HTML5 Canvas

Apuntes para Profesionales

Traducido por:

rortegag

100+ páginas

de consejos y trucos profesionales

Contenidos

Acerca de

Este libro ha sido traducido por rortegag.com

Si desea descargar el libro original, puede descargarlo desde:

https://books.goalkicker.com/HTML5CanvasBook/

Si desea contribuir con una donación, hazlo desde:

https://www.buymeacoffee.com/GoalKickerBooks

Por favor, siéntase libre de compartir este PDF con cualquier persona de forma gratuita, la última versión de este libro se puede descargar desde:

https://goalkicker.com/HTML5CanvasBook

Envíe sus comentarios y correcciones a web@petercv.com

Capítulo 1: Introducción a HTML5 Canvas

Sección 1.1: Detección de la posición del ratón en el canvas

Este ejemplo mostrará cómo obtener la posición del ratón relativa al canvas, de forma que `(0,0)` será la esquina superior izquierda del HTML5 Canvas. Los valores `e.clientX` y `e.clientY` obtendrán las posiciones del ratón relativas a la parte superior del documento, para cambiar esto y que se base en la parte superior del canvas restamos las posiciones `left` y `right` del canvas de los ejes X e Y del cliente.

```
var canvas = document.getElementById("myCanvas");
var ctx = canvas.getContext("2d");
ctx.font = "16px Arial";
canvas.addEventListener("mousemove", function(e) {
      var cRect = canvas.getBoundingClientRect(); // Obtiene la posición CSS y el ancho/alto
      var canvasX = Math.round(e.clientX - cRect.left); // Restar la "izquierda" del canvas
      var canvasY = Math.round(e.clientY - cRect.top); // de las posiciones X/Y para que
      ctx.clearRect(0, 0, canvas.width, canvas.height); // (0,0) parte superior izquierda del
      canvas
      ctx.fillText("X: "+canvasX+", Y: "+canvasY, 10, 20);
});
```

El uso de `Math.round` se debe a asegurar que las posiciones `x,y` son enteras, ya que el rectángulo delimitador del canvas puede no tener posiciones enteras.

Sección 1.2: Tamaño y resolución del canvas

El tamaño de un canvas es el área que ocupa en la página y viene definido por las propiedades CSS `width` y `height`.

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

La resolución del canvas define el número de píxeles que contiene. La resolución se fija estableciendo las propiedades `width` y `height`. Si no se especifica, el canvas por defecto es de 300 por 150 píxeles.

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

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

Esto provocará que cada píxel se estire de forma desigual. El aspecto de los píxeles es 1:2. Cuando el canvas se estira el navegador utilizará el filtrado bilineal. Esto tiene el efecto de difuminar los píxeles que están estirados.

Para obtener los mejores resultados al utilizar el canvas, asegúrate de que la resolución del canvas coincide con el tamaño de la pantalla.

Siguiendo con el estilo CSS anterior para que coincida con el tamaño de la pantalla añadir el canvas con la anchura y la altura ajustada a el mismo número de píxeles que el estilo define.

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

Sección 1.3: rotate

El método `rotate(r)` del contexto 2D rota el canvas la cantidad especificada `r` 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("Hola Mundo", ox, oy);
rotate_ctx = function() {
     // trasladar de modo que el origen sea ahora (ox, oy) el centro del canvas
     ctx.translate(ox, oy);
     // convertir grados a radianes con radianes = (Math.PI/180)*grados
     ctx.rotate((Math.PI / 180) * 15);
     ctx.fillText("Hola Mundo", 0, 0);
     // volver a trasladar
     ctx.translate(-ox, -oy);
};
```

Sección 1.4: Guardar canvas en un archivo de imagen

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

El método puede tomar dos parámetros opcionales `canvas.toDataURL(type, encoderOptions): type` es el formato de la imagen (si se omite, por defecto es `image/png`); `encoderOptions` es un número entre 0 y 1 que indica la calidad de la imagen (por defecto es 0,92).

Aquí dibujamos un canvas y adjuntamos el URI de datos del canvas al enlace "Descargar a miImagen.jpg".

HTML

```
<canvas id="canvas" width=240 height=240 style="background-color:#808080;"></canvas>
<p></p>
<a id="download" download="miImagen.jpg" href="" onclick="descargar_img(this);">Descargar en
miImagen.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);
descargar_img = function(el) {
     // obtener el URI de la imagen del objeto canvas
     var imageURI = canvas.toDataURL("image/jpg");
     el.href = imageURI;
};
```

Sección 1.5: Cómo añadir el elemento HTML5 Canvas 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 una anchura por defecto de 300px y una altura por defecto de 150px.
- Requiere JavaScript porque todo el contenido debe añadirse mediante programación al canvas.

Ejemplo: Crear un elemento HTML5 Canvas utilizando tanto el lenguaje de 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(){
				// añadir un elemento canvas usando javascript
				var canvas=document.createElement('canvas');
				canvas.id='canvasJavascript'
				document.body.appendChild(canvas);
			}); // end $(function(){});
		</script>
	</head>
	<body>
		<!-- añadir un elemento canvas mediante html -->
		<canvas id='canvasHtml5'></canvas>
	</body>
</html>
```

Sección 1.6: Índice de capacidades y usos de HTML5 Canvas

Capacidades del canvas

Canvas te permite dibujar mediante programación en tu página web:

- Imágenes,
- Textos,
- Líneas y curvas.

Los dibujos en canvas pueden estilizarse mucho:

- ancho del trazo,
- color del trazo,
- color de relleno de la forma,
- opacidad,
- sombreado,
- degradados lineales y radiales,
- tipo de letra,
- tamaño de fuente,

- alineación del texto,
- el texto puede estar trazado, relleno o trazado y relleno a la vez,
- cambio de tamaño de la imagen,
- recorte de imagen,
- composición

Usos del canvas

Los dibujos pueden combinarse y colocarse en cualquier lugar del canvas para que sirva para crear:

- Aplicaciones Paint / Sketch,
- Juegos interactivos de ritmo rápido,
- Análisis de datos como tablas, gráficos,
- imágenes tipo Photoshop.

El canvas permite manipular los colores rojo, verde, azul y el componente alfa de las imágenes. Esto permite a canvas manipular imágenes con resultados similares a Photoshop.

- Recolorear cualquier parte de una imagen a nivel de píxel (si utilizas HSL puedes incluso recolorear una imagen conservando la iluminación y la saturación importantes para que el resultado no parezca que alguien le ha dado una bofetada de pintura a la imagen),
- "Knockout" el fondo alrededor de una persona / elemento en una imagen,
- Detectar y rellenar parte de una imagen (por ejemplo, cambiar el color de un pétalo de flor pulsado por el usuario de verde a amarillo, ¡sólo ese pétalo pulsado! -- sólo ese pétalo),
- Haga Deformación de la perspectiva (por ejemplo, envuelva una imagen alrededor de la curva de una taza),
- Examinar una imagen en busca de contenido (por ejemplo, reconocimiento facial),
- Responde a preguntas sobre una imagen: ¿Hay un coche aparcado en esta imagen de mi plaza de aparcamiento?,
- Aplicar filtros de imagen estándar (escala de grises, sepia, etc.)
- Aplica cualquier filtro de imagen exótico que se te ocurra (Detección de bordes Sobel),
- Combina imágenes. Si la querida abuela María no pudo asistir a la reunión familiar, simplemente "photoshopeala" en la imagen de la reunión. Si no te gusta el primo Felipe, elimínalo con "photoshop",
- Reproducir un vídeo / Tomar un fotograma de un vídeo,
- Exporte el contenido del canvas como una imagen .jpg | .png (incluso puede recortar o anotar la imagen opcionalmente y exportar el resultado como una nueva imagen),

Sobre el movimiento y la edición de dibujos en canvas (por ejemplo, para crear un juego de acción):

- Una vez que se ha dibujado algo en el canvas, ese dibujo existente no se puede mover ni editar. Merece la pena aclarar esta idea errónea de que los dibujos del canvas se pueden mover: *Los dibujos existentes en el canvas no pueden no se pueden mover ni modificar*.
- Canvas 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.
- El canvas crea la ilusión de movimiento dibujando algo rápida y repetidamente y volviéndolo a dibujar en una nueva posición. Al igual que la televisión, este redibujado constante da al ojo la ilusión de movimiento.

Sección 1.7: Canvas fuera de pantalla

Muchas veces cuando se trabaja con el canvas necesitará tener un canvas para mantener algunos datos de píxeles. Es fácil crear un canvas fuera de la pantalla, obtener un contexto 2D. Un canvas fuera de pantalla también utilizará los gráficos disponibles disponible.

El siguiente código simplemente crea un canvas y lo rellena con píxeles azules.

```
function createCanvas(width, height){
      var canvas = document.createElement("canvas"); // crear un elemento canvas
      canvas.width = width;
      canvas.height = height;
      return canvas;
}
var myCanvas = createCanvas(256,256); // crear un pequeño canvas de 256 por 256 píxeles
var ctx = myCanvas.getContext("2d");
ctx.fillStyle = "blue";
ctx.fillRect(0,0,256,256);
```

Muchas veces el canvas fuera de pantalla se utilizará para muchas tareas, y es posible que tengas muchos canvas. Para simplificar el uso del canvas puedes adjuntar el contexto del canvas al canvas.

```
function createCanvasCTX(width, height){
      var canvas = document.createElement("canvas"); // crear un elemento canvas
      canvas.width = width;
      canvas.height = height;
      canvas.ctx = canvas.getContext("2d");
      return canvas;
}
var myCanvas = createCanvasCTX(256,256); // crear un pequeño canvas de 256 por 256 píxeles
myCanvas.ctx.fillStyle = "blue";
myCanvas.ctx.fillRect(0,0,256,256);
```

Sección 1.8: 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("Hola Mundo",150,50);
```

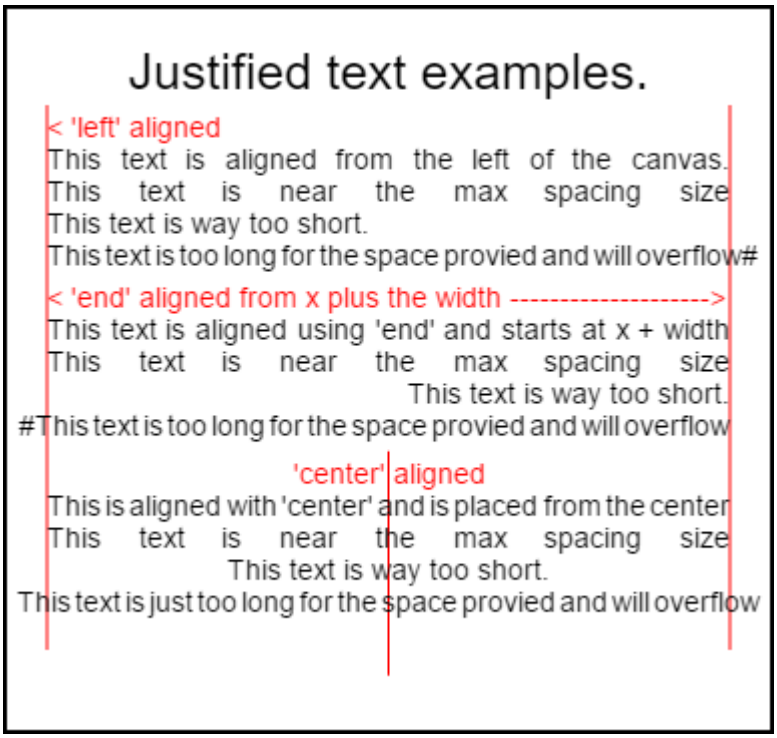
Resultado

Capítulo 2: Texto

Sección 2.1: Texto justificado

Este ejemplo renderiza texto justificado. Añade funcionalidad extra al `CanvasRenderingContext2D` extendiendo su prototipo o como objeto global `justifiedText` (opcional ver Nota A).

Ejemplo de renderizado



El código para representar esta imagen se encuentra en los ejemplos de uso de la parte inferior.

El ejemplo

La función como función anónima de invocación inmediata.

```
(function () {
      const FILL = 0; // const para indicar la representación del texto de relleno
      const STROKE = 1;
      const MEASURE = 2;
      var renderType = FILL; // se utiliza internamente para establecer el texto de relleno o
      trazo
      var maxSpaceSize = 3; // Multiplicador del tamaño máximo del espacio. Si es mayor, no se
      aplica justificación
      var minSpaceSize = 0.5; // Multiplicador para el tamaño mínimo del espacio
      var renderTextJustified = function (ctx, text, x, y, width) {
            var words, wordsWidth, count, spaces, spaceWidth, adjSpace, renderer, i, textAlign,
            useSize, totalWidth;
            textAlign = ctx.textAlign; // obtener la configuración de alineación actual
            ctx.textAlign = 'left';
            wordsWidth = 0;
            words = text.split(' ').map((word) => {
                  var w = ctx.measureText(word).width;
                  wordsWidth += w;
                  return {
                        width: w,
                        word: word,
                  };
            });
```

```
        // count = número de palabras, spaces = número de espacios, spaceWidth tamaño normal
        del espacio
        // adjSpace nuevo tamaño de espacio >= tamaño mín. useSize Tamaño de espacio
        resultante utilizado para renderizar
        count = words.length;
        spaces = count - 1;
        spaceWidth = ctx.measureText(' ').width;
        adjSpace = Math.max(spaceWidth * minSpaceSize, (width - wordsWidth) / space );
        useSize = adjSpace > spaceWidth * maxSpaceSize ? spaceWidth : adjSpace;
        totalWidth = wordsWidth + useSize * spaces;
        if (renderType === MEASURE) {
            // si medir devolver tamaño
            ctx.textAlign = textAlign;
            return totalWidth;
        }
        renderer = renderType === FILL ? ctx.fillText.bind(ctx) : ctx.strokeText.bind(ctx); //
        rellenar o trazo
        switch (textAlign) {
            case 'right':
                x -= totalWidth;
                break;
            case 'end':
                x += width - totalWidth;
                break;
            case 'center': // caída intencionada a través de default
                x -= totalWidth / 2;
            default:
        }
        if (useSize === spaceWidth) {
            // si el tamaño del espacio no cambia
            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;
    };
    // Parsear objeto de configuración vet y set.
    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;
    };
    // definir texto de relleno
    var fillJustifyText = function (text, x, y, width, settings) {
        justifiedTextSettings(settings);
```

```
            renderType = FILL;
            renderTextJustified(this, text, x, y, width);
      };
      // definir texto de trazo
      var strokeJustifyText = function (text, x, y, width, settings) {
            justifiedTextSettings(settings);
            renderType = STROKE;
            renderTextJustified(this, text, x, y, width);
      };
      // definir texto de medida
      var measureJustifiedText = function (text, width, settings) {
            justifiedTextSettings(settings);
            renderType = MEASURE;
            return renderTextJustified(this, text, 0, 0, width);
      };
      // código punto A
      // establecer los prototipos
      CanvasRenderingContext2D.prototype.fillJustifyText = fillJustifyText;
      CanvasRenderingContext2D.prototype.strokeJustifyText = strokeJustifyText;
      CanvasRenderingContext2D.prototype.measureJustifiedText = measureJustifiedText;
      // código punto B
      // código opcional si no desea ampliar el prototipo CanvasRenderingContext2D
      /* Descomentar desde aquí hasta el comentario final
      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);
            }
      }
      hasta aquí */
})();
```

Nota A: Si no desea extender el prototipo `CanvasRenderingContext2D`, elimina del todo el código entre `// punto de código A` y `// punto de código B` y descomente el código marcado `/* Descomentar desde aquí hasta el comentario final`

Cómo utilizarlo

Se añaden tres funciones al `CanvasRenderingContext2D` y están disponibles para todos los objetos de contexto 2D creados.

- `ctx.fillJustifyText( text, x, y, width, [settings]);`
- `ctx.strokeJustifyText( text, x, y, width, [settings]);`
- `ctx.measureJustifiedText( text, width, [settings]);`

La función Rellenar y trazar texto, rellena o traza texto y utiliza los mismos argumentos. `measureJustifiedText` devolverá el ancho real con el que se mostraría el texto. Puede ser igual, menor o mayor que del `width` del argumento, dependiendo de la configuración actual.

Nota: Los argumentos dentro de `[` y `]` son opcionales.

Argumentos de la función

- **`text`:** Cadena de caracteres que contiene el texto a renderizar.
- **`x, y`:** Coordenadas en las que se mostrará el texto.
- **`width`:** Anchura del texto justificado. El texto aumentará/disminuirá los espacios entre palabras para ajustarse al ancho. Si el espacio entre palabras es mayor que `maxSpaceSize` (por defecto = 6) se utilizará el espaciado normal y el texto no ocupará el ancho requerido. Si el espaciado es inferior a `minSpaceSize` (por defecto = 0,5) tiempo normal se utilizará el espacio mínimo y el texto sobrepasará la anchura solicitada.
- **`settings`:** Opcional. Objeto que contiene los tamaños de espacio mínimo y máximo.

El argumento `settings` es opcional y, si no se incluye, el renderizado de texto utilizará la última configuración definida o la predeterminada (que se muestra a continuación).

Tanto `min` como `max` son los tamaños `min` y `max` para el carácter [espacio] que separa las palabras. El valor por defecto `maxSpaceSize = 6` significa que cuando el espacio entre caracteres es > 63 * `ctx.measureText(" ").width` el texto no se justificará. Si el texto a justificar tiene espacios menores que `minSpaceSize = 0.5` (valor por defecto 0.5) * `ctx.measureText(" ").width` el espaciado se ajustará a `minSpaceSize * ctx.measureText(" ").width` y el texto resultante sobrepasará el ancho de justificación.

Se aplican las siguientes reglas, `min` y `max` deben ser números. Si no es así, los valores asociados no se modificados. Si `minSpaceSize` es mayor que `maxSpaceSize` ambos parámetros no son válidos y `min max` no se modificará.

Ejemplo de objeto de configuración con valores por defecto

```
settings = {
     maxSpaceSize : 6; // Multiplicador del tamaño máximo del espacio.
     minSpaceSize : 0.5; // Multiplicador para el tamaño mínimo del espacio
};
```

NOTA: Estas funciones de texto introducen un sutil cambio de comportamiento para la propiedad `textAlign` del contexto 2D. `left`, `right`, `center` y `start` se comportan como se espera, pero `end` no se alineará desde la derecha del argumento `x` de la función, sino desde la derecha de `x + width`.

Nota: los ajustes (tamaño mínimo y máximo del espacio) son globales para todos los objetos de contexto 2D.

Ejemplos de USO

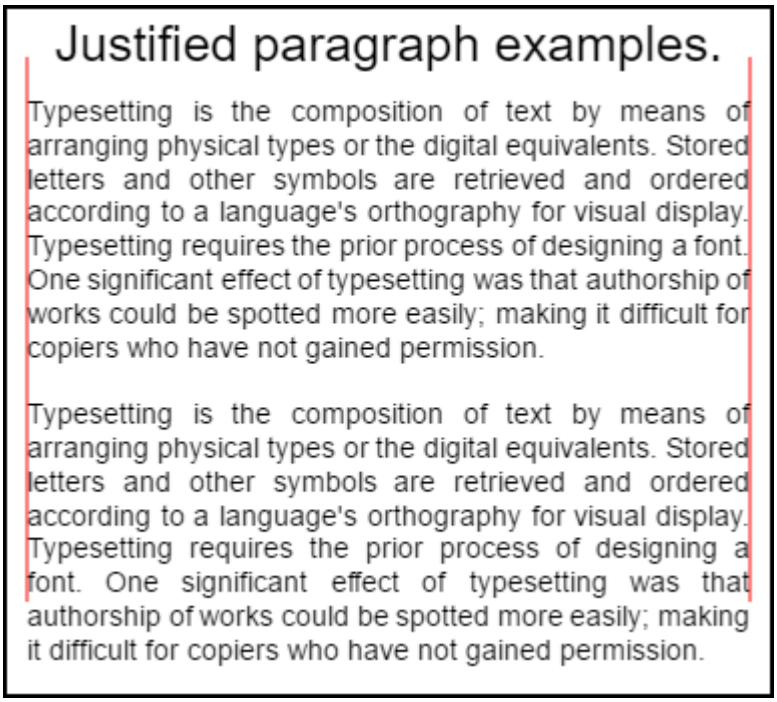
```
var i = 0;
text[i++] = "This text is aligned from the left of the canvas.";
text[i++] = "This text is near the max spacing size";
text[i++] = "This text is way too short.";
text[i++] = "This text is too long for the space provied and will overflow#";
text[i++] = "This text is aligned using 'end' 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 es el contexto 2d
// el canvas es el 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); // se recuerda la configuración
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);
```

Sección 2.2: Párrafos justificados

Renderiza el texto como párrafos justificados. **REQUIERE** el ejemplo **Texto justificado**

Ejemplo de renderizado



El párrafo superior tiene `setting.compact = true` y el inferior `false` y el interlineado es `1.2` en lugar del `1.5` por defecto. Renderizado por ejemplo de uso de código parte inferior de este ejemplo.

Código de ejemplo

```
// Requiere extensiones de texto justificadas
(function () {
      // punto de código A
      if (typeof CanvasRenderingContext2D.prototype.fillJustifyText !== 'function') {
            throw new ReferenceError('Justified Paragraph extension missing required
            CanvasRenderingContext2D justified text extension');
      }
      var maxSpaceSize = 3; // Multiplicador del tamaño máximo del espacio. Si es mayor, no se
      aplica justificación
      var minSpaceSize = 0.5; // Multiplicador para el tamaño mínimo del espacio
      var compact = true; // si es verdadero, intenta introducir tantas palabras como sea
      posible. Si es falso, intenta que el espaciado sea lo más normal posible
      var lineSpacing = 1.5; // space between lines
      const noJustifySetting = {
            // Esta opción desactiva el texto justificado. Se utiliza para representar la última
            línea del párrafo.
            minSpaceSize: 1,
            maxSpaceSize: 1,
      };
      // Parsear objeto de configuración vet y set.
      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) {
        // obtener el tamaño de letra.
        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) => {
            // medir todas las palabras.
            var w = ctx.measureText(word).width;
            return {
                width: w,
                word: word,
            };
        });
        // count = número de palabras, spaces = número de espacios, spaceWidth tamaño normal
        del espacio
        // adjSpace nuevo tamaño de espacio >= tamaño mín. useSize Tamaño de espacio
        resultante utilizado para renderizar
        count = 0;
        lines = [];
        // crea líneas desplazando palabras del array de palabras hasta que el espaciado sea
        óptimo. Si compacto
        // true entonces lo hará y encajará tantas palabras como sea posible. De lo contrario
        intentará que el espaciado lo más
        // lo más cercano posible al espaciado normal
        while (words.length > 0) {
            lastLineWidth = 0;
            lastSize = -1;
            lineFound = false;
            // cada línea debe tener al menos una palabra.
            word = words.shift();
            lineWidth = word.width;
            lineWords = [word.word];
            count = 0;
            while (lineWidth < width && words.length > 0) {
                // Añadir palabras a la línea
                word = words.shift();
                lineWidth += word.width;
                lineWords.push(word.word);
                count += 1;
                spaces = count - 1;
                adjSpace = (width - lineWidth) / spaces;
                if (minS > adjSpace) {
                    // si el espaciado es inferior al mínimo, elimine la última palabra y
                    la línea final
                    lineFound = true;
                    words.unshift(word);
                    lineWords.pop();
                } else {
                    if (!compact) {
```

```
                        // si modo compacto
                        if (adjSpace < spaceWidth) {
                            // si la anchura del espacio es inferior a la normal
                            if (lastSize === -1) {
                                lastSize = adjSpace;
                            }
                            // comprobar si con la última palabra encendida está más
                            cerca de la anchura del espacio
                            if (Math.abs(spaceWidth - adjSpace) < Math.abs(spaceWidth -
                            lastSize)) {
                                lineFound = true; // sí mantenerlo
                            } else {
                                words.unshift(word); // no encaja mejor si se elimina
                                la última palabra
                                lineWords.pop();
                                lineFound = true;
                            }
                        }
                    }
                }
                lastSize = adjSpace; // recuerda el espaciado
            }
            lines.push(lineWords.join(' ')); // y la línea
        }
        // las líneas han sido elaboradas obtenga el tamaño de la fuente, renderice y
        renderice todas las líneas.
        // la última línea puede necesitar ser renderizada como normal para que esté fuera del
        bucle.
        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) {
            // última línea si se alinea a la izquierda o al principio si no se justifica
            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);
            }
        }
        // devuelve detalles sobre el párrafo.
        y += lineSpacing * fontSize;
        return {
            nextLine: y,
            fontSize: fontSize,
            lineHeight: lineSpacing * fontSize,
        };
    }
    // definir relleno
    var fillParagraphText = function (text, x, y, width, settings) {
        justifiedTextSettings(settings);
        settings = {
            minSpaceSize: minSpaceSize,
            maxSpaceSize: maxSpaceSize,
        };
        return justifiedPar(this, text, x, y, width, settings);
    };
    // definir trazo
    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 `CanvasRenderingContext2D`. Si no desea que esto ocurra utilice el ejemplo **Texto justificado** para averiguar cómo cambiar este ejemplo para que forme parte del espacio de nombres global.

NOTA Lanzará un `ReferenceError` si este ejemplo no puede encontrar la función `CanvasRenderingContext2D.prototype.fillJustifyText`

Cómo utilizarlo

```
ctx.fillParaText(text, x, y, width, [settings]);
ctx.strokeParaText(text, x, y, width, [settings]);
```

Véase **Texto justificado** para más detalles sobre los argumentos. Los argumentos entre `[` y `]` son opcionales.

El argumento `settings` tiene dos propiedades adicionales.

- `compact`: Por defecto `true`. Si es `true` intenta empaquetar tantas palabras como sea posible por línea. Si es `false` intenta que el espaciado de palabras lo más cercano posible al espaciado normal.
- `lineSpacing`: Por defecto `1.5`. Espacio por línea por defecto `1.5` la distancia de una línea a la siguiente en términos de tamaño de fuente.

Las propiedades que falten en el objeto de configuración recuperarán sus valores por defecto o los últimos valores válidos. Las propiedades sólo se modificarán si los nuevos valores son válidos. Para los valores válidos `compact` son sólo booleanos `true` o `false`. Los valores verdaderos no se consideran válidos.

Devolver objeto

Las dos funciones devuelven un objeto que contiene información para ayudarle a colocar el siguiente párrafo. El objeto contiene las siguientes propiedades.

- **nextLine** Posición de la línea siguiente después de los píxeles del párrafo
- **fontSize** Tamaño de la fuente. (tenga en cuenta que sólo debe utilizar fuentes definidas en píxeles, por ejemplo, `14px arial`)
- **lineHeight** Distancia en píxeles de una línea a la siguiente

Este ejemplo utiliza un algoritmo sencillo que trabaja línea a línea para encontrar el mejor ajuste para un párrafo. Esto no significa que sea el mejor ajuste (sino el mejor del algoritmo). sobre las líneas generadas. Mover palabras desde el final de una línea al principio de la siguiente, o desde el principio al final. El mejor aspecto se consigue cuando el espaciado de todo el párrafo tiene la menor variación y es lo más parecido al espaciado de texto normal.

