atop';
ctx.drawImage(image, 0, 0);

// set the composite operation
ctx.globalCompositeOperation='color';
ctx.fillStyle = "white";
ctx.globalAlpha = alpha; // alpha 0 = no effect 1 = full effect
ctx.fillRect(0, 0, image.width, image.height);
```



Incrementa el contraste de color con "saturación"

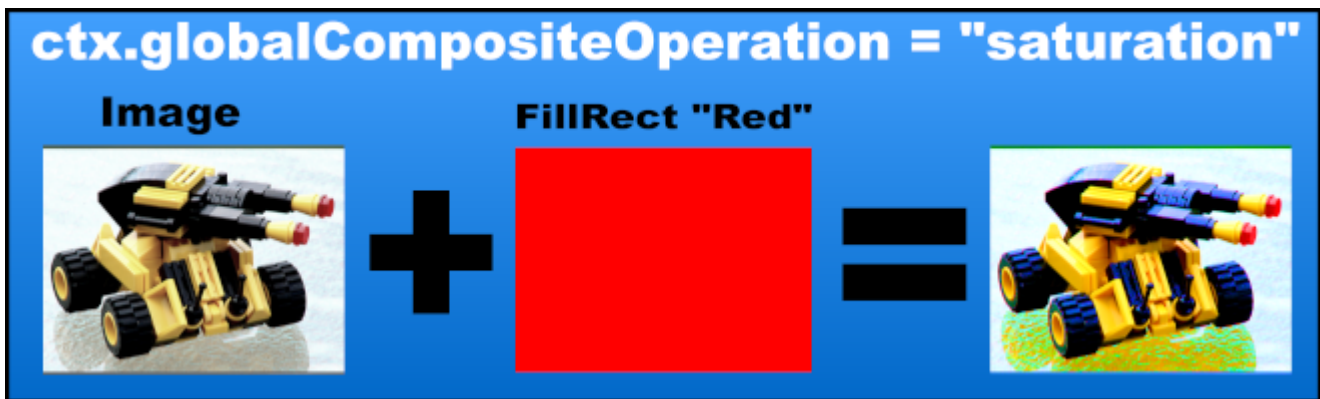
Aumenta el nivel de saturación de una imagen con

```
ctx.globalCompositeOperation = 'saturation';
```

La cantidad del efecto se puede controlar con la configuración alfa o la cantidad de saturación en la superposición de relleno

```
// Render the image
ctx.globalCompositeOperation='source-atop';
ctx.drawImage(image, 0, 0);

// set the composite operation
ctx.globalCompositeOperation = 'saturation';
ctx.fillStyle = "red";
ctx.globalAlpha = alpha; // alpha 0 = no effect 1 = full effect
ctx.fillRect(0, 0, image.width, image.height);
```



Sepia FX con "luminosidad"

Crea un color sepia FX con

```
ctx.globalCompositeOperation = 'luminosity';
```

En este caso, el color sepia se representa primero en la imagen.

La cantidad del efecto se puede controlar con la configuración alfa o la cantidad de saturación en la superposición de relleno

```
// Render the image
ctx.globalCompositeOperation='source-atop';
ctx.fillStyle = "#F80"; // the color of the sepia FX
ctx.fillRect(0, 0, image.width, image.height);

// set the composite operation
ctx.globalCompositeOperation = 'luminosity';

ctx.globalAlpha = alpha; // alpha 0 = no effect 1 = full effect
ctx.drawImage(image, 0, 0);
```



Cambia la opacidad con "globalAlpha"

```
context.globalAlpha=0.50
```

Puede cambiar la opacidad de los nuevos dibujos configurando `globalAlpha` en un valor entre 0.00 (totalmente transparente) y 1.00 (totalmente opaco).

El valor `globalAlpha` predeterminado es 1.00 (totalmente opaco).

Los dibujos existentes no se ven afectados por `globalAlpha`.

```
// draw an opaque rectangle
context.fillRect(10,10,50,50);

// change alpha to 50% -- all new drawings will have 50% opacity
context.globalAlpha=0.50;

// draw a semi-transparent rectangle
context.fillRect(100,10,50,50);
```

Lea Compositing en línea: <https://riptutorial.com/es/html5-canvas/topic/5547/compositing>

Capítulo 8: Diseño de respuesta

Examples

Creación de un lienzo de página completa sensible

Código de inicio para crear y eliminar un lienzo de página completa que responde para cambiar el tamaño de los eventos a través de javascript.

```
var canvas; // Global canvas reference
var ctx; // Global 2D context reference
// Creates a canvas
function createCanvas () {
    const canvas = document.createElement("canvas");
    canvas.style.position = "absolute"; // Set the style
    canvas.style.left = "0px"; // Position in top left
    canvas.style.top = "0px";
    canvas.style.zIndex = 1;
    document.body.appendChild(canvas); // Add to document
    return canvas;
}
// Resizes canvas. Will create a canvas if it does not exist
function sizeCanvas () {
    if (canvas === undefined) { // Check for global canvas reference
        canvas = createCanvas(); // Create a new canvas element
        ctx = canvas.getContext("2d"); // Get the 2D context
    }
    canvas.width = innerWidth; // Set the canvas resolution to fill the page
    canvas.height = innerHeight;
}
// Removes the canvas
function removeCanvas () {
    if (canvas !== undefined) { // Make sure there is something to remove
        removeEventListener("resize", sizeCanvas); // Remove resize event
        document.body.removeChild(canvas); // Remove the canvas from the DOM
        ctx = undefined; // Dereference the context
        canvas = undefined; // Dereference the canvas
    }
}

// Add the resize listener
addEventListener("resize", sizeCanvas);
// Call sizeCanvas to create and set the canvas resolution
sizeCanvas();
// ctx and canvas are now available for use.
```

Si ya no necesita el lienzo, puede eliminarlo llamando a `removeCanvas()`

[Una demostración de este ejemplo en jsfiddle](#)

Coordenadas del ratón después de cambiar el tamaño (o desplazamiento)

Las aplicaciones de lienzo a menudo dependen en gran medida de la interacción del usuario con

el mouse, pero cuando se cambia el tamaño de la ventana, las coordenadas del evento del mouse en las que se basa el lienzo probablemente se cambian porque el cambio de tamaño hace que el lienzo se desplace en una posición diferente con respecto a la ventana. Por lo tanto, el diseño responsivo requiere que la posición de desplazamiento del lienzo se vuelva a calcular cuando se cambie el tamaño de la ventana, y también se vuelva a calcular cuando la ventana se desplace.

Este código escucha los eventos de cambio de tamaño de la ventana y vuelve a calcular las compensaciones utilizadas en los controladores de eventos del mouse:

```
// variables holding the current canvas offset position
//   relative to the window
var offsetX,offsetY;

// a function to recalculate the canvas offsets
function reOffset(){
    var BB=canvas.getBoundingClientRect();
    offsetX=BB.left;
    offsetY=BB.top;
}

// listen for window resizing (and scrolling) events
//   and then recalculate the canvas offsets
window.onscroll=function(e){ reOffset(); }
window.onresize=function(e){ reOffset(); }

// example usage of the offsets in a mouse handler
function handleMouseUp(e){
    // use offsetX & offsetY to get the correct mouse position
    mouseX=parseInt(e.clientX-offsetX);
    mouseY=parseInt(e.clientY-offsetY);
    // ...
}
```

Animaciones de lienzos sensibles sin eventos de cambio de tamaño.

Los eventos de cambio de tamaño de la ventana pueden activarse en respuesta al movimiento del dispositivo de entrada del usuario. Cuando cambia el tamaño de un lienzo, se borra automáticamente y se ve obligado a volver a renderizar el contenido. Para las animaciones, haga esto en cada fotograma a través de la función de bucle principal llamada por `requestAnimationFrame` que hace todo lo posible para mantener la representación sincronizada con el hardware de la pantalla.

El problema con el evento de cambio de tamaño es que cuando se usa el mouse para cambiar el tamaño de la ventana, los eventos pueden activarse muchas veces más rápido que la tasa estándar de 60 fps del navegador. Cuando el evento de cambio de tamaño sale del lienzo, el búfer posterior se presenta al DOM desincronizado con el dispositivo de visualización, lo que puede causar distorsión y otros efectos negativos. También hay una gran cantidad de memoria y distribución innecesarias que pueden afectar aún más a la animación cuando GC se limpia un poco después.

Evento de cambio de tamaño anunciado

Una forma común de lidiar con las altas tasas de activación del evento de cambio de tamaño es rebotar el evento de cambio de tamaño.

```
// Assume canvas is in scope
addEventListener("resize", debouncedResize );

// debounce timeout handle
var debounceTimeoutHandle;

// The debounce time in ms (1/1000th second)
const DEBOUNCE_TIME = 100;

// Resize function
function debouncedResize () {
    clearTimeout(debounceTimeoutHandle); // Clears any pending debounce events

    // Schedule a canvas resize
    debounceTimeoutHandle = setTimeout(resizeCanvas, DEBOUNCE_TIME);
}

// canvas resize function
function resizeCanvas () { ... resize and redraw ... }
```

El ejemplo anterior retrasa el cambio de tamaño del lienzo hasta 100 ms después del evento de cambio de tamaño. Si en ese momento se activan más eventos de cambio de tamaño, se cancela el tiempo de espera de cambio de tamaño existente y se programa uno nuevo. Esto efectivamente consume la mayoría de los eventos de cambio de tamaño.

Todavía tiene algunos problemas, el más notable es el retraso entre cambiar el tamaño y ver el lienzo redimensionado. Reducir el tiempo de rebote mejora esto, pero el cambio de tamaño aún no está sincronizado con el dispositivo de pantalla. También tiene la representación de bucle principal de animación en un lienzo de mal ajuste.

¡Más código puede reducir los problemas! Más código también crea sus propios problemas nuevos.

Simple y el mejor tamaño

Después de haber probado muchas formas diferentes de suavizar el cambio de tamaño del lienzo, desde el complejo absurdamente, hasta simplemente ignorar el problema (¿a quién le importa de todos modos?) Me volví a un amigo de confianza.

KISS es algo de lo que la mayoría de los programadores deberían saber ((**K** eep **I** t **S** ple **S** t **S** upple) *El estúpido se refiere a mí por no haberlo pensado hace años*) y resulta que la mejor solución es la más simple de todas.

Simplemente redimensiona el lienzo desde dentro del bucle principal de animación. Permanece sincronizado con el dispositivo de visualización, no hay una representación innecesaria y la

gestión de recursos es lo mínimo posible mientras se mantiene la velocidad de fotogramas completa. Tampoco necesita agregar un evento de cambio de tamaño a la ventana o cualquier función de cambio de tamaño adicional.

Agregue el tamaño del lugar donde normalmente borraría el lienzo al verificar si el tamaño del lienzo coincide con el tamaño de la ventana. Si no lo cambiamos de tamaño.

```
// Assumes canvas element is in scope as canvas

// Standard main loop function callback from requestAnimationFrame
function mainLoop(time) {

    // Check if the canvas size matches the window size
    if (canvas.width !== innerWidth || canvas.height !== innerHeight) {
        canvas.width = innerWidth;    // resize canvas
        canvas.height = innerHeight;  // also clears the canvas
    } else {
        ctx.clearRect(0, 0, canvas.width, canvas.height); // clear if not resized
    }

    // Animation code as normal.

    requestAnimationFrame(mainLoop);
}
```

Lea Diseño de respuesta en línea: <https://riptutorial.com/es/html5-canvas/topic/5495/disenio-de-respuesta>

Capítulo 9: Gráficos y diagramas

Examples

Línea con puntas de flecha

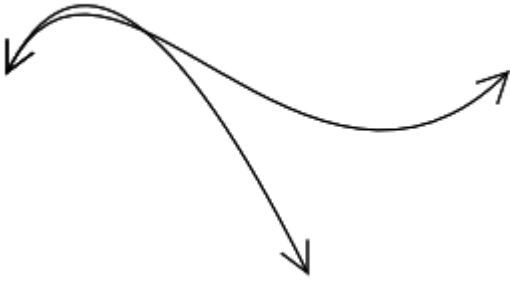


```
// Usage:
drawLineWithArrows(50,50,150,50,5,8,true,true);

// x0,y0: the line's starting point
// x1,y1: the line's ending point
// width: the distance the arrowhead perpendicularly extends away from the line
// height: the distance the arrowhead extends backward from the endpoint
// arrowStart: true/false directing to draw arrowhead at the line's starting point
// arrowEnd: true/false directing to draw arrowhead at the line's ending point

function drawLineWithArrows(x0,y0,x1,y1,aWidth,aLength,arrowStart,arrowEnd){
    var dx=x1-x0;
    var dy=y1-y0;
    var angle=Math.atan2(dy,dx);
    var length=Math.sqrt(dx*dx+dy*dy);
    //
    ctx.translate(x0,y0);
    ctx.rotate(angle);
    ctx.beginPath();
    ctx.moveTo(0,0);
    ctx.lineTo(length,0);
    if(arrowStart){
        ctx.moveTo(aLength,-aWidth);
        ctx.lineTo(0,0);
        ctx.lineTo(aLength,aWidth);
    }
    if(arrowEnd){
        ctx.moveTo(length-aLength,-aWidth);
        ctx.lineTo(length,0);
        ctx.lineTo(length-aLength,aWidth);
    }
    //
    ctx.stroke();
    ctx.setTransform(1,0,0,1,0,0);
}
```

Curva cúbica y cuadrada de Bezier con puntas de flecha



```
// Usage:
var p0={x:50,y:100};
var p1={x:100,y:0};
var p2={x:200,y:200};
var p3={x:300,y:100};

cubicCurveArrowHeads(p0, p1, p2, p3, 15, true, true);

quadraticCurveArrowHeads(p0, p1, p2, 15, true, true);

// or use defaults true for both ends with arrow heads
cubicCurveArrowHeads(p0, p1, p2, p3, 15);

quadraticCurveArrowHeads(p0, p1, p2, 15);

// draws both cubic and quadratic bezier
function bezWithArrowheads(p0, p1, p2, p3, arrowLength, hasStartArrow, hasEndArrow) {
  var x, y, norm, ex, ey;
  function pointsToNormalisedVec(p,pp){
    var len;
    norm.y = pp.x - p.x;
    norm.x = -(pp.y - p.y);
    len = Math.sqrt(norm.x * norm.x + norm.y * norm.y);
    norm.x /= len;
    norm.y /= len;
    return norm;
  }

  var arrowWidth = arrowLength / 2;
  norm = {};
  // defaults to true for both arrows if arguments not included
  hasStartArrow = hasStartArrow === undefined || hasStartArrow === null ? true :
hasStartArrow;
  hasEndArrow = hasEndArrow === undefined || hasEndArrow === null ? true : hasEndArrow;
  ctx.beginPath();
  ctx.moveTo(p0.x, p0.y);
  if (p3 === undefined) {
    ctx.quadraticCurveTo(p1.x, p1.y, p2.x, p2.y);
    ex = p2.x; // get end point
    ey = p2.y;
    norm = pointsToNormalisedVec(p1,p2);
  } else {
    ctx.bezierCurveTo(p1.x, p1.y, p2.x, p2.y, p3.x, p3.y)
    ex = p3.x; // get end point
    ey = p3.y;
  }
}
```

```

    norm = pointsToNormalisedVec(p2,p3);
}
if (hasEndArrow) {
    x = arrowWidth * norm.x + arrowLength * -norm.y;
    y = arrowWidth * norm.y + arrowLength * norm.x;
    ctx.moveTo(ex + x, ey + y);
    ctx.lineTo(ex, ey);
    x = arrowWidth * -norm.x + arrowLength * -norm.y;
    y = arrowWidth * -norm.y + arrowLength * norm.x;
    ctx.lineTo(ex + x, ey + y);
}
if (hasStartArrow) {
    norm = pointsToNormalisedVec(p0,p1);
    x = arrowWidth * norm.x - arrowLength * -norm.y;
    y = arrowWidth * norm.y - arrowLength * norm.x;
    ctx.moveTo(p0.x + x, p0.y + y);
    ctx.lineTo(p0.x, p0.y);
    x = arrowWidth * -norm.x - arrowLength * -norm.y;
    y = arrowWidth * -norm.y - arrowLength * norm.x;
    ctx.lineTo(p0.x + x, p0.y + y);
}

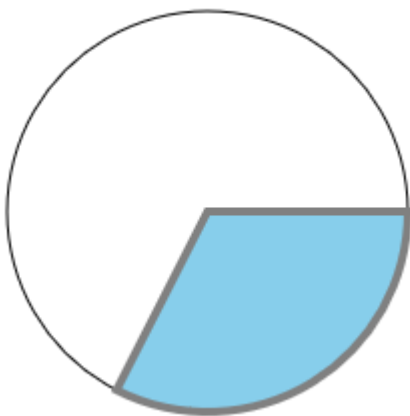
ctx.stroke();
}

function cubicCurveArrowheads(p0, p1, p2, p3, arrowLength, hasStartArrow, hasEndArrow) {
    bezWithArrowheads(p0, p1, p2, p3, arrowLength, hasStartArrow, hasEndArrow);
}
function quadraticCurveArrowheads(p0, p1, p2, arrowLength, hasStartArrow, hasEndArrow) {
    bezWithArrowheads(p0, p1, p2, undefined, arrowLength, hasStartArrow, hasEndArrow);
}
}

```

Cuña

El código solo dibuja la cuña ... círculo dibujado aquí solo para perspectiva.



```

// Usage
var wedge={
    cx:150, cy:150,
    radius:100,
    startAngle:0,
    endAngle:Math.PI*.65
}

```

```
drawWedge(wedge, 'skyblue', 'gray', 4);

function drawWedge(w, fill, stroke, strokewidth) {
    ctx.beginPath();
    ctx.moveTo(w.cx, w.cy);
    ctx.arc(w.cx, w.cy, w.radius, w.startAngle, w.endAngle);
    ctx.closePath();
    ctx.fillStyle=fill;
    ctx.fill();
    ctx.strokeStyle=stroke;
    ctx.lineWidth=strokewidth;
    ctx.stroke();
}
```

Arco con relleno y trazo

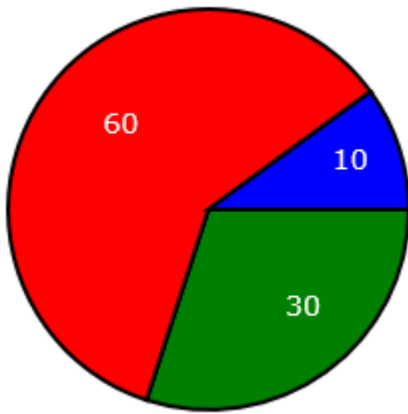


```
// Usage:
var arc={
    cx:150, cy:150,
    innerRadius:75, outerRadius:100,
    startAngle:-Math.PI/4, endAngle:Math.PI
}

drawArc(arc, 'skyblue', 'gray', 4);

function drawArc(a, fill, stroke, strokewidth) {
    ctx.beginPath();
    ctx.arc(a.cx, a.cy, a.innerRadius, a.startAngle, a.endAngle);
    ctx.arc(a.cx, a.cy, a.outerRadius, a.endAngle, a.startAngle, true);
    ctx.closePath();
    ctx.fillStyle=fill;
    ctx.strokeStyle=stroke;
    ctx.lineWidth=strokewidth;
    ctx.fill();
    ctx.stroke();
}
```

Gráfico circular con demo



```
<!doctype html>
<html>
<head>
<style>
  body{ background-color:white; }
  #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

  var canvas = document.getElementById("canvas");
  var ctx = canvas.getContext("2d");
  ctx.lineWidth = 2;
  ctx.font = '14px verdana';

  var PI2 = Math.PI * 2;
  var myColor = ["Green", "Red", "Blue"];
  var myData = [30, 60, 10];
  var cx = 150;
  var cy = 150;
  var radius = 100;

  pieChart(myData, myColor);

  function pieChart(data, colors) {
    var total = 0;
    for (var i = 0; i < data.length; i++) {
      total += data[i];
    }

    var sweeps = []
    for (var i = 0; i < data.length; i++) {
      sweeps.push(data[i] / total * PI2);
    }

    var accumAngle = 0;
    for (var i = 0; i < sweeps.length; i++) {
      drawWedge(accumAngle, accumAngle + sweeps[i], colors[i], data[i]);
      accumAngle += sweeps[i];
    }
  }

  function drawWedge(startAngle, endAngle, fill, label) {
    // draw the wedge
```

```
ctx.beginPath();
ctx.moveTo(cx, cy);
ctx.arc(cx, cy, radius, startAngle, endAngle, false);
ctx.closePath();
ctx.fillStyle = fill;
ctx.strokeStyle = 'black';
ctx.fill();
ctx.stroke();

// draw the label
var midAngle = startAngle + (endAngle - startAngle) / 2;
var labelRadius = radius * .65;
var x = cx + (labelRadius) * Math.cos(midAngle);
var y = cy + (labelRadius) * Math.sin(midAngle);
ctx.fillStyle = 'white';
ctx.fillText(label, x, y);
}

}); // end $(function(){});
</script>
</head>
<body>
  <canvas id="canvas" width=512 height=512></canvas>
</body>
</html>
```

Lea Gráficos y diagramas en línea: <https://riptutorial.com/es/html5-canvas/topic/5492/graficos-y-diagramas>

Capítulo 10: Imágenes

Examples

Recorte de imágenes utilizando lienzo.

Este ejemplo muestra una función de recorte de imagen simple que toma una imagen y recorta las coordenadas y devuelve la imagen recortada.

```
function cropImage(image, croppingCoords) {
  var cc = croppingCoords;
  var workCan = document.createElement("canvas"); // create a canvas
  workCan.width = Math.floor(cc.width); // set the canvas resolution to the cropped image
  size
  workCan.height = Math.floor(cc.height);
  var ctx = workCan.getContext("2d"); // get a 2D rendering interface
  ctx.drawImage(image, -Math.floor(cc.x), -Math.floor(cc.y)); // draw the image offset to
  place it correctly on the cropped region
  image.src = workCan.toDataURL(); // set the image source to the canvas as a data URL
  return image;
}
```

Usar

```
var image = new Image();
image.src = "image URL"; // load the image
image.onload = function () { // when loaded
  cropImage(
    this, {
      x : this.width / 4, // crop keeping the center
      y : this.height / 4,
      width : this.width / 2,
      height : this.height / 2,
    });
  document.body.appendChild(this); // Add the image to the DOM
};
```

El lienzo retenido

Al agregar contenido de fuentes fuera de su dominio o del sistema de archivos local, el lienzo está marcado como contaminado. Intentar acceder a los datos de píxeles o convertirlos a un dataURL producirá un error de seguridad.

```
vr image = new Image();
image.src = "file://myLocalImage.png";
image.onload = function(){
  ctx.drawImage(this, 0, 0);
  ctx.getImageData(0, 0, canvas.width, canvas.height); // throws a security error
}
```

Este ejemplo es solo un esbozo para atraer a alguien con una comprensión detallada detallada.

¿"Context.drawImage" no muestra la imagen en el lienzo?

Asegúrese de que su objeto de imagen esté completamente cargado antes de intentar dibujarlo en el lienzo con `context.drawImage`. De lo contrario, la imagen no se mostrará en silencio.

En JavaScript, las imágenes no se cargan de inmediato. En su lugar, las imágenes se cargan de forma asíncrona y durante el tiempo que demoran en cargar JavaScript continúa ejecutando cualquier código que siga a `image.src`. Esto significa que `context.drawImage` puede ejecutarse con una imagen vacía y, por lo tanto, no mostrará nada.

Ejemplo de asegurarse de que la imagen esté completamente cargada antes de intentar dibujarla con `.drawImage`

```
var img=new Image();
img.onload=start;
img.onerror=function(){alert(img.src+' failed');}
img.src="someImage.png";
function start(){
    // start() is called AFTER the image is fully loaded regardless
    // of start's position in the code
}
```

Ejemplo cargando múltiples imágenes antes de intentar dibujar con alguna de ellas.

Hay más cargadores de imágenes con todas las funciones, pero este ejemplo ilustra cómo hacerlo.

```
// first image
var img1=new Image();
img1.onload=start;
img1.onerror=function(){alert(img1.src+' failed to load.');};
img1.src="imageOne.png";
// second image
var img2=new Image();
img2.onload=start;
img1.onerror=function(){alert(img2.src+' failed to load.');};
img2.src="imageTwo.png";
//
var imgCount=2;
// start is called every time an image loads
function start(){
    // countdown until all images are loaded
    if(--imgCount>0){return;}
    // All the images are now successfully loaded
    // context.drawImage will successfully draw each one
    context.drawImage(img1,0,0);
    context.drawImage(img2,50,0);
}
```

Escala de imagen para ajustar o rellenar.

Escalado para adaptarse

Significa que toda la imagen será visible, pero puede haber algún espacio vacío en los lados o en

la parte superior e inferior si la imagen no tiene el mismo aspecto que el lienzo. El ejemplo muestra la imagen a escala para ajustarse. El azul en los lados se debe al hecho de que la imagen no es el mismo aspecto que el lienzo.



Escalado para llenar

Significa que la imagen se escala para que la imagen cubra todos los píxeles del lienzo. Si el aspecto de la imagen no es el mismo que el lienzo, entonces se recortarán algunas partes de la imagen. El ejemplo muestra la imagen a escala para rellenar. Observe cómo la parte superior e inferior de la imagen ya no son visibles.



Escala de ejemplo para ajustar

```
var image = new Image();
image.src = "imgURL";
image.onload = function(){
    scaleToFit(this);
}

function scaleToFit(img){
    // get the scale
    var scale = Math.min(canvas.width / img.width, canvas.height / img.height);
    // get the top left position of the image
    var x = (canvas.width / 2) - (img.width / 2) * scale;
    var y = (canvas.height / 2) - (img.height / 2) * scale;
    ctx.drawImage(img, x, y, img.width * scale, img.height * scale);
}
```

Escala de ejemplo para llenar

```
var image = new Image();
image.src = "imgURL";
image.onload = function(){
    scaleToFill(this);
}

function scaleToFill(img){
    // get the scale
    var scale = Math.max(canvas.width / img.width, canvas.height / img.height);
    // get the top left position of the image
    var x = (canvas.width / 2) - (img.width / 2) * scale;
    var y = (canvas.height / 2) - (img.height / 2) * scale;
    ctx.drawImage(img, x, y, img.width * scale, img.height * scale);
}
```

La única diferencia entre las dos funciones es obtener la escala. El ajuste usa la escala de ajuste mínima, mientras que el relleno utiliza la escala de ajuste máximo.

Lea Imágenes en línea: <https://riptutorial.com/es/html5-canvas/topic/3210/imagenes>

Capítulo 11: Los tipos de medios y el lienzo.

Observaciones

Este tema trata sobre los distintos tipos de medios y cómo se pueden utilizar con el lienzo en la interfaz 2D.

Los tipos de medios tienen categorías genéricas y específicas de formato.

Tipos de medios

- Animaciones
- Videos
- Imágenes
- Imágenes HD
- Imagen vectorial
- Imágenes animadas

Formatos de medios

- Jpg / Jpeg
- Png
- Gif
- SVG
- M-JPEG
- Webm
- Webp

Imágenes

Hay una gran variedad de formatos de imagen compatibles con los navegadores, aunque ningún navegador los admite a todos. Si tiene formatos de imagen particulares, desea utilizar los [navegadores Wiki y los formatos de imagen compatibles](#) proporciona una buena visión general.

El mejor soporte es para los 3 formatos principales, "jpeg", "png" y "gif" con todos los navegadores principales que brindan soporte.

JPEG

Las imágenes JPEG se adaptan mejor a las fotos y fotos como imágenes. No se prestan a los cuadros, diagramas y texto. Las imágenes JPEG no admiten transparencia.

Canvas puede generar imágenes JPEG a través de `canvas.toDataURL` y `canvas.toBlob` y proporciona una configuración de calidad. Como JPEG no admite la transparencia, todos los píxeles transparentes se mezclarán con el negro para la salida final JPG. La imagen resultante no será una copia perfecta del lienzo.

[JPEG en wikipedia](#)

Png

Las imágenes PNG son imágenes de la más alta calidad y también pueden incluir un canal alfa para píxeles transparentes. Los datos de la imagen están comprimidos pero no producen artefactos como las imágenes JPG.

Debido a la compresión sin pérdida y el soporte de canal alfa, los PNG se utilizan para juegos, imágenes de componentes de interfaz de usuario, cuadros, diagramas y texto. Cuando se usan para fotos como fotos, su tamaño de archivo puede ser mucho más grande que el de JPEG. .

El formato PNG también proporciona soporte de animación, aunque el soporte del navegador es limitado, y el acceso a la animación para usar en el lienzo solo se puede realizar a través de las API y bibliotecas de Javascript

El lienzo se puede usar para codificar imágenes PNG a través de `canvas.toDataURL` y `canvas.toBlob` aunque el formato de salida está limitado a RGBA comprimido de 32 bits. El PNG proporcionará una copia perfecta del pixel del lienzo.

[PNG en wikipedia](#)

GIF

Los GIF se usan para animaciones cortas, pero también se pueden usar para proporcionar gráficos, diagramas y texto de alta calidad como imágenes. Los GIF tienen un soporte de color muy limitado con un máximo de 256 colores por fotograma. Con el procesamiento de imágenes clever, las imágenes gif pueden producir resultados sorprendentemente buenos, especialmente cuando se animan. Los gifs también proporcionan transparencia, aunque esto está limitado a activado o desactivado.

AS con PNG, las animaciones GIF no son accesibles directamente para su uso en el lienzo y necesitará una API o biblioteca de Javascript para obtener acceso. GIF no se puede guardar a través del lienzo y requerirá una API o una biblioteca para hacerlo.

[GIF en wikipedia](#)

Examples

Cargando y mostrando una imagen

Para cargar una imagen y colocarla en el lienzo.

```
var image = new Image(); // see note on creating an image
image.src = "imageURL";
image.onload = function(){
    ctx.drawImage(this,0,0);
}
```

Creando una imagen

Hay varias formas de crear una imagen.

- `new Image()`
- `document.createElement("img")`
- `` Como parte del cuerpo HTML y recuperado con `document.getElementById('myImage')`

La imagen es un `HTMLImageElement`

Propiedad `image.src`

La imagen `src` puede ser cualquier URL de imagen válida o dataURL codificada. Consulte las Observaciones de este tema para obtener más información sobre los formatos de imagen y el soporte.

- `image.src = "http://my.domain.com/images/myImage.jpg"`
- `image.src = "data:image/gif;base64,R0lGODlhAQABAIAAAAUeBAAACwAAAAAAQABAAACAkQBADs="` *

* El dataURL es una imagen gif de 1 por 1 píxel que contiene negro

Observaciones sobre carga y errores.

La imagen comenzará a cargarse cuando se establezca su propiedad `src`. La carga es sincrionse pero el evento `onload` no se llamará hasta que la función o el código haya salido / devuelto.

Si obtiene una imagen de la página (por ejemplo `document.getElementById("myImage")`) y su `src` está configurado puede o no puede haber cargado. Puede verificar el estado de la imagen con `HTMLImageElement.complete` que será `true` si está completo. Esto no significa que la imagen se haya cargado, significa que tiene

- cargado
- Hubo un error
- La propiedad `src` no se ha establecido y es igual a la cadena vacía ""

Si la imagen es de una fuente no confiable y puede no ser accesible por una variedad de razones, generará un evento de error. Cuando esto suceda, la imagen estará en un estado roto. Si intenta dibujarlo en el lienzo, se mostrará el siguiente error.

```
Uncaught DOMException: Failed to execute 'drawImage' on 'CanvasRenderingContext2D': The HTMLImageElement provided is in the 'broken' state.
```

Al proporcionar el evento `image.onerror = myImgErrorHandler` , puede tomar las medidas adecuadas para evitar errores.

Dibujando una imagen svg

Para dibujar una imagen SVG vectorial, la operación no es diferente de una imagen rasterizada: Primero debe cargar su imagen SVG en un elemento `HTMLImage`, luego usar el método

drawImage() .

```
var image = new Image();
image.onload = function(){
    ctx.drawImage(this, 0,0);
}
image.src = "someFile.SVG";
```

Las imágenes SVG tienen algunas ventajas sobre las trama, ya que no perderá calidad, independientemente de la escala que dibuje en su lienzo. Pero cuidado, también puede ser un poco más lento que dibujar una imagen rasterizada.

Sin embargo, las imágenes SVG vienen con más restricciones que las imágenes rasterizadas.

- **Por razones de seguridad, no se puede cargar contenido externo desde una imagen SVG a la que se hace referencia en un elemento de imagen HTML ()**

Sin hoja de estilo externa, sin imagen externa referenciada en los elementos SVGImage (<image/>), sin filtro externo o elemento vinculado por el atributo `xlink:href` (<use xlink:href="anImage.SVG#anElement"/>) o la función `url()` método de atributo, etc.

Además, las hojas de estilo adjuntas en el documento principal no tendrán ningún efecto en el documento SVG una vez referenciado en un elemento HTMLImage.

Finalmente, no se ejecutará ningún script dentro de la imagen SVG.

Solución alternativa: deberá adjuntar todos los elementos externos dentro del propio SVG antes de hacer referencia al elemento HTMLImage. (para imágenes o fuentes, debe adjuntar una versión dataURI de sus recursos externos).

- **El elemento raíz (<svg>) debe tener sus atributos de ancho y alto establecidos en un valor absoluto.**

Si tuviera que usar la longitud relativa (por ejemplo, %), entonces el navegador no podrá saber qué es relativo. Algunos navegadores (Blink) intentarán hacer una conjetura, pero la mayoría simplemente ignorará su imagen y no dibujará nada, sin una advertencia.

- **Algunos navegadores mancharán el lienzo cuando se dibuje una imagen SVG.**

Específicamente, Internet-Explorer <Edge en cualquier caso, y Safari 9 cuando un <foreignObject> está presente en la imagen SVG.

Carga básica y reproducción de un video en el lienzo.

El lienzo se puede usar para mostrar videos de una variedad de fuentes. Este ejemplo muestra cómo cargar un video como un recurso de archivo, mostrarlo y agregar un simple clic en la pantalla para reproducir / pausar para alternar.

Esta pregunta de respuesta automática de Stackoverflow [¿Cómo muestro un video utilizando la etiqueta de lienzo HTML5](#) muestra el siguiente código de ejemplo en acción?

Solo una imagen

Un video es solo una imagen en lo que concierne al lienzo. Puedes dibujarlo como cualquier imagen. La diferencia es que el video puede reproducirse y tiene sonido.

Consigue lienzo y configuración básica.