Como este ejemplo depende del ejemplo de **texto justificado**, el código es muy similar. Es posible que desee mover los dos en una sola función. Sustituya la función `justifiedTextSettings` del otro ejemplo por la utilizada en este ejemplo. A continuación, copie el resto del código de este ejemplo en el cuerpo de la función anónima del ejemplo de **texto Justificado**. Ya no será necesario comprobar las dependencias encontradas en `// Punto de código 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"
// establecer la configuración de parámetros
var setting = {
     maxSpaceSize : 6,
     minSpaceSize : 0.5,
     lineSpacing : 1.2,
     compact : true,
}
// Mostrar los límites izquierdo y derecho.
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";
// Dibujar párrafo
var line = ctx.fillParaText(para, left, y, width, setting); // se recuerda la configuración
// Siguiente párrafo
y = line.nextLine + line.lineHeight;
setting.compact = false;
ctx.fillParaText(para, left, y, width, setting);
```

Nota: Para el texto alineado `left` o `start`, la última línea del párrafo siempre tendrá un espaciado normal. Para el resto de otras alineaciones la última línea se trata como todas las demás.

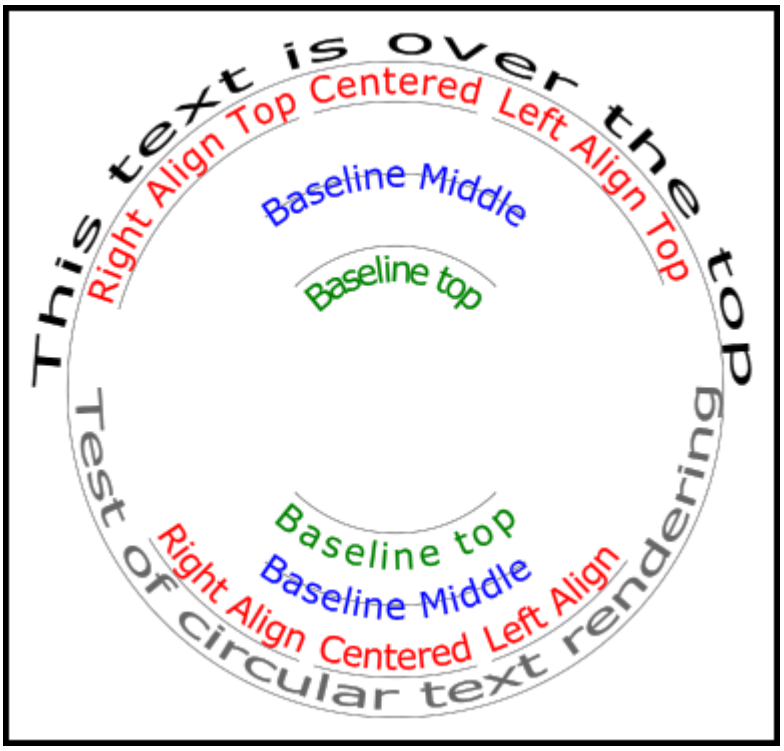
Nota: Puede insertar espacios al principio del párrafo. Aunque esto puede no ser coherente de párrafo a párrafo. Siempre es bueno aprender qué hace una función y modificarla. Un ejercicio sería añadir un ajuste a la configuración que sangrara la primera línea una cantidad fija. Un bucle while tendrá que hacer que la primera palabra aparezca temporalmente más grande (+ sangría) `words[0].width += ?` y luego al renderizar las líneas sangrar la primera línea.

Sección 2.3: Renderizado de texto a lo largo de un arco

Este ejemplo muestra cómo representar texto a lo largo de un arco. Incluye cómo puedes añadir funcionalidad al `CanvasRenderingContext2D` ampliando su prototipo.

Este ejemplo se deriva de la respuesta de StackOverflow Texto Circular.

Ejemplo de renderizado



Código de ejemplo

El ejemplo añade 3 nuevas funciones de renderizado de texto al prototipo de contexto 2D.

- `ctx.fillCircleText(text, x, y, radius, start, end, forward);`
- `ctx.strokeCircleText(text, x, y, radius, start, end, forward);`
- `ctx.measureCircleText(text, radius);`

```
(function () {
     const FILL = 0; // const para indicar la representación del texto de relleno
     const STROKE = 1;
     var renderType = FILL; // se utiliza internamente para establecer el texto de relleno o
     trazo
     const multiplyCurrentTransform = true; // si es true Usar la transformación actual al
     renderizar
     // si es false utiliza coordenadas absolutas que es un poco más rápido
     // después de renderizar el currentTransform se restaura a la transformación por defecto
     // texto del círculo de medida
     // ctx: contexto canvas
     // text: cadena de caracteres a medir
     // r: radio en píxeles
     //
     // devuelve la métrica del tamaño del texto
     //
     // width: Anchura en píxeles del texto
     // angularWidth : anchura angular del texto en radianes
     // pixelAngularSize : anchura angular de un píxel en radianes
     var measure = function (ctx, text, radius) {
          var textWidth = ctx.measureText(text).width; // obtener la anchura de todo el texto
          return {
               width: textWidth,
               angularWidth: (1 / radius) * textWidth,
               pixelAngularSize: 1 / radius,
          };
     };
     // muestra texto a lo largo de un círculo
     // ctx: contexto canvas
     // text: cadena de caracteres a medir
```

```
// x,y: posición del centro del círculo
// r: radio del círculo en píxeles
// start: ángulo en radianes para iniciar.
// [end]: opcional. Si se incluye la alineación del texto se ignora y el texto se
// escala para que quepa entre el inicio y el final;
// [forward]: opcional por defecto true. sí es true la dirección del texto es hacia
adelante, si es false la dirección es hacia atrás
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) {
          // no renderizar cadena de caracteres vacía o transparente
          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; // guardar el textAlign actual para poder restaurarlo al
     final
     dir = forward ? 1 : forward === false ? -1 : 1; // establece dir si no es verdadero o
     falso establece forward como verdadero
     pAS = 1 / radius; // obtener el tamaño angular de un píxel en radianes
     textWidth = ctx.measureText(text).width; // obtener la anchura de todo el texto
     if (end !== undefined && end !== null) {
          // si se indica el final, entonces se ajusta el texto entre el inicio y el final
          pA = ((end - start) / textWidth) * dir;
          wScale = (pA / pAS) * dir;
     } else {
          // si no se indica el final, corregir el inicio y el final para la alineación
          // si no se indica la dirección, cambie la parte superior del texto del círculo
          para leer la dirección correcta
          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': // si está centrado se desplaza a media anchura
                    start -= (pA * textWidth) / 2;
                    end = start + pA * textWidth;
                    break;
               case 'right': // cae intencionadamente hasta el caso "end"
               case 'end':
                    end = start;
                    start -= pA * textWidth;
                    break;
               case 'left': // cae intencionadamente hasta el caso "start"
               case 'start':
                    end = start + pA * textWidth;
          }
     }
     ctx.textAlign = 'center'; // alinear para renderizar
     a = start; // establecer el ángulo inicial
     for (var i = 0; i < text.length; i += 1) {
          // para cada carácter
          aw = ctx.measureText(text[i]).width * pA; // obtener la anchura angular del texto
          var xDx = Math.cos(a + aw / 2); // obtener el vector yAxies del centro x, y, out
          var xDy = Math.sin(a + aw / 2);
          if (multiplyCurrentTransform) {
               // transformar multiplicando la transformación actual
               ctx.save();
```

```
                    if (xDy < 0) {
                        // es el texto al revés. Si es al revés
                        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) {
                        // es el texto al revés. Si lo es, dale la vuelta
                        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); // renderizar el carácter
                } else {
                    ctx.strokeText(text[i], 0, 0); // renderizar el carácter
                }
                if (multiplyCurrentTransform) {
                    // restaurar la transformación actual
                    ctx.restore();
                }
                a += aw; // paso al siguiente ángulo
            }
            // todo listo para limpiar.
            if (!multiplyCurrentTransform) {
                ctx.setTransform(1, 0, 0, 1, 0, 0); // restaurar la transformación
            }
            ctx.textAlign = aligned; // restaurar la alineación del texto
        };
        // definir texto de relleno
        var fillCircleText = function (text, x, y, radius, start, end, forward) {
            renderType = FILL;
            circleText(this, text, x, y, radius, start, end, forward);
        };
        // definir texto de trazo
        var strokeCircleText = function (text, x, y, radius, start, end, forward) {
            renderType = STROKE;
            circleText(this, text, x, y, radius, start, end, forward);
        };
        // definir texto de medida
        var measureCircleTextExt = function (text, radius) {
            return measure(this, text, radius);
        };
        // configurar los prototipos
        CanvasRenderingContext2D.prototype.fillCircleText = fillCircleText;
        CanvasRenderingContext2D.prototype.strokeCircleText = strokeCircleText;
        CanvasRenderingContext2D.prototype.measureCircleText = measureCircleTextExt;
})();
```

Descripción de funciones

Este ejemplo añade 3 funciones al `CanvasRenderingContext2D` **`prototype`**`.fillCircleText`, `strokeCircleText` y `measureCircleText`.

`CanvasRenderingContext2D.fillCircleText(text, x, y, radius, start, [end, [forward]]);`

`CanvasRenderingContext2D.strokeCircleText(text, x, y, radius, start, [end, [forward]]);`

`text`: Texto a representar como String.

`x,y`: Posición del centro del círculo como Number.

`radius`: radio del círculo en píxeles

`start`: ángulo en radianes para empezar.

`[end]`: opcional. Si se incluye `ctx.textAlign` se ignora y el texto se escala para ajustarse entre el inicio y el final.

`[forward]`: opcional por defecto `true`. Si es `true` la dirección del texto es hacia delante, si es `false` la dirección es hacia atrás.

Ambas funciones utilizan `textBaseline` para posicionar el texto verticalmente alrededor del radio. Para obtener los mejores resultados, utiliza `ctx.TextBaseline`.

Las funciones lanzarán un `TypeError` si alguno de los argumentos numéricos es NaN.

Si el argumento de texto se recorta a una cadena de caracteres vacía o ctx.globalAlpha = 0 la función simplemente se cae y no hace nada.

`CanvasRenderingContext2D.measureCircleText(text, radius);`

`- **text:** Cadena de caracteres de texto a medir.`

`- **radius:** radio del círculo en píxeles.`

Devuelve un objeto que contiene varias métricas de tamaño para la representación de texto circular.

`- **width:** Anchura en píxeles del texto tal y como se representaría normalmente.`

`- **angularWidth:** anchura angular del texto en radianes.`

`- **pixelAngularSize:** anchura angular de un píxel en radianes.`

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";
// Texto bajo estirado de Math.PI a 0 (180 - 0 grados)
ctx.fillCircleText(text, centX, centY, rad, Math.PI, 0);
// texto encima centrado a Math.PI * 1.5 ( 270 deg)
ctx.fillCircleText(text, centX, centY, rad, Math.PI * 1.5);
// texto arriba centrado a Math.PI * 1.5 ( 270 deg)
ctx.textBaseline = "top";
ctx.fillCircleText(text, centX, centY, rad, Math.PI * 1.5);
// texto encima centrado a Math.PI * 1.5 ( 270 deg)
ctx.textBaseline = "middle";
ctx.fillCircleText(text, centX, centY, rad, Math.PI * 1.5);
// Utilice measureCircleText para obtener el tamaño angular
var circleTextMetric = ctx.measureCircleText("Text to measure", rad);
console.log(circleTextMetric.width); // anchura del texto si se representa normalmente
console.log(circleTextMetric.angularWidth); // anchura angular del texto
console.log(circleTextMetric.pixelAngularSize); // angular size of a pixel
// Utilice medir texto para dibujar un arco alrededor del texto
ctx.textBaseline = "middle";
var width = ctx.measureCircleText(text, rad).angularWidth;
ctx.fillCircleText(text, centX, centY, rad, Math.PI * 1.5);
// representar el arco alrededor del texto
```

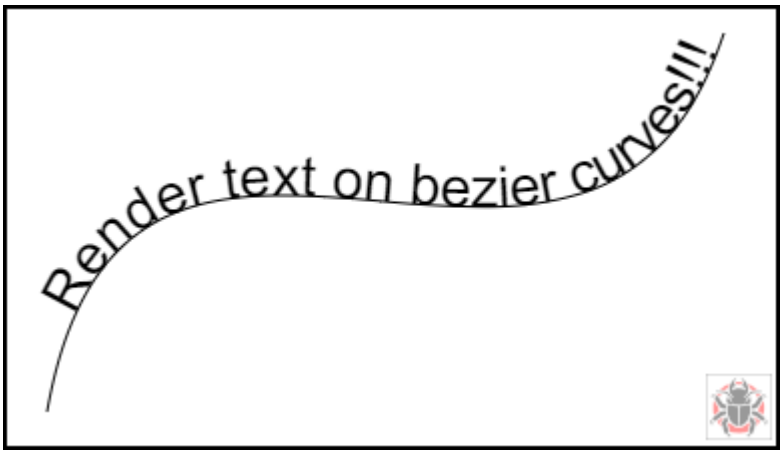
```
ctx.strokeStyle= "red";
ctx.lineWidth = 3;
ctx.beginPath();
ctx.arc(centX, centY, rad + fontSize / 2,Math.PI * 1.5 - width/2,Math.PI*1.5 + width/2);
ctx.arc(centX, centY, rad - fontSize / 2,Math.PI * 1.5 + width/2,Math.PI*1.5 - width/2,true);
ctx.closePath();
ctx.stroke();
```

NOTA: El texto mostrado es sólo una aproximación del texto circular. Por ejemplo, si se renderizan dos "l" las dos líneas no serán paralelas, pero si se renderiza 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 transformar correctamente cada píxel para crear un texto circular.

NOTA: `const multiplyCurrentTransform = true;` definida en este ejemplo se utiliza para establecer el método de transformación utilizado. Si es `false` la transformación para el renderizado de texto circular es absoluta y no depende del estado actual de la transformación. El texto no se verá afectado por ninguna transformación previa de escala, rotar o trasladar. Esto aumentará el rendimiento de la función de renderizado, después de que la función la transformación se establecerá en el valor predeterminado `setTransform(1,0,0,1,0,0)`.

Si `multiplyCurrentTransform = true` (establecido por defecto en este ejemplo) el texto utilizará la transformación actual de modo que el texto puede ser escalado, traducido, sesgado, rotado, etc, pero modificando la transformación actual antes de llamar a las funciones `fillCircleText` y `strokeCircleText`. Dependiendo del estado del contexto 2D esto puede ser algo más lento que `multiplyCurrentTransform = false`.

Sección 2.4: Texto sobre curvas, beziers cúbicos y cuadráticos



`textOnCurve(text,offset,x1,y1,x2,y2,x3,y3,x4,y4)`

Renderiza texto en curvas cuadráticas y cúbicas.

- `text` es el texto a representar
- `offset` distancia del inicio de la curva al texto >= 0
- `x1,y1` - `x3,y3` puntos de la curva cuadrática o
- `x1,y1` - `x4,y4` puntos de la curva cúbica o

Ejemplo de uso

```
textOnCurve("¡Hola Mundo!",50,100,100,200,200,300,100); // dibuja texto en una curva cuadrática
                                                        // 50 píxeles desde el inicio de la curva
textOnCurve("¡Hola Mundo!",50,100,100,200,200,300,100,400,200); // dibuja texto en curva cúbica
                                                                // 50 píxeles desde el inicio de la curva
```

La función y la función de ayuda a la curva

```
// pasar 8 valores para bezier cúbico
// pasar 6 valores para cuadrática
// Renderiza el texto desde el inicio de la curva
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 a la curva está diseñada para aumentar el rendimiento de la búsqueda de puntos en el bezier.

```
// localiza puntos en curvas bezier.
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,vec1,vect;
      vec = {x:0,y:0};
      vec1 = {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
}
```

Sección 2.5: Texto del dibujo

Dibujar en el canvas no se limita a formas e imágenes. También puedes dibujar texto en el canvas.

Para dibujar texto en el canvas, obtenga una referencia al canvas y luego llame al método `fillText` en el contexto.

```
var canvas = document.getElementById('canvas');
var ctx = canvas.getContext('2d');
ctx.fillText("Mi texto", 0, 0);
```

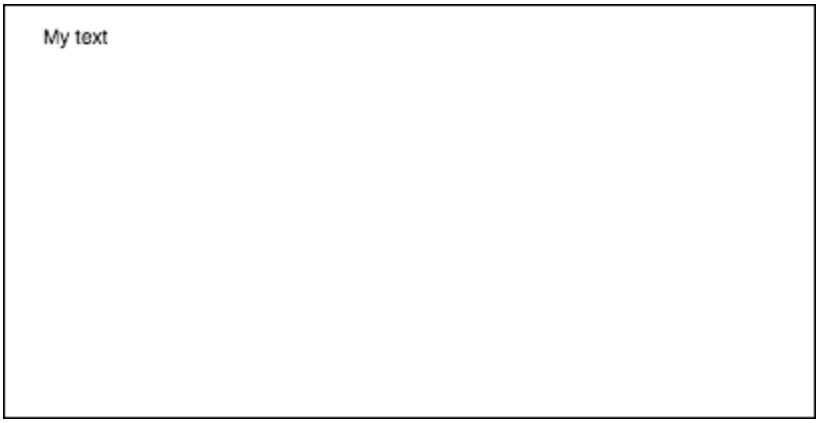
Los tres argumentos necesarios que se pasan a `fillText` son:

1. El texto que desea visualizar
2. La posición horizontal (eje x)
3. La posición vertical (eje y)

Además, hay un cuarto argumento **opcional**, que puede utilizar para especificar la anchura máxima del texto en píxeles. En el ejemplo siguiente, el valor `200` restringe la anchura máxima del texto a 200px:

```
ctx.fillText("Mi texto", 0, 0, 200);
```

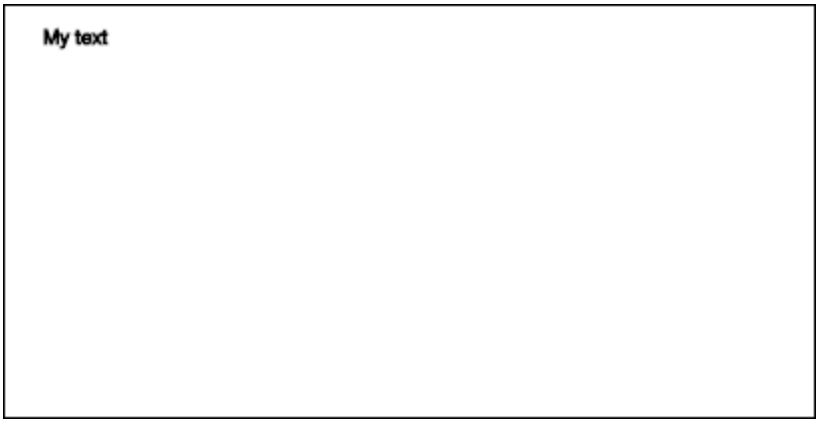
Resultado



También puede dibujar texto sin relleno, y sólo un contorno en su lugar, utilizando el método `strokeText`:

```
ctx.strokeText("Mi texto", 0, 0);
```

Resultado



Sin ninguna propiedad de formato de fuente aplicada, el canvas renderiza el texto a 10px en sans-serif por defecto, haciendo difícil ver la diferencia entre el resultado de los métodos `fillText` y `strokeText`. Consulte el ejemplo Formatear texto para obtener información detallada sobre cómo aumentar el tamaño del texto y aplicar otros cambios estéticos al texto.

Sección 2.6: Dar formato al texto

El formato de fuente por defecto proporcionado por los métodos `fillText` y `strokeText` no es muy atractivo estéticamente. Afortunadamente, la API del canvas proporciona propiedades para dar formato al texto.

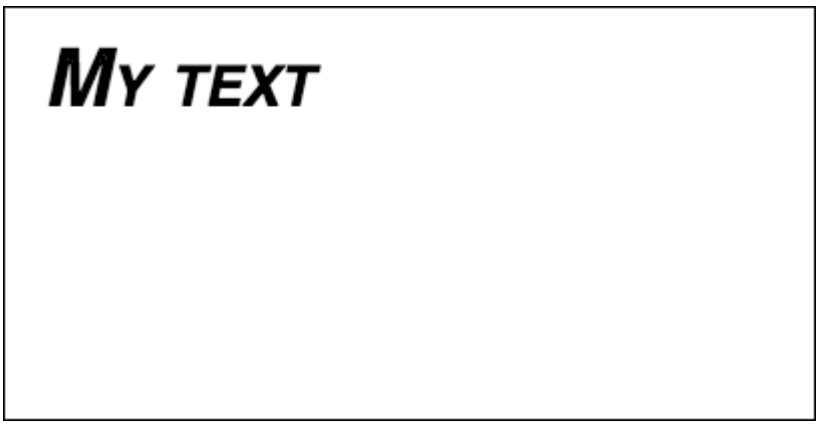
Mediante la propiedad `font` se puede especificar:

- `font-style`
- `font-variant`
- `font-weight`
- `font-size / line-height`
- `font-family`

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 cualquiera de las dos:

- `left`
- `center`
- `right`
- `end` (igual que `right`)
- `start` (igual que `left`)

Por ejemplo

```
ctx.textAlign = "center";
```

Sección 2.7: Envolver texto en párrafos

La API nativa de Canvas no tiene un método para pasar el texto a la línea siguiente cuando se alcanza una anchura máxima deseada. Este ejemplo envuelve el texto en párrafos.

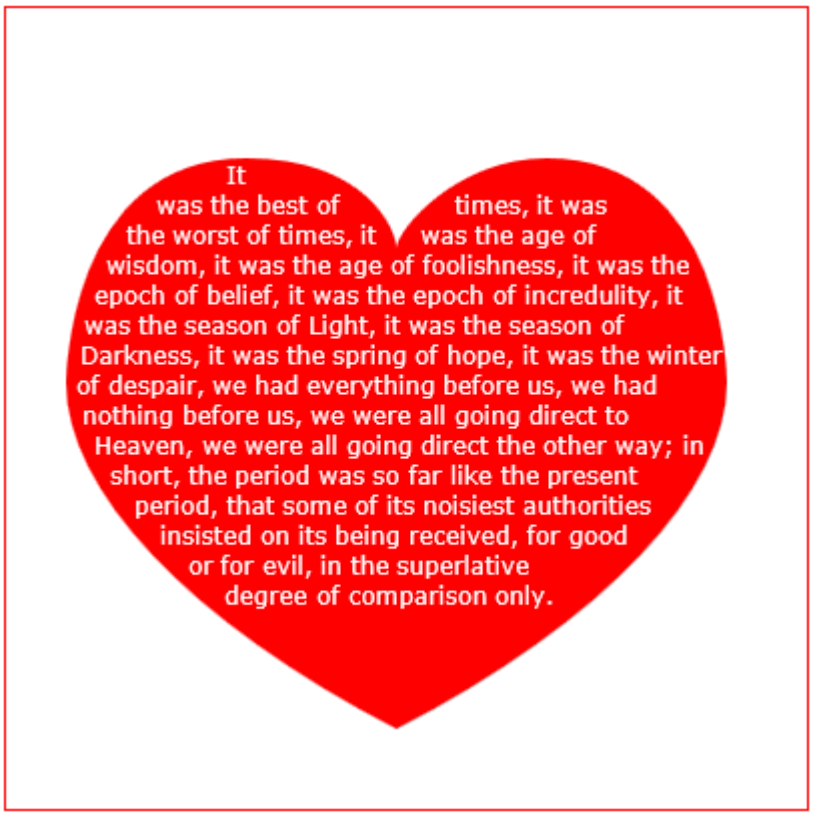
```
function wrapText(text, x, y, maxWidth, fontSize, fontFace){
     var firstY=y;
     var words = text.split(' ');
     var line = '';
     var lineHeight=fontSize*1.286; // una buena aproximación para tamaños de 10-18px
     ctx.font=fontSize+" "+fontFace;
     ctx.textBaseline='top';
     for(var 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);
}
```

Sección 2.8: Dibujar párrafos de texto de forma irregular

Este ejemplo dibuja párrafos de texto en cualquier parte del canvas que tenga píxeles opacos.

Funciona encontrando el siguiente bloque de píxeles opacos que sea lo suficientemente grande como para contener la siguiente palabra especificada y rellenando ese bloque con la palabra especificada.

Los píxeles opacos pueden proceder de cualquier fuente: Comandos de dibujo de rutas y/o imágenes.



```
<!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='Fue la mejor de las épocas, fue la peor de las épocas, fue la
                        edad de la sabiduría, fue la edad de la necedad, fue la época de la
                        creencia, fue la época de la incredulidad, fue la estación de la Luz, fue
                        la estación de la Oscuridad, fue la primavera de la esperanza, fue el
                        invierno de la desesperación, teníamos todo ante nosotros, no teníamos nada
                        ante nosotros, todos íbamos directos al Cielo, todos íbamos directos en la
                        otra dirección; En resumen, el período era tan parecido al actual, que
                        algunas de sus autoridades más ruidosas insistieron en que se recibiera,
                        para bien o para mal, sólo en el grado superlativo de comparación.';
                        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: dibujar Corazón
                        // Nota: la forma puede ser CUALQUIER dibujo opaco, incluso una imagen.
                        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);
                        // rellenar el corazón con texto
                        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();
                        // dibujar palabras secuencialmente en el siguiente bloque de píxeles opacos
                        disponibles
                        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 para que no haya margen
                        izquierdo
                        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;
                }
            }
            // encontrar un bloque rectangular de píxeles opacos
            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);
            }
        }); // fin $(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>
```

Sección 2.9: Rellenar texto con una imagen

Este ejemplo rellena texto con una imagen especificada.

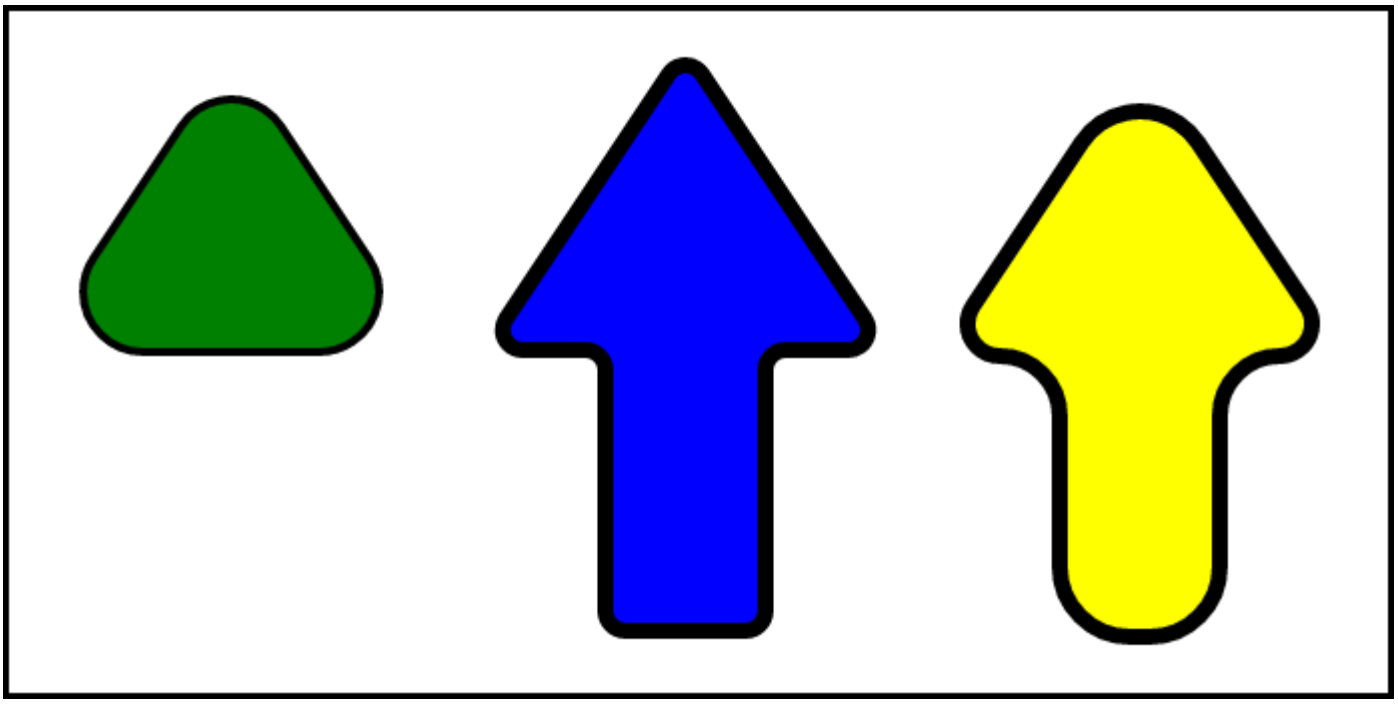
Importante La imagen especificada debe estar completamente cargada antes de llamar a esta función o el dibujo fallará. Utilice `image.onload` para asegurarse de que la imagen está completamente cargada.