```
// It is assumed you know how to add a canvas and correctly size it.
var canvas = document.getElementById("myCanvas"); // get the canvas from the page
var ctx = canvas.getContext("2d");
var videoContainer; // object to hold video and associated info
```

Creando y cargando el video

```
var video = document.createElement("video"); // create a video element
video.src = "urlOffVideo.webm";
// the video will now begin to load.
// As some additional info is needed we will place the video in a
// containing object for convenience
video.autoplay = false; // ensure that the video does not auto play
video.loop = true; // set the video to loop.
videoContainer = { // we will add properties as needed
  video : video,
  ready : false,
};
```

A diferencia de los elementos de imágenes, los videos no tienen que estar completamente cargados para mostrarse en el lienzo. Los videos también proporcionan una gran cantidad de eventos adicionales que se pueden usar para monitorear el estado del video.

En este caso, deseamos saber cuándo el video está listo para reproducirse. `oncanplay` significa que se ha cargado suficiente video para reproducirlo, pero puede que no haya suficiente para reproducirlo hasta el final.

```
video.oncanplay = readyToPlayVideo; // set the event to the play function that
// can be found below
```

Alternativamente, puede usar el `oncanplaythrough` que se disparará cuando se haya cargado suficiente video para que pueda reproducirse hasta el final.

```
video.oncanplaythrough = readyToPlayVideo; // set the event to the play function that
// can be found below
```

Utilice solo uno de los eventos `canPlay` no ambos.

El evento puede jugar (equivalente a la carga de imágenes)

```
function readyToPlayVideo(event){ // this is a reference to the video
  // the video may not match the canvas size so find a scale to fit
  videoContainer.scale = Math.min(
    canvas.width / this.videoWidth,
    canvas.height / this.videoHeight);
  videoContainer.ready = true;
  // the video can be played so hand it off to the display function
  requestAnimationFrame(undateCanvas);
}
```

Mostrando

El video no se reproducirá en el lienzo. Necesitas dibujarlo para cada nuevo marco. Como es difícil saber la velocidad de fotogramas exacta y cuando se producen, el mejor enfoque es mostrar el video como si se estuviera ejecutando a 60 fps. Si la velocidad de fotogramas es menor, entonces simplemente renderiza el mismo fotograma dos veces. Si la velocidad de fotogramas es mayor, no hay nada que se pueda hacer para ver los fotogramas adicionales, por lo que simplemente los ignoramos.

El elemento de video es solo un elemento de imagen y puede dibujarse como cualquier imagen, puede escalar, rotar, panoramizar el video, reflejarlo, desvanecerlo, recortarlo y mostrar solo partes, dibujarlo dos veces la segunda vez con un modo compuesto global Para añadir efectos como lighten, screen, etc.

```
function updateCanvas() {
  ctx.clearRect(0,0,canvas.width,canvas.height); // Though not always needed
                                                // you may get bad pixels from
                                                // previous videos so clear to be
                                                // safe

  // only draw if loaded and ready
  if(videoContainer !== undefined && videoContainer.ready){
    // find the top left of the video on the canvas
    var scale = videoContainer.scale;
    var vidH = videoContainer.video.videoHeight;
    var vidW = videoContainer.video.videoWidth;
    var top = canvas.height / 2 - (vidH / 2 ) * scale;
    var left = canvas.width / 2 - (vidW / 2 ) * scale;
    // now just draw the video the correct size
    ctx.drawImage(videoContainer.video, left, top, vidW * scale, vidH * scale);
    if(videoContainer.video.paused){ // if not playing show the paused screen
      drawPayIcon();
    }
  }
  // all done for display
  // request the next frame in 1/60th of a second
  requestAnimationFrame(updateCanvas);
}
```

Control de pausa de juego básico

Ahora tenemos el video cargado y mostramos todo lo que necesitamos es el control de reproducción. Lo haremos como un click toggle play en la pantalla. Cuando el video se está reproduciendo y el usuario hace clic, el video se detiene. Cuando está en pausa, el clic reanuda la reproducción. Agregaremos una función para oscurecer el video y dibujar un ícono de reproducción (triángulo)

```
function drawPayIcon(){
  ctx.fillStyle = "black"; // darken display
  ctx.globalAlpha = 0.5;
  ctx.fillRect(0,0,canvas.width,canvas.height);
  ctx.fillStyle = "#DDD"; // colour of play icon
  ctx.globalAlpha = 0.75; // partly transparent
```

```
ctx.beginPath(); // create the path for the icon
var size = (canvas.height / 2) * 0.5; // the size of the icon
ctx.moveTo(canvas.width/2 + size/2, canvas.height / 2); // start at the pointy end
ctx.lineTo(canvas.width/2 - size/2, canvas.height / 2 + size);
ctx.lineTo(canvas.width/2 - size/2, canvas.height / 2 - size);
ctx.closePath();
ctx.fill();
ctx.globalAlpha = 1; // restore alpha
}
```

Ahora el evento de pausa de juego

```
function playPauseClick(){
    if(videoContainer !== undefined && videoContainer.ready){
        if(videoContainer.video.paused){
            videoContainer.video.play();
        }else{
            videoContainer.video.pause();
        }
    }
}
// register the event
canvas.addEventListener("click",playPauseClick);
```

Resumen

Reproducir un video es muy fácil usando el lienzo, agregar efecto en tiempo real también es fácil. Sin embargo, existen algunas limitaciones en cuanto a los formatos, cómo puedes jugar y buscar. MDN HTMLMediaElement es el lugar para obtener la referencia completa al objeto de video.

Una vez que la imagen se ha dibujado en el lienzo, puede utilizar `ctx.getImageData` para acceder a los píxeles que contiene. O puede usar `canvas.toDataURL` para tomar una `canvas.toDataURL` y descargarla. (Solo si el video es de una fuente confiable y no mancha el lienzo).

Tenga en cuenta que si el video tiene sonido, si lo reproduce, también reproducirá el sonido.

Feliz videoing

Capturar lienzo y guardar como video webM

Crear un video de WebM desde marcos de lienzo y reproducir en lienzo, o cargar o descargar.

Ejemplo de captura y reproducción de lienzo.

```
name = "CanvasCapture"; // Placed into the Mux and Write Application Name fields of the WebM
header
quality = 0.7; // good quality 1 Best < 0.7 ok to poor
fps = 30; // I have tried all sorts of frame rates and all seem to work
// Do some test to workout what your machine can handle as there
// is a lot of variation between machines.
var video = new Groover.Video(fps,quality,name)
```

```

function capture(){
  if(video.timecode < 5000){ // 5 seconds
    setTimeout(capture,video.frameDelay);
  }else{
    var videoElement = document.createElement("video");
    videoElement.src = URL.createObjectURL(video.toBlob());
    document.body.appendChild(videoElement);
    video = undefined; // DeReference as it is memory hungry.
    return;
  }
  // first frame sets the video size
  video.addFrame(canvas); // Add current canvas frame
}
capture(); // start capture

```

En lugar de hacer un gran esfuerzo solo para ser rechazado, esta es una inserción rápida para ver si es aceptable. Dará los detalles completos si es aceptado. También incluye opciones de captura adicionales para obtener mejores tasas de captura de HD (eliminada de esta versión, puede capturar HD 1080 a 50 fps en buenas máquinas).

Esto fue inspirado por [Wammy](#) pero es una completa reescritura con codificación a medida que avanza la metodología, reduciendo en gran medida la memoria requerida durante la captura. Puede capturar más de 30 segundos mejores datos, manejando algoritmos.

Los cuadros de notas están codificados en imágenes webP. Solo Chrome soporta codificación webP canvas. Para otros navegadores (Firefox y Edge), deberá utilizar un codificador webP de terceros, como [Libwebp](#). La codificación de imágenes WebP a través de Javascript es lenta. (Incluirá la adición de soporte de imágenes webp sin formato si es aceptado).

El codificador webM inspirado por [Whammy: un Javascript en tiempo real WebM](#)

```

var Groover = (function(){
  // ensure webp is supported
  function canEncode(){
    var canvas = document.createElement("canvas");
    canvas.width = 8;
    canvas.height = 8;
    return canvas.toDataURL("image/webp",0.1).indexOf("image/webp") > -1;
  }
  if(!canEncode()){
    return undefined;
  }
  var webmData = null;
  var clusterTimecode = 0;
  var clusterCounter = 0;
  var CLUSTER_MAX_DURATION = 30000;
  var frameNumber = 0;
  var width;
  var height;
  var frameDelay;
  var quality;
  var name;
  const videoMimeType = "video/webm"; // the only one.
  const frameMimeType = 'image/webp'; // can be no other
  const S = String.fromCharCode;

```

```

const dataTypes = {
  object : function(data){ return toBlob(data);},
  number : function(data){ return stream.num(data);},
  string : function(data){ return stream.str(data);},
  array  : function(data){ return data;},
  double2Str : function(num){
    var c = new Uint8Array((new Float64Array([num])).buffer);
    return S(c[7]) + S(c[6]) + S(c[5]) + S(c[4]) + S(c[3]) + S(c[2]) + S(c[1]) +
S(c[0]);
  }
};

const stream = {
  num : function(num){ // writes int
    var parts = [];
    while(num > 0){ parts.push(num & 0xff); num = num >> 8; }
    return new Uint8Array(parts.reverse());
  },
  str : function(str){ // writes string
    var i, len, arr;
    len = str.length;
    arr = new Uint8Array(len);
    for(i = 0; i < len; i++){arr[i] = str.charCodeAt(i);}
    return arr;
  },
  compInt : function(num){ // could not find full details so bit of a guess
    if(num < 128){ // number is prefixed with a bit (1000 is on byte 0100 two,
0010 three and so on)
      num += 0x80;
      return new Uint8Array([num]);
    }else
    if(num < 0x4000){
      num += 0x4000;
      return new Uint8Array([num>>8, num])
    }else
    if(num < 0x200000){
      num += 0x200000;
      return new Uint8Array([num>>16, num>>8, num])
    }else
    if(num < 0x10000000){
      num += 0x10000000;
      return new Uint8Array([num>>24, num>>16, num>>8, num])
    }
  }
}

const ids = { // header names and values
  videoData      : 0x1a45dfa3,
  Version        : 0x4286,
  ReadVersion    : 0x42f7,
  MaxIDLength    : 0x42f2,
  MaxSizeLength  : 0x42f3,
  DocType        : 0x4282,
  DocTypeVersion : 0x4287,
  DocTypeReadVersion : 0x4285,
  Segment        : 0x18538067,
  Info           : 0x1549a966,
  TimecodeScale  : 0x2ad7b1,
  MuxingApp      : 0x4d80,
  WritingApp     : 0x5741,
  Duration       : 0x4489,
  Tracks         : 0x1654ae6b,

```

```

TrackEntry      : 0xae,
TrackNumber     : 0xd7,
TrackUID        : 0x63c5,
FlagLacing      : 0x9c,
Language        : 0x22b59c,
CodecID         : 0x86,
CodecName       : 0x258688,
TrackType       : 0x83,
Video           : 0xe0,
PixelWidth      : 0xb0,
PixelHeight     : 0xba,
Cluster         : 0x1f43b675,
Timecode        : 0xe7,
Frame           : 0xa3,
Keyframe        : 0x9d012a,
FrameBlock      : 0x81,
};
const keyframeD64Header = '\x9d\x01\x2a'; //VP8 keyframe header 0x9d012a
const videoDataPos = 1; // data pos of frame data header
const defaultDelay = dataTypes.double2Str(1000/25);
const header = [ // structure of webM header/chunks what ever they are called.
  ids.videoData, [
    ids.Version, 1,
    ids.ReadVersion, 1,
    ids.MaxIDLength, 4,
    ids.MaxSizeLength, 8,
    ids.DocType, 'webm',
    ids.DocTypeVersion, 2,
    ids.DocTypeReadVersion, 2
  ],
  ids.Segment, [
    ids.Info, [
      ids.TimecodeScale, 1000000,
      ids.MuxingApp, 'Groover',
      ids.WritingApp, 'Groover',
      ids.Duration, 0
    ],
    ids.Tracks, [
      ids.TrackEntry, [
        ids.TrackNumber, 1,
        ids.TrackUID, 1,
        ids.FlagLacing, 0, // always 0
        ids.Language, 'und', // undefined I think this means
        ids.CodecID, 'V_VP8', // These I think must not change
        ids.CodecName, 'VP8', // These I think must not change
        ids.TrackType, 1,
        ids.Video, [
          ids.PixelWidth, 0,
          ids.PixelHeight, 0
        ]
      ]
    ]
  ]
];
function getHeader(){
  header[3][2][3] = name;
  header[3][2][5] = name;
  header[3][2][7] = dataTypes.double2Str(frameDelay);
  header[3][3][1][15][1] = width;
  header[3][3][1][15][3] = height;
  function create(dat){

```

```

    var i,kv,data;
    data = [];
    for(i = 0; i < dat.length; i += 2){
        kv = {i : dat[i]};
        if(Array.isArray(dat[i + 1])){
            kv.d = create(dat[i + 1]);
        }else{
            kv.d = dat[i + 1];
        }
        data.push(kv);
    }
    return data;
}
return create(header);
}
function addCluster(){
    webmData[videoDataPos].d.push({ i: ids.Cluster,d: [{ i: ids.Timecode, d:
Math.round(clusterTimecode)}}]); // Fixed bug with Round
    clusterCounter = 0;
}
function addFrame(frame){
    var VP8, kfS,riff;
    riff = getWebPChunks(atob(frame.toDataURL(frameMimeType, quality).slice(23)));
    VP8 = riff.RIFF[0].WEBP[0];
    kfS = VP8.indexOf(keyframeD64Header) + 3;
    frame = {
        width: ((VP8.charCodeAt(kfS + 1) << 8) | VP8.charCodeAt(kfS)) & 0x3FFF,
        height: ((VP8.charCodeAt(kfS + 3) << 8) | VP8.charCodeAt(kfS + 2)) & 0x3FFF,
        data: VP8,
        riff: riff
    };
    if(clusterCounter > CLUSTER_MAX_DURATION){
        addCluster();
    }
    webmData[videoDataPos].d[webmData[videoDataPos].d.length-1].d.push({
        i: ids.Frame,
        d: S(ids.FrameBlock) + S(Math.round(clusterCounter) >> 8) + S(
Math.round(clusterCounter) & 0xff) + S(128) + frame.data.slice(4),
    });
    clusterCounter += frameDelay;
    clusterTimecode += frameDelay;
    webmData[videoDataPos].d[0].d[3].d = dataTypes.double2Str(clusterTimecode);
}
function startEncoding(){
    frameNumber = clusterCounter = clusterTimecode = 0;
    webmData = getHeader();
    addCluster();
}
function toBlob(vidData){
    var data,i,vData, len;
    vData = [];
    for(i = 0; i < vidData.length; i++){
        data = dataTypes[typeof vidData[i].d](vidData[i].d);
        len = data.size || data.byteLength || data.length;
        vData.push(stream.num(vidData[i].i));
        vData.push(stream.compInt(len));
        vData.push(data)
    }
    return new Blob(vData, {type: videoMimeType});
}
function getWebPChunks(str){

```

```

var offset, chunks, id, len, data;
offset = 0;
chunks = {};
while (offset < str.length) {
  id = str.substr(offset, 4);
  // value will have top bit on (bit 32) so not simply a bitwise operation
  // Warning little endian (Will not work on big endian systems)
  len = new Uint32Array(
    new Uint8Array([
      str.charCodeAt(offset + 7),
      str.charCodeAt(offset + 6),
      str.charCodeAt(offset + 5),
      str.charCodeAt(offset + 4)
    ]).buffer)[0];
  id = str.substr(offset, 4);
  chunks[id] = chunks[id] === undefined ? [] : chunks[id];
  if (id === 'RIFF' || id === 'LIST') {
    chunks[id].push(getWebPChunks(str.substr(offset + 8, len)));
    offset += 8 + len;
  } else if (id === 'WEBP') {
    chunks[id].push(str.substr(offset + 8));
    break;
  } else {
    chunks[id].push(str.substr(offset + 4));
    break;
  }
}
return chunks;
}
function Encoder(fps, _quality = 0.8, _name = "Groover"){
  this.fps = fps;
  this.quality = quality = _quality;
  this.frameDelay = frameDelay = 1000 / fps;
  this.frame = 0;
  this.width = width = null;
  this.timecode = 0;
  this.name = name = _name;
}
Encoder.prototype = {
  addFrame : function(frame){
    if('canvas' in frame){
      frame = frame.canvas;
    }
    if(width === null){
      this.width = width = frame.width,
      this.height = height = frame.height
      startEncoding();
    }else
    if(width !== frame.width || height !== frame.height){
      throw RangeError("Frame size error. Frames must be the same size.");
    }
    addFrame(frame);
    this.frame += 1;
    this.timecode = clusterTimecode;
  },
  toBlob : function(){
    return toBlob(webmData);
  }
}
return {
  Video: Encoder,

```



```
}  
}) ()
```

Lea Los tipos de medios y el lienzo. en línea: <https://riptutorial.com/es/html5-canvas/topic/3689/los-tipos-de-medios-y-el-lienzo->

Capítulo 12: Manipulación de píxeles con "getImageData" y "putImageData"

Examples

Introducción a "context.getImageData"

Html5 Canvas le brinda la posibilidad de obtener y cambiar el color de cualquier píxel en el lienzo.

Puedes usar la manipulación de píxeles de Canvas para:

- Cree un selector de color para una imagen o seleccione un color en una rueda de color.
- Cree filtros de imagen complejos, como desenfoque y detección de bordes.
- Vuelva a colorear cualquier parte de una imagen a nivel de píxel (si usa HSL, puede incluso cambiar el color de una imagen mientras conserva la Iluminación y la Saturación importantes para que el resultado no se vea como si alguien hubiera pintado la imagen).
Nota: Canvas ahora tiene Blend Compositing que también puede recolorar una imagen en algunos casos.
- "Knockout" el fondo alrededor de una persona / elemento en una imagen,
- Cree una herramienta de cubeta de pintura para detectar y rellenar parte de una imagen (p. Ej., Cambie el color de un pétalo de flor con el clic del usuario de verde a amarillo).
- Examine una imagen para el contenido (por ejemplo, reconocimiento facial).

Problemas comunes:

- Por razones de seguridad, `getImageData` está deshabilitado si ha dibujado una imagen que se origina en un dominio diferente al de la propia página web.
- `getImageData` es un método relativamente caro porque crea una gran matriz de datos de píxeles y porque no utiliza la GPU para ayudar en sus esfuerzos. Nota: Canvas ahora tiene una composición mixta que puede realizar la misma manipulación de píxeles que hace `getImageData`.
- Para imágenes `.png`, es posible que `getImageData` no informe exactamente los mismos colores que en el archivo `.png` original porque el navegador puede realizar correcciones gamma y multiplicación previa alfa cuando se dibujan imágenes en el lienzo.

Obteniendo colores de pixel

Use `getImageData` para obtener los colores de píxeles para todo o parte del contenido de su lienzo.

El método `getImageData` devuelve un objeto `ImageData`

El objeto `ImageData` tiene una propiedad `.data` que contiene la información de color de píxel.

La propiedad de `data` es un `Uint8ClampedArray` contiene los datos de color rojo, verde, azul y alfa (opacidad) para todos los píxeles solicitados.

```
// determine which pixels to fetch (this fetches all pixels on the canvas)
var x=0;
var y=0;
var width=canvas.width;
var height=canvas.height;

// Fetch the imageData object
var imageData = context.getImageData(x,y,width,height);

// Pull the pixel color data array from the imageData object
var pixelDataArray = imageData.data;
```

Puede obtener la posición de cualquier píxel [x, y] dentro de `data` matriz de `data` esta manera:

```
// the data[] array position for pixel [x,y]
var n = y * canvas.width + x;
```

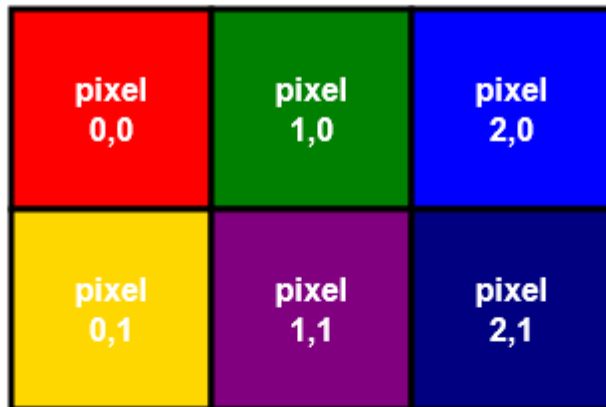
Y luego puedes obtener los valores de rojo, verde, azul y alfa de ese píxel de esta manera:

```
// the RGBA info for pixel [x,y]
var red=data[n];
var green=data[n+1];
var blue=data[n+2];
var alpha=data[n+3];
```

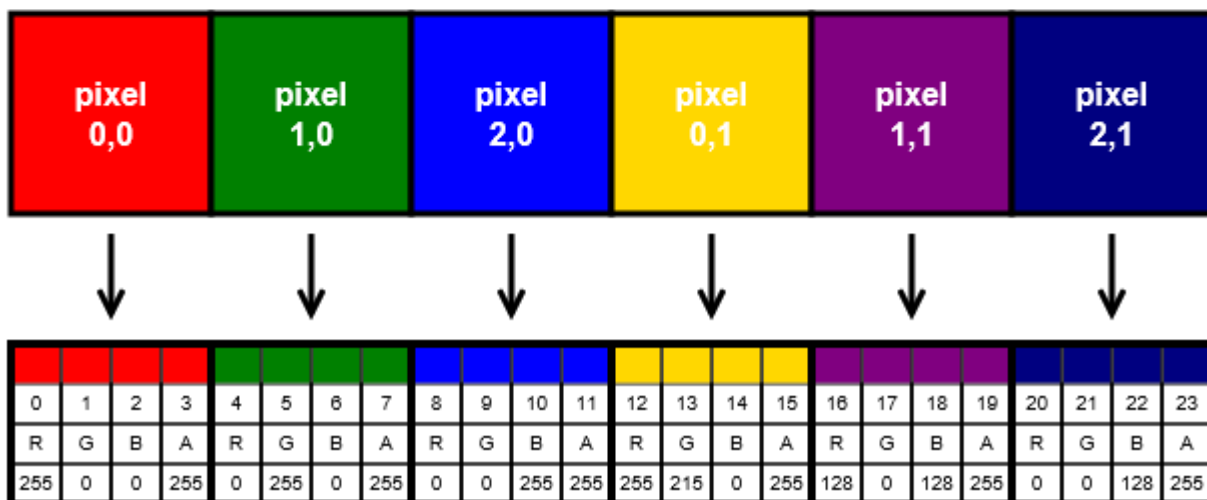
Una ilustración que muestra cómo se estructura la matriz de datos de píxeles.

`context.getImageData` se ilustra a continuación para un pequeño lienzo de 2x3 píxeles:

2x3 pixel canvas



Pixels are arranged sequentially by row
 Each pixel gets 4 array elements
 (Red, Blue, Green & Alpha)



Lea Manipulación de píxeles con "getImageData" y "putImageData" en línea:

<https://riptutorial.com/es/html5-canvas/topic/5573/manipulacion-de-pixeles-con--getimagedata--y--putimagedata->

Capítulo 13: Navegando por un sendero

Examples

Encontrando puntos a lo largo de una curva Bezier cúbica.

Este ejemplo encuentra una matriz de puntos aproximadamente espaciados a lo largo de una curva Bezier cúbica.

Descompone los segmentos de ruta creados con `context.bezierCurveTo` en puntos a lo largo de esa curva.

```
// Return: an array of approximately evenly spaced points along a cubic Bezier curve
//
// Attribution: Stackoverflow's @Blindman67
// Cite: http://stackoverflow.com/questions/36637211/drawing-a-curved-line-in-css-or-canvas-
and-moving-circle-along-it/36827074#36827074
// As modified from the above citation
//
// ptCount: sample this many points at interval along the curve
// pxTolerance: approximate spacing allowed between points
// Ax,Ay,Bx,By,Cx,Cy,Dx,Dy: control points defining the curve
//
function plotCBez (ptCount,pxTolerance,Ax,Ay,Bx,By,Cx,Cy,Dx,Dy) {
  var deltaBAx=Bx-Ax;
  var deltaCBx=Cx-Bx;
  var deltaDCx=Dx-Cx;
  var deltaBAy=By-Ay;
  var deltaCBy=Cy-By;
  var deltaDCy=Dy-Cy;
  var ax,ay,bx,by;
  var lastX=-10000;
  var lastY=-10000;
  var pts=[{x:Ax,y:Ay}];
  for(var i=1;i<ptCount;i++){
    var t=i/ptCount;
    ax=Ax+deltaBAx*t;
    bx=Bx+deltaCBx*t;
    cx=Cx+deltaDCx*t;
    ax+=(bx-ax)*t;
    bx+=(cx-bx)*t;
    //
    ay=Ay+deltaBAy*t;
    by=By+deltaCBy*t;
    cy=Cy+deltaDCy*t;
    ay+=(by-ay)*t;
    by+=(cy-by)*t;
    var x=ax+(bx-ax)*t;
    var y=ay+(by-ay)*t;
    var dx=x-lastX;
    var dy=y-lastY;
    if(dx*dx+dy*dy>pxTolerance){
      pts.push({x:x,y:y});
      lastX=x;
      lastY=y;
    }
  }
}
```

```

    }
  }
  pts.push({x:Dx,y:Dy});
  return(pts);
}

```

Encontrar puntos a lo largo de una curva cuadrática.

Este ejemplo encuentra una matriz de puntos aproximadamente espaciados a lo largo de una curva cuadrática.

Descompone los segmentos de ruta creados con `context.quadraticCurveTo` en puntos a lo largo de esa curva.

```

// Return: an array of approximately evenly spaced points along a Quadratic curve
//
// Attribution: Stackoverflow's @Blindman67
// Cite: http://stackoverflow.com/questions/36637211/drawing-a-curved-line-in-css-or-canvas-
and-moving-circle-along-it/36827074#36827074
// As modified from the above citation
//
// ptCount: sample this many points at interval along the curve
// pxTolerance: approximate spacing allowed between points
// Ax,Ay,Bx,By,Cx,Cy: control points defining the curve
//
function plotQBez(ptCount,pxTolerance,Ax,Ay,Bx,By,Cx,Cy) {
  var deltaBAx=Bx-Ax;
  var deltaCBx=Cx-Bx;
  var deltaBAy=By-Ay;
  var deltaCBy=Cy-By;
  var ax,ay;
  var lastX=-10000;
  var lastY=-10000;
  var pts=[{x:Ax,y:Ay}];
  for(var i=1;i<ptCount;i++){
    var t=i/ptCount;
    ax=Ax+deltaBAx*t;
    ay=Ay+deltaBAy*t;
    var x=ax+((Bx+deltaCBx*t)-ax)*t;
    var y=ay+((By+deltaCBy*t)-ay)*t;
    var dx=x-lastX;
    var dy=y-lastY;
    if(dx*dx+dy*dy>pxTolerance){
      pts.push({x:x,y:y});
      lastX=x;
      lastY=y;
    }
  }
  pts.push({x:Cx,y:Cy});
  return(pts);
}

```

Encontrar puntos a lo largo de una línea

Este ejemplo encuentra una matriz de puntos aproximadamente espaciados a lo largo de una línea.

Descompone los segmentos de ruta creados con `context.lineTo` en puntos a lo largo de esa línea.

```
// Return: an array of approximately evenly spaced points along a line
//
// pxTolerance: approximate spacing allowed between points
// Ax,Ay,Bx,By: end points defining the line
//
function plotLine(pxTolerance,Ax,Ay,Bx,By){
    var dx=Bx-Ax;
    var dy=By-Ay;
    var ptCount=parseInt(Math.sqrt(dx*dx+dy*dy))*3;
    var lastX=-10000;
    var lastY=-10000;
    var pts=[{x:Ax,y:Ay}];
    for(var i=1;i<=ptCount;i++){
        var t=i/ptCount;
        var x=Ax+dx*t;
        var y=Ay+dy*t;
        var dx1=x-lastX;
        var dy1=y-lastY;
        if(dx1*dx1+dy1*dy1>pxTolerance){
            pts.push({x:x,y:y});
            lastX=x;
            lastY=y;
        }
    }
    pts.push({x:Bx,y:By});
    return(pts);
}
```

Encontrar puntos a lo largo de todo un Sendero que contiene curvas y líneas.

Este ejemplo encuentra una matriz de puntos aproximadamente espaciados a lo largo de un Sendero completo.

Descompone todos los segmentos de ruta creados con `context.lineTo` , `context.quadraticCurveTo` y / o `context.bezierCurveTo` en puntos a lo largo de esa ruta.

Uso

```
// Path related variables
var A={x:50,y:100};
var B={x:125,y:25};
var BB={x:150,y:15};
var BB2={x:150,y:185};
var C={x:175,y:200};
var D={x:300,y:150};
var n=1000;
var tolerance=1.5;
var pts;

// canvas related variables
var canvas=document.createElement("canvas");
var ctx=canvas.getContext("2d");
document.body.appendChild(canvas);
canvas.width=378;
canvas.height=256;
```

```

// Tell the Context to plot waypoint in addition to
// drawing the path
plotPathCommands(ctx,n,tolerance);

// Path drawing commands
ctx.beginPath();
ctx.moveTo(A.x,A.y);
ctx.bezierCurveTo(B.x,B.y,C.x,C.y,D.x,D.y);
ctx.quadraticCurveTo(BB.x,BB.y,A.x,A.y);
ctx.lineTo(D.x,D.y);
ctx.strokeStyle='gray';
ctx.stroke();

// Tell the Context to stop plotting waypoints
ctx.stopPlottingPathCommands();

// Demo: Incrementally draw the path using the plotted points
ptsToRects(ctx.getPathPoints());
function ptsToRects(pts){
  ctx.fillStyle='red';
  var i=0;
  requestAnimationFrame(animate);
  function animate(){
    ctx.fillRect(pts[i].x-0.50,pts[i].y-0.50,tolerance,tolerance);
    i++;
    if(i<pts.length){ requestAnimationFrame(animate); }
  }
}
}

```

Un complemento que calcula automáticamente los puntos a lo largo del camino.

Este código modifica los comandos de dibujo de Canvas Context para que los comandos no solo dibujen la línea o curva, sino que también creen una matriz de puntos a lo largo de toda la ruta:

- `startPath`,
- `mover a`,
- `lineTo`,
- `cuadrática`
- `bezierCurveTo`.

¡Nota IMPORTANTE!

Este código modifica las funciones de dibujo reales del Contexto, por lo que cuando `stopPlottingPathCommands` trazar puntos a lo largo de la ruta, debe llamar a los `stopPlottingPathCommands` para devolver las funciones de dibujo del Contexto a su estado no modificado.

El propósito de este Contexto modificado es permitirle "insertar" el cálculo de la matriz de puntos en su código existente sin tener que modificar sus comandos de dibujo de Ruta existentes. Pero, no necesita usar este Contexto modificado, puede llamar por separado las funciones individuales que descomponen una línea, una curva cuadrática y una curva Bezier cúbica y luego concatenar manualmente esas matrices de puntos individuales en una única matriz de puntos para todo el camino

Obtiene una copia de la matriz de puntos resultante utilizando la función `getPathPoints` provista.

Si dibuja múltiples Rutas con el Contexto modificado, la matriz de puntos contendrá un único conjunto concatenado de puntos para todas las múltiples Rutas dibujadas.

Si, en cambio, desea obtener matrices de puntos separadas, puede obtener la matriz actual con `getPathPoints` y luego borrar esos puntos de la matriz con la función `clearPathPoints` provista.