```
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);
}
```

Capítulo 3: Polígonos

Sección 3.1: Renderizar un polígono redondeado

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



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(); // iniciar un nuevo camino
roundedPoly(triangle, cornerRadius);
ctx.fill();
ctx.stroke();
```

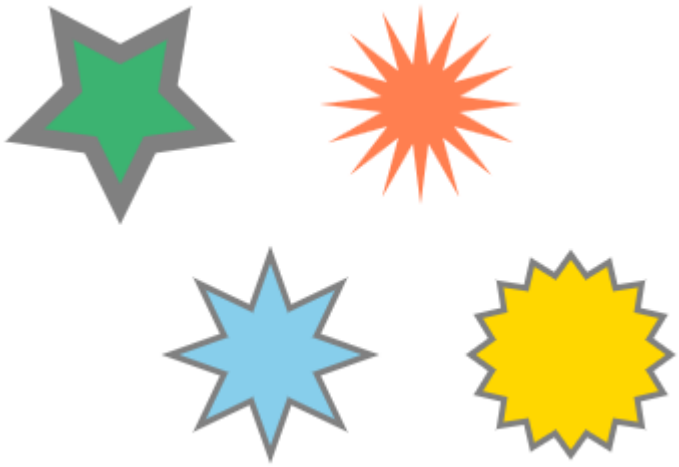
Función de renderizado

```
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) { // convertir puntos en una línea con len y normalizarla
          v.x = pp.x - p.x; // x,y como vec
          v.y = pp.y - p.y;
          v.len = Math.sqrt(v.x * v.x + v.y * v.y); // longitud de vec
          v.nx = v.x / v.len; // normalizado
          v.ny = v.y / v.len;
          v.ang = Math.atan2(v.ny, v.nx); // direccion de vec
     }
     v1 = {};
     v2 = {};
```

```
    len = points.length; // numero de puntos
    p1 = points[len - 1]; // inicio al final de la ruta
    for (i = 0; i < len; i++) { // hacer cada esquina
        p2 = points[(i) % len]; // el punto de la esquina que se está redondeando
        p3 = points[(i + 1) % len];
        // obtener la esquina como vectores fuera de la esquina
        asVec(p2, p1, v1); // vec vuelve del punto de esquina
        asVec(p2, p3, v2); // vec adelante desde el punto de esquina
        // obtener producto cruz de esquinas (asin de angulo)
        sinA = v1.nx * v2.ny - v1.ny * v2.nx; // producto cruz
        // obtener el producto cruz de la primera línea y la segunda línea perpendicular
        sinA90 = v1.nx * v2.nx - v1.ny * -v2.ny; // producto cruzado a la normal de la línea 2
        angle = Math.asin(sinA); // obtener el angulo
        radDirection = 1; // puede que tengas que invertir el radio
        drawDirection = false; // puede que tenga que dibujar el arco en sentido contrario a
        las agujas del reloj
        // encontrar el cuadrante correcto para el centro del círculo
        if (sinA90 < 0) {
            if (angle < 0) {
                angle = Math.PI + angle; // añade 180 para movernos al cuadrante 3
            } else {
                angle = Math.PI - angle; // volver al 2º cuadrante
                radDirection = -1;
                drawDirection = true;
            }
        } else {
            if (angle > 0) {
                radDirection = -1;
                drawDirection = true;
            }
        }
        halfAngle = angle / 2;
        // obtener la distancia de la esquina al punto donde la esquina redonda toca la línea
        lenOut = Math.abs(Math.cos(halfAngle) * radius / Math.sin(halfAngle));
        if (lenOut > Math.min(v1.len / 2, v2.len / 2)) { // fijar si supera la mitad de la
        longitud de la línea
            lenOut = Math.min(v1.len / 2, v2.len / 2);
            // ajustar el radio de redondeo de las esquinas para que encaje
            cRadius = Math.abs(lenOut * Math.sin(halfAngle) / Math.cos(halfAngle));
        } else {
            cRadius = radius;
        }
        x = p2.x + v2.nx * lenOut; // desplazarse desde la esquina a lo largo de la segunda
        línea hasta el punto donde el círculo redondeado toca
        y = p2.y + v2.ny * lenOut;
        x += -v2.ny * cRadius * radDirection; // alejarse de la línea hacia el centro del
        círculo
        y += v2.nx * cRadius * radDirection;
        // x,y es el centro del círculo de la esquina redondeada
        ctx.arc(x, y, cRadius, v1.ang + Math.PI / 2 * radDirection, v2.ang - Math.PI / 2 *
        radDirection, drawDirection); // dibujar el arco en el sentido de las agujas del reloj
        p1 = p2;
        p2 = p3;
    }
    ctx.closePath();
}
```

Sección 3.2: Estrellas

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



```
// Uso:
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: el punto central de la estrella
// points: número de puntos en el exterior de la estrella
// inner: el radio de los puntos interiores de la estrella
// outer: el radio de los puntos exteriores de la Estrella
// fill, stroke: los colores de relleno y trazo a aplicar
// line: la anchura del trazo
function drawStar(centerX, centerY, points, outer, inner, fill, stroke, line) {
      // definir la estrella
      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();
      // dibujar
      ctx.fillStyle=fill;
      ctx.fill();
      ctx.strokeStyle=stroke;
      ctx.lineWidth=line;
      ctx.stroke()
}
```

Sección 3.3: Polígono regular

Un polígono regular tiene todos los lados de igual longitud.

```
// Uso:
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,rotationRa
dians){
      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);
}
```

Capítulo 4: Imágenes

Sección 4.1: ¿Es que "context.drawImage" no muestra la imagen en el canvas?

Asegúrate de que tu objeto imagen está completamente cargado antes de intentar dibujarlo en el canvas con `context.drawImage`.

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

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

```
var img=new Image();
img.onload=start;
img.onerror=function(){
     alert(img.src + ' fallo');
}
img.src="algunaImagen.png";
function start(){
     // start() se ejecuta DESPUÉS de que la imagen se haya cargado completamente.
     // de la posición inicial en el código
}
```

Ejemplo de carga de varias imágenes antes de intentar dibujar con cualquiera de ellas

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

```
// primera imagen
var img1=new Image();
img1.onload=start;
img1.onerror=function(){
     alert(img1.src + ' fallo al cargar.');
};
img1.src="imagenUno.png";
// segunda imagen
var img2=new Image();
img2.onload=start;
img1.onerror=function(){
     alert(img2.src + ' fallo al cargar.');
};
img2.src="imagenDos.png";
//
var imgCount=2;
// start se llama cada vez que se carga una imagen
function start(){
     // cuenta atrás hasta que se cargan todas las imágenes
     if(--imgCount>0){
          return;
     }
     // Todas las imágenes se han cargado correctamente
     // context.drawImage dibujará correctamente cada una de ellas
     context.drawImage(img1,0,0);
     context.drawImage(img2,50,0);
}
```

Sección 4.2: El canvas conservado

Al añadir contenido de fuentes externas a su dominio, o del sistema de archivos local, el canvas se marca como conservado. El intento de acceder a los datos del píxel, o convertir a un `dataURL` lanzará un error de seguridad.

```
var image = new Image();
image.src = "file://miImagenLocal.png";
image.onload = function(){
     ctx.drawImage(this,0,0);
     ctx.getImageData(0,0,canvas.width,canvas.height); // lanza un error de seguridad
}
```

Este ejemplo es sólo un tocho para atraer a alguien con una elaborada comprensión detallada.

Sección 4.3: Recorte de imágenes con canvas

Este ejemplo muestra una sencilla función de recorte de imagen que toma una imagen y unas coordenadas de recorte y devuelve la imagen recortada.

```
function cropImage(image, croppingCoords) {
     var cc = croppingCoords;
     var workCan = document.createElement("canvas"); // crear un canvas
     workCan.width = Math.floor(cc.width); // ajustar la resolución del canvas al tamaño de la
     imagen recortada
     workCan.height = Math.floor(cc.height);
     var ctx = workCan.getContext("2d"); // obtener una interfaz de renderizado 2D
     ctx.drawImage(image, -Math.floor(cc.x), -Math.floor(cc.y)); // dibujar el desplazamiento de
     la imagen para colocarla correctamente en la región recortada
     image.src = workCan.toDataURL(); // establecer la fuente de la imagen en el canvas como una
     URL de datos
     return image;
}
```

Para utilizar

```
var image = new Image();
image.src = "imagen URL"; // cargar la imagen
image.onload = function () { // cuando se carga
     cropImage(
          this, {
          x : this.width / 4, // cosecha manteniendo el centro
          y : this.height / 4,
          width : this.width / 2,
          height : this.height / 2,
     });
     document.body.appendChild(this); // Añade la imagen al DOM
};
```

Sección 4.4: Escalar la imagen para ajustarla o rellenarla

Escala de ajuste

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 canvas. El ejemplo muestra la imagen escalada para ajustarse. El azul de los laterales se debe a que la imagen no tiene el mismo aspecto que el canvas.

Escala para llenar

Significa que la imagen se escala para que todos los píxeles del canvas queden cubiertos por la imagen. Si el aspecto de la imagen no es el mismo que el canvas, algunas partes de la imagen quedarán recortadas. El ejemplo muestra la imagen escalada a relleno. Observa cómo la parte superior e inferior de la imagen ya no son visibles.

Ejemplo Escala de ajuste

```
var image = new Image();
     image.src = "imgURL";
     image.onload = function(){
          scaleToFit(this);
}
function scaleToFit(img){
     // obtener la escala
     var scale = Math.min(canvas.width / img.width, canvas.height / img.height);
     // obtener la posición superior izquierda de la imagen
     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);
}
```

Ejemplo Escala para rellenar

```
var image = new Image();
      image.src = "imgURL";
      image.onload = function(){
            scaleToFill(this);
}
function scaleToFill(img){
      // obtener la escala
      var scale = Math.max(canvas.width / img.width, canvas.height / img.height);
      // obtener la posición superior izquierda de la imagen
      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 la escala. La función de ajuste utiliza la escala de ajuste mínima y la función de relleno utiliza la escala de ajuste máxima.

Capítulo 5: Ruta (sólo sintaxis)

Sección 5.1: createPattern (crea un objeto de estilo de ruta)

```
var pattern = createPattern(imageObject,repeat)
```

Crea un patrón reutilizable (objeto).

El objeto puede asignarse a cualquier `strokeStyle` y/o `fillStyle`.

Entonces `stroke()` o `fill()` pintarán la Ruta con el patrón del objeto.

Argumentos:

- **`imageObject`** es una imagen que se utilizará como patrón. La fuente de la imagen puede ser:
  - `HTMLImageElement` - un elemento `img` o una nueva `Image()`,
  - `HTMLCanvasElement` - un elemento `canvas`,
  - `HTMLVideoElement` - un elemento de vídeo (cogerá el fotograma de vídeo actual),
  - `ImageBitmap`,
  - `Blob`.
- `repeat` determina cómo se repetirá el `imageObject` en el canvas (como un fondo CSS). Este argumento debe estar delimitado por comillas y los valores válidos son:
  - `"repeat"` - el patrón llenará horizontal y verticalmente el canvas,
  - `"repeat-x"` - el patrón sólo se repetirá horizontalmente (1 fila horizontal),
  - `"repeat-y"` - el patrón sólo se repetirá verticalmente (1 fila vertical),
  - `"repeat none"` - el patrón sólo aparece una vez (en la parte superior izquierda).

El objeto patrón es un objeto que puede utilizar (¡y reutilizar!) para hacer que los trazos y rellenos de sus rutas se conviertan en patrones.

Nota al margen: El objeto patrón no es interno al elemento Canvas ni a su Contexto. Se trata de un objeto que puede asignar a cualquier ruta que desee. Incluso puede utilizar este objeto para aplicar el patrón a una ruta en un elemento Canvas diferente.

Sugerencia importante sobre los modelos de canvas

Cuando se crea un objeto patrón, todo el canvas se rellena "invisiblemente" con ese patrón (sujeto al argumento `repeat`).

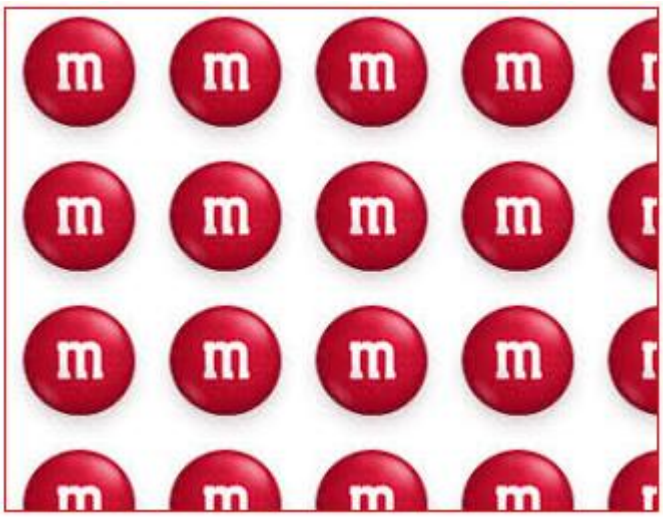
Cuando `stroke()` o `fill()` un trazado, el patrón invisible se revela, pero sólo sobre el trazado que está siendo trazado o rellenado.

1. Empieza con una imagen que quieras utilizar como patrón. Importante: Asegúrese de que su imagen se ha cargado completamente (usando `patternimage.onload`) antes de intentar usarla para crear su patrón.

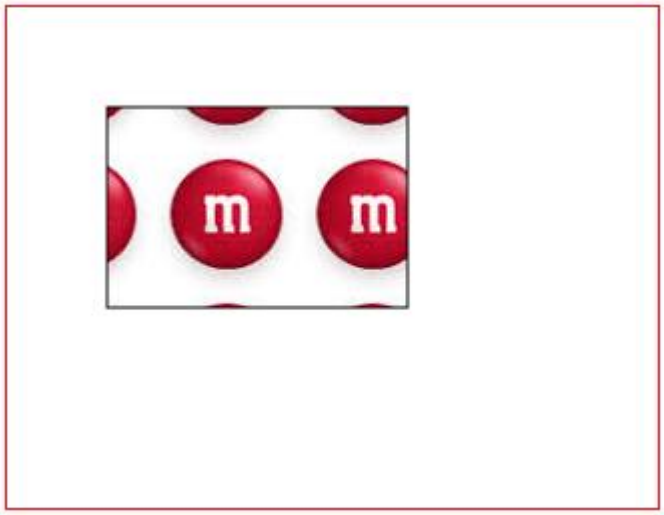
2. Creas un patrón como este:

```
// crear un patrón
var pattern = ctx.createPattern(patternImage,'repeat');
ctx.fillStyle=pattern;
```

3. Entonces Canvas verá "invisiblemente" tu creación de patrones así:



4. Pero hasta que uses `stroke()` o `fill()` con el patrón, no verá nada del patrón en el Canvas.
5. Por último, si traza o rellena un trazado utilizando el patrón, el patrón "invisible" se hace visible en el canvas... pero sólo donde se dibuja el trazado.



```
<!doctype html>
<html>
      <head>
            <style>
                  body{ background-color:white; }
                  #canvas{border:1px solid red; }
            </style>
            <script>
                  window.onload=(function(){
                        // variables relacionadas con canvas
                        var canvas=document.getElementById("canvas");
                        var ctx=canvas.getContext("2d");
                        // rellenar con un patrón
                        var patternImage=new Image();
                        // IMPORTANTE
                        // Utilice siempre .onload para asegurarse de que la imagen se ha
                        // completamente cargada antes de usarla en .createPattern
                        patternImage.onload=function(){
                              // crear un objeto patrón
                              var pattern = ctx.createPattern(patternImage,'repeat');
                              // establecer el fillstyle a ese patrón
                              ctx.fillStyle=pattern;
                              // rellenar un rectángulo con el patrón
                              ctx.fillRect(50,50,150,100);
                              // sólo demo, traza el rectangulo para mayor claridad
                              ctx.strokeRect(50,50,150,100);
                        }
                        patternImage.src='http://i.stack.imgur.com/K9EZl.png';
                  }); // fin window.onload
            </script>
      </head>
      <body>
            <canvas id="canvas" width=325 height=250></canvas>
      </body>
</html>
```

Sección 5.2: stroke (un comando de ruta)

```
context.stroke()
```

Hace que el perímetro de la ruta sea trazado de acuerdo con el `context.strokeStyle` actual y la ruta trazada se dibuja visualmente en el canvas.

Antes de ejecutar `context.stroke` (o `context.fill`) el trazado existe en memoria y aún no está dibujado visualmente en el canvas.

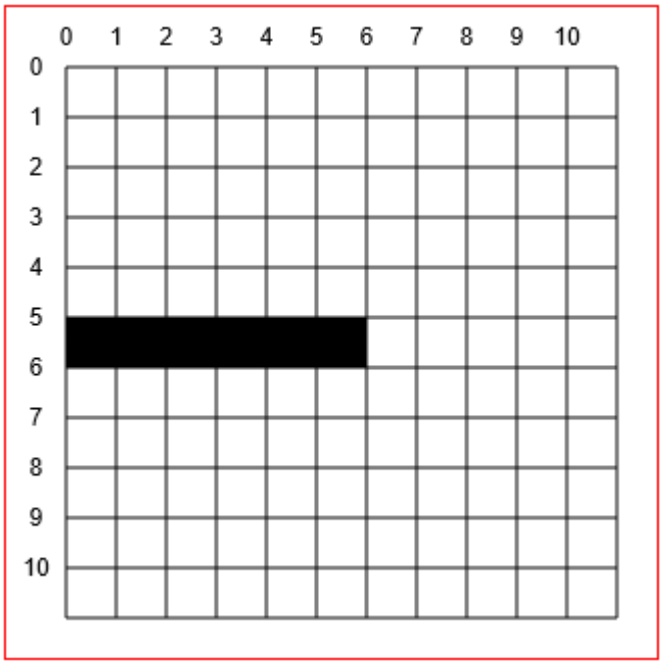
La forma inusual en que se dibujan los trazos

Considere este ejemplo Trayectoria que dibuja una línea negra de 1 píxel desde `[0,5]` hasta `[5,5]`:

```
// dibuja una línea negra de 1 píxel de [0,5] a [5,5]
context.strokeStyle='black';
context.lineWidth=1;
context.beginPath();
context.moveTo(0,5);
context.lineTo(5,5);
context.stroke();
```

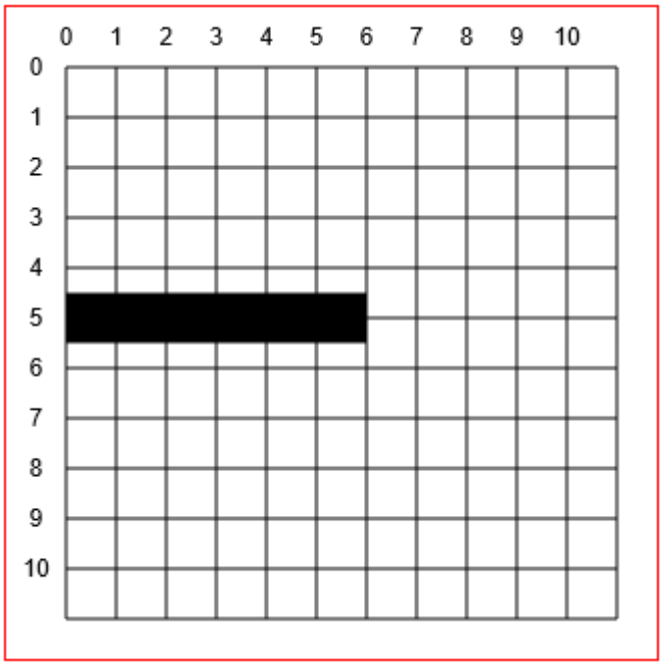
Pregunta: ¿Qué dibuja realmente el navegador en el canvas?

Probablemente esperes obtener 6 píxeles negros en y=5



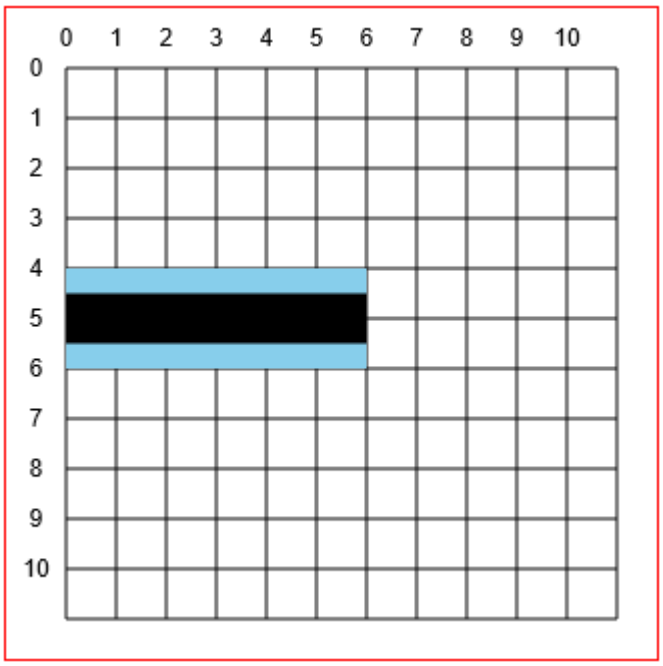
Pero(!) ... ¡El canvas siempre dibuja trazos a medio camino a ambos lados de la ruta definida!

Así que como la línea está definida en `y==5.0` Canvas quiere dibujar la línea entre `y==4.5` y `y==5.5`

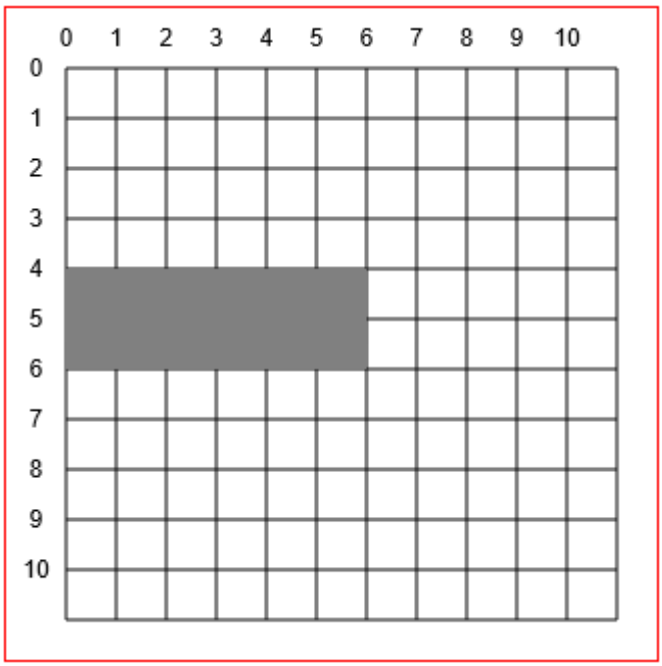


Pero, de nuevo(!) ... ¡La pantalla del ordenador no puede dibujar medios píxeles!

¿Qué hacer con los medios píxeles no deseados (se muestra en azul)?



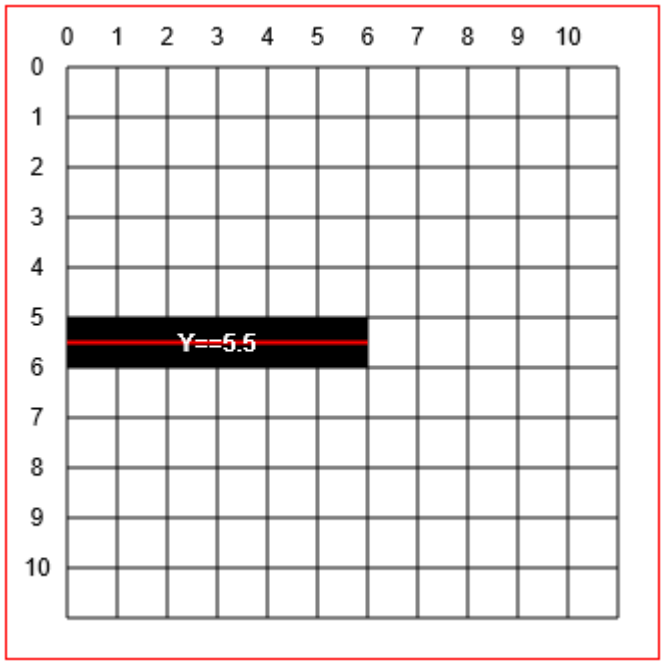
La respuesta es que Canvas en realidad ordena a la pantalla que dibuje una línea de 2 píxeles de ancho de `4.0` a `6.0`. También colorea la línea más clara que el definido `black`. Este extraño comportamiento de dibujo es "anti-aliasing" y ayuda a Canvas a evitar trazos que parezcan dentados.



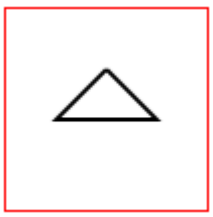
Un truco de ajuste que SÓLO funciona para trazos exactamente horizontales y verticales

Puede obtener una línea negra sólida de 1 píxel especificando que la línea se dibuje en medio píxel:

```
context.moveTo(0,5.5);
context.lineto(5,5.5);
```



Código de ejemplo usando `context.stroke()` para dibujar un trazo en el canvas:



```
<!doctype html>
<html>
      <head>
            <style>
                  body{ background-color:white; }
                  #canvas{border:1px solid red; }
            </style>
            <script>
                  window.onload=(function(){
                        // variables relacionadas con canvas
                        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();
                  }); // fin window.onload
            </script>
      </head>
      <body>
            <canvas id="canvas" width=100 height=100></canvas>
      </body>
</html>
```

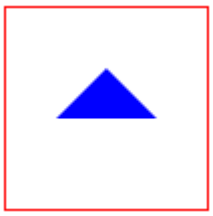
Sección 5.3: fill (un comando de ruta)

```
context.fill()
```

Hace que el interior de la ruta se rellene de acuerdo con el `context.fillStyle` actual y la ruta rellenada se dibuja visualmente en el canvas.

Antes de ejecutar `context.fill` (o `context.stroke`) el trazado existe en memoria y aún no está dibujado visualmente en el canvas.

Código de ejemplo usando `context.fill()` para dibujar una Ruta de relleno en el canvas:



```
<!doctype html>
<html>
      <head>
            <style>
                  body{ background-color:white; }
                  #canvas{border:1px solid red; }
            </style>
            <script>
                  window.onload=(function(){
                        // variables relacionadas con canvas
                        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();
                  }); // fin window.onload
            </script>
      </head>
      <body>
            <canvas id="canvas" width=100 height=100></canvas>
      </body>
</html>
```

Sección 5.4: clip (un comando de ruta)

`context.clip`

Limita los dibujos futuros para que sólo se muestren dentro de la Ruta actual.

Ejemplo: Recortar esta imagen en una ruta triangular

```
<!doctype html>
<html>
      <head>
            <style>
                  body{ background-color:white; }
                  #canvas{border:1px solid red; }
            </style>
            <script>
                  window.onload=(function(){
                        // variables relacionadas con canvas
                        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(){
                              // dibujar un trazado triangular
                              ctx.beginPath();
                              ctx.moveTo(75,50);
                              ctx.lineTo(125,100);
                              ctx.lineTo(25,100);
                              ctx.lineTo(75,50);
                              // recortar dibujos futuros para que aparezcan sólo en el triángulo
                              ctx.clip();
                              // dibujar una imagen
                              ctx.drawImage(img,0,0);
                        }
                  }); // fin window.onload
            </script>
      </head>
      <body>
            <canvas id="canvas" width=150 height=150></canvas>
      </body>
</html>
```

Sección 5.5: Visión general de los comandos básicos de dibujo de trayectorias: líneas y curvas

Ruta

Un trazado define un conjunto de líneas y curvas que pueden dibujarse visiblemente en el canvas.

Una ruta no se dibuja automáticamente en el canvas. Pero las líneas y curvas de la ruta se pueden dibujar en el canvas utilizando un trazo con estilo. Y la forma creada por las líneas y curvas también puede rellenarse con un relleno estilizable.

Las rutas tienen otros usos además de dibujar en el canvas:

- Prueba si una coordenada `x, y` está dentro de la forma de la ruta.
- Definir una región de recorte en la que sólo serán visibles los dibujos que se encuentren dentro de ella. Cualquier dibujo fuera de la región de recorte no se dibujará (`==transparent`) -- similar al `overflow` de CSS.

Los comandos básicos de trazado de rutas son:

- `beginPath`
- `moveTo`
- `lineTo`
- `arc`
- `quadraticCurveTo`
- `bezierCurveTo`
- `arcTo`
- `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 que a menudo se pasa por alto. Si no inicia una nueva ruta, todas las órdenes de ruta emitidas anteriormente se redibujarán automáticamente.

También mueve el "lápiz" de dibujo al origen superior izquierdo del canvas (`==coordinate[0,0]`).

moveTo

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

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

Por defecto, todos los dibujos de rutas están conectados entre sí. Así, el punto final de una línea o curva es el punto de partida de la siguiente línea o curva. Esto puede provocar que se dibuje una línea inesperada que conecte dos dibujos adyacentes. El comando `context.moveTo` básicamente "coge el lápiz de dibujo" y lo coloca en una nueva coordenada de modo que la línea de conexión automática no se dibuje.

lineTo

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

Dibuja un segmento de línea desde la posición actual del lápiz hasta la coordenada [endX,endY].

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

arc

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

Dibuja un arco circular dado un punto central, radio y ángulos inicial y final. Los ángulos se expresan en radianes. Para convertir grados a radianes puedes utilizar esta fórmula: `radians = degrees * Math.PI / 180;`.

El ángulo 0 mira directamente hacia la derecha desde el centro del arco. Para dibujar un círculo completo puedes hacer que anguloFinal = anguloinicial + 360 grados (360 grados == Math.PI2):
`context.arc(10,10,20,0,Math.PI2);`

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

quadraticCurveTo

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

Dibuja una curva cuadrática comenzando en la posición actual del lápiz hasta una coordenada final dada. Otra coordenada de control determina la forma (curvatura) de la curva.

bezierCurveTo

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

Dibuja una curva cúbica de Bézier comenzando en la posición actual del lápiz hasta una coordenada final dada. Otras 2 Las coordenadas de control determinan la forma (curvatura) de la curva.

arcTo

```
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 dos puntos dados: Punto1 y Punto2.

Una línea que conecta la posición actual del lápiz y el inicio del arco se dibuja automáticamente precediendo al arco.

rect

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

Dibuja un rectángulo dada una esquina superior izquierda y una anchura y altura.

El `context.rect` es un comando de dibujo único porque añade rectángulos desconectados. Estos rectángulos no se conectan automáticamente mediante líneas.

closePath

```
context.closePath()
```

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

Por ejemplo, si dibujas 2 líneas formando 2 catetos de un triángulo, `closePath` "cerrará" el triángulo dibujando el tercer cateto del triángulo desde el punto final del segundo cateto hasta el punto inicial del primero.

El nombre de este comando suele dar lugar a malentendidos. `context.closePath` NO es un delimitador final de `context.beginPath`. De nuevo, el comando `closePath` dibuja una línea -- no "cierra" un `beginPath`.

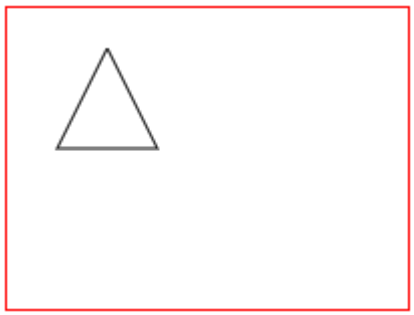
Sección 5.6: lineTo (un comando de ruta)

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

Dibuja un segmento de línea desde la posición actual del lápiz hasta la coordenada `[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 múltiples 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>
```

Sección 5.7: arc (un comando de ruta)

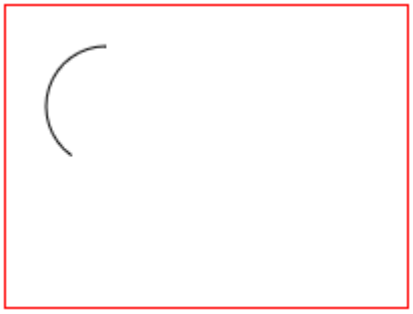
```
context.arc(centerX, centerY, radius, startingRadianAngle, endingRadianAngle)
```

Dibuja un arco circular dado un punto central, radio y ángulos inicial y final. Los ángulos se expresan en radianes.

Para convertir grados a radianes puedes utilizar esta fórmula: `radians = degrees * Math.PI / 180;`.

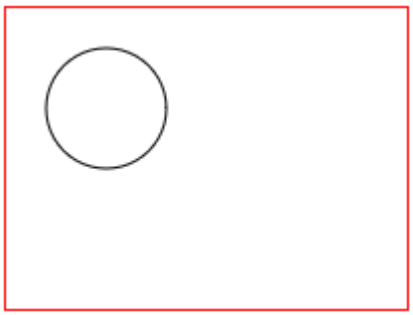
El ángulo 0 mira directamente hacia la derecha desde el centro del arco.

Por defecto, el arco se dibuja en el sentido de las agujas del reloj. Un parámetro opcional `[true|false]` indica que el arco se dibuje en 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(){
                        // obtener una referencia al elemento canvas y su context
                        var canvas=document.getElementById("canvas");
                        var ctx=canvas.getContext("2d");
                        // argumentos
                        var centerX=50;
                        var centerY=50;
                        var radius=30;
                        var startingRadianAngle=Math.PI*2*; // inicio a 90 grados == centroY+radio
                        var endingRadianAngle=Math.PI*2*.75; // final a 270 grados (==PI*2*.75 en
                        radianes)
                        // Un círculo parcial (es decir, un arco) dibujado con el comando "arco".
                        ctx.beginPath();
                        ctx.arc(centerX, centerY, radius, startingRadianAngle, endingRadianAngle);
                        ctx.stroke();
                  }); // fin window.onload
            </script>
      </head>
      <body>
            <canvas id="canvas" width=200 height=150></canvas>
      </body>
</html>
```

Para dibujar un círculo completo puedes hacer que anguloFinal = anguloInicial + 360 grados (360 grados == Math.PI2).

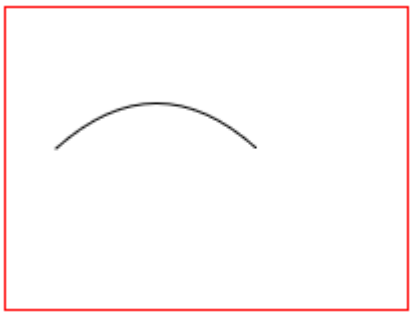


```
<!doctype html>
<html>
      <head>
            <style>
                  body{ background-color:white; }
                  #canvas{border:1px solid red; }
            </style>
            <script>
                  window.onload=(function(){
                        // obtener una referencia al elemento canvas y su contexto
                        var canvas=document.getElementById("canvas");
                        var ctx=canvas.getContext("2d");
                        // argumentos
                        var centerX=50;
                        var centerY=50;
                        var radius=30;
                        var startingRadianAngle=0; // comenzar en 0 grados
                        var endingRadianAngle=Math.PI*2; // final en 360 grados (==PI*2 en radianes)
                        // Un círculo completo dibujado con el comando "arco"
                        ctx.beginPath();
                        ctx.arc(centerX, centerY, radius, startingRadianAngle, endingRadianAngle);
                        ctx.stroke();
                  }); // fin window.onload
            </script>
      </head>
      <body>
            <canvas id="canvas" width=200 height=150></canvas>
      </body>
</html>
```

Sección 5.8: quadraticCurveTo (un comando de ruta)

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

Dibuja una curva cuadrática comenzando en la posición actual del lápiz hasta una coordenada final dada. Otra coordenada de control determina la forma (curvatura) de la curva.

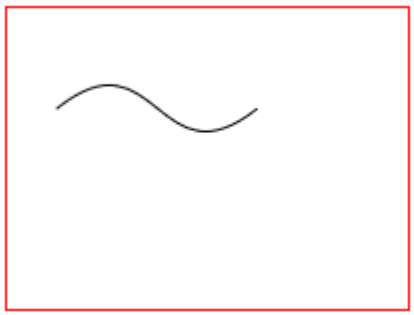


```
<!doctype html>
<html>
      <head>
            <style>
                  body{ background-color:white; }
                  #canvas{border:1px solid red; }
            </style>
            <script>
                  window.onload=(function(){
                        // obtener una referencia al elemento canvas y su contexto
                        var canvas=document.getElementById("canvas");
                        var ctx=canvas.getContext("2d");
                        // argumentos
                        var startX=25;
                        var startY=70;
                        var controlX=75;
                        var controlY=25;
                        var endX=125;
                        var endY=70;
                        // Curva cuadrática dibujada con los comandos "moveTo" y "quadraticCurveTo"
                        ctx.beginPath();
                        ctx.moveTo(startX,startY);
                        ctx.quadraticCurveTo(controlX,controlY,endX,endY);
                        ctx.stroke();
                  }); // fin window.onload
            </script>
      </head>
      <body>
            <canvas id="canvas" width=200 height=150></canvas>
      </body>
</html>
```

Sección 5.9: bezierCurveTo (un comando de ruta)

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

Dibuja una curva cúbica de Bézier comenzando en la posición actual del lápiz hasta una coordenada final dada. Otras 2 Las coordenadas de control determinan la forma (curvatura) de la curva.



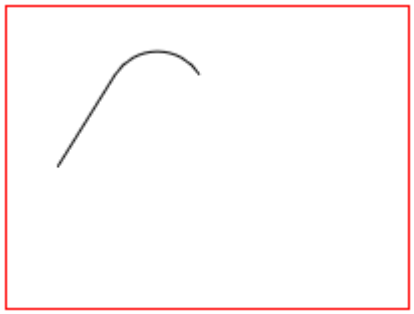
```
<!doctype html>
<html>
      <head>
            <style>
                  body{ background-color:white; }
                  #canvas{border:1px solid red; }
            </style>
            <script>
                  window.onload=(function(){
                        // obtener una referencia al elemento canvas y su contexto
                        var canvas=document.getElementById("canvas");
                        var ctx=canvas.getContext("2d");
                        // argumentos
                        var startX=25;
                        var startY=50;
                        var controlX1=75;
                        var controlY1=10;
                        var controlX2=75;
                        var controlY2=90;
                        var endX=125;
                        var endY=50;
                        // Curva de Bézier cúbica dibujada con los comandos "moveTo" y
                        "bezierCurveTo".
                        ctx.beginPath();
                        ctx.moveTo(startX,startY);
                        ctx.bezierCurveTo(controlX1,controlY1,controlX2,controlY2,endX,endY);
                        ctx.stroke();
                  }); // fin window.onload
            </script>
      </head>
      <body>
            <canvas id="canvas" width=200 height=150></canvas>
      </body>
</html>
```

Sección 5.10: 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 dos puntos dados: Punto1 y Punto2.

Una línea que conecta la posición actual del lápiz y el inicio del arco se dibuja automáticamente precediendo al arco.



```
<!doctype html>
<html>
      <head>
            <style>
                  body{ background-color:white; }
                  #canvas{border:1px solid red; }
            </style>
            <script>
                  window.onload=(function(){
                        // obtener una referencia al elemento canvas y su contexto
                        var canvas=document.getElementById("canvas");
                        var ctx=canvas.getContext("2d");
                        // argumentos
                        var pointX0=25;
                        var pointY0=80;
                        var pointX1=75;
                        var pointY1=0;
                        var pointX2=125;
                        var pointY2=80;
                        var radius=25;
                        // Un arco circular dibujado mediante el comando "arcTo". La línea se dibuja
                        automáticamente.
                        ctx.beginPath();
                        ctx.moveTo(pointX0,pointY0);
                        ctx.arcTo(pointX1, pointY1, pointX2, pointY2, radius);
                        ctx.stroke();
                  }); // fin window.onload
            </script>
      </head>
      <body>
            <canvas id="canvas" width=200 height=150></canvas>
      </body>
</html>
```

Sección 5.11: rect (un comando de ruta)

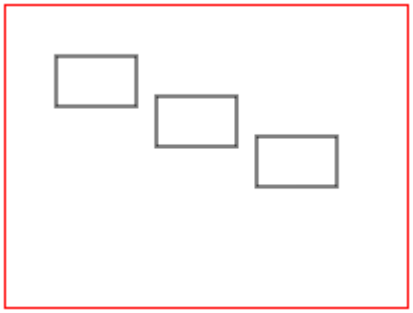
```
context.rect(leftX, topY, width, height)
```

Dibuja un rectángulo dada una esquina superior izquierda y una anchura y altura.

```
<!doctype html>
<html>
      <head>
            <style>
                  body{ background-color:white; }
                  #canvas{border:1px solid red; }
            </style>
            <script>
                  window.onload=(function(){
                        // obtener una referencia al elemento canvas y su contexto
                        var canvas=document.getElementById("canvas");
                        var ctx=canvas.getContext("2d");
                        // argumentos
                        var leftX=25;
                        var topY=25;
                        var width=40;
                        var height=25;
                        // Un rectángulo dibujado con el comando "rect".
                        ctx.beginPath();
                        ctx.rect(leftX, topY, width, height);
                        ctx.stroke();
                  }); // fin 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 añade rectángulos desconectados.

Estos rectángulos desconectados no se conectan automáticamente mediante 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>
```

Sección 5.12: closePath (un comando de ruta)

```
context.closePath()
```

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

Por ejemplo, si dibujas 2 líneas formando 2 catetos de un triángulo, `closePath` "cerrará" el triángulo dibujando el tercer cateto del triángulo desde el punto final del segundo cateto hasta el punto inicial del primero.

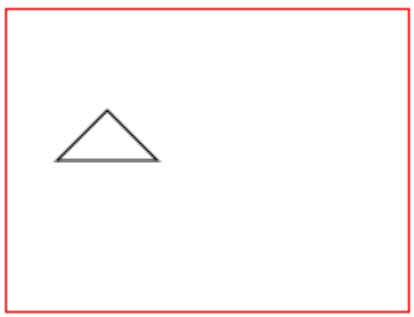
¡Un concepto erróneo explicado!

El nombre de este comando suele dar lugar a malentendidos.

`context.closePath` NO es un delimitador final de `context.beginPath`.

De nuevo, el comando `closePath` dibuja una línea -- no "cierra" un `beginPath`.

Este ejemplo dibuja 2 catetos de un triángulo y utiliza `closePath` para completar (¿cerrar?) el triángulo dibujando el tercer cateto. Lo que `closePath` hace en realidad es trazar 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(){
                        // obtener una referencia al elemento canvas y su contexto
                        var canvas=document.getElementById("canvas");
                        var ctx=canvas.getContext("2d");
                        // argumentos
                        var topVertexX=50;
                        var topVertexY=50;
                        var rightVertexX=75;
                        var rightVertexY=75;
                        var leftVertexX=25;
                        var leftVertexY=75;
                        // Conjunto de segmentos de línea trazados para formar un triángulo
                        utilizando
                        // comandos "moveTo" y "lineTo" multiples
                        ctx.beginPath();
                        ctx.moveTo(topVertexX,topVertexY);
                        ctx.lineTo(rightVertexX,rightVertexY);
                        ctx.lineTo(leftVertexX,leftVertexY);
                        // closePath dibuja el 3er cateto del triángulo
                        ctx.closePath()
                        ctx.stroke();
                  }); // fin window.onload
            </script>
      </head>
      <body>
            <canvas id="canvas" width=200 height=150></canvas>
      </body>
</html>
```

Sección 5.13: beginPath (un comando de ruta)

```
context.beginPath()
```

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

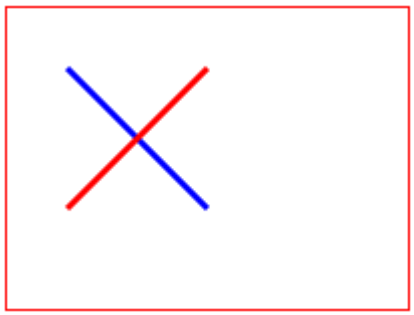
También mueve el "lápiz" de dibujo al origen superior izquierdo del canvas (`==coordinate[0,0]`).

Aunque es opcional, SIEMPRE debe iniciar una ruta con `beginPath`

El descarte es un punto importante que a menudo se pasa por alto. Si no inicia una nueva ruta con `beginPath`, cualquier comando de ruta emitido previamente se redibujará automáticamente.

Estas 2 demos intentan dibujar una "X" con un trazo rojo y otro 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 otro azul.



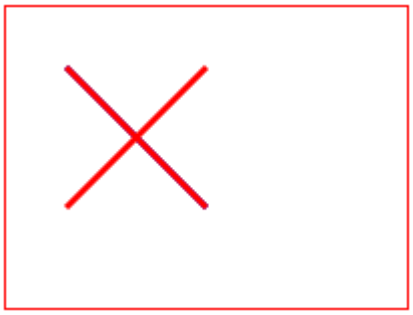
```
<!doctype html>
<html>
	<head>
		<style>
			body{ background-color:white; }
			#canvas{border:1px solid red; }
		</style>
		<script>
			window.onload=(function(){
				// obtener una referencia al elemento canvas y su context
				var canvas=document.getElementById("canvas");
				var ctx=canvas.getContext("2d");
				// dibujar una línea azul
				ctx.beginPath();
				ctx.moveTo(30,30);
				ctx.lineTo(100,100);
				ctx.strokeStyle='blue';
				ctx.lineWidth=3;
				ctx.stroke();
				// dibujar una línea roja
				ctx.beginPath(); // ¡Importante comenzar una nueva ruta!
				ctx.moveTo(100,30);
				ctx.lineTo(30,100);
				ctx.strokeStyle='red';
				ctx.lineWidth=3;
				ctx.stroke();
			}); // fin window.onload
		</script>
	</head>
	<body>
		<canvas id="canvas" width=200 height=150></canvas>
	</body>
</html>
```

Esta segunda demostración omite incorrectamente `beginPath` en el segundo trazo. El resultado es que la "X" tiene incorrectamente los dos trazos rojos.

El segundo `stroke()` dibuja el segundo trazo rojo.

Pero sin un segundo `beginPath`, ese mismo segundo `stroke()` también **redibuja** incorrectamente el primer trazo.

Como el segundo `stroke()` es ahora rojo, el primer trazo azul es **sobrescrito** por un trazo rojo incorrectamente coloreado.



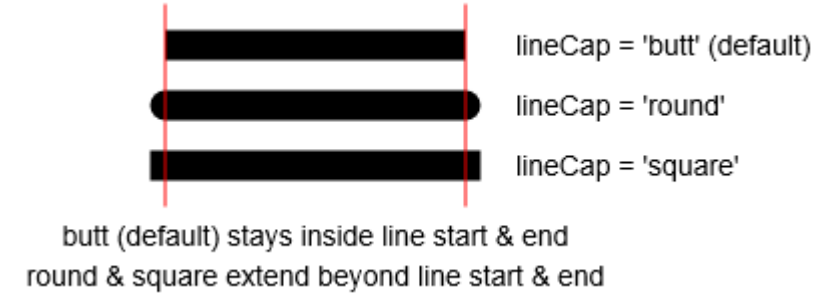
```
<!doctype html>
<html>
      <head>
            <style>
                  body{ background-color:white; }
                  #canvas{border:1px solid red; }
            </style>
            <script>
                  window.onload=(function(){
                        // obtener una referencia al elemento canvas y su context
                        var canvas=document.getElementById("canvas");
                        var ctx=canvas.getContext("2d");
                        // dibujar una línea azul
                        ctx.beginPath();
                        ctx.moveTo(30,30);
                        ctx.lineTo(100,100);
                        ctx.strokeStyle='blue';
                        ctx.lineWidth=3;
                        ctx.stroke();
                        // dibujar una línea roja
                        // Nota: ¡Falta el necesario 'beginPath'!
                        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>
```

Sección 5.14: lineCap (una ruta de atributo de estilo)

```
context.lineCap=capStyle // butt (por defecto), round, square
```

Establece el estilo de tapa de los puntos de inicio y final de línea.

- **`butt`**, el estilo `lineCap` por defecto, muestra tapas cuadradas que no se extienden más allá de los puntos inicial y final de la línea.
- **`round`**, muestra las tapas redondeadas que se extienden más allá de los puntos inicial y final de la línea.
- **`square`**, muestra las tapas cuadradas que se extienden más allá de los puntos inicial y final de la línea.



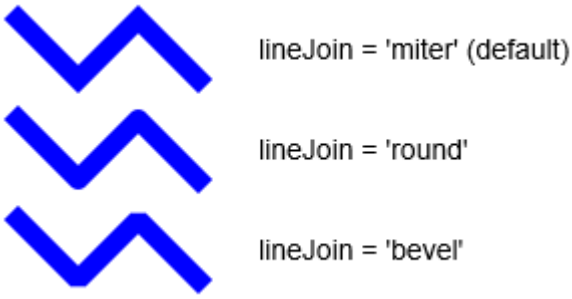

```
<!doctype html>
<html>
     <head>
          <style>
               body{ background-color:white; }
               #canvas{border:1px solid red; }
          </style>
          <script>
               window.onload=(function(){
                    // variables relacionadas con canvas
                    var canvas=document.getElementById("canvas");
                    var ctx=canvas.getContext("2d");
                    ctx.lineWidth=15;
                    // lineCap por defecto: 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);
                    // función de utilidad para trazar una línea
                    function drawLine(startX,startY,endX,endY){
                         ctx.beginPath();
                         ctx.moveTo(startX,startY);
                         ctx.lineTo(endX,endY);
                         ctx.stroke();
                    }
                    // Sólo para demostración,
                    // Reglas para mostrar qué lineCaps se extienden más allá de los puntos
                    finales
                    ctx.lineWidth=1;
                    ctx.strokeStyle='red';
                    drawLine(50,20,50,120);
                    drawLine(200,20,200,120);
               }); // fin window.onload
          </script>
     </head>
     <body>
          <canvas id="canvas" width=300 height=200></canvas>
     </body>
</html>
```

Sección 5.15: lineJoin (una ruta de atributo de estilo)

```
context.lineJoin=joinStyle // miter (por defecto), round, bevel
```

Establece el estilo utilizado para conectar segmentos de línea contiguos.

- **`miter`**, por defecto, une segmentos de línea con una junta afilada.
- **`round`**, une los segmentos de línea con una junta redondeada.
- **`bevel`**, une segmentos de línea con una junta roma.




```
<!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>
```

Sección 5.16: strokeStyle (una ruta de atributo de estilo)

```
context.strokeStyle=color
```

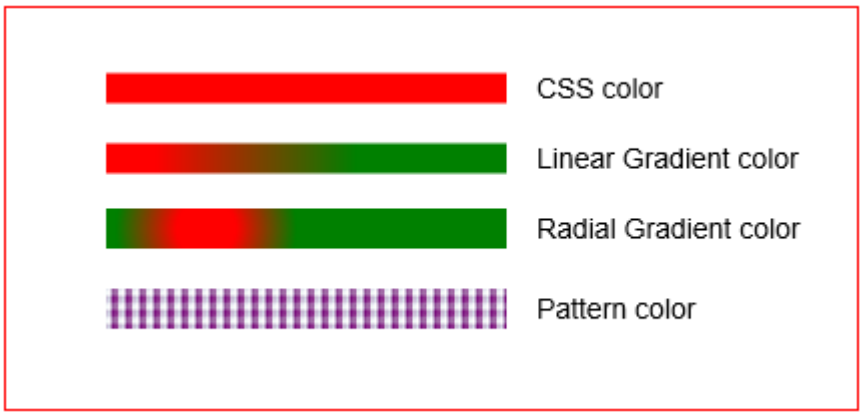
Establece el color que se utilizará para trazar el contorno del trazado actual.

Estas son las opciones de `color` (se deben citar):

- **Un color con nombre CSS**, 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 `red`, `green` y `blue` son enteros 0-255 que indican la intensidad de cada color componente.
- **Un color HSL**, por ejemplo `context.strokeStyle='hsl(hue,saturation,lightness)'` donde `hue` es un entero 0-360 en la rueda de color y `saturation` y `lightness` son porcentajes (0-100%) que indican la intensidad de cada componente.
- **Un color HSLA**, por ejemplo `context.strokeStyle='hsl(hue,saturation,lightness,alpha)'` donde `hue` es un entero 0-360 en la rueda de color y `saturation` y `lightness` son porcentajes (0-100%) indicando la fuerza de cada componente y `alpha` es un valor decimal 0.00-1.00 indicando la opacidad.

También puede especificar estas opciones de color (estas opciones son objetos creados por el contexto):

- **Un gradiente lineal** que es un objeto gradiente lineal creado con `context.createLinearGradient`.
- **Un degradado radial** que es un objeto degradado radial creado con `context.createRadialGradient`.
- **Un patrón** que es un objeto patrón creado con `context.createPattern`.