```
// Modify the Canvas' Context to calculate a set of approximately
//     evenly spaced waypoints as it draws path(s).
function plotPathCommands(ctx, sampleCount, pointSpacing) {
    ctx.mySampleCount=sampleCount;
    ctx.myPointSpacing=pointSpacing;
    ctx.myTolerance=pointSpacing*pointSpacing;
    ctx.myBeginPath=ctx.beginPath;
    ctx.myMoveTo=ctx.moveTo;
    ctx.myLineTo=ctx.lineTo;
    ctx.myQuadraticCurveTo=ctx.quadraticCurveTo;
    ctx.myBezierCurveTo=ctx.bezierCurveTo;
    // don't use myPathPoints[] directly -- use "ctx.getPathPoints"
    ctx.myPathPoints=[];
    ctx.beginPath=function() {
        this.myLastX=0;
        this.myLastY=0;
        this.myBeginPath();
    }
    ctx.moveTo=function(x, y) {
        this.myLastX=x;
        this.myLastY=y;
        this.myMoveTo(x, y);
    }
    ctx.lineTo=function(x, y) {
        var pts=plotLine(this.myTolerance, this.myLastX, this.myLastY, x, y);
        Array.prototype.push.apply(this.myPathPoints, pts);
        this.myLastX=x;
        this.myLastY=y;
        this.myLineTo(x, y);
    }
    ctx.quadraticCurveTo=function(x0, y0, x1, y1) {
        var
pts=plotQBez(this.mySampleCount, this.myTolerance, this.myLastX, this.myLastY, x0, y0, x1, y1);
        Array.prototype.push.apply(this.myPathPoints, pts);
        this.myLastX=x1;
        this.myLastY=y1;
        this.myQuadraticCurveTo(x0, y0, x1, y1);
    }
    ctx.bezierCurveTo=function(x0, y0, x1, y1, x2, y2) {
        var
pts=plotCBez(this.mySampleCount, this.myTolerance, this.myLastX, this.myLastY, x0, y0, x1, y1, x2, y2);
        Array.prototype.push.apply(this.myPathPoints, pts);
        this.myLastX=x2;
        this.myLastY=y2;
        this.myBezierCurveTo(x0, y0, x1, y1, x2, y2);
    }
    ctx.getPathPoints=function() {
        return(this.myPathPoints.slice());
    }
    ctx.clearPathPoints=function() {
        this.myPathPoints.length=0;
    }
}
```

```

    }
    ctx.stopPlottingPathCommands=function() {
        if(!this.myBeginPath){return;}
        this.beginPath=this.myBeginPath;
        this.moveTo=this.myMoveTo;
        this.lineTo=this.myLineTo;
        this.quadraticCurveTo=this.myQuadraticCurveTo;
        this.bezierCurveTo=this.myBezierCurveTo;
        this.myBeginPath=undefined;
    }
}

```

Una demo completa:

```

// Path related variables
var A={x:50,y:100};
var B={x:125,y:25};
var BB={x:150,y:15};
var BB2={x:150,y:185};
var C={x:175,y:200};
var D={x:300,y:150};
var n=1000;
var tolerance=1.5;
var pts;

// canvas related variables
var canvas=document.createElement("canvas");
var ctx=canvas.getContext("2d");
document.body.appendChild(canvas);
canvas.width=378;
canvas.height=256;

// Tell the Context to plot waypoint in addition to
// drawing the path
plotPathCommands(ctx,n,tolerance);

// Path drawing commands
ctx.beginPath();
ctx.moveTo(A.x,A.y);
ctx.bezierCurveTo(B.x,B.y,C.x,C.y,D.x,D.y);
ctx.quadraticCurveTo(BB.x, BB.y, A.x, A.y);
ctx.lineTo(D.x,D.y);
ctx.strokeStyle='gray';
ctx.stroke();

// Tell the Context to stop plotting waypoints
ctx.stopPlottingPathCommands();

// Incrementally draw the path using the plotted points
ptsToRects(ctx.getPathPoints());
function ptsToRects(pts) {
    ctx.fillStyle='red';
    var i=0;
    requestAnimationFrame(animate);
    function animate() {
        ctx.fillRect(pts[i].x-0.50,pts[i].y-0.50,tolerance,tolerance);
        i++;
        if(i<pts.length){ requestAnimationFrame(animate); }
    }
}

```

```

////////////////////////////////////
// A Plug-in
////////////////////////////////////

// Modify the Canvas' Context to calculate a set of approximately
//     evenly spaced waypoints as it draws path(s).
function plotPathCommands(ctx, sampleCount, pointSpacing) {
    ctx.mySampleCount=sampleCount;
    ctx.myPointSpacing=pointSpacing;
    ctx.myTolerance=pointSpacing*pointSpacing;
    ctx.myBeginPath=ctx.beginPath;
    ctx.myMoveTo=ctx.moveTo;
    ctx.myLineTo=ctx.lineTo;
    ctx.myQuadraticCurveTo=ctx.quadraticCurveTo;
    ctx.myBezierCurveTo=ctx.bezierCurveTo;
    // don't use myPathPoints[] directly -- use "ctx.getPathPoints"
    ctx.myPathPoints=[];
    ctx.beginPath=function() {
        this.myLastX=0;
        this.myLastY=0;
        this.myBeginPath();
    }
    ctx.moveTo=function(x, y) {
        this.myLastX=x;
        this.myLastY=y;
        this.myMoveTo(x, y);
    }
    ctx.lineTo=function(x, y) {
        var pts=plotLine(this.myTolerance, this.myLastX, this.myLastY, x, y);
        Array.prototype.push.apply(this.myPathPoints, pts);
        this.myLastX=x;
        this.myLastY=y;
        this.myLineTo(x, y);
    }
    ctx.quadraticCurveTo=function(x0, y0, x1, y1) {
        var
pts=plotQBez(this.mySampleCount, this.myTolerance, this.myLastX, this.myLastY, x0, y0, x1, y1);
        Array.prototype.push.apply(this.myPathPoints, pts);
        this.myLastX=x1;
        this.myLastY=y1;
        this.myQuadraticCurveTo(x0, y0, x1, y1);
    }
    ctx.bezierCurveTo=function(x0, y0, x1, y1, x2, y2) {
        var
pts=plotCBez(this.mySampleCount, this.myTolerance, this.myLastX, this.myLastY, x0, y0, x1, y1, x2, y2);
        Array.prototype.push.apply(this.myPathPoints, pts);
        this.myLastX=x2;
        this.myLastY=y2;
        this.myBezierCurveTo(x0, y0, x1, y1, x2, y2);
    }
    ctx.getPathPoints=function() {
        return(this.myPathPoints.slice());
    }
    ctx.clearPathPoints=function() {
        this.myPathPoints.length=0;
    }
    ctx.stopPlottingPathCommands=function() {
        if(!this.myBeginPath){return;}
        this.beginPath=this.myBeginPath;
    }
}

```

```

    this.moveTo=this.myMoveTo;
    this.lineTo=this.myLineTo;
    this.quadraticCurveTo=this.myQuadraticCurveTo;
    this.bezierCurveTo=this.myBezierCurveTo;
    this.myBeginPath=undefined;
  }
}

////////////////////////////////////
// Helper functions
////////////////////////////////////

// Return: a set of approximately evenly spaced points along a cubic Bezier curve
//
// Attribution: Stackoverflow's @Blindman67
// Cite: http://stackoverflow.com/questions/36637211/drawing-a-curved-line-in-css-or-canvas-
and-moving-circle-along-it/36827074#36827074
// As modified from the above citation
//
// ptCount: sample this many points at interval along the curve
// pxTolerance: approximate spacing allowed between points
// Ax,Ay,Bx,By,Cx,Cy,Dx,Dy: control points defining the curve
//
function plotCBez (ptCount,pxTolerance,Ax,Ay,Bx,By,Cx,Cy,Dx,Dy) {
  var deltaBAx=Bx-Ax;
  var deltaCBx=Cx-Bx;
  var deltaDCx=Dx-Cx;
  var deltaBAy=By-Ay;
  var deltaCBy=Cy-By;
  var deltaDCy=Dy-Cy;
  var ax,ay,bx,by;
  var lastX=-10000;
  var lastY=-10000;
  var pts=[{x:Ax,y:Ay}];
  for(var i=1;i<ptCount;i++){
    var t=i/ptCount;
    ax=Ax+deltaBAx*t;
    bx=Bx+deltaCBx*t;
    cx=Cx+deltaDCx*t;
    ax+=(bx-ax)*t;
    bx+=(cx-bx)*t;
    //
    ay=Ay+deltaBAy*t;
    by=By+deltaCBy*t;
    cy=Cy+deltaDCy*t;
    ay+=(by-ay)*t;
    by+=(cy-by)*t;
    var x=ax+(bx-ax)*t;
    var y=ay+(by-ay)*t;
    var dx=x-lastX;
    var dy=y-lastY;
    if(dx*dx+dy*dy>pxTolerance){
      pts.push({x:x,y:y});
      lastX=x;
      lastY=y;
    }
  }
  pts.push({x:Dx,y:Dy});
  return(pts);
}

```

```

// Return: an array of approximately evenly spaced points along a Quadratic curve
//
// Attribution: Stackoverflow's @Blindman67
// Cite: http://stackoverflow.com/questions/36637211/drawing-a-curved-line-in-css-or-canvas-
and-moving-circle-along-it/36827074#36827074
// As modified from the above citation
//
// ptCount: sample this many points at interval along the curve
// pxTolerance: approximate spacing allowed between points
// Ax,Ay,Bx,By,Cx,Cy: control points defining the curve
//
function plotQBez (ptCount,pxTolerance,Ax,Ay,Bx,By,Cx,Cy) {
    var deltaBAx=Bx-Ax;
    var deltaCBx=Cx-Bx;
    var deltaBAy=By-Ay;
    var deltaCBy=Cy-By;
    var ax,ay;
    var lastX=-10000;
    var lastY=-10000;
    var pts=[{x:Ax,y:Ay}];
    for(var i=1;i<ptCount;i++){
        var t=i/ptCount;
        ax=Ax+deltaBAx*t;
        ay=Ay+deltaBAy*t;
        var x=ax+((Bx+deltaCBx*t)-ax)*t;
        var y=ay+((By+deltaCBy*t)-ay)*t;
        var dx=x-lastX;
        var dy=y-lastY;
        if(dx*dx+dy*dy>pxTolerance){
            pts.push({x:x,y:y});
            lastX=x;
            lastY=y;
        }
    }
    pts.push({x:Cx,y:Cy});
    return(pts);
}

// Return: an array of approximately evenly spaced points along a line
//
// pxTolerance: approximate spacing allowed between points
// Ax,Ay,Bx,By: end points defining the line
//
function plotLine(pxTolerance,Ax,Ay,Bx,By){
    var dx=Bx-Ax;
    var dy=By-Ay;
    var ptCount=parseInt(Math.sqrt(dx*dx+dy*dy))*3;
    var lastX=-10000;
    var lastY=-10000;
    var pts=[{x:Ax,y:Ay}];
    for(var i=1;i<=ptCount;i++){
        var t=i/ptCount;
        var x=Ax+dx*t;
        var y=Ay+dy*t;
        var dx1=x-lastX;
        var dy1=y-lastY;
        if(dx1*dx1+dy1*dy1>pxTolerance){
            pts.push({x:x,y:y});
            lastX=x;
            lastY=y;
        }
    }
}

```

```

    }
  }
  pts.push({x:Bx,y:By});
  return(pts);
}

```

Longitud de una curva cuadrática

Dados los 3 puntos de una curva cuadrática, la siguiente función devuelve la longitud.

```

function quadraticBezierLength(x1,y1,x2,y2,x3,y3)
  var a, e, c, d, u, a1, e1, c1, d1, u1, v1x, v1y;

  v1x = x2 * 2;
  v1y = y2 * 2;
  d = x1 - v1x + x3;
  d1 = y1 - v1y + y3;
  e = v1x - 2 * x1;
  e1 = v1y - 2 * y1;
  c1 = (a = 4 * (d * d + d1 * d1));
  c1 += (b = 4 * (d * e + d1 * e1));
  c1 += (c = e * e + e1 * e1);
  c1 = 2 * Math.sqrt(c1);
  a1 = 2 * a * (u = Math.sqrt(a));
  u1 = b / u;
  a = 4 * c * a - b * b;
  c = 2 * Math.sqrt(c);
  return (a1 * c1 + u * b * (c1 - c) + a * Math.log((2 * u + u1 + c1) / (u1 + c))) / (4 *
a1);
}

```

Derivado de la función de bezier cuadrática $F(t) = a * (1 - t)^2 + 2 * b * (1 - t) * t + c * t^2$

Dividir curvas de bezier en la posición.

Este ejemplo divide curvas cúbicas y bezier en dos.

La función `splitCurveAt` divide la curva en la `position` donde `0.0 = inicio`, `0.5 = medio` y `1 = final`. Puede dividir curvas cuadráticas y cúbicas. El tipo de curva está determinado por el último argumento `x4`. Si no está `undefined` o es `null`, se asume que la curva es cúbica, de lo contrario la curva es una acción cuadrática.

Ejemplo de uso

División de la curva bezier cuadrática en dos.

```

var p1 = {x : 10 , y : 100};
var p2 = {x : 100, y : 200};
var p3 = {x : 200, y : 0};
var newCurves = splitCurveAt(0.5, p1.x, p1.y, p2.x, p2.y, p3.x, p3.y)

var i = 0;
var p = newCurves
// Draw the 2 new curves

```

```
// Assumes ctx is canvas 2d context
ctx.lineWidth = 1;
ctx.strokeStyle = "black";
ctx.beginPath();
ctx.moveTo(p[i++],p[i++]);
ctx.quadraticCurveTo(p[i++], p[i++], p[i++], p[i++]);
ctx.quadraticCurveTo(p[i++], p[i++], p[i++], p[i++]);
ctx.stroke();
```

Dividiendo la curva bezier cúbica en dos

```
var p1 = {x : 10 , y : 100};
var p2 = {x : 100, y : 200};
var p3 = {x : 200, y : 0};
var p4 = {x : 300, y : 100};
var newCurves = splitCurveAt(0.5, p1.x, p1.y, p2.x, p2.y, p3.x, p3.y, p4.x, p4.y)

var i = 0;
var p = newCurves
// Draw the 2 new curves
// Assumes ctx is canvas 2d context
ctx.lineWidth = 1;
ctx.strokeStyle = "black";
ctx.beginPath();
ctx.moveTo(p[i++],p[i++]);
ctx.bezierCurveTo(p[i++], p[i++], p[i++], p[i++], p[i++], p[i++]);
ctx.bezierCurveTo(p[i++], p[i++], p[i++], p[i++], p[i++], p[i++]);
ctx.stroke();
```

La función dividida

splitCurveAt = función (posición, x1, y1, x2, y2, x3, y3, [x4, y4])

Nota: los argumentos dentro de [x4, y4] son opcionales.

Nota: La función tiene algún código `/* */` comentado opcional que se ocupa de los casos de borde en los que las curvas resultantes pueden tener una longitud cero, o quedar fuera del inicio o final de la curva original. Al intentar dividir una curva fuera del rango válido para la `position` ≥ 0 o la `position` ≥ 1 producirá un error de rango. Esto se puede eliminar y funcionará bien, aunque es posible que tenga curvas resultantes que tengan una longitud cero.

```
// With throw RangeError if not 0 < position < 1
// x1, y1, x2, y2, x3, y3 for quadratic curves
// x1, y1, x2, y2, x3, y3, x4, y4 for cubic curves
// Returns an array of points representing 2 curves. The curves are the same type as the split
curve
var splitCurveAt = function(position, x1, y1, x2, y2, x3, y3, x4, y4){
    var v1, v2, v3, v4, quad, retPoints, i, c;

    //
    =====
    // you may remove this as the function will still work and resulting curves will still
render
```

```

// but other curve functions may not like curves with 0 length
//
=====
if(position <= 0 || position >= 1){
    throw RangeError("spliteCurveAt requires position > 0 && position < 1");
}

//
=====
// If you remove the above range error you may use one or both of the following commented
sections
// Splitting curves position < 0 or position > 1 will still create valid curves but they
will
// extend past the end points

//
=====
// Lock the position to split on the curve.
/* optional A
position = position < 0 ? 0 : position > 1 ? 1 : position;
optional A end */

//
=====
// the next commented section will return the original curve if the split results in 0
length curve
// You may wish to uncomment this If you desire such functionality
/* optional B
if(position <= 0 || position >= 1){
    if(x4 === undefined || x4 === null){
        return [x1, y1, x2, y2, x3, y3];
    }else{
        return [x1, y1, x2, y2, x3, y3, x4, y4];
    }
}
optional B end */

retPoints = []; // array of coordinates
i = 0;
quad = false; // presume cubic bezier
v1 = {};
v2 = {};
v4 = {};
v1.x = x1;
v1.y = y1;
v2.x = x2;
v2.y = y2;
if(x4 === undefined || x4 === null){
    quad = true; // this is a quadratic bezier
    v4.x = x3;
    v4.y = y3;
}else{
    v3 = {};
    v3.x = x3;
    v3.y = y3;
    v4.x = x4;
    v4.y = y4;
}
c = position;
retPoints[i++] = v1.x; // start point

```



```

retPoints[i++] = v1.y;

if(quad){ // split quadratic bezier
    retPoints[i++] = (v1.x += (v2.x - v1.x) * c); // new control point for first curve
    retPoints[i++] = (v1.y += (v2.y - v1.y) * c);
    v2.x += (v4.x - v2.x) * c;
    v2.y += (v4.y - v2.y) * c;
    retPoints[i++] = v1.x + (v2.x - v1.x) * c; // new end and start of first and second
curves
    retPoints[i++] = v1.y + (v2.y - v1.y) * c;
    retPoints[i++] = v2.x; // new control point for second curve
    retPoints[i++] = v2.y;
    retPoints[i++] = v4.x; // new endpoint of second curve
    retPoints[i++] = v4.y;
    //=====
    // return array with 2 curves
    return retPoints;
}
retPoints[i++] = (v1.x += (v2.x - v1.x) * c); // first curve first control point

retPoints[i++] = (v1.y += (v2.y - v1.y) * c);
v2.x += (v3.x - v2.x) * c;
v2.y += (v3.y - v2.y) * c;
v3.x += (v4.x - v3.x) * c;
v3.y += (v4.y - v3.y) * c;
retPoints[i++] = (v1.x += (v2.x - v1.x) * c); // first curve second control point
retPoints[i++] = (v1.y += (v2.y - v1.y) * c);
v2.x += (v3.x - v2.x) * c;
v2.y += (v3.y - v2.y) * c;
retPoints[i++] = v1.x + (v2.x - v1.x) * c; // end and start point of first second curves
retPoints[i++] = v1.y + (v2.y - v1.y) * c;
retPoints[i++] = v2.x; // second curve first control point
retPoints[i++] = v2.y;
retPoints[i++] = v3.x; // second curve second control point
retPoints[i++] = v3.y;
retPoints[i++] = v4.x; // endpoint of second curve
retPoints[i++] = v4.y;
//=====
// return array with 2 curves
return retPoints;
}

```

Recorte la curva de bezier.

Este ejemplo le muestra cómo recortar un bezier.

La función `trimBezier` recorta los extremos de la curva devolviendo la curva `fromPos a toPos` . `fromPos` y `toPos` están en el rango de 0 a 1 inclusive, puede recortar curvas cuadráticas y cúbicas. El tipo de curva está determinado por el último argumento `x4` . Si no está `undefined` o es `null` , se asume que la curva es cúbica, de lo contrario la curva es una acción cuadrática.

La curva recortada se devuelve como una matriz de puntos. 6 puntos para curvas cuadráticas y 8 para curvas cúbicas.

Ejemplo de uso

Recorte de una curva cuadrática.

```
var p1 = {x : 10 , y : 100};
var p2 = {x : 100, y : 200};
var p3 = {x : 200, y : 0};
var newCurve = splitCurveAt(0.25, 0.75, p1.x, p1.y, p2.x, p2.y, p3.x, p3.y)

var i = 0;
var p = newCurve
// Draw the trimmed curve
// Assumes ctx is canvas 2d context
ctx.lineWidth = 1;
ctx.strokeStyle = "black";
ctx.beginPath();
ctx.moveTo(p[i++],p[i++]);
ctx.quadraticCurveTo(p[i++], p[i++], p[i++], p[i++]);
ctx.stroke();
```

Recorte de una curva cúbica.

```
var p1 = {x : 10 , y : 100};
var p2 = {x : 100, y : 200};
var p3 = {x : 200, y : 0};
var p4 = {x : 300, y : 100};
var newCurve = splitCurveAt(0.25, 0.75, p1.x, p1.y, p2.x, p2.y, p3.x, p3.y, p4.x, p4.y)

var i = 0;
var p = newCurve
// Draw the trimmed curve
// Assumes ctx is canvas 2d context
ctx.lineWidth = 1;
ctx.strokeStyle = "black";
ctx.beginPath();
ctx.moveTo(p[i++],p[i++]);
ctx.bezierCurveTo(p[i++], p[i++], p[i++], p[i++], p[i++], p[i++]);
ctx.stroke();
```

Ejemplo de función

trimBezier = function (fromPos, toPos, x1, y1, x2, y2, x3, y3, [x4, y4])

Nota: los argumentos dentro de [x4, y4] son opcionales.

Nota: Esta función requiere la función en el ejemplo Dividir curvas Bezier en en esta sección

```
var trimBezier = function(fromPos, toPos, x1, y1, x2, y2, x3, y3, x4, y4){
  var quad, i, s, retBez;
  quad = false;
  if(x4 === undefined || x4 === null){
    quad = true; // this is a quadratic bezier
  }
  if(fromPos > toPos){ // swap is from is after to
    i = fromPos;
    fromPos = toPos
```

```

    toPos = i;
}
// clamp to on the curve
toPos = toPos <= 0 ? 0 : toPos >= 1 ? 1 : toPos;
fromPos = fromPos <= 0 ? 0 : fromPos >= 1 ? 1 : fromPos;
if(toPos === fromPos){
    s = splitBezierAt(toPos, x1, y1, x2, y2, x3, y3, x4, y4);
    i = quad ? 4 : 6;
    retBez = [s[i], s[i+1], s[i], s[i+1], s[i], s[i+1]];
    if(!quad){
        retBez.push(s[i], s[i+1]);
    }
    return retBez;
}
if(toPos === 1 && fromPos === 0){ // no trimming required
    retBez = [x1, y1, x2, y2, x3, y3]; // return original bezier
    if(!quad){
        retBez.push(x4, y4);
    }
    return retBez;
}
if(fromPos === 0){
    if(toPos < 1){
        s = splitBezierAt(toPos, x1, y1, x2, y2, x3, y3, x4, y4);
        i = 0;
        retBez = [s[i++], s[i++], s[i++], s[i++], s[i++], s[i++]];
        if(!quad){
            retBez.push(s[i++], s[i++]);
        }
    }
    return retBez;
}
if(toPos === 1){
    if(fromPos < 1){
        s = splitBezierAt(toPos, x1, y1, x2, y2, x3, y3, x4, y4);
        i = quad ? 4 : 6;
        retBez = [s[i++], s[i++], s[i++], s[i++], s[i++], s[i++]];
        if(!quad){
            retBez.push(s[i++], s[i++]);
        }
    }
    return retBez;
}
s = splitBezierAt(fromPos, x1, y1, x2, y2, x3, y3, x4, y4);
if(quad){
    i = 4;
    toPos = (toPos - fromPos) / (1 - fromPos);
    s = splitBezierAt(toPos, s[i++], s[i++], s[i++], s[i++], s[i++], s[i++]);
    i = 0;
    retBez = [s[i++], s[i++], s[i++], s[i++], s[i++], s[i++]];
    return retBez;
}
i = 6;
toPos = (toPos - fromPos) / (1 - fromPos);
s = splitBezierAt(toPos, s[i++], s[i++], s[i++], s[i++], s[i++], s[i++], s[i++], s[i++], s[i++]);
i = 0;
retBez = [s[i++], s[i++], s[i++], s[i++], s[i++], s[i++], s[i++], s[i++]];
return retBez;
}

```

Longitud de una curva de Bezier cúbica (una aproximación cercana)

Dados los 4 puntos de una curva Bezier cúbica, la siguiente función devuelve su longitud.

Método: La longitud de una curva Bezier cúbica no tiene un cálculo matemático directo. Este método de "fuerza bruta" encuentra una muestra de puntos a lo largo de la curva y calcula la distancia total que abarcan esos puntos.

Precisión: la longitud aproximada es 99 +% precisa utilizando el tamaño de muestreo predeterminado de 40.

```
// Return: Close approximation of the length of a Cubic Bezier curve
//
// Ax,Ay,Bx,By,Cx,Cy,Dx,Dy: the 4 control points of the curve
// sampleCount [optional, default=40]: how many intervals to calculate
// Requires: cubicQxy (included below)
//
function cubicBezierLength(Ax,Ay,Bx,By,Cx,Cy,Dx,Dy,sampleCount){
    var ptCount=sampleCount||40;
    var totDist=0;
    var lastX=Ax;
    var lastY=Ay;
    var dx,dy;
    for(var i=1;i<ptCount;i++){
        var pt=cubicQxy(i/ptCount,Ax,Ay,Bx,By,Cx,Cy,Dx,Dy);
        dx=pt.x-lastX;
        dy=pt.y-lastY;
        totDist+=Math.sqrt(dx*dx+dy*dy);
        lastX=pt.x;
        lastY=pt.y;
    }
    dx=Dx-lastX;
    dy=Dy-lastY;
    totDist+=Math.sqrt(dx*dx+dy*dy);
    return(parseInt(totDist));
}

// Return: an [x,y] point along a cubic Bezier curve at interval T
//
// Attribution: Stackoverflow's @Blindman67
// Cite: http://stackoverflow.com/questions/36637211/drawing-a-curved-line-in-css-or-canvas-
// and-moving-circle-along-it/36827074#36827074
// As modified from the above citation
//
// t: an interval along the curve (0<=t<=1)
// ax,ay,bx,by,cx,cy,dx,dy: control points defining the curve
//
function cubicQxy(t,ax,ay,bx,by,cx,cy,dx,dy){
    ax += (bx - ax) * t;
    bx += (cx - bx) * t;
    cx += (dx - cx) * t;
    ax += (bx - ax) * t;
    bx += (cx - bx) * t;
    ay += (by - ay) * t;
    by += (cy - by) * t;
    cy += (dy - cy) * t;
}
```

```

ay += (by - ay) * t;
by += (cy - by) * t;
return({
  x:ax +(bx - ax) * t,
  y:ay +(by - ay) * t
});
}

```

Encontrar el punto en la curva

Este ejemplo encuentra un punto en una curva bezier o cúbica en la `position` en la que la `position` es la distancia de la unidad en la curva $0 \leq position \leq 1$. La posición se fija al rango, por lo tanto, si los valores <0 o > 1 se pasan, serán conjunto 0,1 respectivamente.

Pasa las coordenadas de la función 6 para el bizantín cuadrático u 8 para el cúbico.

El último argumento opcional es el vector devuelto (punto). Si no se da será creado.

Ejemplo de uso

```

var p1 = {x : 10 , y : 100};
var p2 = {x : 100, y : 200};
var p3 = {x : 200, y : 0};
var p4 = {x : 300, y : 100};
var point = {x : null, y : null};

// for cubic beziers
point = getPointOnCurve(0.5, p1.x, p1.y, p2.x, p2.y, p3.x, p3.y, p4.x, p4.y, point);
// or No need to set point as it is a referance and will be set
getPointOnCurve(0.5, p1.x, p1.y, p2.x, p2.y, p3.x, p3.y, p4.x, p4.y, point);
// or to create a new point
var point1 = getPointOnCurve(0.5, p1.x, p1.y, p2.x, p2.y, p3.x, p3.y, p4.x, p4.y);

// for quadratic beziers
point = getPointOnCurve(0.5, p1.x, p1.y, p2.x, p2.y, p3.x, p3.y, null, null, point);
// or No need to set point as it is a referance and will be set
getPointOnCurve(0.5, p1.x, p1.y, p2.x, p2.y, p3.x, p3.y, null, null, point);
// or to create a new point
var point1 = getPointOnCurve(0.5, p1.x, p1.y, p2.x, p2.y, p3.x, p3.y);

```

La función

getPointOnCurve = function (posición, x1, y1, x2, y2, x3, y3, [x4, y4], [vec])

Nota: los argumentos dentro de [x4, y4] son opcionales.

Nota: x4 , y4 si es null , o undefined significa que la curva es una curva cuadrática. vec es opcional y mantendrá el punto devuelto si se proporciona. Si no se creará.

```

var getPointOnCurve = function(position, x1, y1, x2, y2, x3, y3, x4, y4, vec){

```

```

var vec, quad;
quad = false;
if(vec === undefined){
    vec = {};
}

if(x4 === undefined || x4 === null){
    quad = true;
    x4 = x3;
    y4 = y3;
}

if(position <= 0){
    vec.x = x1;
    vec.y = y1;
    return vec;
}
if(position >= 1){
    vec.x = x4;
    vec.y = y4;
    return vec;
}
c = position;
if(quad){
    x1 += (x2 - x1) * c;
    y1 += (y2 - y1) * c;
    x2 += (x3 - x2) * c;
    y2 += (y3 - y2) * c;
    vec.x = x1 + (x2 - x1) * c;
    vec.y = y1 + (y2 - y1) * c;
    return vec;
}
x1 += (x2 - x1) * c;
y1 += (y2 - y1) * c;
x2 += (x3 - x2) * c;
y2 += (y3 - y2) * c;
x3 += (x4 - x3) * c;
y3 += (y4 - y3) * c;
x1 += (x2 - x1) * c;
y1 += (y2 - y1) * c;
x2 += (x3 - x2) * c;
y2 += (y3 - y2) * c;
vec.x = x1 + (x2 - x1) * c;
vec.y = y1 + (y2 - y1) * c;
return vec;
}

```

Encontrar la extensión de la curva cuadrática

Cuando necesite encontrar el rectángulo delimitador de una curva bezier cuadrática, puede utilizar el siguiente método de ejecución.

```

// This method was discovered by Blindman67 and solves by first normalising the control point
// thereby reducing the algorithm complexity
// x1,y1, x2,y2, x3,y3 Start, Control, and End coords of bezier
// [extent] is optional and if provided the extent will be added to it allowing you to use the
// function
//         to get the extent of many beziers.
// returns extent object (if not supplied a new extent is created)

```

```

// Extent object properties
// top, left, right, bottom, width, height
function getQuadraticCurveExtent(x1, y1, x2, y2, x3, y3, extent) {
    var brx, bx, x, bry, by, y, px, py;

    // solve quadratic for bounds by BM67 normalizing equation
    brx = x3 - x1; // get x range
    bx = x2 - x1; // get x control point offset
    x = bx / brx; // normalise control point which is used to check if maxima is in range

    // do the same for the y points
    bry = y3 - y1;
    by = y2 - y1;
    y = by / bry;

    px = x1; // set defaults in case maxima outside range
    py = y1;

    // find top/left, top/right, bottom/left, or bottom/right
    if (x < 0 || x > 1) { // check if x maxima is on the curve
        px = bx * bx / (2 * bx - brx) + x1; // get the x maxima
    }
    if (y < 0 || y > 1) { // same as x
        py = by * by / (2 * by - bry) + y1;
    }

    // create extent object and add extent
    if (extent === undefined) {
        extent = {};
        extent.left = Math.min(x1, x3, px);
        extent.top = Math.min(y1, y3, py);
        extent.right = Math.max(x1, x3, px);
        extent.bottom = Math.max(y1, y3, py);
    } else { // use supplied extent and extend it to fit this curve
        extent.left = Math.min(x1, x3, px, extent.left);
        extent.top = Math.min(y1, y3, py, extent.top);
        extent.right = Math.max(x1, x3, px, extent.right);
        extent.bottom = Math.max(y1, y3, py, extent.bottom);
    }

    extent.width = extent.right - extent.left;
    extent.height = extent.bottom - extent.top;
    return extent;
}

```

Para obtener una visión más detallada de cómo resolver para obtener más información, consulte la respuesta [Para obtener la extensión de una versión cuadrática](#) que incluye demostraciones ejecutables.

Lea Navegando por un sendero en línea: <https://riptutorial.com/es/html5-canvas/topic/5281/navegando-por-un-sendero>

Capítulo 14: Oscuridad

Examples

Efecto adhesivo utilizando sombras.

Este código agrega sombras que aumentan hacia el exterior a una imagen para crear una versión "adhesiva" de la imagen.

Notas:

- Además de ser un ImageObject, el argumento "img" también puede ser un elemento Canvas. Esto te permite pegar tus propios dibujos personalizados. Si dibujas texto en el argumento del lienzo, también puede pegar ese texto.
- Las imágenes totalmente opacas no tendrán efecto de etiqueta porque el efecto se dibuja alrededor de grupos de píxeles opacos que están bordeados por píxeles transparentes.



```
var canvas=document.createElement("canvas");
var ctx=canvas.getContext("2d");
document.body.appendChild(canvas);
canvas.style.background='navy';
canvas.style.border='1px solid red;';

// Always(!) wait for your images to fully load before trying to drawImage them!
var img=new Image();
img.onload=start;
// put your img.src here...
img.src='http://i.stack.imgur.com/bXaB6.png';
function start(){
    ctx.drawImage(img,20,20);
    var sticker=stickerEffect(img,5);
    ctx.drawImage(sticker, 150,20);
}

function stickerEffect(img,grow){
    var canvas1=document.createElement("canvas");
    var ctx1=canvas1.getContext("2d");
    var canvas2=document.createElement("canvas");
    var ctx2=canvas2.getContext("2d");
    canvas1.width=canvas2.width=img.width+grow*2;
    canvas1.height=canvas2.height=img.height+grow*2;
    ctx1.drawImage(img,grow,grow);
    ctx2.shadowColor='white';
```



```

ctx2.shadowBlur=2;
for(var i=0;i<grow;i++){
    ctx2.drawImage(canvas1,0,0);
    ctx1.drawImage(canvas2,0,0);
}
ctx2.shadowColor='rgba(0,0,0,0)';
ctx2.drawImage(img,grow,grow);
return(canvas2);
}

```

¿Cómo parar más sombras?

Una vez que se activa el sombreado, cada nuevo dibujo del lienzo se sombreadá.

Desactive las sombras adicionales configurando `context.shadowColor` a un color transparente.

```

// start shadowing
context.shadowColor='black';

... render some shadowed drawings ...