```
<!doctype html>
<html>
      <head>
            <style>
                  body{ background-color:white; }
                  #canvas{border:1px solid red; }
            </style>
            <script>
                  window.onload=(function(){
                        // variables relacionadas con canvas
                        var canvas=document.getElementById("canvas");
                        var ctx=canvas.getContext("2d");
                        ctx.lineWidth=15;
                        // trazo utilizando un color CSS: con nombre, RGB, HSL, etc.
                        ctx.strokeStyle='red';
                        drawLine(50,40,250,40);
                        // trazo con un gradiente lineal
                        var gradient = ctx.createLinearGradient(75,75,175,75);
                        gradient.addColorStop(0,'red');
                        gradient.addColorStop(1,'green');
                        ctx.strokeStyle=gradient;
                        drawLine(50,75,250,75);
                        // trazo con gradiente radial
                        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);
                        // trazo utilizando un patrón
                        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/stackoverfl
                        ow/BooMu1.png';
                        // sólo para demostración, dibujar etiquetas por cada trazo
                        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);
                        // utilidad para dibujar una línea
                        function drawLine(startX,startY,endX,endY){
                              ctx.beginPath();
                              ctx.moveTo(startX,startY);
                              ctx.lineTo(endX,endY);
                              ctx.stroke();
                        }
                  }); // fin window.onload
            </script>
      </head>
      <body>
            <canvas id="canvas" width=425 height=200></canvas>
      </body>
</html>
```

Sección 5.17: fillStyle (una ruta de atributo de estilo)

```
context.fillStyle=color
```

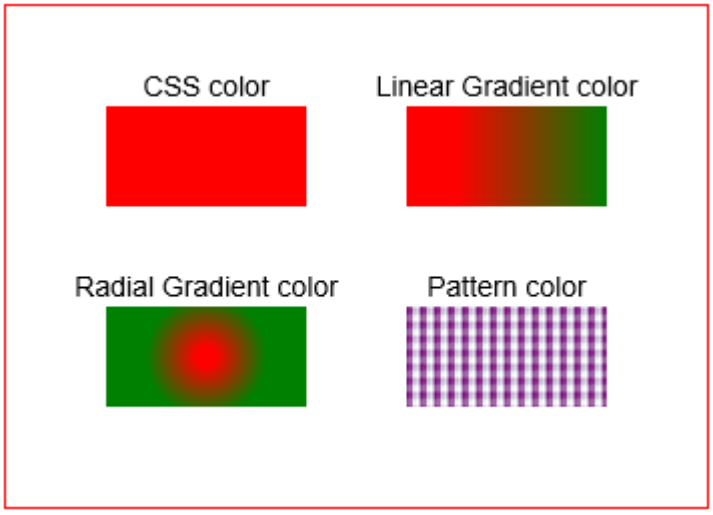
Establece el color que se utilizará para rellenar el interior del trazado actual.

Estas son las opciones de `color` (se deben cotizar):

- **Un color con nombre CSS**, 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 `red`, `green` y `blue` son enteros 0-255 que indican la intensidad de cada color componente.
- **Un color HSL**, por ejemplo `context.strokeStyle='hsl(hue,saturation,lightness)'` donde `hue` es un entero 0-360 en la rueda de colores y `saturation` y `lightness` son porcentajes (0-100%) que indican la intensidad de cada componente.
- **Un color HSLA**, por ejemplo `context.strokeStyle='hsl(hue,saturation,lightness,alpha)'` donde `hue` es un entero 0-360 en la rueda de color y `saturation` y `lightness` son porcentajes (0-100%) que indican la intensidad de cada componente y `alpha` es un valor decimal 0.00-1.00 que indica la opacidad.

También puede especificar estas opciones de color (estas opciones son objetos creados por el contexto):

- **Un gradiente lineal** que es un objeto gradiente lineal creado con `context.createLinearGradient`.
- **Un degradado radial** que es un objeto degradado radial creado con `context.createRadialGradient`.
- **Un patrón** que es un objeto patrón creado con `context.createPattern`.

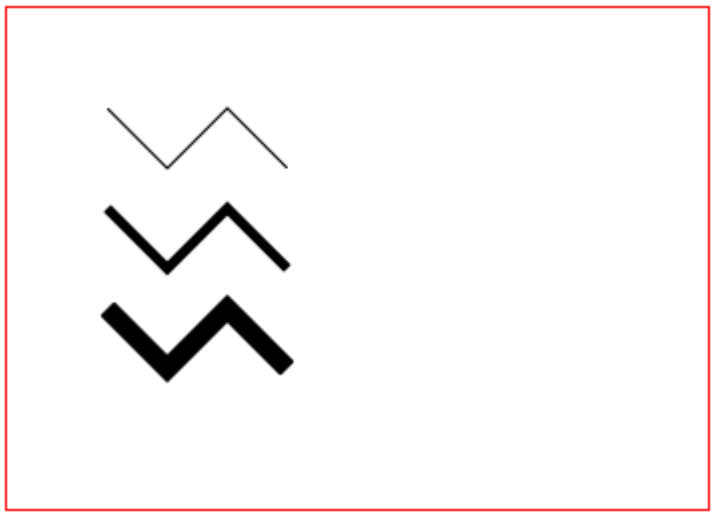


```
<!doctype html>
<html>
      <head>
            <style>
                  body{ background-color:white; }
                  #canvas{border:1px solid red; }
            </style>
            <script>
                  window.onload=(function(){
                        // variables relacionadas con canvas
                        var canvas=document.getElementById("canvas");
                        var ctx=canvas.getContext("2d");
                        // trazo utilizando un color CSS: con nombre, RGB, HSL, etc.
                        ctx.fillStyle='red';
                        ctx.fillRect(50,50,100,50);
                        // trazo con gradiente lineal
                        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);
                        // trazo con gradiente radial
                        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);
                        // trazo utilizando un patrón
                        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';
                        // sólo para demostración, dibujar etiquetas por cada trazo
                        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);
                  }); // fin window.onload
            </script>
      </head>
      <body>
            <canvas id="canvas" width=350 height=250></canvas>
      </body>
</html>
```

Sección 5.18: lineWidth (una ruta de atributo de estilo)

```
context.lineWidth=lineWidth
```

Establece la anchura 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(){
				// variables relacionadas con canvas
				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);
				// utilidad para dibujar una polilínea
				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();
				}
			}); // fin window.onload
		</script>
	</head>
	<body>
		<canvas id="canvas" width=350 height=250></canvas>
	</body>
</html>
```

Sección 5.19: shadowColor, shadowBlur, shadowOffsetX, shadowOffsetY (una ruta de atributo de estilo)

```
shadowColor = color // Color CSS
shadowBlur = width // número entero anchura del desenfoque
shadowOffsetX = distance // la sombra se desplaza horizontalmente por este desplazamiento
shadowOffsetY = distance // la sombra se desplaza verticalmente por este desplazamiento
```

Este conjunto de atributos añadirá una sombra alrededor de un trazado.

Tanto las trayectorias rellenas como las trazadas pueden tener una sombra.

La sombra es más oscura (opaca) en el perímetro de la trayectoria y se aclara gradualmente a medida que se aleja del perímetro de la trayectoria.

- **`shadowColor`** indica qué color CSS se utilizará para crear la sombra.
- **`shadowBlur`** es la distancia sobre la que la sombra se extiende hacia fuera de la trayectoria.
- **`shadowOffsetX`** es una distancia en la que la sombra se desplaza horizontalmente alejándose de la trayectoria. Una distancia positiva desplaza la sombra hacia la derecha, una distancia negativa la desplaza hacia la izquierda.
- **`shadowOffsetY`** es una distancia en la que la sombra se desplaza verticalmente fuera de la trayectoria. Una distancia desplaza la sombra hacia abajo, una distancia negativa la desplaza hacia arriba.

Acerca de `shadowOffsetX` y `shadowOffsetY`

Es importante señalar que *toda la sombra se desplaza en su totalidad*. Esto hará que parte de la sombra se desplace por debajo de los trazados rellenados y, por tanto, parte de la sombra no será visible.

Acerca de los trazos sombreados

Al sombrear un trazo, se sombrean tanto el interior como el exterior del trazo. La sombra es más oscura en el trazo y se aclara a medida que la sombra se extiende hacia fuera en ambas direcciones desde el trazo.

Desactivar el sombreado al terminar

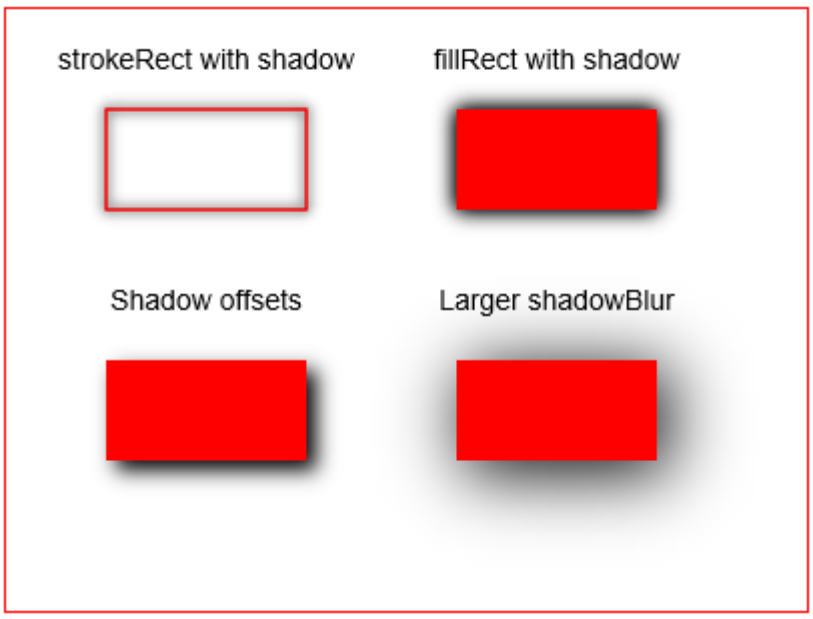
Después de dibujar las sombras, puede que quieras desactivar el sombreado para dibujar más trazados. Para desactivar la sombra se establece el `shadowColor` en transparente.

```
context.shadowColor = 'rgba(0,0,0,0)';
```

Consideraciones sobre el rendimiento

Las sombras (al igual que los degradados) requieren muchos cálculos, por lo que debe utilizarlas con moderación.

Tenga especial cuidado al animar porque dibujar sombras muchas veces por segundo afectará enormemente al rendimiento. Una solución si necesita animar trazados sombreados es crear previamente el trazado sombreado en un segundo "canvas-sombra". segundo "canvas de sombra". El shadow-canvas es un canvas normal que se crea en memoria con `document.createElement` -- no se añade al DOM (es sólo un canvas de montaje). A continuación, dibuje el canvas de sombra en el canvas principal. Esto es mucho más rápido porque los cálculos de sombra no necesitan hacerse muchas veces por segundo. Todo lo que estás haciendo es copiar un canvas pre-construido en tu canvas visible.



```
<!doctype html>
<html>
      <head>
            <style>
                  body{ background-color:white; }
                  #canvas{border:1px solid red; }
            </style>
            <script>
                  window.onload=(function(){
                        // variables relacionadas con canvas
                        var canvas=document.getElementById("canvas");
                        var ctx=canvas.getContext("2d");
                        // trazo sombreado
                        ctx.shadowColor='black';
                        ctx.shadowBlur=6;
                        ctx.strokeStyle='red';
                        ctx.strokeRect(50,50,100,50);
                        // oscurece la sombra acariciando una segunda vez
                        ctx.strokeRect(50,50,100,50);
                        // relleno sombreado
                        ctx.shadowColor='black';
                        ctx.shadowBlur=10;
                        ctx.fillStyle='red';
                        ctx.fillRect(225,50,100,50);
                        // oscurece la sombra acariciando una segunda vez
                        ctx.fillRect(225,50,100,50);
                        // el desplazamiento de la sombra hacia la derecha y hacia abajo
                        ctx.shadowColor='black';
                        ctx.shadowBlur=10;
                        ctx.shadowOffsetX=5;
                        ctx.shadowOffsetY=5;
                        ctx.fillStyle='red';
                        ctx.fillRect(50,175,100,50);
                        // un desenfoque más amplio (==se extiende más allá del camino)
                        ctx.shadowColor='black';
                        ctx.shadowBlur=35;
                        ctx.fillStyle='red';
                        ctx.fillRect(225,175,100,50);
                        // ¡limpia siempre! Desactivar el sombreado
                        ctx.shadowColor='rgba(0,0,0,0)';
                  }); // fin window.onload
            </script>
      </head>
      <body>
            <canvas id="canvas" width=400 height=300></canvas>
      </body>
</html>
```

Sección 5.20: createLinearGradient (crear un objeto de estilo de ruta)

```
var gradient = createLinearGradient( startX, startY, endX, endY )
gradient.addColorStop(gradientPercentPosition, CssColor)
gradient.addColorStop(gradientPercentPosition, CssColor)
[opcionalmente añadir más paradas de color para añadir variedad al degradado]
```

Crea un gradiente lineal reutilizable (objeto).

El objeto puede asignarse a cualquier `strokeStyle` y/o `fillStyle`.

Entonces `stroke()` o `fill()` colorearán la Ruta con los colores del gradiente del objeto.

La creación de un objeto degradado es un proceso de 2 pasos:

1. Crea el objeto gradiente propiamente dicho. Durante la creación se define una línea en el canvas donde el gradiente comenzará y terminará. El objeto gradiente se crea con `var gradient = context.createLinearGradient`.
2. A continuación, añada 2 (o más) colores que formen el degradado. Esto se hace añadiendo múltiples paradas de color al objeto gradiente con `gradient.addColorStop`.

Argumentos:

- **`startX,startY`** es la coordenada del canvas donde comienza el degradado. En el punto inicial (y antes) el canvas es sólidamente del color del `gradientPercentPosition` más bajo.
- **`endX,endY`** es la coordenada del canvas donde termina el degradado. En el punto final (y después) el canvas es sólidamente del color del `gradientPercentPosition` más alto.
- **`gradientPercentPosition`** es un número flotante entre 0.00 y 1.00 asignado a una parada de color. Es básicamente un punto de paso porcentual a lo largo de la línea donde se aplica está parada de color en particular.
    - El gradiente comienza en el porcentaje 0.00 que es `[startX,startY]` en el canvas.
    - El degradado termina en el porcentaje 1.00 que es `[endX,endY]` en el canvas.
    - *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 degradado es un objeto que puede utilizar (¡y reutilizar!) para hacer que los trazos y rellenos de su ruta se vuelvan de color degradado.

Nota al margen: El objeto gradiente no es interno al elemento Canvas ni a su Contexto. Se trata de un objeto que puede asignar a cualquier ruta que desee. Incluso puede utilizar este objeto para colorear una ruta en un en un elemento Canvas diferente.

Las paradas de color son puntos de paso (porcentuales) a lo largo de la línea de gradiente. En cada punto de parada de color, el gradiente es totalmente (==opaco) con el color asignado. Los puntos intermedios a lo largo de la línea de gradiente entre las paradas de color se colorean como degradados de este color y el anterior.

Sugerencia importante sobre los degradados de Canvas

Cuando se crea un objeto degradado, todo el canvas se rellena "invisiblemente" con ese degradado.

Cuando `stroke()` o `fill()` un trazado, el degradado invisible se revela, pero sólo sobre el trazado que se está trazando o rellenando. trazado o rellenado.

1. Si creas un degradado lineal de rojo a magenta como éste:

```
// crea un 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 verá "invisiblemente" tu creación de degradado así:

3. Pero hasta que se use `stroke()` o `fill()` con el gradiente, no verá nada del gradiente en el Canvas.

4. Por último, si traza o rellena un trazado utilizando el degradado, el degradado "invisible" se hace visible en el canvas... pero sólo 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>
```

Sección 5.21: createRadialGradient (crear un objeto de estilo de ruta)

```
var gradient = createRadialGradient(
     centerX1, centerY1, radius1, // este es el círculo de "display"
     centerX2, centerY2, radius2 // este es el círculo de "light casting"
)
gradient.addColorStop(gradientPercentPosition, CssColor)
gradient.addColorStop(gradientPercentPosition, CssColor)
[opcionalmente añadir más paradas de color para añadir variedad al degradado]
```

Crea un degradado radial reutilizable (objeto). El objeto degradado es un objeto que puede utilizar (¡y reutilizar!) para hacer que los trazos y rellenos de tus trazados se coloreen con degradado.

Acerca de...

El degradado radial Canvas es *extremadamente diferente* de los degradados radiales tradicionales.

La definición "oficial" (¡casi indescifrable!) del degradado radial de Canvas se encuentra al final de este mensaje. No lo mires ¡¡Míralo si tienes una disposición débil!!

En términos (casi comprensibles):

- El degradado radial tiene 2 círculos: uno de "fundición" y otro de "visualización".
- El círculo de proyección proyecta luz hacia el círculo de visualización.
- Esa luz es el gradiente.
- La forma de esa luz de gradiente viene determinada por el tamaño relativo y la posición de ambos círculos.

La creación de un objeto degradado es un proceso de 2 pasos:

1. Crea el objeto gradiente propiamente dicho. Durante la creación se define una línea en el canvas donde el gradiente comenzará y terminará. El objeto gradiente se crea con `var gradient = context.radialLinearGradient`.
2. A continuación, añade 2 (o más) colores que formen el degradado. Esto se hace añadiendo múltiples paradas de color al objeto gradiente con `gradient.addColorStop`.

Argumentos:

- **`centerX1,centerY1,radius1`** define un primer círculo donde se mostrará el degradado.
- **`centerX2,centerY2,radius2`** define un segundo círculo que proyecta luz degradada en el primer círculo.
- **`gradientPercentPosition`** es un número flotante entre 0.00 y 1.00 asignado a una parada de color. Es básicamente un punto de ruta porcentual que define dónde se aplica está parada de color particular a lo largo del gradiente.
    - El gradiente comienza en el porcentaje 0,00.
    - El gradiente termina en el 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 gradiente no es interno al elemento Canvas ni a su Contexto. Se trata de un objeto que puede asignar a cualquier ruta que desee. Incluso puede utilizar este objeto para colorear una ruta en un en un elemento Canvas diferente.

Las paradas de color son puntos de paso (porcentuales) a lo largo de la línea de gradiente. En cada punto de parada de color, el gradiente es totalmente (==opaco) con el color asignado. Los puntos intermedios a lo largo de la línea de gradiente entre las paradas de color se colorean como degradados de este color y el anterior.

Sugerencia importante sobre los degradados de Canvas

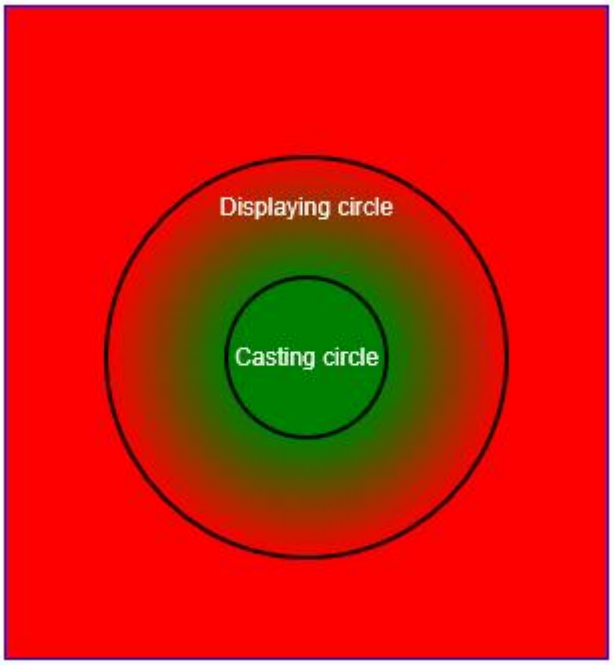
Al crear un objeto degradado, todo el degradado radial se proyecta "invisiblemente" sobre el canvas.

Cuando se use `stroke()` o `fill()` en un trazado, el degradado invisible se revela, pero sólo sobre el trazado que se está trazando o rellenando.

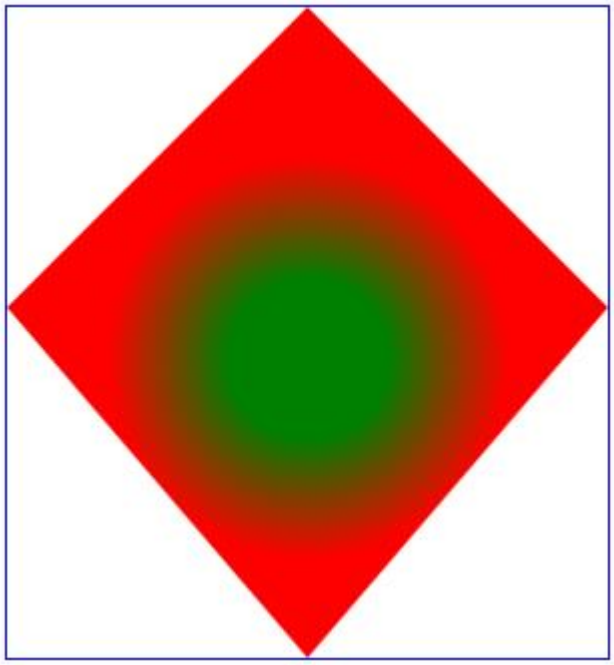
1. Si creas un degradado radial de verde a rojo como este:

```
// crear un radialGradient
var x1=150;
var y1=150;
var x2=280;
var y2=150;
var r1=100;
var r2=120;
var gradient=ctx.createRadialGradient(x1,y1,r1,x2,y2,r2);
gradient.addColorStop(0,'red');
gradient.addColorStop(1,'green');
ctx.fillStyle=gradient;
```

2. Entonces Canvas verá "invisiblemente" tu creación de degradado así:



3. Pero hasta que no se use `stroke()` o `fill()` con el gradiente, no verá nada del gradiente en el Canvas.
4. Por último, si trazas o rellenas un trazado utilizando el degradado, el degradado "invisible" se hace visible en el canvas ... pero sólo 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(){
				// variables relacionadas con canvas
				var canvas=document.getElementById("canvas");
				var ctx=canvas.getContext("2d");
				// crear un gradiente radial
				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;
				// rellenar un trazado con el gradiente
				ctx.beginPath();
				ctx.moveTo(150,0);
				ctx.lineTo(300,150);
				ctx.lineTo(150,325);
				ctx.lineTo(0,150);
				ctx.lineTo(150,0);
				ctx.fill();
			}); // fin window.onload
		</script>
	</head>
	<body>
		<canvas id="canvas" width=300 height=325></canvas>
	</body>
</html>
```

Los aterradores detalles oficiales

¿Quién decide que hace `createRadialGradient?

El W3C publica las especificaciones oficiales recomendadas que los navegadores utilizan para construir el elemento Canvas de HTML5.

La especificación W3C para `createRadialGradient` crípticamente dice así:

¿Qué crea `createRadialGradient`?

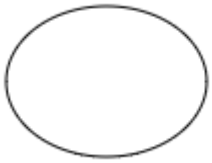
> `createRadialGradient` ... crea efectivamente un cono, tocado por los dos círculos definidos en la creación del gradiente, con la parte del cono antes del círculo inicial (0.0) usando el color del primer desplazamiento, la parte del cono después del círculo final (1.0) usando el color del último desplazamiento, y las áreas fuera del cono con el color del primer desplazamiento.

Cómo funciona internamente

> El método `createRadialGradient(x0, y0, r0, x1, y1, r1)` toma seis argumentos, los tres primeros representando el círculo inicial con origen (x0, y0) y radio r0, y los tres últimos representando el círculo con origen (x1, y1) y radio r1. Los valores están en unidades de espacio de coordenadas. Si r0 o r1 son negativos, se lanzará una excepción `IndexSizeError`. En caso contrario, el método debe devolver un `CanvasGradient` inicializado con los dos círculos especificados.
>
> Los degradados radiales deben renderizarse siguiendo estos pasos:
>
> 1. Si x0 = x1 e y0 = y1 y r0 = r1, entonces el gradiente radial no debe pintar nada. Aborta estos pasos.
> 2. Sea x(ω) = (x1-x0)ω + x0; Sea y(ω) = (y1-y0)ω + y0; Sea r(ω) = (r1-r0)ω + r0 Sea el color en ω el color en esa posición del gradiente (con los colores procedentes de la interpolación).
> 3. Para todos los valores de ω en los que r(ω) > 0, empezando por el valor de ω más próximo al infinito positivo y terminando con el valor de ω más cercano al infinito negativo, dibuje la circunferencia del círculo con radio r(ω) en la posición (x(ω), y(ω)), con el color en ω, pero pintando sólo en las partes del canvas que no hayan sido aún no han sido pintadas por círculos anteriores en este paso para esta representación del gradiente.

Capítulo 6: Rutas

Sección 6.1: Elipse



Nota: Los navegadores están en proceso de añadir un comando de dibujo `context.ellipse` *incorporado, pero este comando no está adoptado universalmente (especialmente no en IE). Los siguientes métodos funcionan en todos los navegadores.*

Dibuja una elipse dada la coordenada superior izquierda deseada:

```
// dibuja una elipse basada en x,y como coordenada superior izquierda
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 dada la coordenada deseada del punto central:

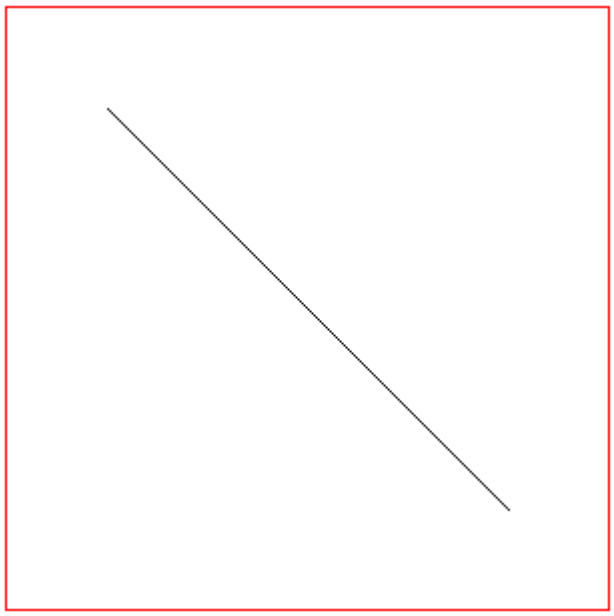
```
// dibuja una elipse basada en cx,cy que son las coordenadas del punto central de la elipse
function drawEllipse2(cx,cy,width,height){
      var PI2=Math.PI*2;
      var ratio=height/width;
      var radius=Math.max(width,height)/2;
      var increment = 1 / radius;
      ctx.beginPath();
      var 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();
}
```

Sección 6.2: Línea sin borrosidad

Cuando Canvas dibuja una línea automáticamente añade anti-aliasing para curar visualmente las "irregularidades". El resultado es una línea menos dentada pero más borrosa.

Esta función dibuja una línea entre 2 puntos sin anti-aliasing utilizando el algoritmo Bresenham's Line. El resultado es una línea nítida sin irregularidades.

Nota importante: *Este método de dibujo píxel a píxel es mucho más lento que* `context.lineTo`.



```
// Uso:
bresenhamLine(50,50,250,250);
// x,y inicio de línea
// xx,yy fin de línea
// se dibujan los píxeles de inicio y final de línea
function bresenhamLine(x, y, xx, yy) {
      var oldFill = ctx.fillStyle; // guardar el antiguo estilo de relleno
      ctx.fillStyle = ctx.strokeStyle; // mover el estilo de trazo a relleno
      xx = Math.floor(xx);
      yy = Math.floor(yy);
      x = Math.floor(x);
      y = Math.floor(y);
      // BRENSENHAM
      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; // valor del error
      var end = false;
      var x1 = x;
      var y1 = y;
      while(!end) {
            ctx.fillRect(x1, y1, 1, 1); // dibujar cada píxel como un recto
            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; // restaurar el antiguo estilo de relleno
}
```

Capítulo 7: Navegar por una ruta

Sección 7.1: Encontrar el punto en la curva

Este ejemplo encuentra un punto en una curva bezier o cúbica en la `position` donde la `position` es la distancia unitaria en la curva 0 <= `position` <= 1. La posición se sujeta al rango, por lo que si se pasan valores < 0 o > 1 se pondrán 0,1 respectivamente.

Pase a la función 6 coordenadas para el bezier cuadrático u 8 para el cúbico.

El último argumento opcional es el vector devuelto (punto). Si no se indica, se creará.