// turn off shadowing.
context.shadowColor='rgba(0,0,0,0)';

```

El sombreado es computacionalmente costoso: ¡caché esa sombra!

¡Advertencia! Aplicar sombras con moderación!

Aplicar el sombreado es costoso y es muy costoso si aplica el sombreado dentro de un ciclo de animación.

En su lugar, almacene en caché una versión sombreada de su imagen (u otro dibujo):

- Al inicio de su aplicación, cree una versión sombreada de su imagen en un segundo lienzo solo en memoria: `var memoryCanvas = document.createElement('canvas') ...`
- Siempre que necesite la versión sombreada, dibuje esa imagen previamente sombreada desde el lienzo en memoria al lienzo visible: `context.drawImage(memoryCanvas,x,y)`



```

var canvas=document.createElement("canvas");
var ctx=canvas.getContext("2d");
var cw=canvas.width;
var ch=canvas.height;

```

```

canvas.style.border='1px solid red;';
document.body.appendChild(canvas);

// Always(!) use "img.onload" to give your image time to
//      fully load before you try drawing it to the Canvas!
var img=new Image();
img.onload=start;
// Put your own img.src here
img.src="http://i.stack.imgur.com/hYFNe.png";
function start(){
    ctx.drawImage(img,0,20);
    var cached=cacheShadowedImage(img,'black',5,3,3);
    for(var i=0;i<5;i++){
        ctx.drawImage(cached,i*(img.width+10),80);
    }
}

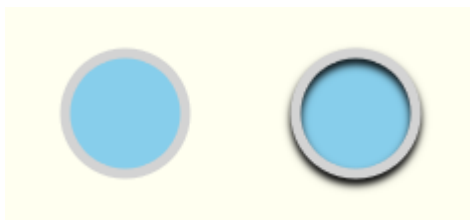
function cacheShadowedImage(img,shadowcolor,blur){
    var c=document.createElement('canvas');
    var cctx=c.getContext('2d');
    c.width=img.width+blur*2+2;
    c.height=img.height+blur*2+2;
    cctx.shadowColor=shadowcolor;
    cctx.shadowBlur=blur;
    cctx.drawImage(img,blur+1,blur+1);
    return(c);
}

```

Añade profundidad visual con sombras

El uso tradicional de la sombra es dar a los dibujos bidimensionales la ilusión de la profundidad 3D.

Este ejemplo muestra el mismo "botón" con y sin sombra



```

var canvas=document.createElement("canvas");
var ctx=canvas.getContext("2d");
document.body.appendChild(canvas);

ctx.fillStyle='skyblue';
ctx.strokeStyle='lightgray';
ctx.lineWidth=5;

// without shadow
ctx.beginPath();
ctx.arc(60,60,30,0,Math.PI*2);
ctx.closePath();
ctx.fill();
ctx.stroke();

// with shadow

```

```
ctx.shadowColor='black';
ctx.shadowBlur=4;
ctx.shadowOffsetY=3;
ctx.beginPath();
ctx.arc(175,60,30,0,Math.PI*2);
ctx.closePath();
ctx.fill();
ctx.stroke();
// stop the shadowing
ctx.shadowColor='rgba(0,0,0,0)';
```

Sombras interiores

Canvas no tiene la `inner-shadow` de CSS.

- El lienzo sombreará el exterior de una forma rellena.
- El lienzo sombreará tanto dentro como fuera de una forma trazada.

Pero es fácil crear sombras internas utilizando la composición.

Trazos con una sombra interior.



Para crear trazos con una sombra interior, use la composición de `destination-in`, lo que hace que el contenido existente permanezca solo donde el contenido existente se superpone al contenido nuevo. El contenido existente que no se superpone con el contenido nuevo se borra.

1. **Trazar una forma con una sombra.** La sombra se extenderá tanto hacia afuera como hacia adentro desde el trazo. Debemos deshacernos de la sombra exterior, dejando solo la sombra interior deseada.
2. **Establezca la composición en `destination-in`** que mantiene la sombra trazada existente solo donde se superponga con cualquier dibujo nuevo.
3. **Rellena la forma.** Esto hace que el trazo y la sombra interior permanezcan mientras se borra la sombra exterior. *¡Bueno no exactamente! Dado que un trazo es medio dentro y medio fuera de la forma rellena, la mitad exterior del trazo también se borrará. La solución es duplicar el `context.lineWidth` por lo que la mitad del trazo de doble tamaño aún está dentro de la forma rellena.*

```
var canvas=document.createElement("canvas");
var ctx=canvas.getContext("2d");
document.body.appendChild(canvas);

// draw an opaque shape -- here we use a rounded rectangle
defineRoundedRect(30,30,100,75,10);
```

```

// set shadowing
ctx.shadowColor='black';
ctx.shadowBlur=10;

// stroke the shadowed rounded rectangle
ctx.lineWidth=4;
ctx.stroke();

// set compositing to erase everything outside the stroke
ctx.globalCompositeOperation='destination-in';
ctx.fill();

// always clean up -- set compositing back to default
ctx.globalCompositeOperation='source-over';

function defineRoundedRect(x,y,width,height,radius) {
  ctx.beginPath();
  ctx.moveTo(x + radius, y);
  ctx.lineTo(x + width - radius, y);
  ctx.quadraticCurveTo(x + width, y, x + width, y + radius);
  ctx.lineTo(x + width, y + height - radius);
  ctx.quadraticCurveTo(x + width, y + height, x + width - radius, y + height);
  ctx.lineTo(x + radius, y + height);
  ctx.quadraticCurveTo(x, y + height, x, y + height - radius);
  ctx.lineTo(x, y + radius);
  ctx.quadraticCurveTo(x, y, x + radius, y);
  ctx.closePath();
}

```

Trazos rellenos con una sombra interior



Para crear rellenos con una sombra interior, siga los pasos 1 a 3 anteriores, pero utilice aún `destination-over` composición de `destination-over`, lo que hace que se dibuje nuevo contenido **bajo el contenido existente**.

4. **Establezca la composición en `destination-over`** que hace que el relleno se dibuje **bajo** la sombra interna existente.
5. **Desactive el sombreado** configurando `context.shadowColor` a un color transparente.
6. **Rellena la forma** con el color deseado. La forma se rellenará debajo de la sombra interior existente.

```

var canvas=document.createElement("canvas");
var ctx=canvas.getContext("2d");
document.body.appendChild(canvas);

// draw an opaque shape -- here we use a rounded rectangle
defineRoundedRect(30,30,100,75,10);

```

```

// set shadowing
ctx.shadowColor='black';
ctx.shadowBlur=10;

// stroke the shadowed rounded rectangle
ctx.lineWidth=4;
ctx.stroke();

// stop shadowing
ctx.shadowColor='rgba(0,0,0,0)';

// set compositing to erase everything outside the stroke
ctx.globalCompositeOperation='destination-in';
ctx.fill();

// set compositing to erase everything outside the stroke
ctx.globalCompositeOperation='destination-over';
ctx.fillStyle='gold';
ctx.fill();

// always clean up -- set compositing back to default
ctx.globalCompositeOperation='source-over';

function defineRoundedRect(x,y,width,height,radius) {
    ctx.beginPath();
    ctx.moveTo(x + radius, y);
    ctx.lineTo(x + width - radius, y);
    ctx.quadraticCurveTo(x + width, y, x + width, y + radius);
    ctx.lineTo(x + width, y + height - radius);
    ctx.quadraticCurveTo(x + width, y + height, x + width - radius, y + height);
    ctx.lineTo(x + radius, y + height);
    ctx.quadraticCurveTo(x, y + height, x, y + height - radius);
    ctx.lineTo(x, y + radius);
    ctx.quadraticCurveTo(x, y, x + radius, y);
    ctx.closePath();
}

```

Rellenos sin trazos con una sombra interior



Para dibujar una forma rellena con una sombra interior, pero sin trazo, puede dibujar el trazo fuera del lienzo y usar `shadowOffsetX` para empujar la sombra hacia atrás en el lienzo.

```

var canvas=document.createElement("canvas");
var ctx=canvas.getContext("2d");
document.body.appendChild(canvas);

// define an opaque shape -- here we use a rounded rectangle
defineRoundedRect(30-500,30,100,75,10);

```

```

// set shadowing
ctx.shadowColor='black';
ctx.shadowBlur=10;
ctx.shadowOffsetX=500;

// stroke the shadowed rounded rectangle
ctx.lineWidth=4;
ctx.stroke();

// stop shadowing
ctx.shadowColor='rgba(0,0,0,0)';

// redefine an opaque shape -- here we use a rounded rectangle
defineRoundedRect(30,30,100,75,10);

// set compositing to erase everything outside the stroke
ctx.globalCompositeOperation='destination-in';
ctx.fill();

// set compositing to erase everything outside the stroke
ctx.globalCompositeOperation='destination-over';
ctx.fillStyle='gold';
ctx.fill();

// always clean up -- set compositing back to default
ctx.globalCompositeOperation='source-over';

function defineRoundedRect(x,y,width,height,radius) {
    ctx.beginPath();
    ctx.moveTo(x + radius, y);
    ctx.lineTo(x + width - radius, y);
    ctx.quadraticCurveTo(x + width, y, x + width, y + radius);
    ctx.lineTo(x + width, y + height - radius);
    ctx.quadraticCurveTo(x + width, y + height, x + width - radius, y + height);
    ctx.lineTo(x + radius, y + height);
    ctx.quadraticCurveTo(x, y + height, x, y + height - radius);
    ctx.lineTo(x, y + radius);
    ctx.quadraticCurveTo(x, y, x + radius, y);
    ctx.closePath();
}

```

Lea Oscuridad en línea: <https://riptutorial.com/es/html5-canvas/topic/5322/oscuridad>

Capítulo 15: Poligonos

Examples

Estrellas

Dibuja estrellas con un estilo flexible (tamaño, colores, número de puntos).



```
// Usage:
drawStar(75,75,5,50,25,'mediumseagreen','gray',9);
drawStar(150,200,8,50,25,'skyblue','gray',3);
drawStar(225,75,16,50,20,'coral','transparent',0);
drawStar(300,200,16,50,40,'gold','gray',3);

// centerX, centerY: the center point of the star
// points: the number of points on the exterior of the star
// inner: the radius of the inner points of the star
// outer: the radius of the outer points of the star
// fill, stroke: the fill and stroke colors to apply
// line: the linewidth of the stroke

function drawStar(centerX, centerY, points, outer, inner, fill, stroke, line) {
  // define the star
  ctx.beginPath();
  ctx.moveTo(centerX, centerY+outer);
  for (var i=0; i < 2*points+1; i++) {
    var r = (i%2 == 0)? outer : inner;
    var a = Math.PI * i/points;
    ctx.lineTo(centerX + r*Math.sin(a), centerY + r*Math.cos(a));
  };
  ctx.closePath();
  // draw
  ctx.fillStyle=fill;
  ctx.fill();
  ctx.strokeStyle=stroke;
  ctx.lineWidth=line;
  ctx.stroke()
}
```

Polígono regular

Un polígono regular tiene todos los lados de igual longitud.



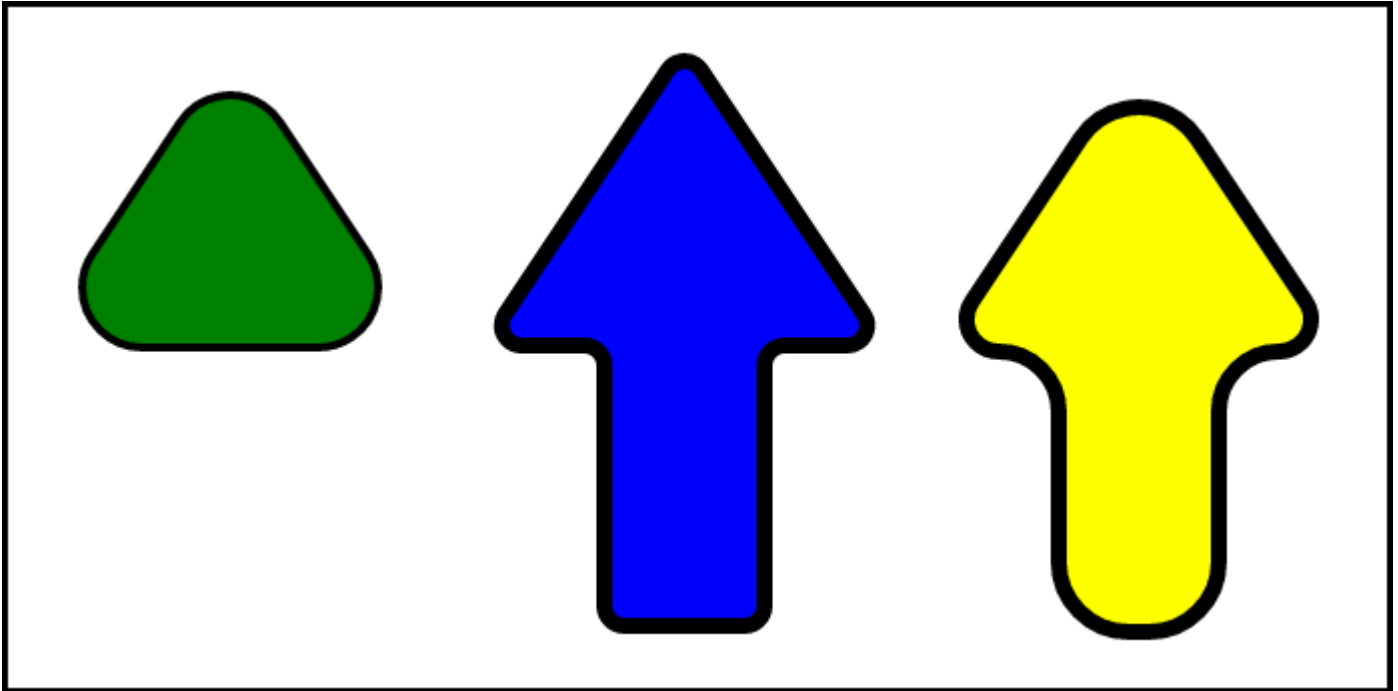
```
// Usage:
drawRegularPolygon(3,25,75,50,6,'gray','red',0);
drawRegularPolygon(5,25,150,50,6,'gray','gold',0);
drawRegularPolygon(6,25,225,50,6,'gray','lightblue',0);
drawRegularPolygon(10,25,300,50,6,'gray','lightgreen',0);

function
drawRegularPolygon(sideCount,radius,centerX,centerY,strokeWidth,strokeColor,fillColor,rotationRadians)

    var angles=Math.PI*2/sideCount;
    ctx.translate(centerX,centerY);
    ctx.rotate(rotationRadians);
    ctx.beginPath();
    ctx.moveTo(radius,0);
    for(var i=1;i<sideCount;i++){
        ctx.rotate(angles);
        ctx.lineTo(radius,0);
    }
    ctx.closePath();
    ctx.fillStyle=fillColor;
    ctx.strokeStyle = strokeColor;
    ctx.lineWidth = strokeWidth;
    ctx.stroke();
    ctx.fill();
    ctx.rotate(angles*-(sideCount-1));
    ctx.rotate(-rotationRadians);
    ctx.translate(-centerX,-centerY);
}
```

Renderiza un polígono redondeado.

Crema una ruta a partir de un conjunto de puntos $[[\{x:?,y:?\},\{x:?,y:?\},\dots,\{x:?,y:?\}]]$ con esquinas redondeadas de radio. Si el ángulo de la esquina es demasiado pequeño para ajustarse al radio o la distancia entre las esquinas no permite espacio, el radio de las esquinas se reduce a un mejor ajuste.



Ejemplo de uso

```
var triangle = [
  { x: 200, y : 50 },
  { x: 300, y : 200 },
  { x: 100, y : 200 }
];
var cornerRadius = 30;
ctx.lineWidth = 4;
ctx.fillStyle = "Green";
ctx.strokeStyle = "black";
ctx.beginPath(); // start a new path
roundedPoly(triangle, cornerRadius);
ctx.fill();
ctx.stroke();
```

Función de render

```
var roundedPoly = function(points,radius){
  var i, x, y, len, p1, p2, p3, v1, v2, sinA, sinA90, radDirection, drawDirection, angle,
  halfAngle, cRadius, lenOut;
  var asVec = function (p, pp, v) { // convert points to a line with len and normalised
    v.x = pp.x - p.x; // x,y as vec
    v.y = pp.y - p.y;
    v.len = Math.sqrt(v.x * v.x + v.y * v.y); // length of vec
    v.nx = v.x / v.len; // normalised
    v.ny = v.y / v.len;
    v.ang = Math.atan2(v.ny, v.nx); // direction of vec
  }
  v1 = {};
  v2 = {};
  len = points.length; // number points
  p1 = points[len - 1]; // start at end of path
  for (i = 0; i < len; i++) { // do each corner
    p2 = points[(i) % len]; // the corner point that is being rounded
    p3 = points[(i + 1) % len];
    // get the corner as vectors out away from corner
```

```

asVec(p2, p1, v1); // vec back from corner point
asVec(p2, p3, v2); // vec forward from corner point
// get corners cross product (asin of angle)
sinA = v1.nx * v2.ny - v1.ny * v2.nx; // cross product
// get cross product of first line and perpendicular second line
sinA90 = v1.nx * v2.nx - v1.ny * -v2.ny; // cross product to normal of line 2
angle = Math.asin(sinA); // get the angle
radDirection = 1; // may need to reverse the radius
drawDirection = false; // may need to draw the arc anticlockwise
// find the correct quadrant for circle center
if (sinA90 < 0) {
    if (angle < 0) {
        angle = Math.PI + angle; // add 180 to move us to the 3 quadrant
    } else {
        angle = Math.PI - angle; // move back into the 2nd quadrant
        radDirection = -1;
        drawDirection = true;
    }
} else {
    if (angle > 0) {
        radDirection = -1;
        drawDirection = true;
    }
}
halfAngle = angle / 2;
// get distance from corner to point where round corner touches line
lenOut = Math.abs(Math.cos(halfAngle) * radius / Math.sin(halfAngle));
if (lenOut > Math.min(v1.len / 2, v2.len / 2)) { // fix if longer than half line
length
    lenOut = Math.min(v1.len / 2, v2.len / 2);
    // ajust the radius of corner rounding to fit
    cRadius = Math.abs(lenOut * Math.sin(halfAngle) / Math.cos(halfAngle));
} else {
    cRadius = radius;
}
x = p2.x + v2.nx * lenOut; // move out from corner along second line to point where
rounded circle touches
y = p2.y + v2.ny * lenOut;
x += -v2.ny * cRadius * radDirection; // move away from line to circle center
y += v2.nx * cRadius * radDirection;
// x,y is the rounded corner circle center
ctx.arc(x, y, cRadius, v1.ang + Math.PI / 2 * radDirection, v2.ang - Math.PI / 2 *
radDirection, drawDirection); // draw the arc clockwise
p1 = p2;
p2 = p3;
}
ctx.closePath();
}

```

Lea Poligonos en línea: <https://riptutorial.com/es/html5-canvas/topic/5493/poligonos>

Capítulo 16: Ruta (solo sintaxis)

Sintaxis

- `context.beginPath ()`
- `context.moveTo (startX, startY)`
- `context.lineTo (endX, endY)`
- `context.arc (centerX, centerY, radio, startingRadianAngle, endingRadianAngle)`
- `context.quadraticCurveTo (controlX, controlY, endX, endY)`
- `context.bezierCurveTo (controlX1, controlY1, controlX2, controlY2, endX, endY)`
- `context.arcTo (pointX1, pointY1, pointX2, pointY2, radio)`
- `context.rect (leftX, topY, width, height);`
- `context.closePath ()`

Examples

Resumen de los comandos básicos de trazado de trayectos: líneas y curvas.

=====

TODO: vincule cada uno de los comandos de dibujo a continuación a sus ejemplos individuales. No sé cómo hacerlo, ya que los enlaces a los ejemplos individuales apuntan hacia la carpeta "borrador".

TODO: Agregar ejemplos para estos comandos de "acción" de ruta: trazo (), relleno (), clip ()

=====

Camino

Una ruta define un conjunto de líneas y curvas que se pueden dibujar visiblemente en el lienzo.

Un trazado no se dibuja automáticamente en el lienzo. Pero las líneas y curvas del camino se pueden dibujar en el lienzo utilizando un trazo con estilo. Y la forma creada por las líneas y curvas también se puede rellenar con un relleno estilizable.

Las rutas tienen usos más allá del dibujo en el lienzo:

- Prueba de golpe si una coordenada x, y está dentro de la forma de la trayectoria.
- Definición de una región de recorte donde solo serán visibles los dibujos dentro de la región de recorte. Cualquier dibujo fuera de la región de recorte no se dibujará (== transparente), similar al desbordamiento de CSS.

Los comandos básicos de dibujo de ruta son:

- `beginPath`
- `mover a`

- `lineTo`
- `arco`
- `cuadrática`
- `bezierCurveTo`
- `arco a`
- `rect`
- `closePath`

Descripción de los comandos básicos de dibujo:

beginPath

```
context.beginPath()
```

Comienza a ensamblar un nuevo conjunto de comandos de ruta y también descarta cualquier ruta previamente ensamblada.

El descarte es un punto importante ya menudo pasado por alto. Si no comienza una nueva ruta, cualquier comando de ruta emitido previamente se volverá a dibujar.

También mueve el "lápiz" del dibujo al origen superior izquierdo del lienzo (== coordenada [0,0]).

mover a

```
context.moveTo(startX, startY)
```

Mueve la ubicación actual del lápiz a la coordenada [startX, startY].

Por defecto, todos los dibujos de ruta están conectados entre sí. Entonces, el punto final de una línea o curva es el punto de inicio de la siguiente línea o curva. Esto puede hacer que se dibuje una línea inesperada que conecta dos dibujos adyacentes. El comando `context.moveTo` básicamente "toma el lápiz del dibujo" y lo coloca en una nueva coordenada para que no se dibuje la línea de conexión automática.

lineTo

```
context.lineTo(endX, endY)
```

Dibuja un segmento de línea desde la ubicación actual del lápiz para coordinar [endX, endY]

Puede ensamblar varios comandos `.lineTo` para dibujar una polilínea. Por ejemplo, puedes ensamblar 3 segmentos de línea para formar un triángulo.

arco

```
context.arc(centerX, centerY, radius, startingRadianAngle, endingRadianAngle)
```

Dibuja un arco circular dado el punto central, el radio y los ángulos de inicio y finalización. Los ángulos se expresan como radianes. Para convertir grados en radianes, puede utilizar esta fórmula: $\text{radians} = \text{degrees} * \text{Math.PI} / 180; .$

El ángulo 0 mira directamente hacia la derecha desde el centro del arco. Para dibujar un círculo completo, puede hacer $\text{endingAngle} = \text{startingAngle} + 360$ grados (360 grados == $\text{Math.PI} * 2$):
`context.arc(10,10,20,0,Math.PI*2);`

Por defecto, el arco se dibuja en el sentido de las agujas del reloj. Un parámetro opcional [true | false] indica que el arco se `context.arc(10,10,20,0,Math.PI*2,true)` sentido contrario a las agujas del reloj: `context.arc(10,10,20,0,Math.PI*2,true)`

cuadrática

```
context.quadraticCurveTo(controlX, controlY, endingX, endingY)
```

Dibuja una curva cuadrática que comienza en la ubicación actual de la pluma hasta una coordenada final dada. Otra coordenada de control dada determina la forma (curvatura) de la curva.

bezierCurveTo

```
context.bezierCurveTo(control1X, control1Y, control2X, control2Y, endingX, endingY)
```

Dibuja una curva de Bézier cúbica que comienza en la ubicación de la pluma actual hasta una coordenada final dada. Otras 2 coordenadas de control dadas determinan la forma (curvatura) de la curva.

arco a

```
context.arcTo(pointX1, pointY1, pointX2, pointY2, radius);
```

Dibuja un arco circular con un radio dado. El arco se dibuja en el sentido de las agujas del reloj dentro de la cuña formada por la ubicación actual del lápiz y se le asignan dos puntos: Punto1 y Punto2.

Una línea que conecta la ubicación actual del lápiz y el inicio del arco se dibuja automáticamente antes del arco.

rect

```
context.rect(leftX, topY, width, height)
```

Dibuja un rectángulo dado una esquina superior izquierda y un ancho y alto.

El `context.rect` es un comando de dibujo único porque agrega rectángulos desconectados. Estos

rectángulos desconectados no se conectan automáticamente por líneas.

closePath

```
context.closePath()
```

Dibuja una línea desde la ubicación actual del lápiz hasta la coordenada de la ruta de inicio.

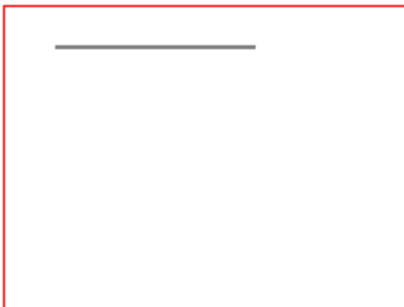
Por ejemplo, si dibujas 2 líneas que forman 2 patas de un triángulo, `closePath` "cerrará" el triángulo dibujando la tercera pata del triángulo desde el punto final de la 2ª pata hasta el punto inicial de la primera pata.

El nombre de este comando a menudo hace que sea mal entendido. `context.closePath` NO es un delimitador final para `context.beginPath`. Nuevamente, el comando `closePath` dibuja una línea, no "cierra" una `beginPath`.

lineTo (un comando de ruta)

```
context.lineTo(endX, endY)
```

Dibuja un segmento de línea desde la ubicación actual del lápiz para coordinar [endX, endY]



```
<!doctype html>
<html>
<head>
<style>
  body{ background-color:white; }
  #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

  // get a reference to the canvas element and it's context
  var canvas=document.getElementById("canvas");
  var ctx=canvas.getContext("2d");

  // arguments
  var startX=25;
  var startY=20;
  var endX=125;
  var endY=20;

  // Draw a single line segment drawn using "moveTo" and "lineTo" commands
  ctx.beginPath();
```

```

    ctx.moveTo(startX, startY);
    ctx.lineTo(endX, endY);
    ctx.stroke();

}); // end window.onload
</script>
</head>
<body>
    <canvas id="canvas" width=200 height=150></canvas>
</body>
</html>

```

Puede ensamblar varios comandos `.lineTo` para dibujar una polilínea. Por ejemplo, puedes ensamblar 3 segmentos de línea para formar un triángulo.



```

<!doctype html>
<html>
<head>
<style>
    body{ background-color:white; }
    #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

    // get a reference to the canvas element and it's context
    var canvas=document.getElementById("canvas");
    var ctx=canvas.getContext("2d");

    // arguments
    var topVertexX=50;
    var topVertexY=20;
    var rightVertexX=75;
    var rightVertexY=70;
    var leftVertexX=25;
    var leftVertexY=70;

    // A set of line segments drawn to form a triangle using
    //     "moveTo" and multiple "lineTo" commands
    ctx.beginPath();
    ctx.moveTo(topVertexX,topVertexY);
    ctx.lineTo(rightVertexX,rightVertexY);
    ctx.lineTo(leftVertexX,leftVertexY);
    ctx.lineTo(topVertexX,topVertexY);
    ctx.stroke();

}); // end window.onload
</script>
</head>

```

```
<body>
  <canvas id="canvas" width=200 height=150></canvas>
</body>
</html>
```

arco (un comando de ruta)

```
context.arc(centerX, centerY, radius, startingRadianAngle, endingRadianAngle)
```

Dibuja un arco circular dado el punto central, el radio y los ángulos de inicio y finalización. Los ángulos se expresan como radianes. Para convertir grados en radianes, puede utilizar esta fórmula: $\text{radians} = \text{degrees} * \text{Math.PI} / 180; .$

El ángulo 0 mira directamente hacia la derecha desde el centro del arco.

Por defecto, el arco se dibuja en el sentido de las agujas del reloj. Un parámetro opcional [true | false] indica que el arco se `context.arc(10,10,20,0,Math.PI*2,true)` sentido contrario a las agujas del reloj: `context.arc(10,10,20,0,Math.PI*2,true)`



```
<!doctype html>
<html>
<head>
<style>
  body{ background-color:white; }
  #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

  // get a reference to the canvas element and its context
  var canvas=document.getElementById("canvas");
  var ctx=canvas.getContext("2d");

  // arguments
  var centerX=50;
  var centerY=50;
  var radius=30;
  var startingRadianAngle=Math.PI*2*; // start at 90 degrees == centerY+radius
  var endingRadianAngle=Math.PI*2*.75; // end at 270 degrees (==PI*2*.75 in radians)

  // A partial circle (i.e. arc) drawn using the "arc" command
  ctx.beginPath();
  ctx.arc(centerX, centerY, radius, startingRadianAngle, endingRadianAngle);
  ctx.stroke();
```

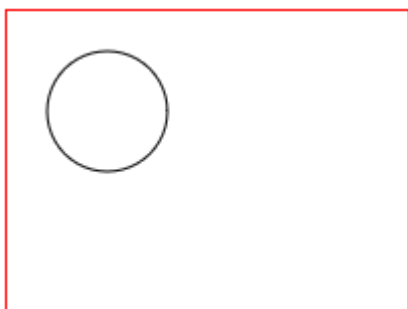


```

}); // end window.onload
</script>
</head>
<body>
  <canvas id="canvas" width=200 height=150></canvas>
</body>
</html>

```

Para dibujar un círculo completo, puede hacer $\text{endingAngle} = \text{startingAngle} + 360$ grados (360 grados == $\text{Math.PI}2$).



```

<!doctype html>
<html>
<head>
<style>
  body{ background-color:white; }
  #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

  // get a reference to the canvas element and its context
  var canvas=document.getElementById("canvas");
  var ctx=canvas.getContext("2d");

  // arguments
  var centerX=50;
  var centerY=50;
  var radius=30;
  var startingRadianAngle=0;          // start at 0 degrees
  var endingRadianAngle=Math.PI*2; // end at 360 degrees (==PI*2 in radians)

  // A complete circle drawn using the "arc" command
  ctx.beginPath();
  ctx.arc(centerX, centerY, radius, startingRadianAngle, endingRadianAngle);
  ctx.stroke();

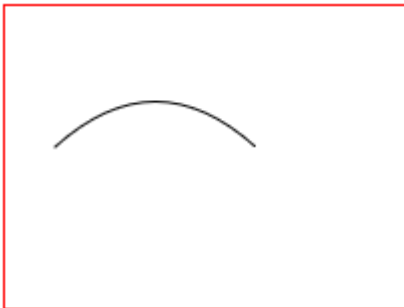
}); // end window.onload
</script>
</head>
<body>
  <canvas id="canvas" width=200 height=150></canvas>
</body>
</html>

```

quadraticCurveTo (un comando de ruta)

```
context.quadraticCurveTo(controlX, controlY, endingX, endingY)
```

Dibuja una curva cuadrática que comienza en la ubicación actual de la pluma hasta una coordenada final dada. Otra coordenada de control dada determina la forma (curvatura) de la curva.



```
<!doctype html>
<html>
<head>
<style>
  body{ background-color:white; }
  #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

  // get a reference to the canvas element and it's context
  var canvas=document.getElementById("canvas");
  var ctx=canvas.getContext("2d");

  // arguments
  var startX=25;
  var startY=70;
  var controlX=75;
  var controlY=25;
  var endX=125;
  var endY=70;

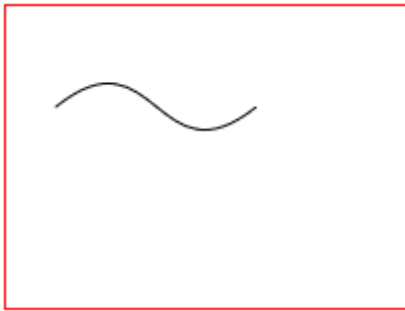
  // A quadratic curve drawn using "moveTo" and "quadraticCurveTo" commands
  ctx.beginPath();
  ctx.moveTo(startX,startY);
  ctx.quadraticCurveTo(controlX,controlY,endX,endY);
  ctx.stroke();

}); // end window.onload
</script>
</head>
<body>
  <canvas id="canvas" width=200 height=150></canvas>
</body>
</html>
```

bezierCurveTo (un comando de ruta)

```
context.bezierCurveTo(control1X, control1Y, control2X, control2Y, endingX, endingY)
```

Dibuja una curva de Bézier cúbica que comienza en la ubicación de la pluma actual hasta una coordenada final dada. Otras 2 coordenadas de control dadas determinan la forma (curvatura) de la curva.



```
<!doctype html>
<html>
<head>
<style>
  body{ background-color:white; }
  #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

  // get a reference to the canvas element and it's context
  var canvas=document.getElementById("canvas");
  var ctx=canvas.getContext("2d");

  // arguments
  var startX=25;
  var startY=50;
  var controlX1=75;
  var controlY1=10;
  var controlX2=75;
  var controlY2=90;
  var endX=125;
  var endY=50;

  // A cubic bezier curve drawn using "moveTo" and "bezierCurveTo" commands
  ctx.beginPath();
  ctx.moveTo(startX,startY);
  ctx.bezierCurveTo(controlX1,controlY1,controlX2,controlY2,endX,endY);
  ctx.stroke();

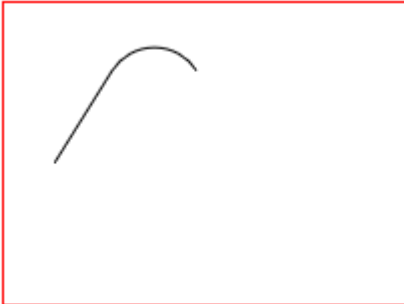
}); // end window.onload
</script>
</head>
<body>
  <canvas id="canvas" width=200 height=150></canvas>
</body>
</html>
```

arcTo (un comando de ruta)

```
context.arcTo(pointX1, pointY1, pointX2, pointY2, radius);
```

Dibuja un arco circular con un radio dado. El arco se dibuja en el sentido de las agujas del reloj dentro de la cuña formada por la ubicación actual del lápiz y se le asignan dos puntos: Punto1 y Punto2.

Una línea que conecta la ubicación actual del lápiz y el inicio del arco se dibuja automáticamente antes del arco.



```
<!doctype html>
<html>
<head>
<style>
  body{ background-color:white; }
  #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

  // get a reference to the canvas element and it's context
  var canvas=document.getElementById("canvas");
  var ctx=canvas.getContext("2d");

  // arguments
  var pointX0=25;
  var pointY0=80;
  var pointX1=75;
  var pointY1=0;
  var pointX2=125;
  var pointY2=80;
  var radius=25;

  // A circular arc drawn using the "arcTo" command. The line is automatically drawn.
  ctx.beginPath();
  ctx.moveTo(pointX0,pointY0);
  ctx.arcTo(pointX1, pointY1, pointX2, pointY2, radius);
  ctx.stroke();

}); // end window.onload
</script>
</head>
<body>
  <canvas id="canvas" width=200 height=150></canvas>
</body>
</html>
```

rect (un comando de ruta)

```
context.rect(leftX, topY, width, height)
```

Dibuja un rectángulo dado una esquina superior izquierda y un ancho y alto.



```
<!doctype html>
<html>
<head>
<style>
  body{ background-color:white; }
  #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

  // get a reference to the canvas element and it's context
  var canvas=document.getElementById("canvas");
  var ctx=canvas.getContext("2d");

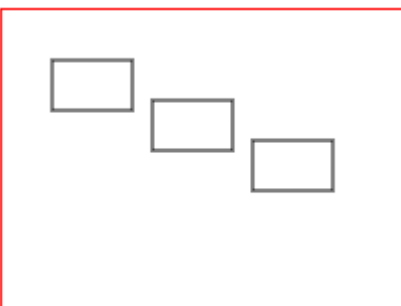
  // arguments
  var leftX=25;
  var topY=25;
  var width=40;
  var height=25;

  // A rectangle drawn using the "rect" command.
  ctx.beginPath();
  ctx.rect(leftX, topY, width, height);
  ctx.stroke();

}); // end window.onload
</script>
</head>
<body>
  <canvas id="canvas" width=200 height=150></canvas>
</body>
</html>
```

El `context.rect` es un comando de dibujo único porque agrega rectángulos desconectados.

Estos rectángulos desconectados no se conectan automáticamente por líneas.



```

<!doctype html>
<html>
<head>
<style>
  body{ background-color:white; }
  #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

  // get a reference to the canvas element and it's context
  var canvas=document.getElementById("canvas");
  var ctx=canvas.getContext("2d");

  // arguments
  var leftX=25;
  var topY=25;
  var width=40;
  var height=25;

  // Multiple rectangles drawn using the "rect" command.
  ctx.beginPath();
  ctx.rect(leftX, topY, width, height);
  ctx.rect(leftX+50, topY+20, width, height);
  ctx.rect(leftX+100, topY+40, width, height);
  ctx.stroke();

}); // end window.onload
</script>
</head>
<body>
  <canvas id="canvas" width=200 height=150></canvas>
</body>
</html>

```

closePath (un comando de ruta)

```
context.closePath()
```

Dibuja una línea desde la ubicación actual del lápiz hasta la coordenada de la ruta de inicio.

Por ejemplo, si dibuja 2 líneas que forman 2 patas de un triángulo, closePath "cerrará" el triángulo dibujando la tercera pata del triángulo desde el punto final de la 2a pata hasta el punto inicial de la primera pata.

¡Un error de explicación!

El nombre de este comando a menudo hace que sea mal entendido.

`context.closePath` NO es un delimitador final para `context.beginPath`.

Nuevamente, el comando closePath dibuja una línea, no "cierra" una beginPath.

Este ejemplo dibuja 2 patas de un triángulo y usa `closePath` para completar (¿cerrar?) El triángulo dibujando la tercera pata. Lo que `closePath` realmente está haciendo es dibujar una línea desde el

punto final del segundo tramo hasta el punto inicial del primer tramo.



```
<!doctype html>
<html>
<head>
<style>
  body{ background-color:white; }
  #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

  // get a reference to the canvas element and it's context
  var canvas=document.getElementById("canvas");
  var ctx=canvas.getContext("2d");

  // arguments
  var topVertexX=50;
  var topVertexY=50;
  var rightVertexX=75;
  var rightVertexY=75;
  var leftVertexX=25;
  var leftVertexY=75;

  // A set of line segments drawn to form a triangle using
  // "moveTo" and multiple "lineTo" commands
  ctx.beginPath();
  ctx.moveTo(topVertexX,topVertexY);
  ctx.lineTo(rightVertexX,rightVertexY);
  ctx.lineTo(leftVertexX,leftVertexY);

  // closePath draws the 3rd leg of the triangle
  ctx.closePath()

  ctx.stroke();

}); // end window.onload
</script>
</head>
<body>
  <canvas id="canvas" width=200 height=150></canvas>
</body>
</html>
```

beginPath (un comando de ruta)

```
context.beginPath()
```

Comienza a ensamblar un nuevo conjunto de comandos de ruta y también descarta cualquier ruta previamente ensamblada.

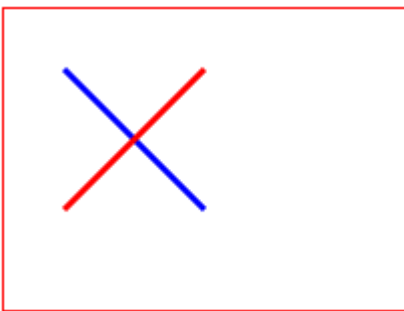
También mueve el "lápiz" del dibujo al origen superior izquierdo del lienzo (== coordenada [0,0]).

Aunque es opcional, SIEMPRE debe iniciar una ruta con `beginPath`

El descarte es un punto importante ya menudo pasado por alto. Si no comienza una nueva ruta con `beginPath`, cualquier comando de ruta emitido previamente se volverá a dibujar.

Estas dos demostraciones intentan dibujar una "X" con un trazo rojo y un trazo azul.

Esta primera demostración utiliza correctamente `beginPath` para iniciar su segundo trazo rojo. El resultado es que la "X" tiene correctamente un trazo rojo y uno azul.



```
<!doctype html>
<html>
<head>
<style>
  body{ background-color:white; }
  #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

  // get a reference to the canvas element and it's context
  var canvas=document.getElementById("canvas");
  var ctx=canvas.getContext("2d");

  // draw a blue line
  ctx.beginPath();
  ctx.moveTo(30,30);
  ctx.lineTo(100,100);
  ctx.strokeStyle='blue';
  ctx.lineWidth=3;
  ctx.stroke();

  // draw a red line
  ctx.beginPath(); // Important to begin a new path!
  ctx.moveTo(100,30);
  ctx.lineTo(30,100);
  ctx.strokeStyle='red';
  ctx.lineWidth=3;
  ctx.stroke();

}); // end window.onload
</script>
```



```
</head>
<body>
  <canvas id="canvas" width=200 height=150></canvas>
</body>
</html>
```

Esta segunda demostración deja incorrectamente a `beginPath` en el segundo trazo. El resultado es que la "X" tiene incorrectamente ambos trazos rojos.

El segundo `stroke()` es dibuja el segundo trazo rojo.

Pero sin un segundo `beginPath`, ese mismo segundo `stroke()` también **vuelve a dibujar** incorrectamente el primer trazo.

Dado que el segundo `stroke()` ahora tiene un estilo rojo, el primer trazo azul se **sobrescribe** con un trazo rojo de color incorrecto.



```
<!doctype html>
<html>
<head>
<style>
  body{ background-color:white; }
  #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

  // get a reference to the canvas element and it's context
  var canvas=document.getElementById("canvas");
  var ctx=canvas.getContext("2d");

  // draw a blue line
  ctx.beginPath();
  ctx.moveTo(30,30);
  ctx.lineTo(100,100);
  ctx.strokeStyle='blue';
  ctx.lineWidth=3;
  ctx.stroke();

  // draw a red line
  // Note: The necessary 'beginPath' is missing!
  ctx.moveTo(100,30);
  ctx.lineTo(30,100);
  ctx.strokeStyle='red';
  ctx.lineWidth=3;
  ctx.stroke();
```

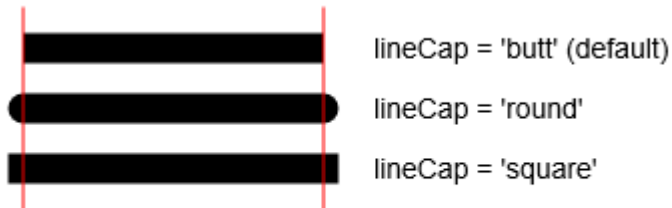
```
}); // end window.onload
</script>
</head>
<body>
  <canvas id="canvas" width=200 height=150></canvas>
</body>
</html>
```

lineCap (un atributo de estilo de ruta)

```
context.lineCap=capStyle // butt (default), round, square
```

Establece el estilo de tapa de los puntos de inicio de línea y los puntos finales.

- **Butt** , el estilo predeterminado de lineCap, muestra mayúsculas cuadradas que no se extienden más allá de los puntos de inicio y final de la línea.
- **redondeado** , muestra mayúsculas redondeadas que se extienden más allá de los puntos inicial y final de la línea.
- **cuadrado** , muestra casquillos cuadrados que se extienden más allá de los puntos inicial y final de la línea.



butt (default) stays inside line start & end
round & square extend beyond line start & end

```
<!doctype html>
<html>
<head>
<style>
  body{ background-color:white; }
  #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

  // canvas related variables
  var canvas=document.getElementById("canvas");
  var ctx=canvas.getContext("2d");
  ctx.lineWidth=15;

  // lineCap default: butt
  ctx.lineCap='butt';
  drawLine(50,40,200,40);

  // lineCap: round
  ctx.lineCap='round';
  drawLine(50,70,200,70);
```

```

// lineCap: square
ctx.lineCap='square';
drawLine(50,100,200,100);

// utility function to draw a line
function drawLine(startX,startY,endX,endY){
    ctx.beginPath();
    ctx.moveTo(startX,startY);
    ctx.lineTo(endX,endY);
    ctx.stroke();
}

// For demo only,
// Rulers to show which lineCaps extend beyond endpoints
ctx.lineWidth=1;
ctx.strokeStyle='red';
drawLine(50,20,50,120);
drawLine(200,20,200,120);

}); // end window.onload
</script>
</head>
<body>
    <canvas id="canvas" width=300 height=200></canvas>
</body>
</html>

```

lineJoin (un atributo de estilo de ruta)

```
context.lineJoin=joinStyle // miter (default), round, bevel
```

Establece el estilo utilizado para conectar los segmentos de línea adyacentes.

- **Mitre** , el valor predeterminado, une segmentos de línea con una unión afilada.
- **redondear** , une segmentos de línea con una junta redondeada.
- **Bisel** , une segmentos de línea con una junta embotada.



lineJoin = 'miter' (default)

lineJoin = 'round'

lineJoin = 'bevel'

```

<!doctype html>
<html>
<head>
<style>
    body{ background-color:white; }
    #canvas{border:1px solid red; }
</style>

```

```

<script>
window.onload=(function(){

    // canvas related variables
    var canvas=document.getElementById("canvas");
    var ctx=canvas.getContext("2d");
    ctx.lineWidth=15;

    // lineJoin: miter (default)
    ctx.lineJoin='miter';
    drawPolyline(50,30);

    // lineJoin: round
    ctx.lineJoin='round';
    drawPolyline(50,80);

    // lineJoin: bevel
    ctx.lineJoin='bevel';
    drawPolyline(50,130);

    // utility to draw polyline
    function drawPolyline(x,y){
        ctx.beginPath();
        ctx.moveTo(x,y);
        ctx.lineTo(x+30,y+30);
        ctx.lineTo(x+60,y);
        ctx.lineTo(x+90,y+30);
        ctx.stroke();
    }

}); // end window.onload
</script>
</head>
<body>
    <canvas id="canvas" width=300 height=200></canvas>
</body>
</html>

```

strokeStyle (un atributo de estilo de ruta)

```
context.strokeStyle=color
```

Establece el color que se utilizará para trazar el contorno de la ruta actual.

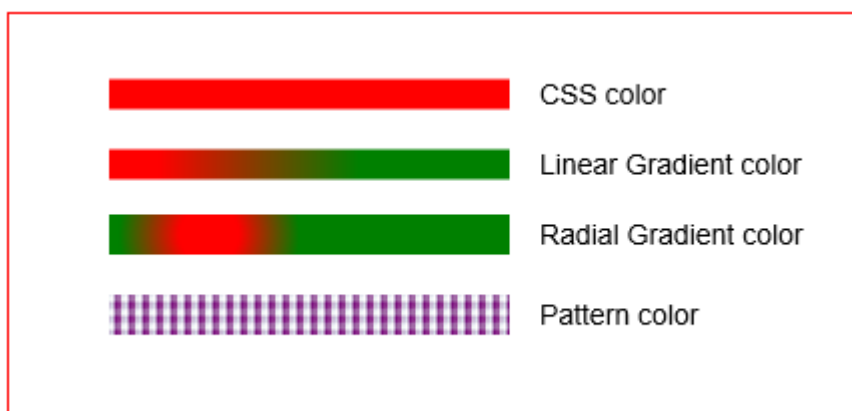
Estas son opciones de `color` (estas deben ser citadas):

- **Un CSS con nombre de color** , por ejemplo `context.strokeStyle='red'`
- **Un color hexadecimal** , por ejemplo `context.strokeStyle='#FF0000'`
- **Un color RGB** , por ejemplo `context.strokeStyle='rgb(red,green,blue)'` donde rojo, verde y azul son números enteros 0-255 que indican la intensidad de cada color componente.
- **Un color HSL** , por ejemplo `context.strokeStyle='hsl(hue,saturation,lightness)'` donde el tono es un número entero 0-360 en la rueda de color y la saturación y la luminosidad son porcentajes (0-100%) que indican la fuerza de cada componente .

- **Un color HSLA** , por ejemplo `context.strokeStyle='hsl(hue,saturation,lightness,alpha)'` donde el tono es un número entero 0-360 en la rueda de color y la saturación y la luminosidad son porcentajes (0-100%) que indican la fuerza de cada componente y alfa es un valor decimal de 0.00-1.00 que indica la opacidad.

También puede especificar estas opciones de color (estas opciones son objetos creados por el contexto):

- **Un degradado lineal** que es un objeto de degradado lineal creado con `context.createLinearGradient`
- **Un degradado radial** que es un objeto de degradado radial creado con `context.createRadialGradient`
- **Un patrón** que es un objeto de patrón creado con `context.createPattern`



```

<!doctype html>
<html>
<head>
<style>
  body{ background-color:white; }
  #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

  // canvas related variables
  var canvas=document.getElementById("canvas");
  var ctx=canvas.getContext("2d");
  ctx.lineWidth=15;

  // stroke using a CSS color: named, RGB, HSL, etc
  ctx.strokeStyle='red';
  drawLine(50,40,250,40);

  // stroke using a linear gradient
  var gradient = ctx.createLinearGradient(75,75,175,75);
  gradient.addColorStop(0,'red');
  gradient.addColorStop(1,'green');
  ctx.strokeStyle=gradient;
  drawLine(50,75,250,75);

  // stroke using a radial gradient
  var gradient = ctx.createRadialGradient(100,110,15,100,110,45);

```

```

gradient.addColorStop(0, 'red');
gradient.addColorStop(1, 'green');
ctx.strokeStyle=gradient;
ctx.lineWidth=20;
drawLine(50,110,250,110);

// stroke using a pattern
var patternImage=new Image();
patternImage.onload=function(){
    var pattern = ctx.createPattern(patternImage, 'repeat');
    ctx.strokeStyle=pattern;
    drawLine(50,150,250,150);
}
patternImage.src='https://dl.dropboxusercontent.com/u/139992952/stackoverflow/BooMul.png';

// for demo only, draw labels by each stroke
ctx.textBaseline='middle';
ctx.font='14px arial';
ctx.fillText('CSS color',265,40);
ctx.fillText('Linear Gradient color',265,75);
ctx.fillText('Radial Gradient color',265,110);
ctx.fillText('Pattern color',265,150);

// utility to draw a line
function drawLine(startX,startY,endX,endY){
    ctx.beginPath();
    ctx.moveTo(startX,startY);
    ctx.lineTo(endX,endY);
    ctx.stroke();
}

}); // end window.onload
</script>
</head>
<body>
    <canvas id="canvas" width=425 height=200></canvas>
</body>
</html>

```

fillStyle (un atributo de estilo de ruta)

```
context.fillStyle=color
```

Establece el color que se utilizará para rellenar el interior de la ruta actual.

Estas son opciones de color (estas deben ser citadas):

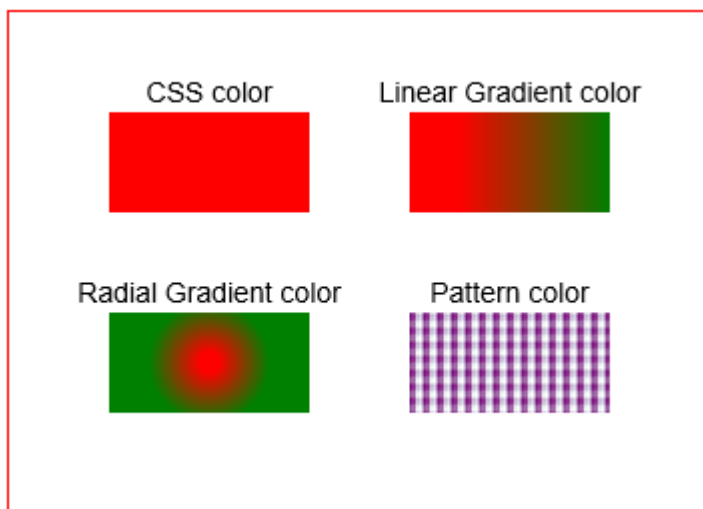
- **Un CSS con nombre de color** , por ejemplo `context.fillStyle='red'`
- **Un color hexadecimal** , por ejemplo `context.fillStyle='#FF0000'`
- **Un color RGB** , por ejemplo `context.fillStyle='rgb(red,green,blue)'` donde rojo, verde y azul son números enteros 0-255 que indican la intensidad de cada color del componente.
- **Un color HSL** , por ejemplo `context.fillStyle='hsl(hue,saturation,lightness)'` donde el tono es un número entero 0-360 en la rueda de color y la saturación y la luminosidad son

porcentajes (0-100%) que indican la fuerza de cada componente .

- **Un color HSLA** , por ejemplo `context.fillStyle='hsl(hue,saturation,lightness,alpha)'` donde el tono es un número entero 0-360 en la rueda de color y la saturación y la luminosidad son porcentajes (0-100%) que indican la fuerza de cada componente y alfa es un valor decimal de 0.00-1.00 que indica la opacidad.

También puede especificar estas opciones de color (estas opciones son objetos creados por el contexto):

- **Un degradado lineal** que es un objeto de degradado lineal creado con `context.createLinearGradient`
- **Un degradado radial** que es un objeto de degradado radial creado con `context.createRadialGradient`
- **Un patrón** que es un objeto de patrón creado con `context.createPattern`



```
<!doctype html>
<html>
<head>
<style>
  body{ background-color:white; }
  #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

  // canvas related variables
  var canvas=document.getElementById("canvas");
  var ctx=canvas.getContext("2d");

  // stroke using a CSS color: named, RGB, HSL, etc
  ctx.fillStyle='red';
  ctx.fillRect(50,50,100,50);

  // stroke using a linear gradient
  var gradient = ctx.createLinearGradient(225,50,300,50);
  gradient.addColorStop(0,'red');
  gradient.addColorStop(1,'green');
  ctx.fillStyle=gradient;
```

```

ctx.fillRect(200,50,100,50);

// stroke using a radial gradient
var gradient = ctx.createRadialGradient(100,175,5,100,175,30);
gradient.addColorStop(0,'red');
gradient.addColorStop(1,'green');
ctx.fillStyle=gradient;
ctx.fillRect(50,150,100,50);

// stroke using a pattern
var patternImage=new Image();
patternImage.onload=function(){
    var pattern = ctx.createPattern(patternImage,'repeat');
    ctx.fillStyle=pattern;
    ctx.fillRect(200,150,100,50);
}
patternImage.src='http://i.stack.imgur.com/ixrWe.png';

// for demo only, draw labels by each stroke
ctx.fillStyle='black';
ctx.textAlign='center';
ctx.textBaseline='middle';
ctx.font='14px arial';
ctx.fillText('CSS color',100,40);
ctx.fillText('Linear Gradient color',250,40);
ctx.fillText('Radial Gradient color',100,140);
ctx.fillText('Pattern color',250,140);

}); // end window.onload
</script>
</head>
<body>
    <canvas id="canvas" width=350 height=250></canvas>
</body>
</html>

```

lineWidth (un atributo de estilo de ruta)

```
context.lineWidth=lineWidth
```

Establece el ancho de la línea que trazará el contorno del trazado.




```

<!doctype html>
<html>
<head>
<style>
  body{ background-color:white; }
  #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

  // canvas related variables
  var canvas=document.getElementById("canvas");
  var ctx=canvas.getContext("2d");

  ctx.lineWidth=1;
  drawPolyline(50,50);

  ctx.lineWidth=5;
  drawPolyline(50,100);

  ctx.lineWidth=10;
  drawPolyline(50,150);

  // utility to draw a polyline
  function drawPolyline(x,y){
    ctx.beginPath();
    ctx.moveTo(x,y);
    ctx.lineTo(x+30,y+30);
    ctx.lineTo(x+60,y);
    ctx.lineTo(x+90,y+30);
    ctx.stroke();
  }

}); // end window.onload
</script>
</head>
<body>
  <canvas id="canvas" width=350 height=250></canvas>
</body>
</html>

```

shadowColor, shadowBlur, shadowOffsetX, shadowOffsetY (atributos de estilo de ruta)

```

shadowColor = color           // CSS color
shadowBlur = width            // integer blur width
shadowOffsetX = distance      // shadow is moved horizontally by this offset
shadowOffsetY = distance      // shadow is moved vertically by this offset

```

Este conjunto de atributos agregará una sombra alrededor de una ruta.

Tanto los caminos rellenos como los trazados pueden tener una sombra.

La sombra es más oscura (opaca) en el perímetro de la trayectoria y se vuelve gradualmente más clara a medida que se aleja del perímetro de la trayectoria.

- **shadowColor** indica qué color CSS se usará para crear la sombra.
- **shadowBlur** es la distancia sobre la cual la sombra se extiende hacia afuera desde el camino.
- **shadowOffsetX** es una distancia por la cual la sombra se desplaza horizontalmente alejándose del camino. Una distancia positiva mueve la sombra hacia la derecha, una distancia negativa mueve la sombra hacia la izquierda.
- **shadowOffsetY** es una distancia por la cual la sombra se desplaza verticalmente alejándose del camino. Una distancia positiva mueve la sombra hacia abajo, una distancia negativa mueve la sombra hacia arriba.

Acerca de shadowOffsetX y shadowOffsetY

Es importante tener en cuenta que *toda la sombra se desplaza en su totalidad* . Esto hará que parte de la sombra se desplace por debajo de los caminos rellenos y, por lo tanto, parte de la sombra no será visible.

Sobre trazos sombreados

Al sombrear un trazo, tanto el interior como el exterior del trazo están sombreados. La sombra es más oscura en el trazo y se aclara cuando la sombra se extiende hacia afuera en ambas direcciones desde el trazo.

Desactivar el sombreado cuando haya terminado

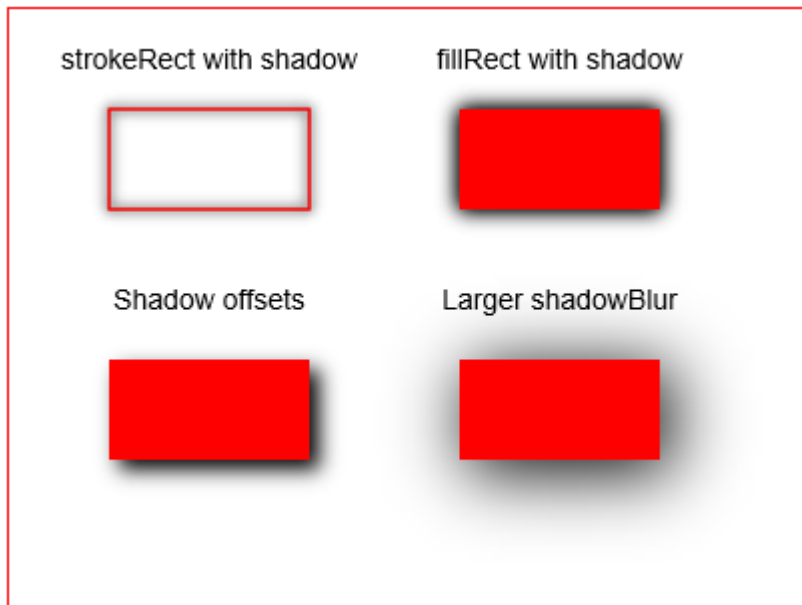
Después de dibujar sus sombras, es posible que desee desactivar el sombreado para dibujar más caminos. Para desactivar el sombreado, configure el `shadowColor` la `shadowColor` en transparente.

```
context.shadowColor = 'rgba(0,0,0,0)';
```

Consideraciones de rendimiento

Las sombras (como los gradientes) requieren cálculos extensos y, por lo tanto, debe usar sombras con moderación.

Tenga especial cuidado al animar, porque dibujar sombras muchas veces por segundo tendrá un gran impacto en el rendimiento. Una solución alternativa si necesita animar rutas sombreadas es crear previamente la ruta sombreada en un segundo "lienzo de sombras". El lienzo de la sombra es un lienzo normal que se crea en la memoria con `document.createElement` , no se agrega al DOM (solo es un lienzo provisional). Luego dibuja el lienzo de sombras en el lienzo principal. Esto es mucho más rápido porque los cálculos de sombra no necesitan realizarse muchas veces por segundo. Todo lo que estás haciendo es copiar un lienzo creado previamente en tu lienzo visible.



```

<!doctype html>
<html>
<head>
<style>
  body{ background-color:white; }
  #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

  // canvas related variables
  var canvas=document.getElementById("canvas");
  var ctx=canvas.getContext("2d");

  // shadowed stroke
  ctx.shadowColor='black';
  ctx.shadowBlur=6;
  ctx.strokeStyle='red';
  ctx.strokeRect(50,50,100,50);
  // darken the shadow by stroking a second time
  ctx.strokeRect(50,50,100,50);

  // shadowed fill
  ctx.shadowColor='black';
  ctx.shadowBlur=10;
  ctx.fillStyle='red';
  ctx.fillRect(225,50,100,50);
  // darken the shadow by stroking a second time
  ctx.fillRect(225,50,100,50);

  // the shadow offset rightward and downward
  ctx.shadowColor='black';
  ctx.shadowBlur=10;
  ctx.shadowOffsetX=5;
  ctx.shadowOffsetY=5;
  ctx.fillStyle='red';
  ctx.fillRect(50,175,100,50);

  // a wider blur (==extends further from the path)
  ctx.shadowColor='black';
  ctx.shadowBlur=35;

```

```

ctx.fillStyle='red';
ctx.fillRect (225,175,100,50);

// always clean up! Turn off shadowing
ctx.shadowColor='rgba(0,0,0,0)';

}); // end window.onload
</script>
</head>
<body>
  <canvas id="canvas" width=400 height=300></canvas>
</body>
</html>

```

createLinearGradient (crea un objeto de estilo de ruta)

```

var gradient = createLinearGradient( startX, startY, endX, endY )
gradient.addColorStop(gradientPercentPosition, CssColor)
gradient.addColorStop(gradientPercentPosition, CssColor)
[optionally add more color stops to add to the variety of the gradient]

```

Crea un degradado lineal reutilizable (objeto).

El objeto se puede asignar a cualquier `strokeStyle` y / o `fillStyle` .

Luego, el trazo () o el relleno () colorearán la ruta con los colores degradados del objeto.

Crear un objeto degradado es un proceso de 2 pasos:

1. Crea el objeto degradado en sí. Durante la creación, se define una línea en el lienzo donde el gradiente comenzará y terminará. El objeto de gradiente se crea con `var gradient = context.createLinearGradient` .
2. Luego, agregue 2 (o más) colores que conforman el degradado. Esto se hace agregando múltiples paradas de color al objeto de `gradient.addColorStop` con `gradient.addColorStop` .

Argumentos:

- **startX, startY** es la coordenada del lienzo donde comienza el degradado. En el punto de inicio (y antes), el lienzo es sólidamente el color de la `gradientPercentPosition` más baja.
- **endX, endY** es la coordenada del lienzo donde termina el degradado. En el punto final (y después), el lienzo es sólidamente el color de `gradientPercentPosition` más alto.
- **gradientPercentPosition** es un número flotante entre 0.00 y 1.00 asignado a una parada de color. Básicamente es un punto porcentual a lo largo de la línea donde se aplica esta parada de color en particular.
 - El gradiente comienza en el porcentaje 0.00 que es [startX, startY] en el lienzo.
 - El gradiente termina en el porcentaje 1.00 que es [endX, endY] en el lienzo.
 - *Nota técnica:* el término "porcentaje" no es técnicamente correcto ya que los valores van de 0.00 a 1.00 en lugar de 0% a 100%.

- **CssColor** es un color CSS asignado a esta parada de color en particular.

El objeto de degradado es un objeto que puede usar (y reutilizar) para hacer que los trazos y rellenos de su trayectoria se vuelvan de color degradado.

Nota al margen: el objeto de degradado no es interno al elemento Canvas ni a su Contexto. Es un objeto JavaScript separado y reutilizable que puede asignar a cualquier ruta que desee. Incluso puede usar este objeto para colorear una ruta en un elemento de lienzo diferente (!)

Las paradas de color son puntos de referencia (porcentaje) a lo largo de la línea de degradado. En cada punto de parada de color, el degradado está completamente coloreado (== opacamente) con su color asignado. Los puntos intermedios a lo largo de la línea de gradiente entre las paradas de color se colorean como gradientes del color anterior y posterior.

¡Indirecta importante sobre gradientes de la lona!

Cuando creas un objeto de degradado, todo el lienzo se rellena "invisiblemente" con ese degradado.

Cuando traza `stroke()` o `fill()` un camino, se revela el gradiente invisible, pero solo se revela sobre ese camino que se está trazando o relleno.

1. Si creas un degradado lineal de rojo a magenta como este:

```
// create a linearGradient
var gradient=ctx.createLinearGradient(100,0,canvas.width-100,0);
gradient.addColorStop(0,'red');
gradient.addColorStop(1,'magenta');
ctx.fillStyle=gradient;
```

2. Entonces Canvas "invisiblemente" verá tu creación de gradiente de esta manera:



3. Pero hasta que `stroke()` o `fill()` con el degradado, no verá ninguno del degradado en el Lienzo.
4. Finalmente, si traza o rellena un trazado utilizando el degradado, el degradado "invisible" se hace visible en el Lienzo ... pero solo donde se dibuja el trazado.



```
<!doctype html>
<html>
<head>
<style>
  body{ background-color:white; }
  #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

  // canvas related variables
  var canvas=document.getElementById("canvas");
  var ctx=canvas.getContext("2d");

  // Create a linearGradient
  // Note: Nothing visually appears during this process
  var gradient=ctx.createLinearGradient(100,0,canvas.width-100,0);
  gradient.addColorStop(0,'red');
  gradient.addColorStop(1,'magenta');

  // Create a polyline path
  // Note: Nothing visually appears during this process
  var x=20;
  var y=40;
  ctx.lineCap='round';
  ctx.lineJoin='round';
  ctx.lineWidth=15;
  ctx.beginPath();
  ctx.moveTo(x,y);
  ctx.lineTo(x+30,y+50);
  ctx.lineTo(x+60,y);
  ctx.lineTo(x+90,y+50);
  ctx.lineTo(x+120,y);
  ctx.lineTo(x+150,y+50);
  ctx.lineTo(x+180,y);
  ctx.lineTo(x+210,y+50);
  ctx.lineTo(x+240,y);
  ctx.lineTo(x+270,y+50);
  ctx.lineTo(x+300,y);
  ctx.lineTo(x+330,y+50);
  ctx.lineTo(x+360,y);

  // Set the stroke style to be the gradient
  // Note: Nothing visually appears during this process
  ctx.strokeStyle=gradient;

  // stroke the path
  // FINALLY! The gradient-stroked path is visible on the canvas
  ctx.stroke();
```

```
}); // end window.onload
</script>
</head>
<body>
  <canvas id="canvas" width=400 height=150></canvas>
</body>
</html>
```

createRadialGradient (crea un objeto de estilo de ruta)

```
var gradient = createRadialGradient(
  centerX1, centerY1, radius1,    // this is the "display" circle
  centerX2, centerY2, radius2    // this is the "light casting" circle
)
gradient.addColorStop(gradientPercentPosition, CssColor)
gradient.addColorStop(gradientPercentPosition, CssColor)
[optionally add more color stops to add to the variety of the gradient]
```

Creando un degradado radial reutilizable (objeto). El objeto de degradado es un objeto que puede usar (y reutilizar) para hacer que los trazos y rellenos de su trayectoria se vuelvan de color degradado.

Acerca de...

El degradado radial de Canvas es *extremadamente diferente* de los degradados radiales tradicionales.

La definición "oficial" (¡casi indescifrable!) Del degradado radial de Canvas se encuentra al final de esta publicación. ¡No lo mires si tienes una disposición débil!

En términos (casi comprensibles):

- El degradado radial tiene 2 círculos: un círculo de "lanzamiento" y un círculo de "visualización".
- El círculo de lanzamiento arroja luz en el círculo de visualización.
- Esa luz es el gradiente.
- La forma de esa luz de gradiente está determinada por el tamaño relativo y la posición de ambos círculos.

Crear un objeto degradado es un proceso de 2 pasos:

1. Crea el objeto degradado en sí. Durante la creación, se define una línea en el lienzo donde el gradiente comenzará y terminará. El objeto de gradiente se crea con `var gradient = context.radialLinearGradient .`
2. Luego, agregue 2 (o más) colores que conforman el degradado. Esto se hace agregando múltiples paradas de color al objeto de `gradient.addColorStop` con `gradient.addColorStop .`

Argumentos:

- **centerX1, centerY1, radius1** define un primer círculo donde se mostrará el degradado.

- **centerX2, centroY2, radio2** define un segundo círculo que está proyectando luz de gradiente en el primer círculo.
- **gradientPercentPosition** es un número flotante entre 0.00 y 1.00 asignado a una parada de color. Básicamente es un porcentaje de punto de referencia que define dónde se aplica esta parada de color en particular a lo largo del degradado.
 - El gradiente comienza en porcentaje de 0.00.
 - El gradiente termina en porcentaje 1.00.
 - *Nota técnica:* el término "porcentaje" no es técnicamente correcto ya que los valores van de 0.00 a 1.00 en lugar de 0% a 100%.
- **CssColor** es un color CSS asignado a esta parada de color en particular.

Nota al margen: el objeto de degradado no es interno al elemento Canvas ni a su Contexto. Es un objeto JavaScript separado y reutilizable que puede asignar a cualquier ruta que desee. Incluso puede usar este objeto para colorear una ruta en un elemento de lienzo diferente (!)

Las paradas de color son puntos de referencia (porcentaje) a lo largo de la línea de degradado. En cada punto de parada de color, el degradado está completamente coloreado (== opacamente) con su color asignado. Los puntos intermedios a lo largo de la línea de gradiente entre las paradas de color se colorean como gradientes del color anterior y posterior.

¡Indirecta importante sobre gradientes de la lona!

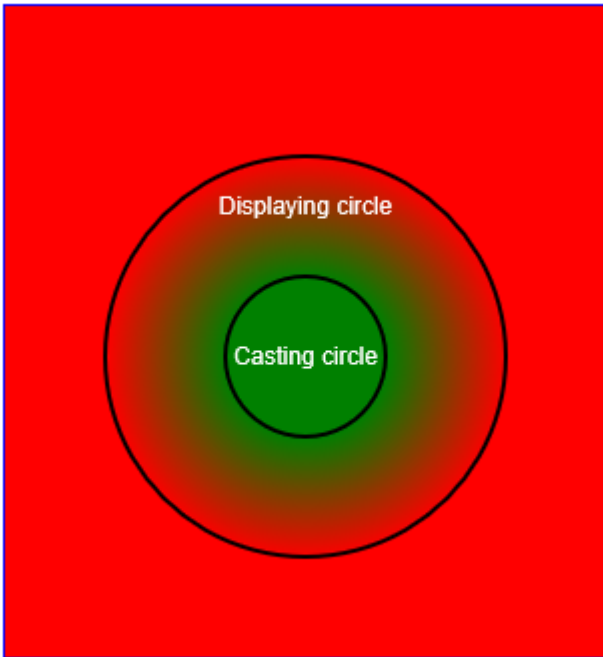
Cuando creas un objeto de degradado, todo el degradado radial se "invisiblemente" se proyecta sobre el lienzo.

Cuando traza `stroke()` o `fill()` un camino, se revela el gradiente invisible, pero solo se revela sobre ese camino que se está trazando o rellenando.

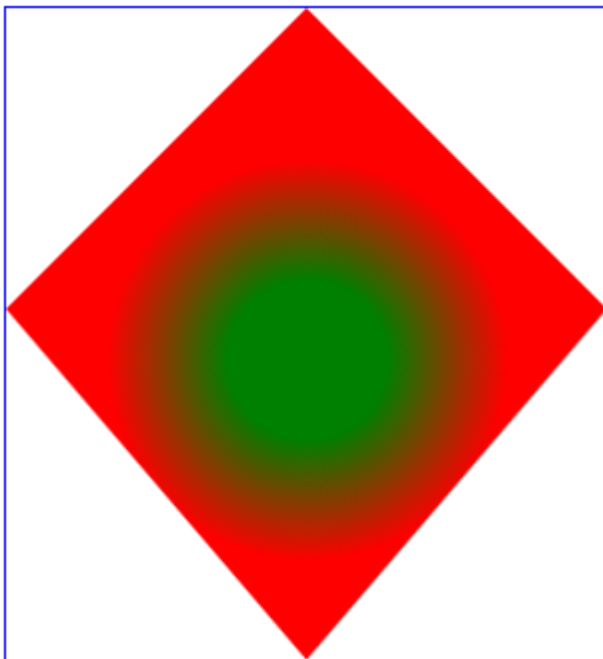
1. Si creas un degradado radial de verde a rojo como este:

```
// create a radialGradient
var x1=150;
var y1=150;
var x2=280;
var y2=150;
var r1=100;
var r2=120;
var gradient=context.createRadialGradient(x1,y1,r1,x2,y2,r2);
gradient.addColorStop(0,'red');
gradient.addColorStop(1,'green');
context.fillStyle=gradient;
```

2. Entonces Canvas "invisiblemente" verá tu creación de gradiente de esta manera:



3. Pero hasta que `stroke()` o `fill()` con el degradado, no verá ninguno del degradado en el Lienzo.
4. Finalmente, si traza o rellena un trazado utilizando el degradado, el degradado "invisible" se hace visible en el Lienzo ... pero solo donde se dibuja el trazado.