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};
// para beziers cúbicos
point = getPointOnCurve(0.5, p1.x, p1.y, p2.x, p2.y, p3.x, p3.y, p4.x, p4.y, point);
// o No hay necesidad de establecer el punto, ya que es una referencia y se establecerá
getPointOnCurve(0.5, p1.x, p1.y, p2.x, p2.y, p3.x, p3.y, p4.x, p4.y, point);
// o para crear un nuevo punto
var point1 = getPointOnCurve(0.5, p1.x, p1.y, p2.x, p2.y, p3.x, p3.y, p4.x, p4.y);
// para beziers cuadráticos
point = getPointOnCurve(0.5, p1.x, p1.y, p2.x, p2.y, p3.x, p3.y, null, null, point);
// o No hay necesidad de establecer el punto, ya que es una referencia y se establecerá
getPointOnCurve(0.5, p1.x, p1.y, p2.x, p2.y, p3.x, p3.y, null, null, point);
// o para crear un nuevo punto
var point1 = getPointOnCurve(0.5, p1.x, p1.y, p2.x, p2.y, p3.x, p3.y);
```

La función

```
getPointOnCurve = function(position, 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 un bezier cuadrático. `vec` es opcional y contendrá el punto devuelto si se suministra. Si no, será creado.

```
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;
}
```

Sección 7.2: Hallar la extensión de una curva cuadrática

Cuando necesites encontrar el rectángulo que delimita una curva bezier cuadrática puedes utilizar el siguiente método performance siguiente.

```
// Este método fue descubierto por Blindman67 y resuelve normalizando primer el punto de control
// reduciendo así la complejidad del algoritmo
// coordenadas x1,y1, x2,y2, x3,y3 start, control y end de bezier
// [extent] es opcional y, si se proporciona, se le añadirá la extensión, lo que le permitirá
// utilizar la función
// para obtener la extensión de muchos beziers.
// devuelve el objeto extent (si no se proporciona, se crea un nuevo extent)
// Propiedades del objeto Extent
// top, left,right,bottom,width,height
function getQuadraticCurevExtent(x1, y1, x2, y2, x3, y3, extent) {
      var brx, bx, x, bry, by, y, px, py;
      // resolver cuadrática para límites por BM67 ecuación de normalización
      brx = x3 - x1; // obtener rango x
      bx = x2 - x1; // obtener desplazamiento del punto de control x
      x = bx / brx; // normalizar el punto de control que se utiliza para comprobar si el máximo
      está  dentro del intervalo
      // haz lo mismo para los puntos y
      bry = y3 - y1;
      by = y2 - y1;
      y = by / bry;
      px = x1; // establecer valores por defecto en caso de que los máximos estén fuera de rango
      py = y1;
      // busca top/left, top/right, bottom/left, o bottom/right
      if (x < 0 || x > 1) { // comprobar si x máximo está en la curva
            px = bx * bx / (2 * bx - brx) + x1; // obtener el máximo x
      }
      if (y < 0 || y > 1) { // lo mismo para y
            py = by * by / (2 * by - bry) + y1;
      }
      // crear objeto extent y añadir 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 { // utilizar la extensión aplicada y ampliarla para que se ajuste a esta curva
            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;
}
```

Sección 7.3: Encontrar puntos a lo largo de una curva cúbica de Bézier

Este ejemplo encuentra un array de puntos aproximadamente espaciados uniformemente a lo largo de una curva cúbica de Bézier.

Descompone los segmentos 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
```

```
//
// Atribución: @Blindman67 de StackOverflow
// Cita: http://stackoverflow.com/questions/36637211/drawing-a-curved-line-in-css-or-canvas-and-
moving-circlealong-it/36827074#36827074
// Modificado a partir de la cita anterior
//
// ptCount: muestrear tantos puntos a intervalos a lo largo de la curva
// pxTolerance: espacio aproximado permitido entre puntos
// Ax,Ay,Bx,By,Cx,Cy,Dx,Dy: puntos de control que definen la curva
//
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);
}
```

Sección 7.4: Encontrar puntos a lo largo de una curva cuadrática

Este ejemplo encuentra un conjunto de puntos aproximadamente espaciados uniformemente a lo largo de una curva cuadrática.

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

```
// Devuelve: un array de puntos aproximadamente espaciados uniformemente a lo largo de una curva
cuadrática
//
// Atribución: @Blindman67 de StackOverflow
// Cita: http://stackoverflow.com/questions/36637211/drawing-a-curved-line-in-css-or-canvas-and-
moving-circlealong-it/36827074#36827074
// Modificado a partir de la cita anterior
//
// ptCount: muestrear tantos puntos a intervalos a lo largo de la curva
// pxTolerance: espacio aproximado permitido entre puntos
// Ax,Ay,Bx,By,Cx,Cy: puntos de control que definen la curva
//
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);
}
```

Sección 7.5: Encontrar puntos a lo largo de una línea

Este ejemplo encuentra un array de puntos aproximadamente espaciados uniformemente a lo largo de una línea.

Descompone los segmentos 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);
}
```

Sección 7.6: Encontrar puntos a lo largo de una Ruta completa que contenga curvas y líneas

Este ejemplo encuentra un conjunto de puntos aproximadamente espaciados uniformemente a lo largo de una Ruta completa.

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

```
// Variables relacionadas con la ruta
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;
// variables relacionadas con canvas
var canvas=document.createElement("canvas");
var ctx=canvas.getContext("2d");
document.body.appendChild(canvas);
canvas.width=378;
canvas.height=256;
// Indicar al Context que trace un waypoint además de dibujar la ruta
plotPathCommands(ctx,n,tolerance);
// Comandos de dibujo de rutas
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();
// Decirle al Context que deje de trazar waypoints
ctx.stopPlottingPathCommands();
// Demostración: Dibujar incrementalmente la ruta utilizando los puntos trazados
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 plug-in que calcula automáticamente los puntos de ruta

Este código modifica los comandos de dibujo de estos Contextos de Canvas para que los comandos no sólo dibujen la línea o curva, sino que también creen un array de puntos a lo largo de toda la ruta:

- `beginPath`,
- `moveTo`,
- `lineTo`,
- `quadraticCurveTo`,
- `bezierCurveTo`.

Nota importante

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

El propósito de este Contexto modificado es permitirle "enchufar" el cálculo del array de puntos en su código existente sin tener que modificar sus comandos de dibujo de Ruta. Pero, no es necesario utilizar este Contexto modificado - puede llamar por separado a las funciones individuales que descomponen una línea, una curva cuadrática y una curva cúbica Bezier y luego concatenar manualmente esas matrices de puntos individuales en un único array de puntos para toda la ruta.

Obtén una copia del array de puntos resultante utilizando la función `getPathPoints` suministrada.

Si dibuja varias rutas con el contexto modificado, el array de puntos contendrá un único conjunto concatenado de puntos para todas las rutas dibujadas.

Si, por el contrario, quieres obtener matrices de puntos separadas, puedes obtener el array actual con `getPathPoints` y luego borrar esos puntos del array con la función suministrada `clearPathPoints`.

```
// Modifica el Contexto del Canvas para calcular un conjunto de puntos de ruta aproximadamente
// espaciados uniformemente a medida que dibuja la(s) ruta(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;
      // no uses myPathPoints[] directamente -- usa "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 demostración completa

```
// Variables relacionadas con la ruta
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;
// variables relacionadas con canvas
var canvas=document.createElement("canvas");
var ctx=canvas.getContext("2d");
document.body.appendChild(canvas);
canvas.width=378;
canvas.height=256;
// Indicar al Context que trace un waypoint además de dibujar la ruta
plotPathCommands(ctx,n,tolerance);
// Comandos de dibujo de rutas
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();
// Decir al Context que deje de trazar waypoints
ctx.stopPlottingPathCommands();
// Dibuja la ruta de forma incremental utilizando los puntos trazados
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 Plug-in
////////////////////////////////////////
// Modifica el Contexto del Canvas para calcular un conjunto de puntos de ruta aproximadamente
// espaciados uniformemente mientras dibuja la(s) ruta(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;
     // no uses myPathPoints[] directamente -- usa "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;
    }
}
////////////////////////////////
// Funciones auxiliares
////////////////////////////////
// Devuelve: un conjunto de puntos aproximadamente espaciados uniformemente a lo largo de una curva
cúbica de Bézier
//
// Atribución: @Blindman67 de StackOverflow
// Cita: http://stackoverflow.com/questions/36637211/drawing-a-curved-line-in-css-or-canvas-and-
moving-circlealong-it/36827074#36827074
// Modificado a partir de la cita anterior
//
// ptCount: muestrear tantos puntos a intervalos a lo largo de la curva
// pxTolerance: espacio aproximado permitido entre puntos
// Ax,Ay,Bx,By,Cx,Cy,Dx,Dy: puntos de control que definen la curva
//
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);
}
// Devuelve: un array de puntos aproximadamente espaciados uniformemente a lo largo de una curva
cuadrática
//
// Atribución: @Blindman67 de StackOverflow
// Cita: http://stackoverflow.com/questions/36637211/drawing-a-curved-line-in-css-or-canvas-and-
moving-circlealong-it/36827074#36827074
// Modificado a partir de la cita anterior
//
// ptCount: muestrear tantos puntos a intervalos a lo largo de la curva
// pxTolerance: espacio aproximado permitido entre puntos
// Ax,Ay,Bx,By,Cx,Cy: puntos de control que definen la curva
//
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);
}
// Devuelve: un array de puntos aproximadamente espaciados uniformemente a lo largo de una línea
//
// pxTolerance: espacio aproximado permitido entre puntos
// Ax,Ay,Bx,By: puntos finales que definen la línea
//
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);
}
```

Sección 7.7: Curvas de Bézier divididas en la posición

Este ejemplo divide las curvas cúbicas y Bézier en dos.

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

Ejemplo de uso

División en dos de una curva de Bézier cuadrática.

```
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
// Dibujar las 2 nuevas curvas
// Asume que ctx es el contexto canvas 2d
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();
```

División en dos de una curva cúbica de Bézier.

```
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
// Dibujar las 2 nuevas curvas
// Asume que ctx es el contexto canvas 2d
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 split

`splitCurveAt = function(position, x1, y1, x2, y2, x3, y3, [x4, y4])`

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

Nota: La función tiene algún código opcional comentado `/* */` que trata los casos extremos en los que las curvas resultantes pueden tener longitud cero, o caer fuera del inicio o los extremos de la curva original. En efecto intentar dividir una curva fuera del rango válido para `position >= 0` o `position >= 1` arrojará un error de rango error. Esto se puede quitar y funcionará bien, aunque puede tener curvas resultantes que tienen cero.

```
// Con lanzar RangeError si no 0 < position < 1
// x1, y1, x2, y2, x3, y3 para curvas cuadráticas
// x1, y1, x2, y2, x3, y3, x4, y4 para curvas cúbicas
// Devuelve un array de puntos que representan 2 curvas. Las curvas son del mismo tipo que la curva
dividida.
var splitCurveAt = function(position, x1, y1, x2, y2, x3, y3, x4, y4){
     var v1, v2, v3, v4, quad, retPoints, i, c;
     //=============================================================================================
     // puedes eliminar esto ya que la función seguirá funcionando y las curvas resultantes
     seguirán renderizándose
     // pero a otras funciones de curvas puede que no les gusten las curvas con longitud 0
```

```
//=============================================================================================
if(position <= 0 || position >= 1){
     throw RangeError("spliteCurveAt requires position > 0 && position < 1");
}
//=============================================================================================
// Si elimina el error de rango anterior, puede utilizar una o ambas de las siguientes
secciones comentadas
// La división de curvas posición < 0 o position > 1 seguirá creando curvas válidas, pero
serán
// se extienden más allá de los puntos finales
//=============================================================================================
// Bloquear la posición para dividir en la curva.
/* opcional A
position = position < 0 ? 0 : position > 1 ? 1 : position;
opcional A fin */
//=============================================================================================
// la siguiente sección comentada devolverá la curva original si la división resulta de
longitud 0 curva
// Si lo desea, puede descomentar esta opción.
/* opcional 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];
     }
}
opcional B fin */
retPoints = []; // array de coordenadas
i = 0;
quad = false; // suponer bezier cúbico
v1 = {};
v2 = {};
v4 = {};
v1.x = x1;
v1.y = y1;
v2.x = x2;
v2.y = y2;
if(x4 === undefined || x4 === null){
     quad = true; // esto es un bezier cuadrático
     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; // punto inicial
retPoints[i++] = v1.y;
if(quad){ // bézier cuadrático dividido
     retPoints[i++] = (v1.x += (v2.x - v1.x) * c); // nuevo punto de control para la primera
     curva
     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; // nuevo final e inicio de la primera y
     segunda curvas
     retPoints[i++] = v1.y + (v2.y - v1.y) * c;
     retPoints[i++] = v2.x; // nuevo punto de control para la segunda curva
     retPoints[i++] = v2.y;
     retPoints[i++] = v4.x; // nuevo punto final de la segunda curva
```

```
            retPoints[i++] = v4.y;
        //=======================================================
        // devolver array con 2 curvas
            return retPoints;
        }
        retPoints[i++] = (v1.x += (v2.x - v1.x) * c); // primera curva primer punto de control
        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); // primera curva segundo punto de control
        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; // punto final e inicial de la primera segunda
        curva
        retPoints[i++] = v1.y + (v2.y - v1.y) * c;
        retPoints[i++] = v2.x; // segunda curva primer punto de control
        retPoints[i++] = v2.y;
        retPoints[i++] = v3.x; // segunda curva segundo punto de control
        retPoints[i++] = v3.y;
        retPoints[i++] = v4.x; // punto final de la segunda curva
        retPoints[i++] = v4.y;
        //=======================================================
        // devolver array con 2 curvas
        return retPoints;
}
```

Sección 7.8: Recortar curva de Bézier

Este ejemplo 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 inclusivo, puede recortar curvas cuadráticas y cúbicas. El tipo de curva viene determinado por el último argumento x, `x4`. Si no es **`undefined`** o **`null`** entonces asume que la curva es cúbica de lo contrario la curva es cuadrática.

La curva recortada se devuelve como un array 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
// Dibujar la curva recortada
// Asume que ctx es el contexto canvas 2d
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
// Dibujar la curva recortada
// Asume que ctx es el contexto canvas 2d
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 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; // esto es un bezier cuadrático
     }
     if(fromPos > toPos){ // intercambiar fromPos despues de toPos
          i = fromPos;
          fromPos = toPos
          toPos = i;
     }
     // abrazadera en la curva
     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 es necesario recortar
          retBez = [x1, y1, x2, y2, x3, y3]; // devolver el bezier original
          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++]);
        i = 0;
        retBez = [s[i++], s[i++], s[i++], s[i++], s[i++], s[i++], s[i++], s[i++]];
        return retBez;
}
```

Sección 7.9: Longitud de una curva cúbica de Bézier (una aproximación)

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

Método: La longitud de una curva cúbica de Bézier 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 recorrida por esos puntos.

Precisión: La longitud aproximada tiene una precisión del 99+% utilizando el tamaño de muestreo predeterminado de 40.

```
// Devolución: Aproximación de la longitud de una curva cúbica de Bezier
//
// Ax,Ay,Bx,By,Cx,Cy,Dx,Dy: los 4 puntos de control de la curva
// sampleCount [optional, default=40]: cuántos intervalos hay que calcular
// Requiere: cubicQxy (incluido más abajo)
//
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));
}
// Devuelve: un punto [x,y] a lo largo de una curva cúbica de Bézier en el intervalo T
//
// Atribución: @Blindman67 de StackOverflow
// Cita: http://stackoverflow.com/questions/36637211/drawing-a-curved-line-in-css-or-canvas-and-
moving-circlealong-it/36827074#36827074
// Modificado a partir de la cita anterior
//
// t: un intervalo a lo largo de la curva (0<=t<=1)
// ax,ay,bx,by,cx,cy,dx,dy: puntos de control que definen la curva
//
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
     });
}
```

Sección 7.10: 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);
}
```

Derivada de la función bézier cuadrática F(t) = a * (1 - t)2 + 2 * b * (1 - t) * t + c * t2

Capítulo 8: Arrastrar formas e imágenes en el canvas

Sección 8.1: Cómo se "mueven" REALMENTE las formas y las imágenes en el Canvas

Un problema: Canvas sólo recuerda píxeles, no formas ni imágenes

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

Puede que te sorprenda que, al igual que una imagen, si dibujas un círculo en un canvas no puedes arrastrar ese círculo alrededor del canvas. Eso es porque el canvas no recordará dónde dibujó el círculo.

```
// ¡¡este arco (==circulo) no es arrastrable!!
context.beginPath();
context.arc(20, 30, 15, 0, Math.PI*2);
context.fillStyle='blue';
context.fill();
```

Lo que el Canvas NO sabe...

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

Lo que el Canvas SÍ sabe...

El canvas conoce el color de cada píxel de su superficie de dibujo.

El canvas puede decirte que en x,y==[20,30] hay un píxel azul, pero no sabe si este píxel azul forma parte de un círculo.

Lo que significa...

Esto significa que todo lo que se dibuja en el canvas es permanente: inamovible e inmutable.

- Canvas no puede mover el círculo ni cambiar su tamaño.
- Canvas no puede volver a colorear el círculo ni borrarlo.
- Canvas no puede decir si el ratón se cierne sobre el círculo.
- Canvas no puede decir si el círculo está colisionando con otro círculo.
- Canvas no puede permitir que un usuario arrastre el círculo alrededor del canvas.

Pero Canvas puede dar la I-L-U-S-I-O-N de movimiento

El canvas puede dar la ilusión de movimiento borrando continuamente el círculo y volviéndolo a dibujar en una posición diferente. Al redibujar el canvas muchas veces por segundo, se engaña al ojo haciéndole ver que el círculo se mueve a través del canvas.

- Borrar el canvas
- Actualizar la posición del círculo
- Vuelve a dibujar el círculo en su nueva posición
- Repite, repite, repite...

Este código da la ilusión de movimiento al redibujar continuamente un círculo en nuevas posiciones.

```
// crear un canvas
var canvas=document.createElement("canvas");
var ctx=canvas.getContext("2d");
ctx.fillStyle='red';
document.body.appendChild(canvas);
// una variable que indica la posición X de un círculo
var circleX=20;
// empezar a animar el círculo a través del canvas
// borrando y volviendo a dibujar continuamente el círculo
// en nuevas posiciones
requestAnimationFrame(animate);
function animate(){
      // actualizar la posición X del círculo
      circleX++;
      // redibujar el círculo en su nueva posición
      ctx.clearRect(0,0,canvas.width,canvas.height);
      ctx.beginPath();
      ctx.arc( circleX, 30,15,0,Math.PI*2 );
      ctx.fill();
      // solicitar otro bucle animate()
      requestAnimationFrame(animate);
}
```

Sección 8.2: Arrastrar círculos y rectángulos en el canvas

¿Qué es una "Forma"?

Normalmente, para guardar las formas se crea un objeto JavaScript "shape" que representa cada forma.

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

Por supuesto, en realidad no estás guardando formas. En su lugar, está guardando la definición de cómo dibujar las formas.

A continuación, coloca cada objeto de forma en un array para facilitar la consulta.

```
// guardar información relevante sobre las formas dibujadas en el canvas
var shapes=[];
// define un círculo y guárdalo en el array shapes[].
shapes.push( {x:10, y:20, radius:15, fillcolor:'blue'} );
// define un rectángulo y guárdalo en el array shapes[].
shapes.push( {x:10, y:100, width:50, height:35, fillcolor:'red'} );
```

Uso de los eventos del ratón para hacer Dragging

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

Al pasar el ratón (`mousedowm`):

Comprueba si hay algún objeto bajo el ratón. Si una forma está bajo el ratón, el usuario está intentando arrastrar esa forma. Así que mantiene una referencia a esa forma y establece un indicador `true/false isDragging` que indica que se está produciendo un arrastre.

Al mover el ratón (`mousemove`):

Calcula la distancia que el ratón ha sido arrastrado desde el último evento `mousemove` y cambia la posición de la forma arrastrada por esa distancia. Para cambiar la posición de la forma, se cambian las propiedades de posición `x,y` en el objeto de esa forma.

Al subir o bajar el ratón (`mouseup` o `mouseout`):

El usuario tiene la intención de detener la operación de arrastre, así que borra la bandera "`isDragging`". El arrastre se ha completado.

Demostración: Arrastrar círculos y rectángulos en el canvas

Esta demo arrastra círculos y rectángulos sobre el canvas respondiendo a los eventos del ratón y dando la ilusión de movimiento borrando y redibujando.

```
// variables relacionadas con canvas
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';
// utilizado para calcular la posición del canvas con respecto a la ventana
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(); }
// guardar información relevante sobre las formas dibujadas en el canvas
var shapes=[];
// define un círculo y guárdalo en el array shapes[].
shapes.push( {x:30, y:30, radius:15, color:'blue'} );
// define un rectángulo y guárdalo en el array shapes[].
shapes.push( {x:100, y:-1, width:75, height:35, color:'red'} );
// vars relacionados con el arrastre
var isDragging=false;
var startX,startY;
// contiene el índice de la forma arrastrada (si existe)
var selectedShapeIndex;
// dibujar las formas en el canvas
drawAll();
// escuchar eventos del ratón
canvas.onmousedown=handleMouseDown;
canvas.onmousemove=handleMouseMove;
canvas.onmouseup=handleMouseUp;
canvas.onmouseout=handleMouseOut;
// dado ratón X & Y (mx & my) y objeto forma
// devolver true/false si el ratón está dentro de la forma
function isMouseInShape(mx,my,shape){
     if(shape.radius){
          // esto es un círculo
          var dx=mx-shape.x;
          var dy=my-shape.y;
          // prueba matemática para ver si el ratón está dentro de un círculo
          if(dx*dx+dy*dy<shape.radius*shape.radius){
               // sí, el ratón está dentro de este círculo
               return(true);
          }
```

```
        } else if(shape.width){
            // esto es un rectángulo
            var rLeft=shape.x;
            var rRight=shape.x+shape.width;
            var rTop=shape.y;
            var rBott=shape.y+shape.height;
            // prueba matemática para ver si el ratón está dentro del rectángulo
            if( mx>rLeft && mx<rRight && my>rTop && my<rBott){
                return(true);
            }
        }
        // el ratón no está en ninguna de las formas
        return(false);
}
function handleMouseDown(e){
        // indica al navegador que estamos gestionando este evento
        e.preventDefault();
        e.stopPropagation();
        // calcular la posición actual del ratón
        startX=parseInt(e.clientX-offsetX);
        startY=parseInt(e.clientY-offsetY);
        // probar la posición del ratón con todas las formas
        // publicar el resultado si el ratón está en una forma
        for(var i=0;i<shapes.length;i++){
            if(isMouseInShape(startX,startY,shapes[i])){
                // el ratón está dentro de esta forma
                // selecciona esta forma
                selectedShapeIndex=i;
                // establece el indicador isDragging
                isDragging=true;
                // y volver (==dejar de buscar más formas bajo el ratón)
                return;
            }
        }
}
function handleMouseUp(e){
        // volver si no estamos arrastrando
        if(!isDragging){return;}
        // indica al navegador que estamos gestionando este evento
        e.preventDefault();
        e.stopPropagation();
        // el arrastre ha terminado -- borra la bandera isDragging
        isDragging=false;
}
function handleMouseOut(e){
        // devolver si no estamos arrastrando
        if(!isDragging){return;}
        // indica al navegador que estamos gestionando este evento
        e.preventDefault();
        e.stopPropagation();
        // el arrastre ha terminado -- borra la bandera isDragging
        isDragging=false;
}
function handleMouseMove(e){
        // devolver si no estamos arrastrando
        if(!isDragging){return;}
        // indica al navegador que estamos gestionando este evento
        e.preventDefault();
        e.stopPropagation();
        // calcular la posición actual del ratón
        mouseX=parseInt(e.clientX-offsetX);
        mouseY=parseInt(e.clientY-offsetY);
        // ¿cuánto se ha arrastrado el ratón desde su posición anterior?
        var dx=mouseX-startX;
```

```
        var dy=mouseY-startY;
        // mover la forma seleccionada por la distancia de arrastre
        var selectedShape=shapes[selectedShapeIndex];
        selectedShape.x+=dx;
        selectedShape.y+=dy;
        // borrar el canvas y volver a dibujar todas las formas
        drawAll();
        // actualizar la posición inicial de arrastre (== la posición actual del ratón)
        startX=mouseX;
        startY=mouseY;
}
// borrar el canvas y volver a dibujar todas las formas en sus posiciones actuales
function drawAll(){
        ctx.clearRect(0,0,cw,ch);
        for(var i=0;i<shapes.length;i++){
                var shape=shapes[i];
                if(shape.radius){
                        // es un círculo
                        ctx.beginPath();
                        ctx.arc(shape.x,shape.y,shape.radius,0,Math.PI*2);
                        ctx.fillStyle=shape.color;
                        ctx.fill();
                }else if(shape.width){
                        // es un rectángulo
                        ctx.fillStyle=shape.color;
                        ctx.fillRect(shape.x,shape.y,shape.width,shape.height);
                }
        }
}
```

Sección 8.3: Arrastrar formas irregulares por el canvas

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

Los círculos y rectángulos tienen pruebas matemáticas para comprobar si el ratón está dentro de ellos. Esto hace que las pruebas de círculos y rectángulos de forma fácil, rápida y eficaz. Puede "probar" cientos de círculos o rectángulos en una fracción de segundo.

También puede arrastrar formas irregulares. Pero las formas irregulares no tienen una prueba matemática rápida. Afortunadamente, las formas irregulares tienen una prueba de acierto incorporada para determinar si un punto (ratón) está dentro de la forma: `context.isPointInPath`. Aunque `isPointInPath` funciona bien, no es tan eficiente como las pruebas de acierto puramente matemáticas: a menudo es hasta 10 veces más lento que las pruebas de acierto puramente matemáticas.

Un requisito cuando se utiliza `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 del trazado (como arriba), pero no necesita `stroke()` o `fill()` la Ruta antes de probarlo con `isPointInPath`. De esta forma puede probar Rutas dibujadas previamente sin tener que sobrescribir (trazar/rellenar) esas Rutas previas en el propio Canvas.

La forma irregular no tiene por qué ser tan común como el triángulo cotidiano. También puede probar cualquier irregular.

Este ejemplo comentado muestra cómo arrastrar formas Ruta irregulares, así como círculos y rectángulos:

```
// variables relacionadas con canvas
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';
```

```
// utilizado para calcular la posición del canvas con respecto a la ventana
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(); }
// guardar información relevante sobre las formas dibujadas en el canvas
var shapes=[];
// define un círculo y guárdalo en el array shapes[].
shapes.push( {x:20, y:20, radius:15, color:'blue'} );
// define un rectángulo y guárdalo en el array shapes[].
shapes.push( {x:100, y:-1, width:75, height:35, color:'red'} );
// define una trayectoria triangular y guárdala en el array shapes[].
shapes.push( {x:0, y:0, points:[{x:50,y:30},{x:75,y:60},{x:25,y:60}],color:'green'} );
// variables relacionadas con el arrastre
var isDragging=false;
var startX,startY;
// contiene el índice de la forma arrastrada (si existe)
var selectedShapeIndex;
// dibujar las formas en el canvas
drawAll();
// escuchar los eventos del ratón
canvas.onmousedown=handleMouseDown;
canvas.onmousemove=handleMouseMove;
canvas.onmouseup=handleMouseUp;
canvas.onmouseout=handleMouseOut;
// dado ratón X & Y (mx & my) y objeto forma
// devolver true/false si el ratón está dentro de la forma
function isMouseInShape(mx,my,shape){
     if(shape.radius){
          // esto es un círculo
          var dx=mx-shape.x;
          var dy=my-shape.y;
          // prueba matemática para ver si el ratón está dentro de un círculo
          if(dx*dx+dy*dy<shape.radius*shape.radius){
               // si el ratón está dentro de este círculo
               return(true);
          }
     } else if(shape.width){
          // esto es un rectángulo
          var rLeft=shape.x;
          var rRight=shape.x+shape.width;
          var rTop=shape.y;
          var rBott=shape.y+shape.height;
          // prueba matemática para ver si el ratón está dentro del rectángulo
          if( mx>rLeft && mx<rRight && my>rTop && my<rBott){
               return(true);
          }
     } else if(shape.points){
          // se trata de un recorrido polylineal
          // En primer lugar, vuelva a definir la ruta (¡no es necesario trazar/rellenar!)
          defineIrregularPath(shape);
          // A continuación, prueba de acierto con isPointInPath
          if(ctx.isPointInPath(mx,my)){
               return(true);
          }
     }
     // el ratón no está en ninguna de las formas
     return(false);
```