```
<!doctype html>
<html>
<head>
<style>
  body{ background-color:white; padding:10px; }
  #canvas{border:1px solid blue; }
</style>
<script>
window.onload=(function(){
```

```

// canvas related vars
var canvas=document.getElementById("canvas");
var ctx=canvas.getContext("2d");

// create a radial gradient
var x1=150;
var y1=175;
var x2=350;
var y2=175;
var r1=100;
var r2=40;
x2=x1;
var gradient=ctx.createRadialGradient(x1,y1,r1,x2,y2,r2);
gradient.addColorStop(0,'red');
gradient.addColorStop(1,'green');
ctx.fillStyle=gradient;

// fill a path with the gradient
ctx.beginPath();
ctx.moveTo(150,0);
ctx.lineTo(300,150);
ctx.lineTo(150,325);
ctx.lineTo(0,150);
ctx.lineTo(150,0);
ctx.fill();

}); // end window.onload
</script>
</head>
<body>
  <canvas id="canvas" width=300 height=325></canvas>
</body>
</html>

```

Los detalles oficiales de miedo

¿Quién decide lo que hace `createRadialGradient`?

El [W3C](#) emite las especificaciones oficiales recomendadas que los navegadores utilizan para construir el elemento Canvas Html5.

La [especificación W3C para createRadialGradient](#) lee críticamente de la siguiente manera:

¿ createRadialGradient **crea** createRadialGradient

`createRadialGradient` ... crea efectivamente un cono, tocado por los dos círculos definidos en la creación del degradado, con la parte del cono antes del círculo inicial (0.0) utilizando el color del primer desplazamiento, la parte del cono después del círculo final (1.0) usando el color del último desplazamiento y las áreas fuera del cono que no se han tocado con el degradado (negro transparente).

¿Cómo funciona internamente?

El `createRadialGradient(x0, y0, r0, x1, y1, r1)` toma seis argumentos, los primeros

tres representan el círculo inicial con origen (x_0, y_0) y radio r_0 , y los últimos tres representan el círculo final con origen (x_1, y_1) y radio r_1 . Los valores están en unidades de espacio de coordenadas. Si cualquiera de r_0 o r_1 son negativos, se debe lanzar una excepción `IndexSizeError`. De lo contrario, el método debe devolver un `CanvasGradient` radial inicializado con los dos círculos especificados.

Los gradientes radiales deben representarse siguiendo estos pasos:

1. Si $x_0 = x_1$ e $y_0 = y_1$ y $r_0 = r_1$, entonces el gradiente radial no debe pintar nada. Abortar estos pasos.
2. Sea $x(\omega) = (x_1 - x_0)\omega + x_0$; Sea $y(\omega) = (y_1 - y_0)\omega + y_0$; Sea $r(\omega) = (r_1 - r_0)\omega + r_0$. Sea el color en ω el color en esa posición en el degradado (con los colores provenientes de la interpolación y extrapolación descritos anteriormente).
3. Para todos los valores de ω donde $r(\omega) > 0$, comenzando con el valor de ω más cercano al infinito positivo y terminando con el valor de ω más cercano al infinito negativo, dibuje la circunferencia del círculo con el radio $r(\omega)$ en la posición $(x(\omega), y(\omega))$, con el color en, pero solo pintando en las partes del lienzo que aún no han sido pintadas por círculos anteriores en este paso para esta representación del degradado.

`createPattern` (crea un objeto de estilo de ruta)

```
var pattern = createPattern(imageObject, repeat)
```

Creación de un patrón reutilizable (objeto).

El objeto se puede asignar a cualquier `strokeStyle` y / o `fillStyle`.

Luego, trazar `()` o rellenar `()` pintará la ruta con el patrón del objeto.

Argumentos:

- **imageObject** es una imagen que se usará como patrón. La fuente de la imagen puede ser:
 - `HTMLImageElement` --- un elemento `img` o una nueva imagen `()`,
 - `HTMLCanvasElement` --- un elemento `canvas`,
 - `HTMLVideoElement` --- un elemento de video (tomará el cuadro de video actual)
 - `ImageBitmap`,
 - Gota.
- **la repetición** determina cómo se repetirá el `imageObject` en el lienzo (como en un fondo CSS). Este argumento debe estar delimitado por comillas y los valores válidos son:
 - "repetir" --- el patrón llenará horizontal y verticalmente el lienzo
 - "repeat-x" --- el patrón solo se repetirá horizontalmente (1 fila horizontal)
 - "repetir-y" --- el patrón solo se repetirá verticalmente (1 fila vertical)
 - "no repetir" --- el patrón aparece solo una vez (arriba a la izquierda)

El objeto de patrón es un objeto que puede usar (y reutilizar) para hacer que los trazos y rellenos de su ruta se conviertan en patrones.

Nota al margen: el objeto de patrón no es interno al elemento Canvas ni a su contexto. Es un objeto JavaScript separado y reutilizable que puede asignar a cualquier ruta que desee. Incluso puede usar este objeto para aplicar un patrón a una ruta en un elemento de lienzo diferente (!)

Importante pista sobre los patrones de lienzo!

Cuando creas un objeto de patrón, todo el lienzo se rellena "invisiblemente" con ese patrón (sujeto al argumento de `repeat`).

Cuando traza `stroke()` o `fill()` un camino, se revela el patrón invisible, pero solo se revela sobre ese camino que se está trazando o relleno.

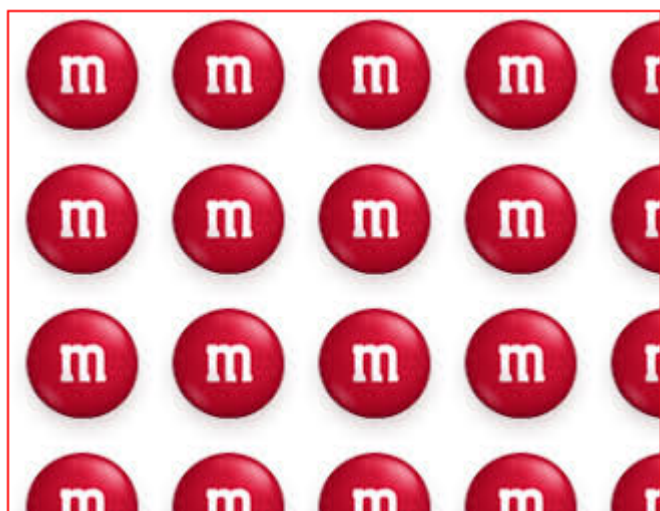
1. Comience con una imagen que desee utilizar como patrón. Importante (!): Asegúrese de que su imagen se haya cargado completamente (usando `patternimage.onload`) antes de intentar usarla para crear su patrón.



2. Usted crea un patrón como este:

```
// create a pattern
var pattern = ctx.createPattern(patternImage, 'repeat');
ctx.fillStyle=pattern;
```

3. Entonces Canvas "invisiblemente" verá la creación de tu patrón de esta manera:



4. Pero hasta que `stroke()` o `fill()` con el patrón, no verá ninguno del patrón en el lienzo.
5. Finalmente, si trazas o rellenas una ruta utilizando el patrón, el patrón "invisible" se hace visible en el Lienzo ... pero solo donde se dibuja la ruta.



```
<!doctype html>
<html>
<head>
<style>
  body{ background-color:white; }
  #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

  // canvas related variables
  var canvas=document.getElementById("canvas");
  var ctx=canvas.getContext("2d");

  // fill using a pattern
  var patternImage=new Image();
  // IMPORTANT!
  // Always use .onload to be sure the image has
  //   fully loaded before using it in .createPattern
  patternImage.onload=function(){
    // create a pattern object
    var pattern = ctx.createPattern(patternImage, 'repeat');
    // set the fillstyle to that pattern
    ctx.fillStyle=pattern;
    // fill a rectangle with the pattern
    ctx.fillRect(50,50,150,100);
    // demo only, stroke the rect for clarity
    ctx.strokeRect(50,50,150,100);
  }
  patternImage.src='http://i.stack.imgur.com/K9EZ1.png';

}); // end window.onload
</script>
</head>
<body>
  <canvas id="canvas" width=325 height=250></canvas>
</body>
</html>
```

trazo (un comando de ruta)

```
context.stroke()
```

Hace que el perímetro de la ruta sea trazada de acuerdo con el `context.strokeStyle` actual. `context.strokeStyle` y la ruta trazada se dibuja visualmente en el lienzo.

Antes de ejecutar `context.stroke` (o `context.fill`), la ruta existe en la memoria y aún no está dibujada visualmente en el lienzo.

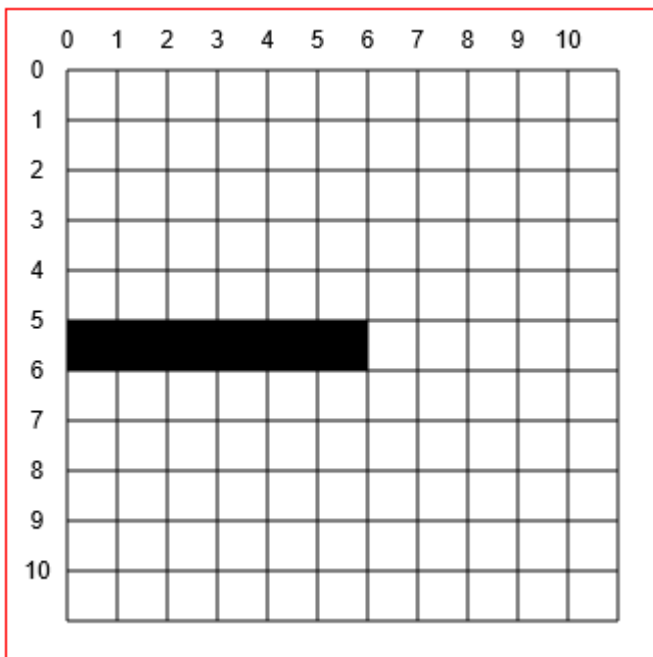
La inusual forma en que se dibujan los trazos.

Considere este camino de ejemplo que dibuja una línea negra de 1 píxel de `[0,5]` a `[5,5]` :

```
// draw a 1 pixel black line from [0,5] to [5,5]
context.strokeStyle='black';
context.lineWidth=1;
context.beginPath();
context.moveTo(0,5);
context.lineTo(5,5);
context.stroke();
```

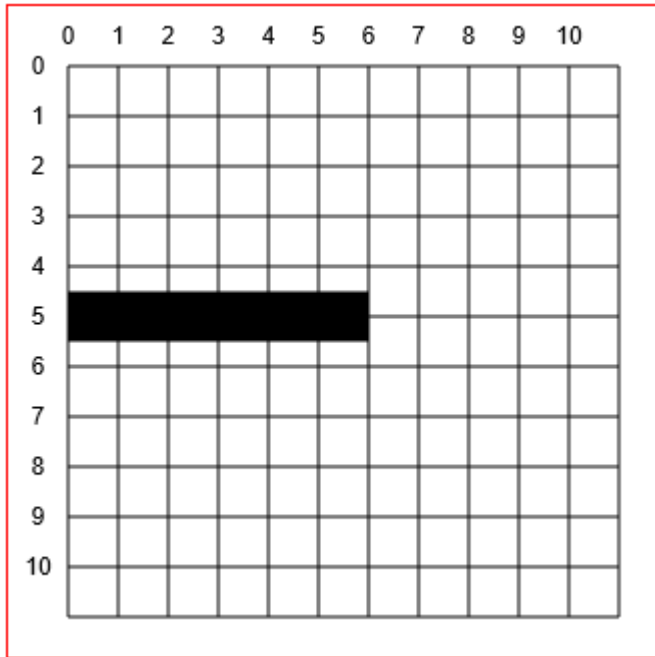
Pregunta: ¿Qué dibuja realmente el navegador en el lienzo?

Probablemente esperas obtener 6 píxeles negros en $y = 5$



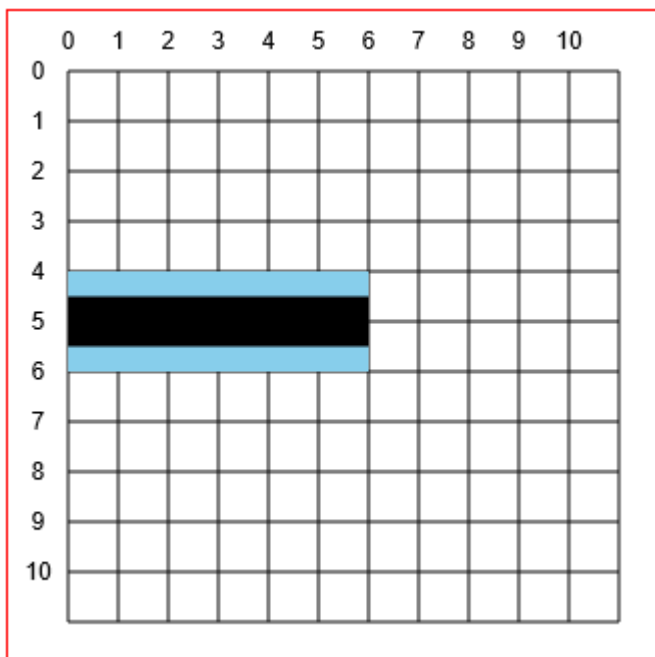
Pero (!) ... ¡El lienzo siempre dibuja trazos a mitad de camino hacia cualquier lado del camino definido!

Entonces, como la línea está definida en $y=5.0$ Canvas quiere dibujar la línea entre $y=4.5$ y $y=5.5$

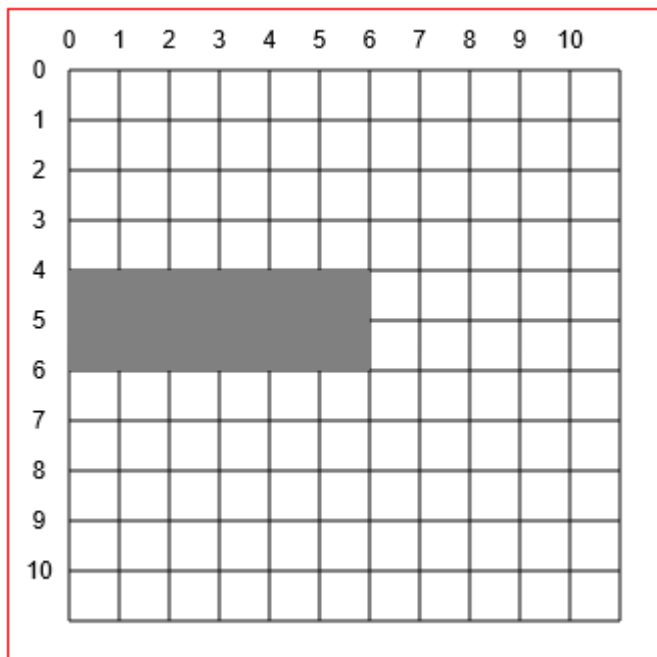


Pero, otra vez (!) ... ¡La pantalla de la computadora no puede dibujar medio píxeles!

Entonces, ¿qué se debe hacer con los medios píxeles no deseados (que se muestran en azul a continuación)?



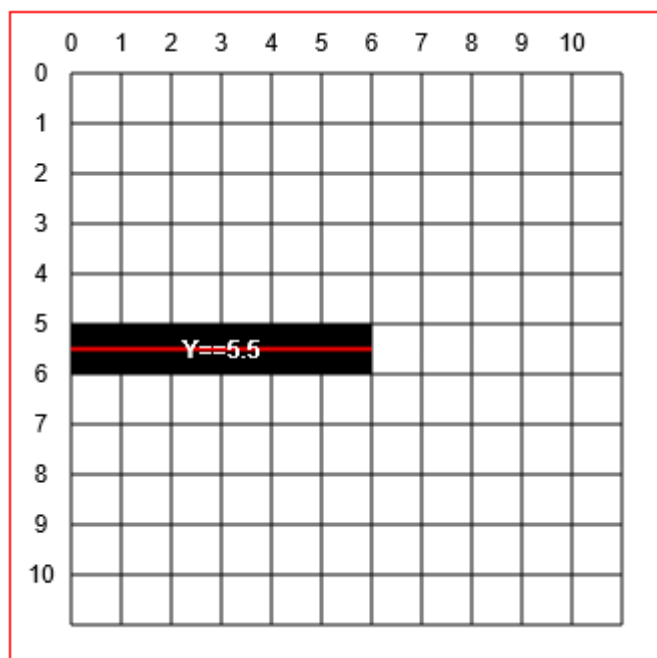
La respuesta es que Canvas en realidad ordena a la pantalla dibujar una línea de 2 píxeles de ancho de 4.0 a 6.0 . También colorea la línea más clara que el `black` definido. Este extraño comportamiento de dibujo es "anti-aliasing" y ayuda a Canvas a evitar trazos que parezcan irregulares.



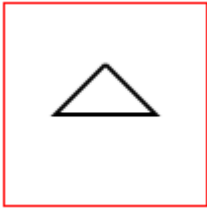
Un truco de ajuste que SOLO funciona para movimientos horizontales y verticales exactamente

Puede obtener una línea negra continua de 1 píxel especificando que la línea se dibuje en el medio píxel:

```
context.moveTo(0, 5.5);
context.lineTo(5, 5.5);
```



Código de ejemplo que usa `context.stroke()` para dibujar una ruta trazada en el lienzo:



```
<!doctype html>
<html>
<head>
<style>
  body{ background-color:white; }
  #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

  // canvas related variables
  var canvas=document.getElementById("canvas");
  var ctx=canvas.getContext("2d");

  ctx.beginPath();
  ctx.moveTo(50,30);
  ctx.lineTo(75,55);
  ctx.lineTo(25,55);
  ctx.lineTo(50,30);
  ctx.lineWidth=2;
  ctx.stroke();

}); // end window.onload
</script>
</head>
<body>
  <canvas id="canvas" width=100 height=100></canvas>
</body>
</html>
```

Rellenar (un comando de ruta)

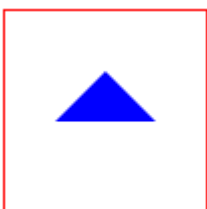
```
context.fill()
```

Hace que el interior de la ruta se rellene de acuerdo con el `context.fillStyle` actual.

`context.fillStyle` relleno y la ruta rellena se dibuja visualmente en el lienzo.

Antes de ejecutar `context.fill` (o `context.stroke`), la ruta existe en la memoria y aún no está dibujada visualmente en el lienzo.

Ejemplo de código usando `context.fill()` para dibujar una ruta llena en el lienzo:



```
<!doctype html>
<html>
<head>
<style>
  body{ background-color:white; }
  #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

  // canvas related variables
  var canvas=document.getElementById("canvas");
  var ctx=canvas.getContext("2d");

  ctx.beginPath();
  ctx.moveTo(50,30);
  ctx.lineTo(75,55);
  ctx.lineTo(25,55);
  ctx.lineTo(50,30);
  ctx.fillStyle='blue';
  ctx.fill();

}); // end window.onload
</script>
</head>
<body>
  <canvas id="canvas" width=100 height=100></canvas>
</body>
</html>
```

clip (un comando de ruta)

```
context.clip
```

Limita los dibujos futuros para mostrar solo dentro de la ruta actual.

Ejemplo: Clip de esta imagen en un camino triangular



```
<!doctype html>
<html>
<head>
<style>
  body{ background-color:white; }
  #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

  // canvas related variables
  var canvas=document.getElementById("canvas");
  var ctx=canvas.getContext("2d");

  var img=new Image();
  img.onload=start;
  img.src='http://i.stack.imgur.com/1CqWf.jpg'

  function start(){
    // draw a triangle path
    ctx.beginPath();
    ctx.moveTo(75,50);
    ctx.lineTo(125,100);
    ctx.lineTo(25,100);
    ctx.lineTo(75,50);

    // clip future drawings to appear only in the triangle
    ctx.clip();

    // draw an image
    ctx.drawImage(img,0,0);
  }

}); // end window.onload
</script>
</head>
<body>
  <canvas id="canvas" width=150 height=150></canvas>
</body>
</html>
```

Lea Ruta (solo sintaxis) en línea: <https://riptutorial.com/es/html5-canvas/topic/3241/ruta--solo-sintaxis->

Capítulo 17: Texto

Examples

Dibujo de texto

Dibujar en lienzo no se limita a formas e imágenes. También puede dibujar texto al lienzo.

Para dibujar texto en el lienzo, obtenga una referencia al lienzo y luego llame al método `fillText` en el contexto.

```
var canvas = document.getElementById('canvas');
var ctx = canvas.getContext('2d');
ctx.fillText("My text", 0, 0);
```

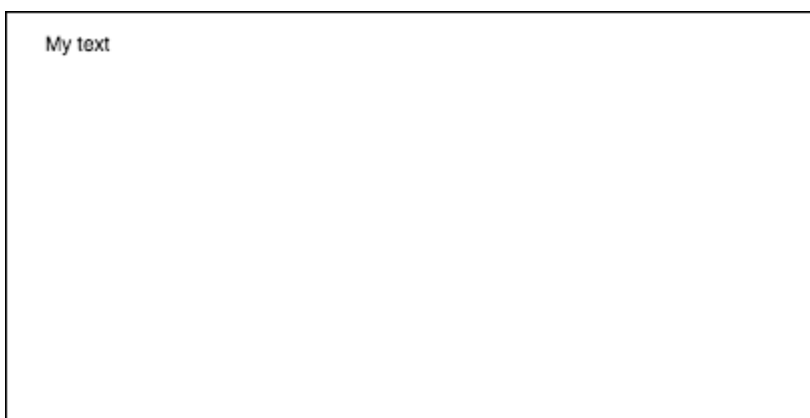
Los tres argumentos **requeridos** que se pasan a `fillText` son:

1. El texto que desea mostrar.
2. La posición horizontal (eje x)
3. La posición vertical (eje y)

Además, hay un cuarto argumento **opcional**, que puede utilizar para especificar el ancho máximo de su texto en píxeles. En el siguiente ejemplo, el valor de `200` restringe el ancho máximo del texto a 200 px:

```
ctx.fillText("My text", 0, 0, 200);
```

Resultado:



También puede dibujar texto sin relleno, y solo un contorno en su lugar, utilizando el método `strokeText`:

```
ctx.strokeText("My text", 0, 0);
```

Resultado:



Sin las propiedades de formato de fuente aplicadas, el lienzo representa el texto a 10px en sans-serif de forma predeterminada, lo que dificulta ver la diferencia entre el resultado de los métodos `fillText` y `strokeText` . Consulte el [ejemplo de Formato de texto](#) para obtener detalles sobre cómo aumentar el tamaño del texto y aplicar otros cambios estéticos al texto.

Formato de texto

El formato de fuente predeterminado proporcionado por los métodos `fillText` y `strokeText` no es muy atractivo estéticamente. Afortunadamente, la API del lienzo proporciona propiedades para formatear texto.

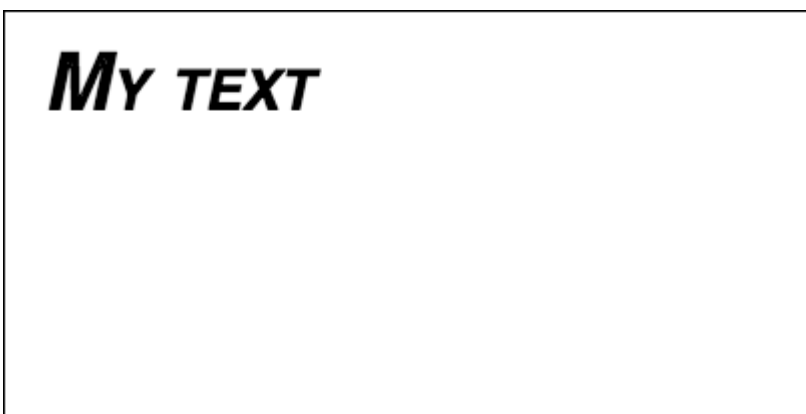
Usando la propiedad de `font` puede especificar:

- Estilo de fuente
- variante de fuente
- peso de fuente
- tamaño de fuente / altura de línea
- Familia tipográfica

Por ejemplo:

```
ctx.font = "italic small-caps bold 40px Helvetica, Arial, sans-serif";  
ctx.fillText("My text", 20, 50);
```

Resultado:



Usando la propiedad `textAlign` , también puede cambiar la alineación del texto a:

- izquierda
- centrar
- Correcto
- final (igual que a la derecha)
- inicio (igual a la izquierda)

Por ejemplo:

```
ctx.textAlign = "center";
```

Envolver el texto en párrafos

Native Canvas API no tiene un método para ajustar el texto en la siguiente línea cuando se alcanza el ancho máximo deseado. Este ejemplo envuelve el texto en párrafos.

```
function wrapText(text, x, y, maxWidth, fontSize, fontFace){
  var firstY=y;
  var words = text.split(' ');
  var line = '';
  var lineHeight=fontSize*1.286; // a good approx for 10-18px sizes

  ctx.font=fontSize+" "+fontFace;
  ctx.textBaseline='top';

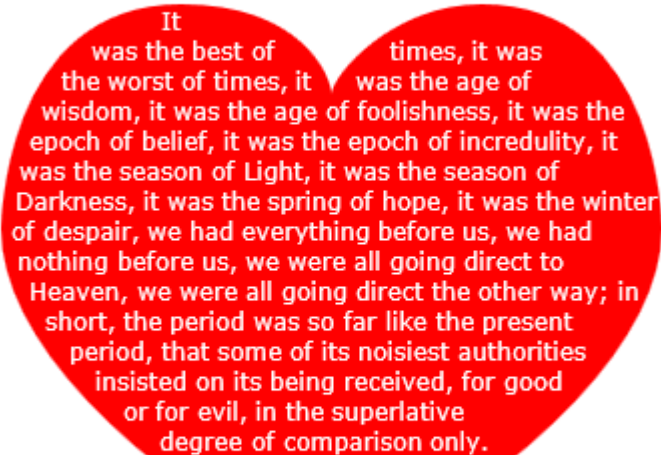
  for(var n = 0; n < words.length; n++) {
    var testLine = line + words[n] + ' ';
    var metrics = ctx.measureText(testLine);
    var testWidth = metrics.width;
    if(testWidth > maxWidth) {
      ctx.fillText(line, x, y);
      if(n<words.length-1){
        line = words[n] + ' ';
        y += lineHeight;
      }
    }
    else {
      line = testLine;
    }
  }
  ctx.fillText(line, x, y);
}
```

Dibuja párrafos de texto en formas irregulares

Este ejemplo dibuja párrafos de texto en cualquier parte del lienzo que tenga píxeles opacos.

Funciona encontrando el siguiente bloque de píxeles opacos que es lo suficientemente grande como para contener la siguiente palabra especificada y rellenando ese bloque con la palabra especificada.

Los píxeles opacos pueden provenir de cualquier fuente: comandos de dibujo de ruta y / o imágenes.



It was the best of times, it was the worst of times, it was the age of wisdom, it was the age of foolishness, it was the epoch of belief, it was the epoch of incredulity, it was the season of Light, it was the season of Darkness, it was the spring of hope, it was the winter of despair, we had everything before us, we had nothing before us, we were all going direct to Heaven, we were all going direct the other way; in short, the period was so far like the present period, that some of its noisiest authorities insisted on its being received, for good or for evil, in the superlative degree of comparison only.

```
<!doctype html>
<html>
<head>
<style>
  body{ background-color:white; padding:10px; }
  #canvas{border:1px solid red;}
</style>
<script>
window.onload=(function(){

  var canvas=document.getElementById("canvas");
  var ctx=canvas.getContext("2d");
  var cw=canvas.width;
  var ch=canvas.height;

  var fontsize=12;
  var fontface='verdana';
  var lineHeight=parseInt(fontsize*1.286);

  var text='It was the best of times, it was the worst of times, it was the age of wisdom,
it was the age of foolishness, it was the epoch of belief, it was the epoch of incredulity, it
was the season of Light, it was the season of Darkness, it was the spring of hope, it was the
winter of despair, we had everything before us, we had nothing before us, we were all going
direct to Heaven, we were all going direct the other way; in short, the period was so far like
the present period, that some of its noisiest authorities insisted on its being received, for
good or for evil, in the superlative degree of comparison only.';
  var words=text.split(' ');
  var wordWidths=[];
  ctx.font=fontsize+'px '+fontface;
  for(var i=0;i<words.length;i++){ wordWidths.push(ctx.measureText(words[i]).width); }
  var spaceWidth=ctx.measureText(' ').width;
  var wordIndex=0
  var data=[];
```

```

// Demo: draw Heart
// Note: the shape can be ANY opaque drawing -- even an image
ctx.scale(3,3);
ctx.beginPath();
ctx.moveTo(75,40);
ctx.bezierCurveTo(75,37,70,25,50,25);
ctx.bezierCurveTo(20,25,20,62.5,20,62.5);
ctx.bezierCurveTo(20,80,40,102,75,120);
ctx.bezierCurveTo(110,102,130,80,130,62.5);
ctx.bezierCurveTo(130,62.5,130,25,100,25);
ctx.bezierCurveTo(85,25,75,37,75,40);
ctx.fillStyle='red';
ctx.fill();
ctx.setTransform(1,0,0,1,0,0);

// fill heart with text
ctx.fillStyle='white';
var imgDataData=ctx.getImageData(0,0,cw,ch).data;
for(var i=0;i<imgDataData.length;i+=4){
    data.push(imgDataData[i+3]);
}
placeWords();

// draw words sequentially into next available block of
// available opaque pixels
function placeWords(){
    var sx=0;
    var sy=0;
    var y=0;
    var wordIndex=0;
    ctx.textBaseline='top';
    while(y<ch && wordIndex<words.length){
        sx=0;
        sy=y;
        var startingIndex=wordIndex;
        while(sx<cw && wordIndex<words.length){
            var x=getRect(sx,sy,lineHeight);
            var available=x-sx;
            var spacer=spaceWidth; // spacer=0 to have no left margin
            var w=spacer+wordWidths[wordIndex];
            while(available>=w){
                ctx.fillText(words[wordIndex],spacer+sx,sy);
                sx+=w;
                available-=w;
                spacer=spaceWidth;
                wordIndex++;
                w=spacer+wordWidths[wordIndex];
            }
            sx=x+1;
        }
        y=(wordIndex>startingIndex)?y+lineHeight:y+1;
    }
}

// find a rectangular block of opaque pixels
function getRect(sx,sy,height){
    var x=sx;
    var y=sy;
    var ok=true;
    while(ok){
        if(data[y*cw+x]<250){ok=false;}
    }
}

```



```

        y++;
        if(y>=sy+height){
            y=sy;
            x++;
            if(x>=cw){ok=false;}
        }
    }
    return(x);
}

}); // end $(function(){});
</script>
</head>
<body>
    <h4>Note: the shape must be closed and alpha=250 inside</h4>
    <canvas id="canvas" width=400 height=400></canvas>
</body>
</html>

```

Rellena texto con una imagen

Este ejemplo llena el texto con una imagen específica.

¡Importante! La imagen especificada debe estar completamente cargada antes de llamar a esta función o el dibujo fallará. Use `image.onload` para asegurarse de que la imagen esté completamente cargada.



```

function drawImageInsideText (canvas, x, y, img, text, font) {
    var c=canvas.cloneNode();
    var ctx=c.getContext('2d');
    ctx.font=font;
    ctx.fillText(text, x, y);
    ctx.globalCompositeOperation='source-atop';
    ctx.drawImage(img, 0, 0);
    canvas.getContext('2d').drawImage(c, 0, 0);
}

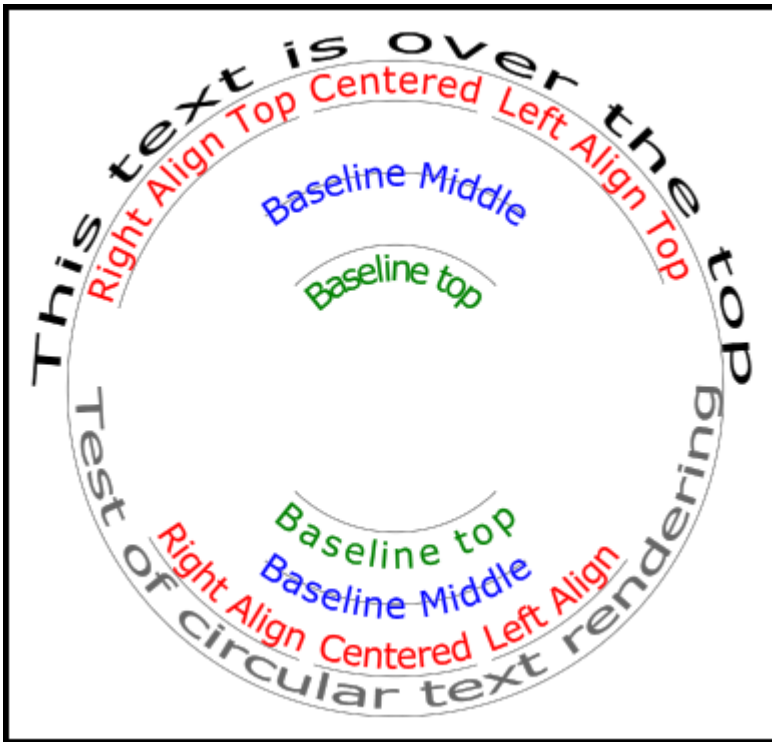
```

Representación de texto a lo largo de un arco.

Este ejemplo muestra cómo representar texto a lo largo de un arco. Incluye cómo puedes agregar funcionalidad a `CanvasRenderingContext2D` al extender su prototipo.

Este ejemplo se deriva de la respuesta de Stackoverflow [Circular Text](#).

Representación de ejemplo



Código de ejemplo

El ejemplo agrega 3 nuevas funciones de representación de texto al prototipo de contexto 2D.

- **ctx.fillCircleText (texto, x, y, radio, inicio, final, avance);**
- **ctx.strokeCircleText (texto, x, y, radio, inicio, final, adelante);**
- **ctx.measureCircleText (texto, radio);**

```
(function(){
  const FILL = 0;          // const to indicate filltext render
  const STROKE = 1;
  var renderType = FILL; // used internal to set fill or stroke text
  const multiplyCurrentTransform = true; // if true Use current transform when rendering
  // if false use absolute coordinates which is a
  little quicker
  // after render the currentTransform is restored to
  default transform

  // measure circle text
  // ctx: canvas context
  // text: string of text to measure
  // r: radius in pixels
  //
  // returns the size metrics of the text
  //
  // width: Pixel width of text
  // angularWidth : angular width of text in radians
```

```

// pixelAngularSize : angular width of a pixel in radians
var measure = function(ctx, text, radius){
    var textWidth = ctx.measureText(text).width; // get the width of all the text
    return {
        width            : textWidth,
        angularWidth     : (1 / radius) * textWidth,
        pixelAngularSize : 1 / radius
    };
}

// displays text along a circle
// ctx: canvas context
// text: string of text to measure
// x,y: position of circle center
// r: radius of circle in pixels
// start: angle in radians to start.
// [end]: optional. If included text align is ignored and the text is
//        scaled to fit between start and end;
// [forward]: optional default true. if true text direction is forwards, if false
direction is backward
var circleText = function (ctx, text, x, y, radius, start, end, forward) {
    var i, textWidth, pA, pAS, a, aw, wScale, aligned, dir, fontSize;
    if(text.trim() === "" || ctx.globalAlpha === 0){ // dont render empty string or
transparent
        return;
    }
    if(isNaN(x) || isNaN(y) || isNaN(radius) || isNaN(start) || (end !== undefined && end
!&& null && isNaN(end))){ //
        throw TypeError("circle text arguments requires a number for x,y, radius, start,
and end.")
    }
    aligned = ctx.textAlign; // save the current textAlign so that it can be
restored at end
    dir = forward ? 1 : forward === false ? -1 : 1; // set dir if not true or false set
forward as true
    pAS = 1 / radius; // get the angular size of a pixel in radians
    textWidth = ctx.measureText(text).width; // get the width of all the text
    if (end !== undefined && end !== null) { // if end is supplied then fit text between
start and end
        pA = ((end - start) / textWidth) * dir;
        wScale = (pA / pAS) * dir;
    } else { // if no end is supplied correct start and end for alignment
// if forward is not given then swap top of circle text to read the correct
direction
        if(forward === null || forward === undefined){
            if(((start % (Math.PI * 2)) + Math.PI * 2) % (Math.PI * 2) > Math.PI){
                dir = -1;
            }
        }
        pA = -pAS * dir ;
        wScale = -1 * dir;
        switch (aligned) {
            case "center": // if centered move around half width
                start -= (pA * textWidth) / 2;
                end = start + pA * textWidth;
                break;
            case "right": // intentionally falls through to case "end"
            case "end":
                end = start;
                start -= pA * textWidth;
                break;
        }
    }
}

```

```

        case "left": // intentionally falls through to case "start"
        case "start":
            end = start + pA * textWidth;
        }
    }

    ctx.textAlign = "center"; // align for rendering
    a = start; // set the start angle
    for (var i = 0; i < text.length; i += 1) { // for each character
        aw = ctx.measureText(text[i]).width * pA; // get the angular width of the text
        var xDx = Math.cos(a + aw / 2); // get the xAxes vector from the center
x,y out
        var xDy = Math.sin(a + aw / 2);
        if(multiplyCurrentTransform){ // transform multiplying current transform
            ctx.save();
            if (xDy < 0) { // is the text upside down. If it is flip it
                ctx.transform(-xDy * wScale, xDx * wScale, -xDx, -xDy, xDx * radius + x,
xDy * radius + y);
            } else {
                ctx.transform(-xDy * wScale, xDx * wScale, xDx, xDy, xDx * radius + x, xDy
* radius + y);
            }
        }else{
            if (xDy < 0) { // is the text upside down. If it is flip it
                ctx.setTransform(-xDy * wScale, xDx * wScale, -xDx, -xDy, xDx * radius +
x, xDy * radius + y);
            } else {
                ctx.setTransform(-xDy * wScale, xDx * wScale, xDx, xDy, xDx * radius + x,
xDy * radius + y);
            }
        }
        if(renderType === FILL){
            ctx.fillText(text[i], 0, 0); // render the character
        }else{
            ctx.strokeText(text[i], 0, 0); // render the character
        }
        if(multiplyCurrentTransform){ // restore current transform
            ctx.restore();
        }
        a += aw; // step to the next angle
    }
    // all done clean up.
    if(!multiplyCurrentTransform){
        ctx.setTransform(1, 0, 0, 1, 0, 0); // restore the transform
    }
    ctx.textAlign = aligned; // restore the text alignment
}
// define fill text
var fillCircleText = function(text, x, y, radius, start, end, forward){
    renderType = FILL;
    circleText(this, text, x, y, radius, start, end, forward);
}
// define stroke text
var strokeCircleText = function(text, x, y, radius, start, end, forward){
    renderType = STROKE;
    circleText(this, text, x, y, radius, start, end, forward);
}
// define measure text
var measureCircleTextExt = function(text, radius){
    return measure(this, text, radius);
}
}

```

```
// set the prototypes
CanvasRenderingContext2D.prototype.fillCircleText = fillCircleText;
CanvasRenderingContext2D.prototype.strokeCircleText = strokeCircleText;
CanvasRenderingContext2D.prototype.measureCircleText = measureCircleTextExt;
})();
```

Descripciones de funciones

Este ejemplo agrega 3 funciones al `CanvasRenderingContext2D` `prototype` . `fillCircleText` , `strokeCircleText` , y `measureCircleText`

CanvasRenderingContext2D.fillCircleText (texto, x, y, radio, inicio, [final, [adelante]]);

CanvasRenderingContext2D.strokeCircleText (texto, x, y, radio, inicio, [final, [adelante]]);

- **texto:** Texto para renderizar como cadena.
- **x , y :** Posición del centro del círculo como números.
- **radio:** radio del círculo en píxeles
- **inicio:** ángulo en radianes para comenzar.
- **[final]:** opcional. Si se incluye, se ignora `ctx.textAlign` y el texto se escala para que se ajuste entre el inicio y el final.
- **[adelante]:** opción predeterminada 'verdadero'. si la dirección verdadera del texto es hacia adelante, si la dirección "falsa" es hacia atrás.

Ambas funciones utilizan `textBaseline` para colocar el texto verticalmente alrededor del radio. Para obtener los mejores resultados, use `ctx.TextBaseline` .

Las funciones lanzarán un `TypeError` es cualquiera de los argumentos numéricos como NaN.

Si el argumento de `text` recorta a una cadena vacía o `ctx.globalAlpha = 0` la función simplemente se `ctx.globalAlpha = 0` y no hace nada.

CanvasRenderingContext2D.measureCircleText (texto, radio);

```
- **text:** String of text to measure.
- **radius:** radius of circle in pixels.
```

Devuelve un objeto que contiene varias métricas de tamaño para representar texto circular

- **width:** Pixel width of text as it would normally be rendered
- **angularWidth:** angular width of text in radians.
- **pixelAngularSize:** angular width of a pixel in radians.

Ejemplos de uso

```
const rad = canvas.height * 0.4;
const text = "Hello circle TEXT!";
const fontSize = 40;
const centX = canvas.width / 2;
const centY = canvas.height / 2;
ctx.clearRect(0,0,canvas.width,canvas.height)

ctx.font = fontSize + "px verdana";
ctx.textAlign = "center";
ctx.textBaseline = "bottom";
ctx.fillStyle = "#000";
ctx.strokeStyle = "#666";

// Text under stretched from Math.PI to 0 (180 - 0 deg)
ctx.fillCircleText(text, centX, centY, rad, Math.PI, 0);

// text over top centered at Math.PI * 1.5 ( 270 deg)
ctx.fillCircleText(text, centX, centY, rad, Math.PI * 1.5);

// text under top centered at Math.PI * 1.5 ( 270 deg)
ctx.textBaseline = "top";
ctx.fillCircleText(text, centX, centY, rad, Math.PI * 1.5);

// text over top centered at Math.PI * 1.5 ( 270 deg)
ctx.textBaseline = "middle";
ctx.fillCircleText(text, centX, centY, rad, Math.PI * 1.5);

// Use measureCircleText to get angular size
var circleTextMetric = ctx.measureCircleText("Text to measure", rad);
console.log(circleTextMetric.width);           // width of text if rendered normally
console.log(circleTextMetric.angularWidth);    // angular width of text
console.log(circleTextMetric.pixelAngularSize); // angular size of a pixel

// Use measure text to draw a arc around the text
ctx.textBaseline = "middle";
var width = ctx.measureCircleText(text, rad).angularWidth;
ctx.fillCircleText(text, centX, centY, rad, Math.PI * 1.5);

// render the arc around the text
ctx.strokeStyle= "red";
ctx.lineWidth = 3;
ctx.beginPath();
ctx.arc(centX, centY, rad + fontSize / 2,Math.PI * 1.5 - width/2,Math.PI*1.5 + width/2);
ctx.arc(centX, centY, rad - fontSize / 2,Math.PI * 1.5 + width/2,Math.PI*1.5 - width/2,true);
ctx.closePath();
```

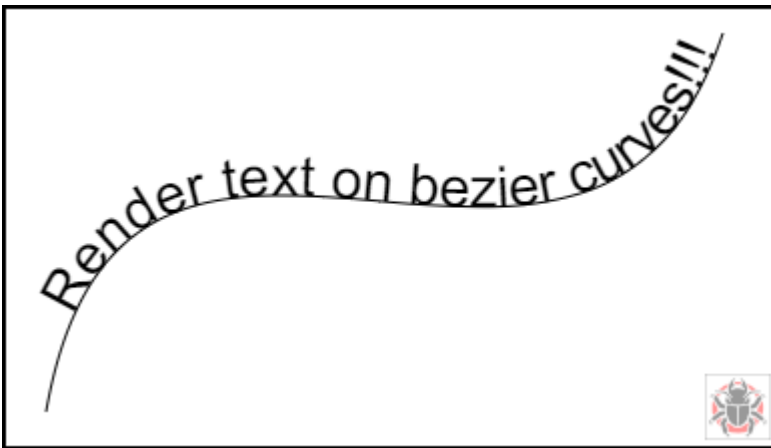
```
ctx.stroke();
```

NOTA: El texto representado es solo una aproximación de texto circular. Por ejemplo, si se representan dos l, las dos líneas no serán paralelas, pero si representa una "H", los dos bordes serán paralelos. Esto se debe a que cada carácter se representa lo más cerca posible de la dirección requerida, en lugar de que cada píxel se transforme correctamente para crear un texto circular.

NOTA: `const multiplyCurrentTransform = true;` definido en este ejemplo se utiliza para establecer el método de transformación utilizado. Si es `false` la transformación para la representación de texto circular es absoluta y no depende del estado de transformación actual. El texto no se verá afectado por ninguna escala anterior, rotación o transformación. Esto aumentará el rendimiento de la función de procesamiento, después de que la función se llame, la transformación se establecerá en el `setTransform(1,0,0,1,0,0)` predeterminado `setTransform(1,0,0,1,0,0)`

Si `multiplyCurrentTransform = true` (establecido como predeterminado en este ejemplo), el texto usará la transformación actual para que el texto se pueda escalar, inclinar, rotar, etc., pero modificar la transformación actual antes de llamar a las funciones `fillCircleText` y `strokeCircleText`. Dependiendo del estado actual del contexto 2D, esto puede ser algo más lento que `multiplyCurrentTransform = false`

Texto en curva, beziers cúbicos y cuadráticos.



textOnCurve (texto, desplazamiento, x1, y1, x2, y2, x3, y3, x4, y4)

Representa el texto en curvas cuadráticas y cúbicas.

- `text` es el texto a representar
- distancia de `offset` desde el inicio de la curva hasta el texto ≥ 0
- `x1, y1 - x3, y3` puntos de curva cuadrática o
- `x1, y1 - x4, y4` puntos de curva cúbica o

Ejemplo de uso:

```

textOnCurve("Hello world!",50,100,100,200,200,300,100); // draws text on quadratic curve
// 50 pixels from start of curve

textOnCurve("Hello world!",50,100,100,200,200,300,100,400,200);
// draws text on cubic curve
// 50 pixels from start of curve

```

La función y la función de ayudante curver

```

// pass 8 values for cubic bezier
// pass 6 values for quadratic
// Renders text from start of curve
var textOnCurve = function(text,offset,x1,y1,x2,y2,x3,y3,x4,y4){
  ctx.save();
  ctx.textAlign = "center";
  var widths = [];
  for(var i = 0; i < text.length; i++){
    widths[widths.length] = ctx.measureText(text[i]).width;
  }
  var ch = curveHelper(x1,y1,x2,y2,x3,y3,x4,y4);
  var pos = offset;
  var cpos = 0;

  for(var i = 0; i < text.length; i++){
    pos += widths[i] / 2;
    cpos = ch.forward(pos);
    ch.tangent(cpos);
    ctx.setTransform(ch.vect.x, ch.vect.y, -ch.vect.y, ch.vect.x, ch.vec.x, ch.vec.y);
    ctx.fillText(text[i],0,0);

    pos += widths[i] / 2;
  }
  ctx.restore();
}

```

La función de ayuda de la curva está diseñada para aumentar el rendimiento de los puntos de búsqueda en el bezier.

```

// helper function locates points on bezier curves.
function curveHelper(x1, y1, x2, y2, x3, y3, x4, y4){
  var tx1, ty1, tx2, ty2, tx3, ty3, tx4, ty4;
  var a,b,c,u;
  var vec,currentPos,vecl,vect;
  vec = {x:0,y:0};
  vecl = {x:0,y:0};
  vect = {x:0,y:0};
  quad = false;
  currentPos = 0;
  currentDist = 0;
  if(x4 === undefined || x4 === null){
    quad = true;
    x4 = x3;
    y4 = y3;
  }
  var estLen = Math.sqrt((x4 - x1) * (x4 - x1) + (y4 - y1) * (y4 - y1));
  var onePix = 1 / estLen;
  function posAtC(c){

```



```

    tx1 = x1; ty1 = y1;
    tx2 = x2; ty2 = y2;
    tx3 = x3; ty3 = y3;
    tx1 += (tx2 - tx1) * c;
    ty1 += (ty2 - ty1) * c;
    tx2 += (tx3 - tx2) * c;
    ty2 += (ty3 - ty2) * c;
    tx3 += (x4 - tx3) * c;
    ty3 += (y4 - ty3) * c;
    tx1 += (tx2 - tx1) * c;
    ty1 += (ty2 - ty1) * c;
    tx2 += (tx3 - tx2) * c;
    ty2 += (ty3 - ty2) * c;
    vec.x = tx1 + (tx2 - tx1) * c;
    vec.y = ty1 + (ty2 - ty1) * c;
    return vec;
}
function posAtQ(c){
    tx1 = x1; ty1 = y1;
    tx2 = x2; ty2 = y2;
    tx1 += (tx2 - tx1) * c;
    ty1 += (ty2 - ty1) * c;
    tx2 += (x3 - tx2) * c;
    ty2 += (y3 - ty2) * c;
    vec.x = tx1 + (tx2 - tx1) * c;
    vec.y = ty1 + (ty2 - ty1) * c;
    return vec;
}
function forward(dist){
    var step;
    helper.posAt(currentPos);

    while(currentDist < dist){
        vec1.x = vec.x;
        vec1.y = vec.y;
        currentPos += onePix;
        helper.posAt(currentPos);
        currentDist += step = Math.sqrt((vec.x - vec1.x) * (vec.x - vec1.x) + (vec.y -
vec1.y) * (vec.y - vec1.y));

    }
    currentPos -= ((currentDist - dist) / step) * onePix
    currentDist -= step;
    helper.posAt(currentPos);
    currentDist += Math.sqrt((vec.x - vec1.x) * (vec.x - vec1.x) + (vec.y - vec1.y) *
(vec.y - vec1.y));
    return currentPos;
}

function tangentQ(pos){
    a = (1-pos) * 2;
    b = pos * 2;
    vect.x = a * (x2 - x1) + b * (x3 - x2);
    vect.y = a * (y2 - y1) + b * (y3 - y2);
    u = Math.sqrt(vect.x * vect.x + vect.y * vect.y);
    vect.x /= u;
    vect.y /= u;
}
function tangentC(pos){
    a = (1-pos)
    b = 6 * a * pos;

```

```

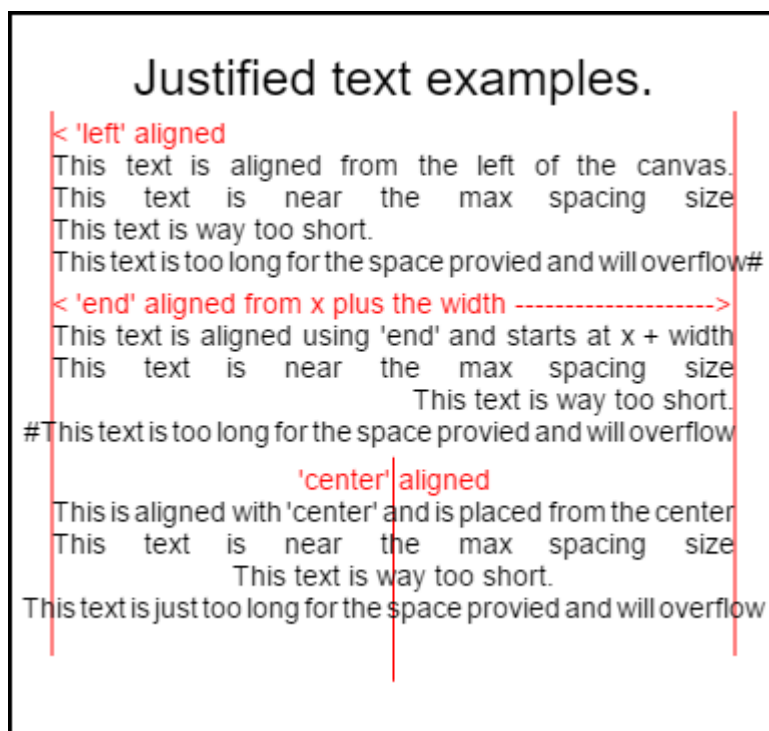
a *= 3 * a;
c = 3 * pos * pos;
vect.x = -x1 * a + x2 * (a - b) + x3 * (b - c) + x4 * c;
vect.y = -y1 * a + y2 * (a - b) + y3 * (b - c) + y4 * c;
u = Math.sqrt(vect.x * vect.x + vect.y * vect.y);
vect.x /= u;
vect.y /= u;
}
var helper = {
  vec : vec,
  vect : vect,
  forward : forward,
}
if(quad){
  helper.posAt = posAtQ;
  helper.tangent = tangentQ;
}else{
  helper.posAt = posAtC;
  helper.tangent = tangentC;
}
return helper
}

```

Texto justificado

Este ejemplo muestra el texto justificado. Agrega funcionalidad adicional a `CanvasRenderingContext2D` al extender su prototipo o como un objeto global `justifiedText` (opcional, ver Nota A).

Ejemplo de renderizado.



El código para representar esta imagen se encuentra en los ejemplos de uso en la parte inferior .

El ejemplo

La función como una función anónima inmediatamente invocada.

```
(function(){
  const FILL = 0;          // const to indicate filltext render
  const STROKE = 1;
  const MEASURE = 2;
  var renderType = FILL; // used internal to set fill or stroke text

  var maxSpaceSize = 3; // Multiplier for max space size. If greater then no justification
  applied
  var minSpaceSize = 0.5; // Multiplier for minimum space size
  var renderTextJustified = function(ctx,text,x,y,width){
    var words, wordsWidth, count, spaces, spaceWidth, adjSpace, renderer, i, textAlign,
    useSize, totalWidth;
    textAlign = ctx.textAlign; // get current align settings
    ctx.textAlign = "left";
    wordsWidth = 0;
    words = text.split(" ").map(word => {
      var w = ctx.measureText(word).width;
      wordsWidth += w;
      return {
        width : w,
        word : word,
      };
    });
    // count = num words, spaces = number spaces, spaceWidth normal space size
    // adjSpace new space size >= min size. useSize Resulting space size used to render
    count = words.length;
    spaces = count - 1;
    spaceWidth = ctx.measureText(" ").width;
    adjSpace = Math.max(spaceWidth * minSpaceSize, (width - wordsWidth) / spaces);
    useSize = adjSpace > spaceWidth * maxSpaceSize ? spaceWidth : adjSpace;
    totalWidth = wordsWidth + useSize * spaces
    if(renderType === MEASURE){ // if measuring return size
      ctx.textAlign = textAlign;
      return totalWidth;
    }
    renderer = renderType === FILL ? ctx.fillText.bind(ctx) : ctx.strokeText.bind(ctx); //
    fill or stroke
    switch(textAlign){
      case "right":
        x -= totalWidth;
        break;
      case "end":
        x += width - totalWidth;
        break;
      case "center": // intentional fall through to default
        x -= totalWidth / 2;
      default:
    }
    if(useSize === spaceWidth){ // if space size unchanged
      renderer(text,x,y);
    } else {
      for(i = 0; i < count; i += 1){
        renderer(words[i].word,x,y);
        x += words[i].width;
        x += useSize;
      }
    }
  }
});
```

```

    }
  }
  ctx.textAlign = textAlign;
}
// Parse vet and set settings object.
var justifiedTextSettings = function(settings){
  var min,max;
  var vetNumber = (num, defaultNum) => {
    num = num !== null && num !== null && !isNaN(num) ? num : defaultNum;
    if(num < 0){
      num = defaultNum;
    }
    return num;
  }
  if(settings === undefined || settings === null){
    return;
  }
  max = vetNumber(settings.maxSpaceSize, maxSpaceSize);
  min = vetNumber(settings.minSpaceSize, minSpaceSize);
  if(min > max){
    return;
  }
  minSpaceSize = min;
  maxSpaceSize = max;
}
// define fill text
var fillJustifyText = function(text, x, y, width, settings){
  justifiedTextSettings(settings);
  renderType = FILL;
  renderTextJustified(this, text, x, y, width);
}
// define stroke text
var strokeJustifyText = function(text, x, y, width, settings){
  justifiedTextSettings(settings);
  renderType = STROKE;
  renderTextJustified(this, text, x, y, width);
}
// define measure text
var measureJustifiedText = function(text, width, settings){
  justifiedTextSettings(settings);
  renderType = MEASURE;
  return renderTextJustified(this, text, 0, 0, width);
}
// code point A
// set the prototypes
CanvasRenderingContext2D.prototype.fillJustifyText = fillJustifyText;
CanvasRenderingContext2D.prototype.strokeJustifyText = strokeJustifyText;
CanvasRenderingContext2D.prototype.measureJustifiedText = measureJustifiedText;
// code point B

// optional code if you do not wish to extend the CanvasRenderingContext2D prototype
/* Uncomment from here to the closing comment
window.justifiedText = {
  fill : function(ctx, text, x, y, width, settings){
    justifiedTextSettings(settings);
    renderType = FILL;
    renderTextJustified(ctx, text, x, y, width);
  },
  stroke : function(ctx, text, x, y, width, settings){
    justifiedTextSettings(settings);
    renderType = STROKE;
  }
};

```

```

        renderTextJustified(ctx, text, x, y, width);
    },
    measure : function(ctx, text, width, settings){
        justifiedTextSettings(settings);
        renderType = MEASURE;
        return renderTextJustified(ctx, text, 0, 0, width);
    }
}
to here*/
})();

```

Nota A: Si no desea extender el prototipo de `CanvasRenderingContext2D` del ejemplo todo el código entre `// code point A` y `// code point B` y elimine el comentario del código marcado `/* Uncomment from here to the closing comment`

Cómo utilizar

Se agregan tres funciones a `CanvasRenderingContext2D` y están disponibles para todos los objetos de contexto 2D creados.

- `ctx.fillJustifyText` (texto, x, y, ancho, [configuración]);
- `ctx.strokeJustifyText` (texto, x, y, ancho, [configuración]);
- `ctx.measureJustifiedText` (texto, ancho, [configuración]);

Función de texto de relleno y trazo Texto de relleno o trazo y use los mismos argumentos.

`measureJustifiedText` devolverá el ancho real al que se representaría el texto. Esto puede ser igual, menor o mayor que el `width` del argumento dependiendo de la configuración actual.

Nota: los argumentos dentro de `[y]` son opcionales.

Argumentos de función

- **texto:** cadena que contiene el texto que se va a representar.
- **x, y:** Coordina para representar el texto en.
- **ancho:** ancho del texto justificado. El texto aumentará / disminuirá los espacios entre las palabras para ajustarse al ancho. Si el espacio entre palabras es mayor que `maxSpaceSize` (predeterminado = 6), se utilizará el espaciado normal y el texto no llenará el ancho requerido. Si el espaciado es menor que `minSpaceSize` (predeterminado = 0.5) el espaciado normal de tiempo, entonces se usa el tamaño de espacio mínimo y el texto sobrepasará el ancho solicitado
- **Configuraciones:** Opcional. Objeto que contiene tamaños de espacio mínimo y máximo.

El argumento de `settings` es opcional y, si no se incluye, la representación de texto usará la última configuración definida o la predeterminada (que se muestra a continuación).

Tanto el mínimo como el máximo son los tamaños mínimo y máximo para el carácter [espacio]

que separa las palabras. El valor predeterminado de `maxSpaceSize = 6` significa que cuando el espacio entre los caracteres es $> 63 * \text{ctx.measureText}("") .width$ el texto no se justificará. Si el texto que se va a justificar tiene espacios menores que `minSpaceSize = 0.5` (valor predeterminado $0.5 * \text{ctx.measureText}(" ") .width$) el espaciado se establecerá en `minSpaceSize * \text{ctx.measureText}(" ") .width` y el texto resultante se rebasará El ancho justificante.

Se aplican las siguientes reglas, min y max deben ser números. Si no, entonces los valores asociados no serán cambiados. Si `minSpaceSize` es más grande que `maxSpaceSize` ambas configuraciones de entrada no son válidas y el máximo máximo no se cambiará.

Ejemplo de objeto de configuración con valores predeterminados

```
settings = {
  maxSpaceSize : 6; // Multiplier for max space size.
  minSpaceSize : 0.5; // Multiplier for minimum space size
};
```

NOTA: Estas funciones de texto introducen un cambio de comportamiento sutil para la propiedad `textAlign` del contexto 2D. 'Izquierda', 'derecha', 'centro' y 'inicio' se comportan como se espera, pero 'final' no se alinearán desde la derecha del argumento de la función `x` sino desde la derecha de `x + width`

Nota: las configuraciones (tamaño de espacio mínimo y máximo) son globales para todos los objetos de contexto 2D.

Ejemplos de uso

```
var i = 0;
text[i++] = "This text is aligned from the left of the canvas.";
text[i++] = "This text is near the max spacing size";
text[i++] = "This text is way too short.";
text[i++] = "This text is too long for the space provied and will overflow#";
text[i++] = "This text is aligned using 'end' and starts at x + width";
text[i++] = "This text is near the max spacing size";
text[i++] = "This text is way too short.";
text[i++] = "#This text is too long for the space provied and will overflow";
text[i++] = "This is aligned with 'center' and is placed from the center";
text[i++] = "This text is near the max spacing size";
text[i++] = "This text is way too short.";
text[i++] = "This text is just too long for the space provied and will overflow";

// ctx is the 2d context
// canvas is the canvas

ctx.clearRect(0,0,w,h);
ctx.font = "25px arial";
ctx.textAlign = "center"
var left = 20;
var center = canvas.width / 2;
var width = canvas.width-left*2;
var y = 40;
var size = 16;
```

```

var i = 0;
ctx.fillText("Justified text examples.",center,y);
y+= 40;
ctx.font = "14px arial";
ctx.textAlign = "left"
var ww = ctx.measureJustifiedText(text[0], width);
var setting = {
    maxSpaceSize : 6,
    minSpaceSize : 0.5
}
ctx.strokeStyle = "red"
ctx.beginPath();
ctx.moveTo(left,y - size * 2);
ctx.lineTo(left, y + size * 15);
ctx.moveTo(canvas.width - left,y - size * 2);
ctx.lineTo(canvas.width - left, y + size * 15);
ctx.stroke();
ctx.textAlign = "left";
ctx.fillStyle = "red";
ctx.fillText("< 'left' aligned",left,y - size)
ctx.fillStyle = "black";
ctx.fillJustifyText(text[i++], left, y, width, setting); // settings is remembered
ctx.fillJustifyText(text[i++], left, y+=size, width);
ctx.fillJustifyText(text[i++], left, y+=size, width);
ctx.fillJustifyText(text[i++], left, y+=size, width);
y += 2.3*size;
ctx.fillStyle = "red";
ctx.fillText("< 'end' aligned from x plus the width ----->",left,y - size)
ctx.fillStyle = "black";
ctx.textAlign = "end";
ctx.fillJustifyText(text[i++], left, y, width);
ctx.fillJustifyText(text[i++], left, y+=size, width);
ctx.fillJustifyText(text[i++], left, y+=size, width);
ctx.fillJustifyText(text[i++], left, y+=size, width);

y += 40;
ctx.strokeStyle = "red"
ctx.beginPath();
ctx.moveTo(center,y - size * 2);
ctx.lineTo(center, y + size * 5);
ctx.stroke();
ctx.textAlign = "center";
ctx.fillStyle = "red";
ctx.fillText("'center' aligned",center,y - size)
ctx.fillStyle = "black";
ctx.fillJustifyText(text[i++], center, y, width);
ctx.fillJustifyText(text[i++], center, y+=size, width);
ctx.fillJustifyText(text[i++], center, y+=size, width);
ctx.fillJustifyText(text[i++], center, y+=size, width);

```

Párrafos justificados.

Representa el texto como párrafos justificados. **REQUIERE** el ejemplo de **texto justificado**

Ejemplo de render

Justified paragraph examples.

Typesetting is the composition of text by means of arranging physical types or the digital equivalents. Stored letters and other symbols are retrieved and ordered according to a language's orthography for visual display. Typesetting requires the prior process of designing a font. One significant effect of typesetting was that authorship of works could be spotted more easily; making it difficult for copiers who have not gained permission.

Typesetting is the composition of text by means of arranging physical types or the digital equivalents. Stored letters and other symbols are retrieved and ordered according to a language's orthography for visual display. Typesetting requires the prior process of designing a font. One significant effect of typesetting was that authorship of works could be spotted more easily; making it difficult for copiers who have not gained permission.

El párrafo superior tiene **setting.compact = true** y bottom **false** y el interlineado es **1.2** en lugar del **1.5** predeterminado. Representado por el código de uso de la parte inferior de este ejemplo.

Código de ejemplo