```
}
function handleMouseDown(e){
     // indica al navegador que estamos gestionando este evento
     e.preventDefault();
     e.stopPropagation();
     // calcular la posición actual del ratón
     startX=parseInt(e.clientX-offsetX);
     startY=parseInt(e.clientY-offsetY);
     // probar la posición del ratón con todas las formas
     // publicar el resultado si el ratón está en una forma
     for(var i=0;i<shapes.length;i++){
          if(isMouseInShape(startX,startY,shapes[i])){
               // el ratón está dentro de esta forma
               // selecciona esta forma
               selectedShapeIndex=i;
               // establece el indicador isDragging
               isDragging=true;
               // y volver (==dejar de buscar más formas bajo el ratón)
               return;
          }
     }
}
function handleMouseUp(e){
     // devolver si no estamos arrastrando
     if(!isDragging){return;}
     // indica al navegador que estamos gestionando este evento
     e.preventDefault();
     e.stopPropagation();
     // el arrastre ha terminado -- borra la bandera isDragging
     isDragging=false;
}
function handleMouseOut(e){
     // devolver si no estamos arrastrando
     if(!isDragging){return;}
     // indica al navegador que estamos gestionando este evento
     e.preventDefault();
     e.stopPropagation();
     // el arrastre ha terminado -- borra la bandera isDragging
     isDragging=false;
}
function handleMouseMove(e){
     // devolver si no estamos arrastrando
     if(!isDragging){return;}
     // indica al navegador que estamos gestionando este evento
     e.preventDefault();
     e.stopPropagation();
     // calcular la posición actual del ratón
     mouseX=parseInt(e.clientX-offsetX);
     mouseY=parseInt(e.clientY-offsetY);
     // ¿cuánto se ha arrastrado el ratón desde su posición anterior?
     var dx=mouseX-startX;
     var dy=mouseY-startY;
     // mover la forma seleccionada por la distancia de arrastre
     var selectedShape=shapes[selectedShapeIndex];
     selectedShape.x+=dx;
     selectedShape.y+=dy;
     // borrar el canvas y redibujar todas las formas
     drawAll();
     // actualizar la posición inicial de arrastre (== la posición actual del ratón)
     startX=mouseX;
     startY=mouseY;
}
// borrar el canvas y volver a dibujar todas las formas en sus posiciones actuales
function drawAll(){
```

```
    ctx.clearRect(0,0,cw,ch);
    for(var i=0;i<shapes.length;i++){
        var shape=shapes[i];
        if(shape.radius){
            // es un círculo
            ctx.beginPath();
            ctx.arc(shape.x,shape.y,shape.radius,0,Math.PI*2);
            ctx.fillStyle=shape.color;
            ctx.fill();
        } else if(shape.width){
            // es un rectángulo
            ctx.fillStyle=shape.color;
            ctx.fillRect(shape.x,shape.y,shape.width,shape.height);
        } else if(shape.points){
            // es un recorrido polilineal
            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();
}
```

Sección 8.4: Arrastrar imágenes por el canvas

Vea este Ejemplo para una explicación general de arrastrar Formas alrededor del Canvas.

Este ejemplo comentado muestra cómo arrastrar imágenes por el canvas.

```
// variables relacionadas con canvas
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';
// utilizado para calcular la posición del canvas con respecto a la Ventana
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(); }
// guardar información relevante sobre las formas dibujadas en el canvas
var shapes=[];
// variables relacionadas con el arrastre
var isDragging=false;
var startX,startY;
// contiene el índice de la forma arrastrada (si existe)
var selectedShapeIndex;
```

```
// cargar la imagen
var card=new Image();
card.onload=function(){
      // definir una imagen y guardarla en el array shapes[]
      shapes.push( {x:30, y:10, width:127, height:150, image:card} );
      // dibujar las formas en el canvas
      drawAll();
      // escuchar eventos del ratón
      canvas.onmousedown=handleMouseDown;
      canvas.onmousemove=handleMouseMove;
      canvas.onmouseup=handleMouseUp;
      canvas.onmouseout=handleMouseOut;
};
// pon aquí el src de la imagen
card.src='https://dl.dropboxusercontent.com/u/139992952/stackoverflow/card.png';
// dado ratón X & Y (mx & my) y objeto forma
// devolver true/false si el ratón está dentro de la forma
function isMouseInShape(mx,my,shape){
      // ¿esta forma es una imagen?
      if(shape.image){
            // esto es un rectángulo
            var rLeft=shape.x;
            var rRight=shape.x+shape.width;
            var rTop=shape.y;
            var rBott=shape.y+shape.height;
            // prueba matemática para ver si el ratón está dentro de la imagen
            if( mx>rLeft && mx<rRight && my>rTop && my<rBott){
                  return(true);
            }
      }
      // el ratón no está en ninguna de estas formas
      return(false);
}
function handleMouseDown(e){
      // indica al navegador que estamos gestionando este evento
      e.preventDefault();
      e.stopPropagation();
      // calcular la posición actual del ratón
      startX=parseInt(e.clientX-offsetX);
      startY=parseInt(e.clientY-offsetY);
      // comprobar la posición del ratón en todas las formas publicar el resultado si el ratón está
      en una forma
      for(var i=0;i<shapes.length;i++){
            if(isMouseInShape(startX,startY,shapes[i])){
                  // el ratón está dentro de esta forma seleccione esta forma
                  selectedShapeIndex=i;
                  // establece el indicador isDragging
                  isDragging=true;
                  // y devolver (==dejar de buscar más formas bajo el ratón)
                  return;
            }
      }
}
function handleMouseUp(e){
      // devolver si no estamos arrastrando
      if(!isDragging){return;}
      // indica al navegador que estamos gestionando este evento
      e.preventDefault();
      e.stopPropagation();
      // el arrastre ha terminado -- borra la bandera isDragging
      isDragging=false;
}
function handleMouseOut(e){
      // devolver si no estamos arrastrando
```

```
    if(!isDragging){return;}
    // indica al navegador que estamos gestionando este evento
    e.preventDefault();
    e.stopPropagation();
    // el arrastre ha terminado -- borra la bandera isDragging
    isDragging=false;
}
function handleMouseMove(e){
    // devolver si no estamos arrastrando
    if(!isDragging){return;}
    // indica al navegador que estamos gestionando este evento
    e.preventDefault();
    e.stopPropagation();
    // calcular la posición actual del ratón
    mouseX=parseInt(e.clientX-offsetX);
    mouseY=parseInt(e.clientY-offsetY);
    // ¿cuánto se ha arrastrado el ratón desde su posición anterior?
    var dx=mouseX-startX;
    var dy=mouseY-startY;
    // mover la forma seleccionada por la distancia de arrastre
    var selectedShape=shapes[selectedShapeIndex];
    selectedShape.x+=dx;
    selectedShape.y+=dy;
    // borrar el canvas y redibujar todas las formas
    drawAll();
    // actualizar la posición inicial de arrastre (== la posición actual del ratón)
    startX=mouseX;
    startY=mouseY;
}
// borrar el canvas y volver a dibujar todas las formas en sus posiciones actuales
function drawAll(){
    ctx.clearRect(0,0,cw,ch);
    for(var i=0;i<shapes.length;i++){
        var shape=shapes[i];
        if(shape.image){
            // es una imagen
            ctx.drawImage(shape.image,shape.x,shape.y);
        }
    }
}
```

Capítulo 9: Tipos de media y canvas

Sección 9.1: Cargar y reproducir un vídeo en el canvas

El canvas puede utilizarse para mostrar vídeo de diversas fuentes. Este ejemplo muestra cómo cargar un vídeo como un recurso de archivo, mostrarlo y añadir un sencillo interruptor de reproducción/pausa en pantalla.

Esta pregunta de auto-respuesta de StackOverflow Cómo mostrar un vídeo utilizando la etiqueta canvas de HTML5 muestra lo siguiente código de ejemplo en acción.

Sólo una imagen

Un vídeo es sólo una imagen en lo que respecta al canvas. Se puede dibujar como cualquier imagen. La diferencia es que el vídeo puede reproducirse y tiene sonido.

Obtener el canvas y la configuración básica

```
// Se supone que sabes cómo añadir un canvas y dimensionarlo correctamente.
var canvas = document.getElementById("myCanvas"); // obtener el canvas de la página
var ctx = canvas.getContext("2d");
var videoContainer; // objeto que contiene el vídeo y la información asociada
```

Creación y carga del video

```
var video = document.createElement("video"); // crear un elemento de vídeo
video.src = "urlOffVideo.webm";
// el vídeo comenzará a cargarse.
// Como se necesita alguna información adicional colocaremos el vídeo en un
// objeto contenedor por comodidad
video.autoPlay = false; // asegúrese de que el vídeo no se reproduce automáticamente
video.loop = true; // configura el vídeo en bucle.
videoContainer = { // añadiremos propiedades según sea necesario
     video : video,
     ready : false,
};
```

A diferencia de los elementos de imagen, los vídeos no tienen que estar completamente cargados para mostrarse en el canvas. Los vídeos también ofrecen serie de eventos adicionales que pueden utilizarse para supervisar el estado del vídeo.

En este caso deseamos saber cuándo el vídeo está listo para reproducirse. `oncanplay` significa que se ha cargado una parte suficiente del vídeo para reproducir una parte, pero puede que no haya suficiente para reproducir hasta el final.

```
video.oncanplay = readyToPlayVideo; // establecer el evento en la función de reproducción que se puede encontrar a continuación
```

Alternativamente, puedes utilizar `oncanplaythrough` que se disparará cuando suficiente del vídeo se ha cargado para que pueda ser reproducirse hasta el final.

```
video.oncanplaythrough = readyToPlayVideo; // establecer el evento en la función de reproducción que se puede encontrar a continuación
```

Utiliza sólo uno de los eventos `canPlay`, no ambos.

El evento `canPlay` (equivalente a `image.onload`)

```
function readyToPlayVideo(event){ // esta es una referencia al video
      // es posible que el vídeo no coincida con el tamaño del canvas, así que busque una escala
      que se ajuste
      videoContainer.scale = Math.min(
      canvas.width / this.videoWidth,
      canvas.height / this.videoHeight);
      videoContainer.ready = true;
      // el vídeo se puede reproducir así que pásalo a la función de visualización
      requestAnimationFrame(undateCanvas);
}
```

Visualización

El vídeo no se reproducirá solo en el canvas. Hay que dibujarlo en cada nuevo fotograma. Como es difícil conocer la frecuencia de imagen exacta y cuándo se producen, lo mejor es mostrar el vídeo como si se ejecutara a 60 fps. Si es menor, entonces simplemente es renderizar el mismo fotograma dos veces. Si la velocidad de fotogramas es mayor, no hay nada que se pueda hacer para ver los fotogramas adicionales, así que simplemente los ignoramos.

El elemento de vídeo es sólo un elemento de imagen y se puede dibujar como cualquier imagen, puede escalar, rotar, desplazar el vídeo, reflejarlo, desvanecerlo, recortarlo y mostrar sólo partes, dibujarlo dos veces la segunda vez con un modo compuesto global para añadir FX como aclarar, pantalla, etc.

```
function updateCanvas(){
      ctx.clearRect(0,0,canvas.width,canvas.height); // Aunque no siempre es necesario
      // puede obtener píxeles malos de
      // vídeos anteriores tan claros para ser
      // seguro sólo si está cargado y listo
      if(videoContainer !== undefined && videoContainer.ready){
            // encontrar la parte superior izquierda del vídeo en el 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;
            // ahora solo dibuja el video del tamaño correcto
            ctx.drawImage(videoContainer.video, left, top, vidW * scale, vidH * scale);
            if(videoContainer.video.paused){ // si no se está reproduciendo muestra la pantalla de
            pausa
                  drawPayIcon();
            }
      }
      // todo listo para su visualización
      // solicitar el siguiente fotograma en 1/60 de segundo
      requestAnimationFrame(updateCanvas);
}
```

Control básico de reproducción y pausa

Ahora que tenemos el vídeo cargado y visualizado todo lo que necesitamos es el control de reproducción. Vamos a hacer como un juego de palanca de clic en la pantalla. Cuando el vídeo se está reproduciendo y el usuario hace clic, el vídeo se pausa. Cuando se pone en pausa, el clic reanuda la reproducción. Añadiremos una función para oscurecer el vídeo y dibujar un icono de reproducción (triángulo).

```
function drawPayIcon(){
      ctx.fillStyle = "black"; // oscurecer la pantalla
      ctx.globalAlpha = 0.5;
      ctx.fillRect(0,0,canvas.width,canvas.height);
      ctx.fillStyle = "#DDD"; // color del icono de juego
      ctx.globalAlpha = 0.75; // parcialmente transparente
      ctx.beginPath(); // crear la ruta para el icono
      var size = (canvas.height / 2) * 0.5; // el tamaño del icono
      ctx.moveTo(canvas.width/2 + size/2, canvas.height / 2); // empezar por la punta
      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; // restaurar alfa
}
```

Ahora el evento de pausa de reproducción

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

Resumen

Reproducir un vídeo es muy fácil utilizando el canvas, añadir efectos en tiempo real también es sencillo. No obstante, hay algunas limitaciones en cuanto a formatos, forma de jugar y búsqueda. MDN HTMLMediaElement es el lugar para obtener la referencia completa a el objeto de vídeo.

Una vez dibujada la imagen en el canvas, puedes utilizar `ctx.getImageData` para acceder a los píxeles que contiene. También puedes utilizar `canvas.toDataURL` para capturar una imagen fija y descargarla. (Sólo si el vídeo procede de una fuente de confianza y no contamine el canvas).

Tenga en cuenta que, si el vídeo tiene sonido, al reproducirlo también se reproducirá el sonido.

Feliz vídeo.

Sección 9.2: Capturar canvas y guardar como vídeo webM

Creación de un vídeo WebM a partir de fotogramas del canvas y reproducción en el canvas, o carga, o descarga.

Ejemplo de canvas de captura y reproducción

```
name = "CanvasCapture"; // Se coloca en los campos Mux y Write Application Name de la cabecera
WebM
quality = 0.7; // buena calidad 1 Mejor < 0,7 de aceptable a pobre
fps = 30; // He probado todo tipo de velocidades de fotogramas y todos parecen funcionar
// Haga algunas pruebas para entrenar lo que su máquina puede manejar ya que hay mucha variación
entre las máquinas.
var video = new Groover.Video(fps,quality,name)
function capture(){
      if(video.timecode < 5000){ // 5 segundos
            setTimeout(capture,video.frameDelay);
      }else{
            var videoElement = document.createElement("video");
            videoElement.src = URL.createObjectURL(video.toBlob());
            document.body.appendChild(videoElement);
            video = undefined; // DeReference ya que consume mucha memoria.
            return;
      }
      // el primer fotograma establece el tamaño del vídeo
      video.addFrame(canvas); // Añadir el marco del canvas actual
}
capture(); // iniciar captura
```

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

Esto fue inspirado por Wammy, pero es una reescritura completa con codificar a medida que la metodología, lo que reduce en gran medida la memoria necesaria durante la captura. Puede capturar más de 30 segundos mejores los datos y el manejo de algoritmos.

Nota: Los fotogramas se codifican en imágenes webP. Sólo Chrome soporta la codificación webP canvas. Para otros navegadores (Firefox y Edge) tendrá que utilizar un codificador webP de "terceros" como Libwebp Javascript La codificación de imágenes WebP a través de Javascript es lenta. (incluirá la adición de soporte para imágenes webp en bruto si son aceptadas).

El codificador webM inspirado en Whammy: A Real Time Javascript WebM

```
var Groover = (function(){
      // asegurate de que webp es compatible
      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"; // el único.
      const frameMimeType = 'image/webp'; // no puede ser otro
      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){ // escribe int
            var parts = [];
            while(num > 0){ parts.push(num & 0xff); num = num >> 8; }
                  return new Uint8Array(parts.reverse());
            },
            str : function(str){ // escribe 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){ // no pude encontrar todos los detalles
                  if(num < 128){ // el número va precedido de un bit (1000 está en el byte 0100
                  dos, 0010 tres, etc.)
                        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 = { // nombres y valores de cabecera
            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'; //Cabecera de fotograma clave VP8 0x9d012a
const videoDataPos = 1; // posición de datos de la cabecera de datos de la trama
const defaultDelay = dataTypes.double2Str(1000/25);
const header = [ // estructura de las cabeceras/chunks de webM, como quiera que se
llamen.
        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, // siempre 0
                                ids.Language, 'und', // undefined Creo que esto significa
                                ids.CodecID, 'V_VP8', // Creo que esto no debe cambiar
                                ids.CodecName, 'VP8', // Creo que esto no debe cambiar
                                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)}
            ]}
        ); // Corregido el error con 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);
```

```
                    // el valor tendrá el bit superior activado (bit 32), por lo que no será una
                    simple operación bit a bit
                    // Advertencia little endian (No funcionará en sistemas big endian)
                    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,
     }
})()
```

Aquí una explicación de que significa Endian.

Sección 9.3: Dibujar una imagen SVG

Para dibujar una imagen vectorial SVG, el funcionamiento no difiere del de una imagen rasterizada:

Primero debe cargar la imagen SVG en un elemento `HTMLImage` y, a continuación, utilizar el método `drawImage()`.

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

Las imágenes SVG tienen algunas ventajas sobre las rasterizadas, ya que no perderás calidad, sea cual sea la escala a la que la dibujes en tu canvas. Pero cuidado, también puede ser un poco más lento que dibujar una imagen rasterizada.

Sin embargo, las imágenes SVG tienen más restricciones que las rasterizadas.

- **Por motivos de seguridad, no se puede cargar contenido externo desde una imagen SVG referenciada en un `HTMLImageElement(<img>)`.**
  Sin hoja de estilo externa, sin imagen externa referenciada en elementos `SVGImage` (`<image/>`), sin filtro externo o elemento vinculado mediante el atributo `xlink:href` **`<use`** `xlink:href="anImage.SVG#anElement"` **`/>`** o el método de atributo `funcIRI(url())`, etc.
  Además, las hojas de estilo añadidas en el documento principal no tendrán ningún efecto en el documento SVG una vez que se haga referencia a ellas en un elemento `HTMLImage`.
  Por último, no se ejecutará ningún script dentro de la imagen SVG.
  Solución: tendrá que añadir todos los elementos externos dentro del propio SVG antes de hacer referencia al elemento `HTMLImage`. (En el caso de imágenes o fuentes, deberá añadir una versión `dataURI` de los recursos externos).

- **Los atributos de anchura y altura del elemento raíz (`<svg>`) deben tener un valor absoluto.**
  Si utiliza una longitud relativa (por ejemplo, `%`), el navegador no podrá saber a qué es relativa. Algunos navegadores (Blink) intentarán adivinarlo, pero la mayoría simplemente ignorará tu imagen y no dibujará nada, sin ninguna advertencia.

- **Algunos navegadores mancharán el canvas cuando se haya dibujado en él una imagen SVG.**
  En concreto, Internet-Explorer < Edge en cualquier caso, y Safari 9 cuando un **`<foreignObject>`** está presente en la imagen SVG.

Sección 9.4: Cargar y visualizar una imagen

Para cargar una imagen y colocarla en el canvas.

```
var image = new Image(); // ver nota sobre la creación de una imagen
      image.src = "imageURL";
      image.onload = function(){
      ctx.drawImage(this,0,0);
}
```

Crear una imagen

Hay varias formas de crear una imagen.

- `new Image()`
- `document.createElement("img")`
- **`<img`** `src = 'imageUrl' id='myImage'` **`>`** Como parte del cuerpo HTML y recuperado con `document.getElementById('myImage')`

La imagen es un `HTMLImageElement`

Propiedad image.src

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

- `image.src = http://my.domain.com/images/myImage.jpg`
- `image.src = ""` *

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

Observaciones sobre la carga y los errores

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

Si obtiene una imagen de la página (por ejemplo `document.getElementById("myImage")`) y se establece su src puede o no se haya cargado. Puedes comprobar el estado de la imagen con `HTMLImageElement.complete` que será `true` si está completa. Esto no significa que la imagen se haya cargado, significa que o bien

- ha cargado
- hubo un error
- la propiedad src no se ha establecido y es igual a la cadena de caracteres vacía `""`

Si la imagen procede de una fuente poco fiable y puede que no sea accesible por diversas razones, generará un evento de error evento. Cuando esto ocurra, la imagen estará en un estado roto. Si intenta dibujarlo en el canvas, se producirá el siguiente error arrojará el siguiente error.

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

Suministrando el evento `image.onerror = myImgErrorHandler` puedes tomar las acciones apropiadas para prevenir errores.

Capítulo 10: Animación

Sección 10.1: Utiliza requestAnimationFrame() NO setInterval() para los bucles de animación

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

- El código de animación se sincroniza con la actualización de la pantalla para mayor eficacia. El código de borrado + redibujado se programa, pero no se ejecuta inmediatamente. El navegador ejecutará el código borrado + redibujado sólo cuando la pantalla esté lista para actualizarse. Esta sincronización con el ciclo de actualización aumenta el rendimiento de la animación, ya que proporciona al código el máximo tiempo disponible para completarse.
- Cada bucle siempre se completa antes de que se permita el inicio de otro bucle. Esto evita el "tearing", en el que el usuario ve una versión incompleta del dibujo. El ojo nota especialmente el “tearing” y se distrae cuando el “tearing” ocurre. Evitar el desgarro hace que la animación parezca más fluida y coherente.
- La animación se detiene automáticamente cuando el usuario cambia a otra pestaña del navegador. Esto ahorra energía en dispositivos móviles porque el dispositivo no gasta energía en calcular una animación que el usuario no puede ver en ese momento.

La mayoría de pantallas de los dispositivos se refrescarán unas 60 veces por segundo, por lo que `requestAnimationFrame` puede redibujar continuamente a unos 60 "frames" por segundo. El ojo ve el movimiento a 20-30 fotogramas por segundo, por lo que `requestAnimationFrame` puede crear fácilmente la ilusión de movimiento.

Observa que `requestAnimationFrame` se recupera al final de cada `animateCircle`. Esto se debe a que cada `requestAnimatonFrame` sólo solicita una única ejecución 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(){
                        // variables relacionadas con canvas
                        var canvas=document.getElementById("canvas");
                        var ctx=canvas.getContext("2d");
                        var cw=canvas.width;
                        var ch=canvas.height;
                        // iniciar la animación
                        requestAnimationFrame(animate);
                        function animate(currentTime){
                              // dibujar un círculo aleatorio completo
                              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);
                        }
                  }); // fin $(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.

Sección 10.2: Animar una imagen en el canvas

Este ejemplo carga y anima una imagen en el canvas.

¡Consejo importante! Asegúrate de dar tiempo a tu 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(){
                        // variables relacionadas con canvas
                        var canvas=document.getElementById("canvas");
                        var ctx=canvas.getContext("2d");
                        var cw=canvas.width;
                        var ch=canvas.height;
                        // variables relacionadas con la animación
                        var minX=20; // Mantener la animación de la imagen
                        var maxX=250; // entre minX y maxX
                        var x=minX; // La coordenada X actual
                        var speedX=1; // La imagen se moverá a 1px por bucle
                        var direction=1; // La dirección de la imagen: 1==hacia la derecha, -1==hacia
                        la izquierda
                        var y=20; // La coordenada Y
                        // Cargar una nueva imagen
                        // ¡¡¡IMPORTANTE!!! ¡Debes darle tiempo a la imagen para que cargue usando
                        img.onload!
                        var img=new Image();
                        img.onload=start;
                        img.src="https://dl.dropboxusercontent.com/u/139992952/stackoverflow/sun.pm
                        g";
                        function start(){
                              // la imagen está completamente cargada así que empieza a animar
                              requestAnimationFrame(animate);
                        }
                        function animate(time){
                              // limpiar el canvas
                              ctx.clearRect(0,0,cw,ch);
                              // dibujar
                              ctx.drawImage(img,x,y);
                              // actualizar
                              x += speedX * direction;
                              // mantener "x" dentro de min & max
                              if(x<minX){
                                    x=minX; direction*=-1;
                              }
                              if(x>maxX){
                                    x=maxX; direction*=-1;
                              }
                              // solicitar otro bucle de animación
                              requestAnimationFrame(animate);
                        }
                  }); // fin $(function(){});
            </script>
      </head>
      <body>
            <canvas id="canvas" width=512 height=512></canvas>
      </body>
</html>
```

Sección 10.3: Establecer la velocidad de fotogramas mediante requestAnimationFrame

Usando `requestAnimationFrame` puede que en algunos sistemas se actualice a más frames por segundo que los 60fps. 60fps es la velocidad por defecto si el renderizado puede seguir el ritmo. Algunos sistemas funcionarán a 120 fps, quizá más.

Si utiliza el siguiente método, sólo debe utilizar frecuencias de cuadro que sean divisiones enteras de 60, de modo que `(60 / FRAMES_PER_SECOND) % 1 === 0` es `true`, u obtendrá frecuencias de cuadro inconsistentes.

```
const FRAMES_PER_SECOND = 30; // Los valores válidos son 60,30,20,15,10...
// establece el tiempo mim para renderizar el siguiente fotograma
const FRAME_MIN_TIME = (1000/60) * (60 / FRAMES_PER_SECOND) - (1000/60) * 0.5;
var lastFrameTime = 0; // la última hora del fotograma
function update(time){
      if(time-lastFrameTime < FRAME_MIN_TIME){ // omitir la trama si la llamada es demasiado
      temprana
            requestAnimationFrame(update);
            return; // devolver ya que no hay nada que hacer
      }
      lastFrameTime = time; // recordar la hora del fotograma renderizado
      // renderizar el marco
      requestAnimationFrame(update); // obtener el próximo frame
}
requestAnimationFrame(update); // iniciar animación
```

Sección 10.4: Alivio con las ecuaciones de Robert Penners

Una flexibilización hace que alguna **variable** cambie de forma **irregular** a lo largo de una **duración**.

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

- una coordenada X,
- el ancho de un rectángulo,
- un ángulo de giro,
- el componente rojo de un color R,G,B.
- cualquier cosa que pueda expresarse como un número.

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

- un periodo de tiempo,
- una distancia por recorrer,
- una cantidad de bucles de animación a ejecutar,
- cualquier cosa que pueda expresarse.

"irregular" significa que la variable progresa de los valores iniciales a los finales de forma irregular:

- más rápido al principio y más lento al final, o viceversa,
- sobrepasa el final, pero retrocede hasta el final cuando termina la duración,
- avanza/retrocede elásticamente de forma repetida durante la duración,
- "rebota" en el final mientras se detiene al terminar la duración.