```
// Requires justified text extensions
(function(){
  // code point A
  if(typeof CanvasRenderingContext2D.prototype.fillJustifyText !== "function"){
    throw new ReferenceError("Justified Paragraph extension missing required
CanvasRenderingContext2D justified text extension");
  }
  var maxSpaceSize = 3; // Multiplier for max space size. If greater then no justificatoin
applied
  var minSpaceSize = 0.5; // Multiplier for minimum space size
  var compact = true; // if true then try and fit as many words as possible. If false then
try to get the spacing as close as possible to normal
  var lineSpacing = 1.5; // space between lines
  const noJustifySetting = { // This setting forces justified text off. Used to render last
line of paragraph.
    minSpaceSize : 1,
    maxSpaceSize : 1,
  }

  // Parse vet and set settings object.
  var justifiedTextSettings = function(settings){
    var min, max;
    var vetNumber = (num, defaultNum) => {
      num = num !== null && num !== null && !isNaN(num) ? num : defaultNum;
      return num < 0 ? defaultNum : num;
    }
    if(settings === undefined || settings === null){ return; }
    compact = settings.compact === true ? true : settings.compact === false ? false :
compact;
    max = vetNumber(settings.maxSpaceSize, maxSpaceSize);
    min = vetNumber(settings.minSpaceSize, minSpaceSize);
```



```

    lineSpacing = vetNumber(settings.lineSpacing, lineSpacing);
    if(min > max){ return; }
    minSpaceSize = min;
    maxSpaceSize = max;
}
var getFontSize = function(font){ // get the font size.
    var numFind = /[0-9]+/;
    var number = numFind.exec(font)[0];
    if(isNaN(number)){
        throw new ReferenceError("justifiedPar Cant find font size");
    }
    return Number(number);
}
function justifiedPar(ctx, text, x, y, width, settings, stroke){
    var spaceWidth, minS, maxS, words, count, lines, lineWidth, lastLineWidth, lastSize,
i, renderer, fontSize, adjSpace, spaces, word, lineWords, lineFound;
    spaceWidth = ctx.measureText(" ").width;
    minS = spaceWidth * minSpaceSize;
    maxS = spaceWidth * maxSpaceSize;
    words = text.split(" ").map(word => { // measure all words.
        var w = ctx.measureText(word).width;
        return {
            width : w,
            word : word,
        };
    });
    // count = num words, spaces = number spaces, spaceWidth normal space size
    // adjSpace new space size >= min size. useSize Resulting space size used to render
    count = 0;
    lines = [];
    // create lines by shifting words from the words array until the spacing is optimal.
If compact
    // true then will true and fit as many words as possible. Else it will try and get the
spacing as
    // close as possible to the normal spacing
    while(words.length > 0){
        lastLineWidth = 0;
        lastSize = -1;
        lineFound = false;
        // each line must have at least one word.
        word = words.shift();
        lineWidth = word.width;
        lineWords = [word.word];
        count = 0;
        while(lineWidth < width && words.length > 0){ // Add words to line
            word = words.shift();
            lineWidth += word.width;
            lineWords.push(word.word);
            count += 1;
            spaces = count - 1;
            adjSpace = (width - lineWidth) / spaces;
            if(minS > adjSpace){ // if spacing less than min remove last word and finish
line
                lineFound = true;
                words.unshift(word);
                lineWords.pop();
            }else{
                if(!compact){ // if compact mode
                    if(adjSpace < spaceWidth){ // if less than normal space width
                        if(lastSize === -1){
                            lastSize = adjSpace;

```

```

        }
        // check if with last word on if its closer to space width
        if(Math.abs(spaceWidth - adjSpace) < Math.abs(spaceWidth -
lastSize)){
            lineFound = true; // yes keep it
        }else{
            words.unshift(word); // no better fit if last word removes
            lineWords.pop();
            lineFound = true;
        }
    }
}
    }
    lastSize = adjSpace; // remember spacing
}
    lines.push(lineWords.join(" ")); // and the line
}
// lines have been worked out get font size, render, and render all the lines. last
// line may need to be rendered as normal so it is outside the loop.
fontSize = getFontSize(ctx.font);
renderer = stroke === true ? ctx.strokeJustifyText.bind(ctx) :
ctx.fillJustifyText.bind(ctx);
for(i = 0; i < lines.length - 1; i++){
    renderer(lines[i], x, y, width, settings);
    y += lineSpacing * fontSize;
}
if(lines.length > 0){ // last line if left or start aligned for no justify
    if(ctx.textAlign === "left" || ctx.textAlign === "start"){
        renderer(lines[lines.length - 1], x, y, width, noJustifySetting);
        ctx.measureJustifiedText("", width, settings);
    }else{
        renderer(lines[lines.length - 1], x, y, width);
    }
}
// return details about the paragraph.
y += lineSpacing * fontSize;
return {
    nextLine : y,
    fontSize : fontSize,
    lineHeight : lineSpacing * fontSize,
};
}
// define fill
var fillParagraphText = function(text, x, y, width, settings){
    justifiedTextSettings(settings);
    settings = {
        minSpaceSize : minSpaceSize,
        maxSpaceSize : maxSpaceSize,
    };
    return justifiedPar(this, text, x, y, width, settings);
}
// define stroke
var strokeParagraphText = function(text, x, y, width, settings){
    justifiedTextSettings(settings);
    settings = {
        minSpaceSize : minSpaceSize,
        maxSpaceSize : maxSpaceSize,
    };
    return justifiedPar(this, text, x, y, width, settings,true);
}
CanvasRenderingContext2D.prototype.fillParaText = fillParagraphText;

```

```
CanvasRenderingContext2D.prototype.strokeParaText = strokeParagraphText;
})();
```

NOTA esto extiende el prototipo de `CanvasRenderingContext2D`. Si no desea que esto suceda, utilice el **texto Justificado de ejemplo** para averiguar cómo cambiar este ejemplo para que sea parte del espacio de nombres global.

NOTA `CanvasRenderingContext2D.prototype.fillJustifyText` un `ReferenceError` si este ejemplo no puede encontrar la función `CanvasRenderingContext2D.prototype.fillJustifyText`

Cómo utilizar

```
ctx.fillParaText(text, x, y, width, [settings]);
ctx.strokeParaText(text, x, y, width, [settings]);
```

Consulte el **texto justificado** para obtener detalles sobre los argumentos. Los argumentos entre [y] son opcionales.

El argumento de `settings` tiene dos propiedades adicionales.

- **compacto**: predeterminado `true`. Si es verdadero intenta empaquetar tantas palabras como sea posible por línea. Si es falso, intenta obtener el espacio entre palabras lo más cercano posible al espacio normal.
- **lineSpacing** Default `1.5`. Espacio por línea por defecto `1.5` la distancia de en línea a la siguiente en términos de tamaño de fuente

Las propiedades que faltan en el objeto de configuración se predeterminarán a sus valores predeterminados o a los últimos valores válidos. Las propiedades solo se cambiarán si los nuevos valores son válidos. Para los valores válidos `compact` los valores booleanos son `true` o `false` valores de verdad no se consideran válidos

Devolver objeto

Las dos funciones devuelven un objeto que contiene información para ayudarlo a colocar el siguiente párrafo. El objeto contiene las siguientes propiedades.

- **nextLine** Posición de la siguiente línea después de los píxeles del párrafo.
- Tamaño de la fuente **fontSize**. (Tenga en cuenta que solo use las fuentes definidas en píxeles, por ejemplo, `14px arial`)
- **lineHeight** Distance en píxeles de una línea a la siguiente

Este ejemplo utiliza un algoritmo simple que trabaja una línea a la vez para encontrar el mejor ajuste para un párrafo. Esto no significa que sea el mejor ajuste (más bien el mejor del algoritmo) Es posible que desee mejorar el algoritmo creando un algoritmo de línea de paso múltiple sobre las líneas generadas. Mover palabras desde el final de una línea hasta el comienzo de la siguiente, o desde el inicio hasta el final. La mejor apariencia se logra cuando el espaciado en

todo el párrafo tiene la variación más pequeña y es el más cercano al espaciado normal del texto.

Como este ejemplo depende del ejemplo de **texto justificado**, el código es muy similar. Es posible que desee mover los dos en una función. Reemplace la función `justifiedTextSettings` en el otro ejemplo con la utilizada en este ejemplo. Luego copie todo el resto del código de este ejemplo en el cuerpo de la función anónima del ejemplo de **texto justificado**. Ya no tendrá que probar las dependencias encontradas en `// Code point A` Se puede eliminar.

Ejemplo de uso

```
ctx.font = "25px arial";
ctx.textAlign = "center"

var left = 10;
var center = canvas.width / 2;
var width = canvas.width-left*2;
var y = 20;
var size = 16;
var i = 0;
ctx.fillText("Justified paragraph examples.",center,y);
y+= 30;
ctx.font = "14px arial";
ctx.textAlign = "left"
// set para settings
var setting = {
  maxSpaceSize : 6,
  minSpaceSize : 0.5,
  lineSpacing : 1.2,
  compact : true,
}
// Show the left and right bounds.
ctx.strokeStyle = "red"
ctx.beginPath();
ctx.moveTo(left,y - size * 2);
ctx.lineTo(left, y + size * 15);
ctx.moveTo(canvas.width - left,y - size * 2);
ctx.lineTo(canvas.width - left, y + size * 15);
ctx.stroke();
ctx.textAlign = "left";
ctx.fillStyle = "black";

// Draw paragraph
var line = ctx.fillParaText(para, left, y, width, setting); // settings is remembered

// Next paragraph
y = line.nextLine + line.lineHeight;
setting.compact = false;
ctx.fillParaText(para, left, y, width, setting);
```

Nota: Para el texto alineado a la `left` o `start` la última línea del párrafo siempre tendrá un espaciado normal. Para todas las demás alineaciones, la última línea se trata como todas las demás.

Nota: Puede insertar el inicio del párrafo con espacios. Aunque esto puede no ser consistente de un párrafo a otro. Siempre es bueno aprender lo que hace una función

y modificarla. Un ejercicio sería agregar una configuración a la configuración que sangra la primera línea en una cantidad fija. Indica que el bucle while necesitará hacer que la primera palabra aparezca más grande (+ sangría) `words[0].width += ?` y luego, al renderizar las líneas, sangra la primera línea.

Lea Texto en línea: <https://riptutorial.com/es/html5-canvas/topic/5235/texto>

Capítulo 18: Transformaciones

Examples

Dibujar rápidamente muchas imágenes traducidas, escaladas y rotadas.

Hay muchas situaciones en las que desea dibujar una imagen que se gira, se escala y se traduce. La rotación debe ocurrir alrededor del centro de la imagen. Esta es la forma más rápida de hacerlo en el lienzo 2D. Estas funciones se adaptan bien a los juegos en 2D, donde la expectativa es generar unos cientos de imágenes de hasta más de 1000 imágenes cada 60 segundos. (Depende del hardware)

```
// assumes that the canvas context is in ctx and in scope
function drawImageRST(image, x, y, scale, rotation){
    ctx.setTransform(scale, 0, 0, scale, x, y); // set the scale and translation
    ctx.rotate(rotation); // add the rotation
    ctx.drawImage(image, -image.width / 2, -image.height / 2); // draw the image offset by
    half its width and height
}
```

Una variante también puede incluir el valor alfa que es útil para los sistemas de partículas.

```
function drawImageRST_Alpha(image, x, y, scale, rotation, alpha){
    ctx.setTransform(scale, 0, 0, scale, x, y); // set the scale and translation
    ctx.rotate(rotation); // add the rotation
    ctx.globalAlpha = alpha;
    ctx.drawImage(image, -image.width / 2, -image.height / 2); // draw the image offset by
    half its width and height
}
```

Es importante tener en cuenta que ambas funciones dejan el contexto del lienzo en un estado aleatorio. Aunque las funciones no se verán afectadas por otros renderizando mi ser. Cuando haya terminado de representar las imágenes, es posible que deba restaurar la transformación predeterminada

```
ctx.setTransform(1, 0, 0, 1, 0, 0); // set the context transform back to the default
```

Si usa la versión alfa (segundo ejemplo) y luego la versión estándar, tendrá que asegurarse de que se restaure el estado alfa global

```
ctx.globalAlpha = 1;
```

Un ejemplo del uso de las funciones anteriores para representar algunas partículas y algunas imágenes.

```
// assume particles to contain an array of particles
for(var i = 0; i < particles.length; i++){
    var p = particles[i];
```

```

drawImageRST_Alpha(p.image, p.x, p.y, p.scale, p.rot, p.alpha);
// no need to rest the alpha in the loop
}
// you need to reset the alpha as it can be any value
ctx.globalAlpha = 1;

drawImageRST(myImage, 100, 100, 1, 0.5); // draw an image at 100,100
// no need to reset the transform
drawImageRST(myImage, 200, 200, 1, -0.5); // draw an image at 200,200
ctx.setTransform(1,0,0,1,0,0); // reset the transform

```

Girar una imagen o camino alrededor de su punto central



Los pasos 1 a 5 a continuación permiten que cualquier imagen o forma de trayectoria se mueva a cualquier lugar del lienzo y se gire a cualquier ángulo sin cambiar ninguna de las coordenadas del punto original de la imagen / forma de trayectoria.

1. Mueva el origen del lienzo [0,0] al punto central de la forma

```
context.translate( shapeCenterX, shapeCenterY );
```

2. Gire el lienzo en el ángulo deseado (en radianes)

```
context.rotate( radianAngle );
```

3. Mueva el origen del lienzo de nuevo a la esquina superior izquierda

```
context.translate( -shapeCenterX, -shapeCenterY );
```

4. Dibuja la imagen o la forma del camino usando sus coordenadas originales.

```
context.fillRect( shapeX, shapeY, shapeWidth, shapeHeight );
```

5. ¡Siempre limpia! Establecer el estado de transformación de nuevo a donde estaba antes de # 1

- **Paso # 5, Opción # 1:** Deshacer todas las transformaciones en orden inverso

```
// undo #3
context.translate( shapeCenterX, shapeCenterY );
// undo #2
context.rotate( -radianAngle );
// undo #1
context.translate( -shapeCenterX, shapeCenterY );
```

- **Paso # 5, Opción # 2:** Si el lienzo estaba en un estado sin transformar (el valor predeterminado) antes de comenzar el paso # 1, puede deshacer los efectos de los pasos # 1-3 restableciendo todas las transformaciones a su estado predeterminado

```
// set transformation to the default state (==no transformation applied)
context.setTransform(1,0,0,1,0,0)
```

Código de ejemplo de demostración:

```
// canvas references & canvas styling
var canvas=document.createElement("canvas");
canvas.style.border='1px solid red';
document.body.appendChild(canvas);
canvas.width=378;
canvas.height=256;
var ctx=canvas.getContext("2d");
ctx.fillStyle='green';
ctx.globalAlpha=0.35;

// define a rectangle to rotate
var rect={ x:100, y:100, width:175, height:50 };

// draw the rectangle unrotated
ctx.fillRect( rect.x, rect.y, rect.width, rect.height );

// draw the rectangle rotated by 45 degrees (==PI/4 radians)
ctx.translate( rect.x+rect.width/2, rect.y+rect.height/2 );
ctx.rotate( Math.PI/4 );
ctx.translate( -rect.x-rect.width/2, -rect.y-rect.height/2 );
ctx.fillRect( rect.x, rect.y, rect.width, rect.height );
```

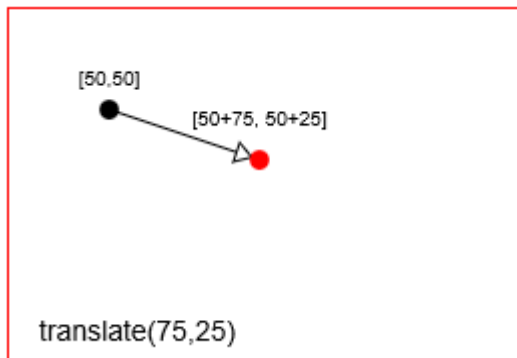
Introducción a las transformaciones

Las transformaciones alteran la posición inicial de un punto dado al mover, rotar y escalar ese punto.

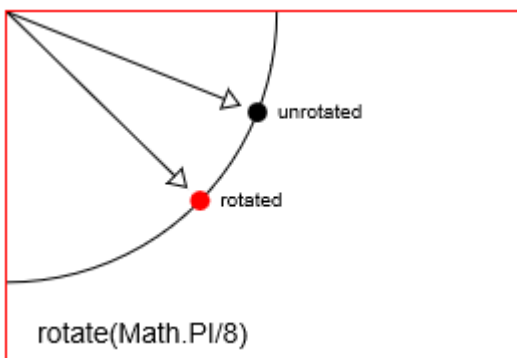
- **Traducción:** Mueve un punto por una `distanceX` y una `distanceY`.
- **Rotación:** Gira un punto en un `radian angle` alrededor de su punto de rotación. El punto de rotación predeterminado en Lienzo HTML es el origen superior izquierdo [`x = 0, y = 0`] del Lienzo. Pero puedes reposicionar el punto de rotación usando traducciones.
- **Escalado:** escala la posición de un punto mediante un `scalingFactorX` y `scalingFactorY` de `scalingFactorY` desde su punto de escala. El punto de escala predeterminado en el Lienzo HTML es el origen superior izquierdo [`x = 0, y = 0`] del Lienzo. Pero puedes reposicionar el punto de escala usando traducciones.

También puede hacer transformaciones menos comunes, como cizallamiento (sesgo), configurando directamente la matriz de transformación del lienzo usando `context.transform`.

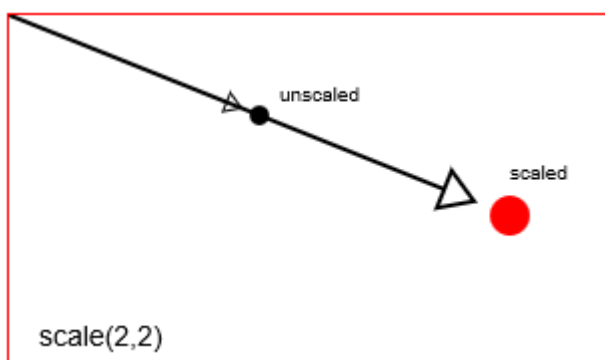
Traducir (== mover) un punto con `context.translate(75, 25)`



Girar un punto con `context.rotate(Math.PI/8)`



Escala un punto con `context.scale(2, 2)`



Canvas realmente logra transformaciones al alterar todo el sistema de coordenadas del canvas.

- `context.translate` moverá el origen del lienzo [0,0] desde la esquina superior izquierda a una nueva ubicación.
- `context.rotate` rotará todo el sistema de coordenadas del lienzo alrededor del origen.
- `context.scale` escalará todo el sistema de coordenadas del lienzo alrededor del origen. Piense en esto como un aumento del tamaño de cada x, y en el lienzo: `every x*=scaleX` y `every y*=scaleY`.

Las transformaciones del lienzo son persistentes. Todos los dibujos nuevos continuarán transformándose hasta que restablezca la transformación del lienzo a su estado predeterminado

(== totalmente sin transformar). Puede restablecer la configuración predeterminada con:

```
// reset context transformations to the default (untransformed) state
context.setTransform(1,0,0,1,0,0);
```

Una matriz de transformación para realizar un seguimiento de las formas traducidas, rotadas y escaladas

Canvas le permite `context.translate`, `context.rotate` y `context.scale` para dibujar su forma en la posición y el tamaño que necesita.

Canvas utiliza una matriz de transformación para realizar un seguimiento eficiente de las transformaciones.

- Puedes cambiar la matriz de Canvas con `context.transform`
- Puede cambiar la matriz de Canvas con los comandos de `translate`, `rotate` & `scale` individuales
- Puede sobrescribir completamente la matriz de Canvas con `context.setTransform`,
- *Pero no puede leer la matriz de transformación interna de Canvas, es solo de escritura.*

¿Por qué usar una matriz de transformación?

Una matriz de transformación le permite agregar muchas traducciones individuales, rotaciones y escalas en una matriz única y fácil de volver a aplicar.

Durante animaciones complejas, puede aplicar docenas (o cientos) de transformaciones a una forma. Al utilizar una matriz de transformación, puede (casi) volver a aplicar esas docenas de transformaciones con una sola línea de código.

Algunos ejemplos de uso:

- **Probar si el mouse está dentro de una forma que ha traducido, rotado y escalado**

Existe un `context.isPointInPath` incorporado que comprueba si un punto (por ejemplo, el mouse) está dentro de una forma de trayectoria, pero esta prueba incorporada es muy lenta en comparación con las pruebas que usan una matriz.

Comprobar de manera eficiente si el mouse está dentro de una forma implica tomar la posición del mouse informada por el navegador y transformarla de la misma manera que se transformó la forma. Luego, puede aplicar la prueba de impacto como si la forma no se hubiera transformado.

- **Redibuje una forma que haya sido traducida, girada y escalada extensivamente.**

En lugar de volver a aplicar transformaciones individuales con múltiples `.translate`, `.rotate`, `.scale`, puede aplicar todas las transformaciones agregadas en una sola línea de código.

- **Formas de prueba de colisión que han sido traducidas, rotadas y escaladas.**

Puede usar geometría y trigonometría para calcular los puntos que conforman las formas transformadas, pero es más rápido usar una matriz de transformación para calcular esos puntos.

Una matriz de transformación "clase"

Este código refleja los comandos nativos de transformación `context.translate`, `context.rotate`, `context.scale`. A diferencia de la matriz del lienzo nativo, esta matriz es legible y reutilizable.

Métodos:

- `translate`, `rotate`, `scale` los comandos de transformación de contexto y te permite alimentar transformaciones en la matriz. La matriz mantiene eficientemente las transformaciones agregadas.
- `setContextTransform` toma un contexto y establece la matriz de ese contexto igual a esta matriz de transformación. Esto vuelve a aplicar de manera eficiente todas las transformaciones almacenadas en esta matriz al contexto.
- `resetContextTransform` restablece la transformación del contexto a su estado predeterminado (== sin transformar).
- `getTransformedPoint` toma un punto de coordenadas sin transformar y lo convierte en un punto transformado.
- `getScreenPoint` toma un punto de coordenadas transformado y lo convierte en un punto sin transformar.
- `getMatrix` devuelve las transformaciones agregadas en forma de matriz de matriz.

Código:

```
var TransformationMatrix=( function(){
  // private
  var self;
  var m=[1,0,0,1,0,0];
  var reset=function(){ var m=[1,0,0,1,0,0]; }
  var multiply=function(mat) {
    var m0=m[0]*mat[0]+m[2]*mat[1];
    var m1=m[1]*mat[0]+m[3]*mat[1];
    var m2=m[0]*mat[2]+m[2]*mat[3];
    var m3=m[1]*mat[2]+m[3]*mat[3];
    var m4=m[0]*mat[4]+m[2]*mat[5]+m[4];
    var m5=m[1]*mat[4]+m[3]*mat[5]+m[5];
    m=[m0,m1,m2,m3,m4,m5];
  }
  var screenPoint=function(transformedX,transformedY){
    // invert
    var d =1/(m[0]*m[3]-m[1]*m[2]);
    im=[ m[3]*d, -m[1]*d, -m[2]*d, m[0]*d, d*(m[2]*m[5]-m[3]*m[4]), d*(m[1]*m[4]-
m[0]*m[5]) ];
    // point
```

```

        return({
            x:transformedX*im[0]+transformedY*im[2]+im[4],
            y:transformedX*im[1]+transformedY*im[3]+im[5]
        });
    }
    var transformedPoint=function(screenX,screenY){
        return({
            x:screenX*m[0] + screenY*m[2] + m[4],
            y:screenX*m[1] + screenY*m[3] + m[5]
        });
    }
    // public
    function TransformationMatrix(){
        self=this;
    }
    // shared methods
    TransformationMatrix.prototype.translate=function(x,y){
        var mat=[ 1, 0, 0, 1, x, y ];
        multiply(mat);
    };
    TransformationMatrix.prototype.rotate=function(rAngle){
        var c = Math.cos(rAngle);
        var s = Math.sin(rAngle);
        var mat=[ c, s, -s, c, 0, 0 ];
        multiply(mat);
    };
    TransformationMatrix.prototype.scale=function(x,y){
        var mat=[ x, 0, 0, y, 0, 0 ];
        multiply(mat);
    };
    TransformationMatrix.prototype.skew=function(radianX,radianY){
        var mat=[ 1, Math.tan(radianY), Math.tan(radianX), 1, 0, 0 ];
        multiply(mat);
    };
    TransformationMatrix.prototype.reset=function(){
        reset();
    }
    TransformationMatrix.prototype.setContextTransform=function(ctx){
        ctx.setTransform(m[0],m[1],m[2],m[3],m[4],m[5]);
    }
    TransformationMatrix.prototype.resetContextTransform=function(ctx){
        ctx.setTransform(1,0,0,1,0,0);
    }
    TransformationMatrix.prototype.getTransformedPoint=function(screenX,screenY){
        return(transformedPoint(screenX,screenY));
    }
    TransformationMatrix.prototype.getScreenPoint=function(transformedX,transformedY){
        return(screenPoint(transformedX,transformedY));
    }
    TransformationMatrix.prototype.getMatrix=function(){
        var clone=[m[0],m[1],m[2],m[3],m[4],m[5]];
        return(clone);
    }
    // return public
    return(TransformationMatrix);
})();

```

Manifestación:

Esta demostración utiliza la Matriz de transformación "Clase" anterior para:

- Seguimiento (== guardar) la matriz de transformación de un rectángulo.
- Redibuje el rectángulo transformado sin utilizar comandos de transformación de contexto.
- Probar si el ratón ha hecho clic dentro del rectángulo transformado.

Código:

```

<!doctype html>
<html>
<head>
<style>
  body{ background-color:white; }
  #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

  var canvas=document.getElementById("canvas");
  var ctx=canvas.getContext("2d");
  var cw=canvas.width;
  var ch=canvas.height;
  function reOffset(){
    var BB=canvas.getBoundingClientRect();
    offsetX=BB.left;
    offsetY=BB.top;
  }
  var offsetX,offsetY;
  reOffset();
  window.onscroll=function(e){ reOffset(); }
  window.onresize=function(e){ reOffset(); }

  // Transformation Matrix "Class"

  var TransformationMatrix=( function(){
    // private
    var self;
    var m=[1,0,0,1,0,0];
    var reset=function(){ var m=[1,0,0,1,0,0]; }
    var multiply=function(mat){
      var m0=m[0]*mat[0]+m[2]*mat[1];
      var m1=m[1]*mat[0]+m[3]*mat[1];
      var m2=m[0]*mat[2]+m[2]*mat[3];
      var m3=m[1]*mat[2]+m[3]*mat[3];
      var m4=m[0]*mat[4]+m[2]*mat[5]+m[4];
      var m5=m[1]*mat[4]+m[3]*mat[5]+m[5];
      m=[m0,m1,m2,m3,m4,m5];
    }
    var screenPoint=function(transformedX,transformedY){
      // invert
      var d =1/(m[0]*m[3]-m[1]*m[2]);
      im=[ m[3]*d, -m[1]*d, -m[2]*d, m[0]*d, d*(m[2]*m[5]-m[3]*m[4]), d*(m[1]*m[4]-
m[0]*m[5]) ];
      // point
      return({
        x:transformedX*im[0]+transformedY*im[2]+im[4],
        y:transformedX*im[1]+transformedY*im[3]+im[5]
      });
    }
  }
  var transformedPoint=function(screenX,screenY){

```

```

        return({
            x:screenX*m[0] + screenY*m[2] + m[4],
            y:screenX*m[1] + screenY*m[3] + m[5]
        });
    }
    // public
    function TransformationMatrix(){
        self=this;
    }
    // shared methods
    TransformationMatrix.prototype.translate=function(x,y){
        var mat=[ 1, 0, 0, 1, x, y ];
        multiply(mat);
    };
    TransformationMatrix.prototype.rotate=function(rAngle){
        var c = Math.cos(rAngle);
        var s = Math.sin(rAngle);
        var mat=[ c, s, -s, c, 0, 0 ];
        multiply(mat);
    };
    TransformationMatrix.prototype.scale=function(x,y){
        var mat=[ x, 0, 0, y, 0, 0 ];
        multiply(mat);
    };
    TransformationMatrix.prototype.skew=function(radianX,radianY){
        var mat=[ 1, Math.tan(radianY), Math.tan(radianX), 1, 0, 0 ];
        multiply(mat);
    };
    TransformationMatrix.prototype.reset=function(){
        reset();
    }
    TransformationMatrix.prototype.setContextTransform=function(ctx){
        ctx.setTransform(m[0],m[1],m[2],m[3],m[4],m[5]);
    }
    TransformationMatrix.prototype.resetContextTransform=function(ctx){
        ctx.setTransform(1,0,0,1,0,0);
    }
    TransformationMatrix.prototype.getTransformedPoint=function(screenX,screenY){
        return(transformedPoint(screenX,screenY));
    }
    TransformationMatrix.prototype.getScreenPoint=function(transformedX,transformedY){
        return(screenPoint(transformedX,transformedY));
    }
    TransformationMatrix.prototype.getMatrix=function(){
        var clone=[m[0],m[1],m[2],m[3],m[4],m[5]];
        return(clone);
    }
    // return public
    return(TransformationMatrix);
})();

// DEMO starts here

// create a rect and add a transformation matrix
// to track it's translations, rotations & scalings
var rect={x:30,y:30,w:50,h:35,matrix:new TransformationMatrix()};

// draw the untransformed rect in black
ctx.strokeRect(rect.x, rect.y, rect.w, rect.h);
// Demo: label
ctx.font='11px arial';

```

```

ctx.fillText('Untransformed Rect',rect.x,rect.y-10);

// transform the canvas & draw the transformed rect in red
ctx.translate(100,0);
ctx.scale(2,2);
ctx.rotate(Math.PI/12);
// draw the transformed rect
ctx.strokeStyle='red';
ctx.strokeRect(rect.x, rect.y, rect.w, rect.h);
ctx.font='6px arial';
// Demo: label
ctx.fillText('Same Rect: Translated, rotated & scaled',rect.x,rect.y-6);
// reset the context to untransformed state
ctx.setTransform(1,0,0,1,0,0);

// record the transformations in the matrix
var m=rect.matrix;
m.translate(100,0);
m.scale(2,2);
m.rotate(Math.PI/12);

// use the rect's saved transformation matrix to reposition,
//      resize & redraw the rect
ctx.strokeStyle='blue';
drawTransformedRect(rect);

// Demo: instructions
ctx.font='14px arial';
ctx.fillText('Demo: click inside the blue rect',30,200);

// redraw a rect based on it's saved transformation matrix
function drawTransformedRect(r){
    // set the context transformation matrix using the rect's saved matrix
    m.setContextTransform(ctx);
    // draw the rect (no position or size changes needed!)
    ctx.strokeRect( r.x, r.y, r.w, r.h );
    // reset the context transformation to default (==untransformed);
    m.resetContextTransform(ctx);
}

// is the point in the transformed rectangle?
function isPointInTransformedRect(r,transformedX,transformedY){
    var p=r.matrix.getScreenPoint(transformedX,transformedY);
    var x=p.x;
    var y=p.y;
    return(x>r.x && x<r.x+r.w && y>r.y && y<r.y+r.h);
}

// listen for mousedown events
canvas.onmousedown=handleMouseDown;
function handleMouseDown(e){
    // tell the browser we're handling this event
    e.preventDefault();
    e.stopPropagation();
    // get mouse position
    mouseX=parseInt(e.clientX-offsetX);
    mouseY=parseInt(e.clientY-offsetY);
    // is the mouse inside the transformed rect?
    if(isPointInTransformedRect(rect,mouseX,mouseY)){
        alert('You clicked in the transformed Rect');
    }
}

```

```
}

// Demo: redraw transformed rect without using
//       context transformation commands
function drawTransformedRect(r,color){
    var m=r.matrix;
    var tl=m.getTransformedPoint(r.x,r.y);
    var tr=m.getTransformedPoint(r.x+r.w,r.y);
    var br=m.getTransformedPoint(r.x+r.w,r.y+r.h);
    var bl=m.getTransformedPoint(r.x,r.y+r.h);
    ctx.beginPath();
    ctx.moveTo(tl.x,tl.y);
    ctx.lineTo(tr.x,tr.y);
    ctx.lineTo(br.x,br.y);
    ctx.lineTo(bl.x,bl.y);
    ctx.closePath();
    ctx.strokeStyle=color;
    ctx.stroke();
}

}); // end window.onload
</script>
</head>
<body>
    <canvas id="canvas" width=512 height=250></canvas>
</body>
</html>
```

Lea Transformaciones en línea: <https://riptutorial.com/es/html5-canvas/topic/5494/transformaciones>

Creditos

S. No	Capítulos	Contributors
1	Empezando con html5-canvas	almcd , Blindman67 , Community , Daniel Dees , Kaiido , markE , ndugger , Spencer Wieczorek , Stephen Leppik , user2314737
2	Animación	Blindman67 , markE
3	Arrastrando formas de ruta e imágenes sobre lienzo	markE
4	Borrar la pantalla	bjanes , Blindman67 , Kaiido , markE , Mike C , Ronen Ness
5	Caminos	Blindman67 , markE
6	Colisiones e Intersecciones	Blindman67 , markE
7	Compositing	Blindman67 , markE
8	Diseño de respuesta	Blindman67 , markE , mnoronha
9	Gráficos y diagramas	Blindman67 , markE
10	Imágenes	Blindman67 , Kaiido , markE
11	Los tipos de medios y el lienzo.	Blindman67 , Bobby , Kaiido
12	Manipulación de píxeles con "getImageData" y "putImageData"	markE
13	Navegando por un sendero	Blindman67 , markE
14	Oscuridad	markE
15	Poligonos	Blindman67 , markE
16	Ruta (solo sintaxis)	AgataB , markE
17	Texto	almcd , Blindman67 , markE , RamenChef