Atribución: Robert Penner ha creado el "patrón oro" de las funciones de flexibilización.

Cita: https://github.com/danro/jquery-easing/blob/master/jquery.easing.js

```
// t: tiempo transcurrido dentro de la duración (currentTime-startTime),
// b: valor inicial,
// c: cambio total desde el valor inicial (endingValue-startingValue),
// d: duración total
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

```
// incluir el objeto Easings de arriba
var Easings = ...
// Demostración
var startTime;
var beginningValue=50; // coordenada x inicial
var endingValue=450; // coordenada x final
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);
    }
}
```

Sección 10.5: Animar en un intervalo especificado (añadir un nuevo rectángulo cada 1 segundo)

Este ejemplo añade una 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(){
                        // variables relacionadas con canvas
                        var canvas=document.getElementById("canvas");
                        var ctx=canvas.getContext("2d");
                        var cw=canvas.width;
                        var ch=canvas.height;
                        // variables de intervalo de animación
                        var nextTime=0; // la siguiente animación comienza en "nextTime"
                        var duration=1000; // ejecutar animación cada 1000ms
                        var x=20; // la X donde se dibuja el siguiente rectangulo
                        // iniciar la animación
                        requestAnimationFrame(animate);
                        function animate(currentTime){
                              // esperar a que se produzca nextTime
                              if(currentTime<nextTime){
                                    // solicitar otro bucle de animación
                                    requestAnimationFrame(animate);
                                    // el tiempo no ha transcurrido, así que devuelve
                                    return;
                              }
                              // establece nextTime
                              nextTime=currentTime+duration;
                              // añadir otro rectángulo cada 1000ms
                              ctx.fillStyle='#'+Math.floor(Math.random()*16777215).toString(16);
                              ctx.fillRect(x,30,30,30);
                              // actualizar la posición X para el siguiente rectángulo
                              x+=30;
                              // solicitar otro bucle de animación
                              requestAnimationFrame(animate);
                        }
                  }); // fin $(function(){});
            </script>
      </head>
      <body>
            <canvas id="canvas" width=512 height=512></canvas>
      </body>
</html>
```

Sección 10.6: Animar a una hora determinada (un reloj animado)

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

Código anotado:

```
<!doctype html>
<html>
      <head>
            <style>
                  body{ background-color:white; }
                  #canvas{border:1px solid red; }
            </style>
            <script>
                  window.onload=(function(){
                        // variables relacionadas con canvas
                        var canvas=document.getElementById("canvas");
                        var ctx=canvas.getContext("2d");
                        var cw=canvas.width;
                        var ch=canvas.height;
                        // estilo canvas para el reloj
                        ctx.strokeStyle='lightgray';
                        ctx.fillStyle='skyblue';
                        ctx.lineWidth=5;
                        // caché de valores utilizados con frecuencia
                        var PI=Math.PI;
                        var fullCircle=PI*2;
                        var sa=-PI/2; // == el ángulo de las 12 horas en context.arc
                        // iniciar la animación
                        requestAnimationFrame(animate);
                        function animate(currentTime){
                              // obtener el valor actual de los segundos del reloj del Sistema
                              var date=new Date();
                              var seconds=date.getSeconds();
                              // limpiar el canvas
                              ctx.clearRect(0,0,cw,ch);
                              // dibujar un círculo completo (== la esfera del reloj);
                              ctx.beginPath();
                              ctx.moveTo(100,100);
                              ctx.arc(100,100,75,0,fullCircle);
                              ctx.stroke();
                              // dibujar una aguja que represente el valor actual de los segundos
                              ctx.beginPath();
                              ctx.moveTo(100,100);
                              ctx.arc(100,100,75,sa,sa+fullCircle*seconds/60);
                              ctx.fill();
                              // solicitar otro bucle de animación
                              requestAnimationFrame(animate);
                        }
                  }); // fin $(function(){});
            </script>
      </head>
      <body>
            <canvas id="canvas" width=512 height=512></canvas>
      </body>
</html>
```

Sección 10.7: No dibujes animaciones en tus manejadores de eventos (una simple aplicación de sketch)

Durante el `mousemove`, salen 30 eventos de ratón por segundo. Puede que no sea capaz de redibujar los dibujos a 30 veces por segundo. Incluso si puede, probablemente está desperdiciando potencia de cálculo dibujando cuando el navegador no está listo para dibujar (desperdicio == a través de ciclos de refresco de pantalla).

Por lo tanto, tiene sentido separar los eventos de entrada de los usuarios (como el movimiento del ratón) del dibujo de las animaciones.

- En los manejadores de eventos, guarde todas las variables de eventos que controlan dónde se posicionan los dibujos en el canvas. Pero no dibujes nada.
- En un bucle `requestAnimationFrame`, renderiza todos los dibujos en el Canvas usando la información guardada.

Al no dibujar en los manejadores de eventos, no estás forzando a Canvas a tratar de refrescar dibujos complejos a la velocidad de los eventos del ratón.

Haciendo todo el dibujo en `requestAnimationFrame` obtienes todos los beneficios descritos aquí Usa `requestanimationFrame` y 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);
                        }
                        // variables del canvas
                        var canvas=document.getElementById("canvas");
                        var ctx=canvas.getContext("2d");
                        var cw=canvas.width;
                        var ch=canvas.height;
                        // establecer el estilo del canvas
                        ctx.strokeStyle='skyblue';
                        ctx.lineJoint='round';
                        ctx.lineCap='round';
                        ctx.lineWidth=6;
                        // gestión del desplazamiento y cambio de tamaño de las ventanas
                        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();
                        }
                        // variables para guardar los puntos creados durante la gestión del
                        movimiento del ratón
                        var points=[];
                        var lastLength=0;
                        // iniciar el bucle de animación
                        requestAnimationFrame(draw);
                        canvas.onmousemove=function(e){
                              handleMouseMove(e);
                        }
                        function handleMouseMove(e){
                              // indica al navegador que estamos gestionando este evento
                              e.preventDefault();
                              e.stopPropagation();
                              // obtener la posición del ratón
                              mouseX=parseInt(e.clientX-offsetX);
```

```
                    mouseY=parseInt(e.clientY-offsetY);
                    // guarda la posición del ratón en el array points[] pero no dibujes
                    nada
                    points.push({x:mouseX,y:mouseY});
                }
                function draw(){
                    // ¿No hay puntos adicionales? Solicita otro frame y devuelvelo
                    var length=points.length;
                    if(length==lastLength){
                        requestAnimationFrame(draw);
                        return;
                    }
                    // dibujar los puntos adicionales
                    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();
                    // solicitar otro bucle de animación
                    requestAnimationFrame(draw);
                }
            }); // fin window.onload
        </script>
    </head>
    <body>
        <h4>Move mouse over Canvas to sketch</h4>
        <canvas id="canvas" width=512 height=512></canvas>
    </body>
</html>
```

Sección 10.8: Animación simple con contexto 2D y requestAnimationFrame

Este ejemplo le mostrará cómo crear una animación sencilla utilizando el canvas y el contexto 2D. Se asume que sabes cómo crear y añadir un canvas al DOM y obtener el contexto.

```
// thieste ejemplo asume que ctx y canvas han sido creados
const textToDisplay = "This is an example that uses the canvas to animate some text.";
const textStyle = "white";
const BGStyle = "black"; // estilo de fondo
const textSpeed = 0.2; // en píxeles por milisegundo
const textHorMargin = 8; // tener el texto un poco fuera del canvas
ctx.font = Math.floor(canvas.height * 0.8) + "px arial"; // ajustar el tamaño de la fuente al 80%
de la altura del canvas
var textWidth = ctx.measureText(textToDisplay).width; // obtener el ancho del texto
var totalTextSize = (canvas.width + textHorMargin * 2 + textWidth);
ctx.textBaseline = "middle"; // no poner el texto en el centro vertical
ctx.textAlign = "left"; // alinear hacia la izquierda
var textX = canvas.width + 8; // empezar con el texto fuera de la pantalla a la derecha
var textOffset = 0; // cuánto se ha desplazado el texto
var startTime;
// esta función se llama una vez por fotograma, es decir, aproximadamente 16,66 ms (60fps)
function update(time){ // el tiempo es pasado por requestAnimationFrame
     if(startTime === undefined){ // obtener una referencia para la hora de inicio si éste es el
     primer fotograma
          startTime = time;
     }
     ctx.fillStyle = BGStyle;
     ctx.fillRect(0, 0, canvas.width, canvas.height); // limpiar el canvas dibujando sobre él
     textOffset = ((time - startTime) * textSpeed) % (totalTextSize); // mover el texto a la
     izquierda
     ctx.fillStyle = textStyle; // establecer el estilo del texto
     ctx.fillText(textToDisplay, textX - textOffset, canvas.height / 2); // renderizar el texto
     requestAnimationFrame(update);// todo hecho pedir el siguiente fotograma
}
requestAnimationFrame(update);// para empezar a solicitar el primer fotograma
```

Una demostración de este ejemplo en JSFiddle.

Sección 10.9: Animar desde [x0,y0] hasta [x1,y1]

Utiliza vectores para calcular incrementales `[x,y]` desde `[startX,startY]` hasta `[endX,endY]`.

```
// dx es la distancia total que hay que recorrer en la dirección X
var dx = endX - startX;
// dy es la distancia total a recorrer en la dirección Y
var dy = endY - startY;
// utilizar un pct (porcentaje) para recorrer las distancias totales
// empezar en 0% que == el punto de partida
// final al 100% que == entonces punto final
var pct=0;
// utiliza dx & dy para calcular dónde está la corriente [x,y] en un pct dado
var x = startX + dx * pct/100;
var y = startY + dx * pct/100;
```

Código de ejemplo:

```
// variables del canvas
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;
// estilos de canvas
ctx.strokeStyle='skyblue';
ctx.fillStyle='blue';
// variables de animacion
var pct=101;
var startX=20;
var startY=50;
var endX=225;
var endY=100;
var dx=endX-startX;
var dy=endY-startY;
// iniciar el bucle de animación
requestAnimationFrame(animate);
// escuchar los eventos del ratón
window.onmousedown=(function(e){handleMouseDown(e);});
window.onmouseup=(function(e){handleMouseUp(e);});
// bucle en constante funcionamiento
// animará el punto desde startX,startY hasta endX,endY
function animate(time){
     // demostracion: volver a ejecutar la animación
     if(++pct>100){
          pct=0;
     }
     // actualizar
     x=startX+dx*pct/100;
     y=startY+dy*pct/100;
     // dibujar
     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()
     // solicitar otro bucle de animación
     requestAnimationFrame(animate);
}
```

Capítulo 11: Colisiones e intersecciones

Sección 11.1: ¿Colisionan 2 círculos?

```
// objetos círculo: { x:, y:, radio: }
// devuelve true si los 2 círculos colisionan
// c1 y c2 son los círculos definidos anteriormente
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);
}
```

Sección 11.2: ¿Colisionan 2 rectángulos?

```
// objetos rectángulo { x:, y:, width:, height: }
// devuelve true si los 2 rectángulos colisionan
// r1 y r2 son rectángulos como los definidos anteriormente
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
     );
}
```

Sección 11.3: ¿Colisionan un círculo y un rectángulo?

```
// objeto rectángulo: { x:, y:, width:, height: }
// objecto círculo: { x:, y:, radius: }
// devuelve true si el rectángulo y el círculo colisionan
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);
}
```

Sección 11.4: ¿Se interceptan 2 segmentos de línea?

La función de este ejemplo devuelve `true` si dos segmentos de recta se cruzan y `false` en caso contrario.

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

```
// objeto punto: {x:, y:}
// p0 y p1 forman un segmento, p2 y p3 forman el segundo segmento
// Devuelve true si los segmentos de las líneas se interceptan
var lineSegmentsIntercept = (function(){ // como singleton para poder utilizar el cierre
      var v1, v2, v3, cross, u1, u2; // variable de trabajo están cerradas por lo que no necesitan
      creación
      // cada vez que se llama a la función. Esto aumenta considerablemente el rendimiento.
      v1 = {x : null, y : null}; // línea p0, p1 como vector
      v2 = {x : null, y : null}; // línea p2, p3 como vector
      v3 = {x : null, y : null}; // la línea de p0 a p2 como vector
      function lineSegmentsIntercept (p0, p1, p2, p3) {
            v1.x = p1.x - p0.x; // línea p0, p1 como vector
            v1.y = p1.y - p0.y;
            v2.x = p3.x - p2.x; // línea p2, p3 como vector
            v2.y = p3.y - p2.y;
            if((cross = v1.x * v2.y - v1.y * v2.x) === 0){ // cross prod 0 si las líneas son
            paralelas
                  return false; // no interceptar
            }
            v3 = {x : p0.x - p2.x, y : p0.y - p2.y}; // la línea de p0 a p2 como vector
            u2 = (v1.x * v3.y - v1.y * v3.x) / cross; // obtener la distancia unitaria a lo largo
            de la línea p2 p3
            // código punto B
            if (u2 >= 0 && u2 <= 1){ // es el intercepto en la línea p2, p3
                  u1 = (v2.x * v3.y - v2.y * v3.x) / cross; // obtener la distancia unitaria en la
                  línea p0, p1;
                  // código punto A
                  return (u1 >= 0 && u1 <= 1); // devuelve true si está en línea si no false.
                  // código punto A fin
            }
            return false; // no interceptar;
            // código punto B fin
      }
      return lineSegmentsIntercept; // función de retorno con cierre para optimización.
})();
```

Ejemplo de uso

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

El ejemplo se puede modificar fácilmente para devolver el punto de intercepción. Sustituir el código entre el `punto A` y el `final de A` por

```
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 `punto de código B` y el `final B` por

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

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

Sección 11.5: ¿Colisionan una línea y un círculo?

```
// de [x0,y0] a [x1,y1] definen un segmento de línea
// [cx,cy] es el punto central del círculo, cr es el radio del círculo
function isCircleSegmentColliding(x0,y0,x1,y1,cx,cy,cr){
      // calc distancia delta: punto de origen a inicio de línea
      var dx=cx-x0;
      var dy=cy-y0;
      // calc distancia delta: inicio a fin de línea
      var dxx=x1-x0;
      var dyy=y1-y0;
      // Calc posición en línea normalizada entre 0.00 y 1.00
      // == producto punto dividido por las distancias de línea delta al cuadrado
      var t=(dx*dxx+dy*dyy)/(dxx*dxx+dyy*dyy);
      // calc pt más cercano en línea
      var x=x0+dxx*t;
      var y=y0+dyy*t;
      // resultados de la pinza al estar en el segmento
      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 );
}
```

Sección 11.6: ¿Colisionan una línea y el rectángulo?

```
// var rect={x:,y:,width:,height:};
// var line={x1:,y1:,x2:,y2:};
// Obtener el punto de intersección del segmento de línea y el rectángulo (si existe)
function lineRectCollide(line,rect){
      // p=punto inicial de la línea, p2=punto final de la línea
      var p={x:line.x1,y:line.y1};
      var p2={x:line.x2,y:line.y2};
      // línea recta superior
      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; }
      // línea recta derecha
      var q=q2;
      var q2={x:rect.x+rect.width,y:rect.y+rect.height};
      if(lineSegmentsCollide(p,p2,q,q2)){ return true; }
      // línea recta inferior
      var q=q2;
      var q2={x:rect.x,y:rect.y+rect.height};
      if(lineSegmentsCollide(p,p2,q,q2)){ return true; }
      // línea recta izquierda
      var q=q2;
      var q2={x:rect.x,y:rect.y};
      if(lineSegmentsCollide(p,p2,q,q2)){ return true; }
      // no se cruza con ninguno de los 4 lados rectos
      return(false);
}
// objeto punto: {x:, y:}
// p0 y p1 forman un segmento, p2 y p3 forman el segundo segmento
// Obtener el punto de intersección de 2 segmentos de línea (si existe)
// Atribución: 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);
      // Comprobar si coincide
      // Si el denominador y el numerador de ua y ub son 0, las dos rectas son coincidentes.
      if(unknownA==0 && unknownB==0 && denominator==0){return(null);}
      // Comprobar si es paralelo
      // Si el denominador de las ecuaciones de ua y ub es 0 entonces las dos rectas son paralelas.
      if (denominator == 0) return null;
      // comprobar si los segmentos de línea colisionan
      unknownA /= denominator;
      unknownB /= denominator;
      var isIntersecting=(unknownA>=0 && unknownA<=1 && unknownB>=0 && unknownB<=1)
      return(isIntersecting);
}
```

Sección 11.7: ¿Colisionan 2 polígonos convexos?

Utiliza el Teorema del Eje Separador para determinar si 2 polígonos convexos se intersecan.

LOS POLÍGONOS DEBEN SER CONVEXOS

Atribución: Markus Jarderot @ ¿Cómo comprobar la intersección entre 2 rectángulos rotados?

```
// los objetos polígono son un array de vértices que forman el polígono
// var polygon1=[{x:100,y:100},{x:150,y:150},{x:50,y:150},...];
// LOS POLÍGONOS DEBEN SER CONVEXOS
// devuelve true si los 2 polígonos colisionan
function convexPolygonsCollide(a, b){
     var polygons = [a, b];
     var minA, maxA, projected, i, i1, j, minB, maxB;
     for (i = 0; i < polygons.length; i++) {
          // para cada polígono, mira cada arista del polígono y determina si separa las dos
          formas
          var polygon = polygons[i];
          for (i1 = 0; i1 < polygon.length; i1++) {
               // coge 2 vértices para crear una arista
               var i2 = (i1 + 1) % polygon.length;
               var p1 = polygon[i1];
               var p2 = polygon[i2];
               // hallar la línea perpendicular a esta arista
               var normal = { x: p2.y - p1.y, y: p1.x - p2.x };
               minA = maxA = undefined;
               // para cada vértice de la primera forma, proyéctalo sobre la línea perpendicular
               a la arista y anota el mínimo y el máximo de estos valores
               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;
                    }
               }
               // para cada vértice de la segunda forma, proyéctalo sobre la línea perpendicular
               a la arista y anota el mínimo y el máximo de estos valores
               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;
                    }
               }
               // si no hay solapamiento entre los proyectos, la arista que estamos mirando
               separa los dos polígonos, y sabemos que no hay solapamiento
               if (maxA < minB || maxB < minA) {
                    return false;
               }
          }
     }
     return true;
};
```

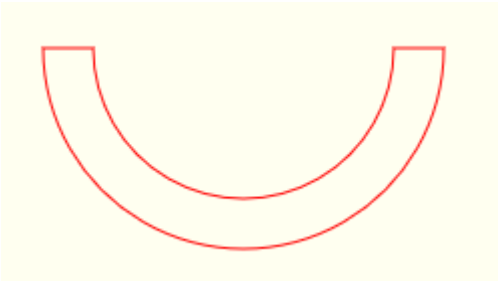
Sección 11.8: ¿Colisionan 2 polígonos? (se permiten tanto polígonos cóncavos como convexos)

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

```
// los objetos polígono son un array de vértices que forman el polígono
// var polygon1=[{x:100,y:100},{x:150,y:150},{x:50,y:150},...];
// Los polígonos pueden ser tanto cóncavos como convexos
// devuelve true si los 2 polígonos colisionan
function polygonsCollide(p1,p2){
     // convertir vértices en puntos de línea
     var lines1=verticesToLinePoints(p1);
     var lines2=verticesToLinePoints(p2);
     // prueba de intersecciones entre cada lado de poly1 y cada lado de poly2
     for(i=0; i<lines1.length; i++){
          for(j=0; j<lines2.length; j++){
               // comprobar si los lados se cruzan
               var p0=lines1[i][0];
               var p1=lines1[i][1];
               var p2=lines2[j][0];
               var p3=lines2[j][1];
               // encontrado una intersección -- los polígonos colisionan
               if(lineSegmentsCollide(p0,p1,p2,p3)){
                    return(true);
               }
          }
     }
     // ninguno de los lados se cruza
     return(false);
}
// helper: convertir vértices en puntos de línea
function verticesToLinePoints(p){
     // asegúrese de que los polys son de cierre automático
     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: probar intersecciones de líneas
// objeto punto: {x:, y:}
// p0 y p1 forman un segmento, p2 y p3 forman el segundo segment
// Obtener el punto de intersección de 2 segmentos de línea (si existe)
// Atribución: 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);
     // Comprobar si coincide
     // Si el denominador y el numerador de la ua y la ub son 0, entonces las dos rectas son
     coincidentes.
     if(unknownA==0 && unknownB==0 && denominator==0){return(null);}
     // Test if Parallel
     // Si el denominador de las ecuaciones de ua y ub es 0 entonces las dos rectas son paralelas.
     if (denominator == 0) return null;
     // comprobar si los segmentos de línea colisionan
     unknownA /= denominator;
     unknownB /= denominator;
     var isIntersecting=(unknownA>=0 && unknownA<=1 && unknownB>=0 && unknownB<=1)
     return(isIntersecting);
}
```

Sección 11.9: ¿Un punto X,Y está dentro de un arco?

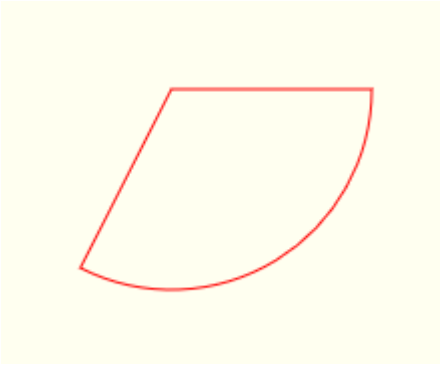
Comprueba si el punto `[x,y]` está dentro de un arco cerrado.



```
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);
}
```

Sección 11.10: ¿Es un punto X,Y dentro de una cuña?

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



```
// objetos en forma de cuña: {cx:,cy:,radius:,startAngle:,endAngle:}
// var wedge={
//    cx:150, cy:150, // punto central
//    radius:100,
//    startAngle:0, endAngle:Math.PI
// }
// Devuelve true si el punto x,y está dentro de la cuña cerrada
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);
}
```

Sección 11.11: ¿Está un punto X,Y dentro de un círculo?

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

```
// objetos circulares: {cx:,cy:,radius:,startAngle:,endAngle:}
// var circle={
//    cx:150, cy:150, // centerpoint
//    radius:100,
// }
// Devuelve true si el punto x,y está dentro del círculo
function isPointInCircle(x,y,circle){
     var dx=x-circle.cx;
     var dy=y-circle.cy;
     return(dx*dx+dy*dy<circle.radius*circle.radius);
}
```

Sección 11.12: ¿Está un punto X,Y dentro de un rectángulo?

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

```
// objetos rectángulo: {x:, y:, width:, height: }
// var rect={x:10, y:15, width:25, height:20}
// Devuelve true si el punto x,y está dentro del rectángulo
function isPointInRectangle(x,y,rect){
     return(x>rect.x && x<rect.x+rect.width && y>rect.y && y<rect.y+rect.height);
}
```

Capítulo 12: Limpiar la pantalla

Sección 12.1: Rectángulos

Puedes utilizar el método `clearRect` para borrar cualquier sección rectangular del canvas.

```
// Limpiar todo el canvas
ctx.clearRect(0, 0, canvas.width, canvas.height);
```

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

Para solucionar este problema, es posible restablecer la matriz de transformación antes de borrar el canvas.

```
ctx.save(); // Guardar el estado actual del contexto
ctx.setTransform(1, 0, 0, 1, 0, 0); // Restablecer la matriz de transformación
ctx.clearRect(0, 0, canvas.width, canvas.height); // Limpiar el canvas
ctx.restore(); // Revertir el estado del contexto, incluida la matriz de transformación
```

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

Sección 12.2: Limpiar el canvas con degradado

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

Para limpiar con un degradado

```
// crear el degradado de fondo una vez
var bgGrad = ctx.createLinearGradient(0,0,0,canvas.height);
bgGrad.addColorStop(0,"#0FF");
bgGrad.addColorStop(1,"#08F");
// Cada vez que necesites limpiar el canvas
ctx.fillStyle = bgGrad;
ctx.fillRect(0,0,canvas.width,canvas.height);
```

Esto es aproximadamente la mitad de rápido `0.008ms` como `clearRect` pero con `0.004ms` no debe afectar negativamente a ninguna animación en tiempo real. (Los tiempos variarán considerablemente en función del dispositivo, la resolución, el navegador y la configuración del navegador. Los tiempos son sólo para comparación)

Sección 12.3: Limpiar canvas mediante operación compuesta

Limpiar el canvas mediante la operación de composición. Esto limpiará el canvas 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.

Sección 12.4: Datos de imagen sin procesar

Es posible escribir directamente en los datos de la imagen renderizada utilizando `putImageData`. Creando nuevos datos de imagen y asignarlos al canvas, despejará 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 renderizada.

Sección 12.5: Formas complejas

Es posible borrar regiones con formas complejas cambiando la propiedad `globalCompositeOperation`.

```
// Todos los píxeles que se dibujen serán transparentes
ctx.globalCompositeOperation = 'destination-out';
// Limpiar una sección triangular
ctx.globalAlpha = 1; // asegurar de que alfa es 1
ctx.fillStyle = '#000'; // asegurarse de que el fillStyle actual no tiene ninguna transparencia
ctx.beginPath();
ctx.moveTo(10, 0);
ctx.lineTo(0, 10);
ctx.lineTo(20, 10);
ctx.fill();
// Empezar a dibujar normalmente de nuevo
ctx.globalCompositeOperation = 'source-over';
```

Capítulo 13: Diseño adaptable

Sección 13.1: Creación de un canvas de página completa adaptable

Código de inicio para crear y eliminar un canvas de página completa que responde a eventos de cambio de tamaño a través de JavaScript.

```
var canvas; // Referencia global del canvas
var ctx; // Referencia global del contexto 2D
// Crea un canvas
function createCanvas () {
      const canvas = document.createElement("canvas");
      canvas.style.position = "absolute"; // Establecer el estilo
      canvas.style.left = "0px"; // Posición en la parte superior izquierda
      canvas.style.top = "0px";
      canvas.style.zIndex = 1;
      document.body.appendChild(canvas); // Añadir al documento
      return canvas;
}
// Cambia el tamaño del canvas. Creará un canvas si no existe
function sizeCanvas () {
      if (canvas === undefined) { // Comprobación de la referencia global del lienzo
            canvas = createCanvas(); // Crear un nuevo elemento canvas
            ctx = canvas.getContext("2d"); // Obtener el contexto 2D
      }
      canvas.width = innerWidth; // Establecer la resolución del canvas para rellenar la página
      canvas.height = innerHeight;
}
// Elimina el canvas
function removeCanvas () {
      if (canvas !== undefined) { // Asegura de que hay algo que quitar
            removeEventListener("resize", sizeCanvas); // Eliminar evento de cambio de tamaño
            document.body.removeChild(canvas); // Eliminar el canvas del DOM
            ctx = undefined; // Dereferencia al contexto
            canvas = undefined; // Dereferencia al canvas
      }
}
// Añadir el receptor de redimensionamiento
addEventListener("resize", sizeCanvas);
// Llama a sizeCanvas para crear y establecer la resolución del canvas
sizeCanvas();
// ctx y canvas ya están disponibles para su uso.
```

Si ya no necesitas el canvas puedes eliminarlo llamando a `removeCanvas()`.

Una demostración de este ejemplo en JSFiddle.

Sección 13.2: Coordenadas del ratón tras redimensionar (o desplazarse)

Las aplicaciones de canvas a menudo dependen en gran medida de la interacción del usuario con el ratón, pero cuando se cambia el tamaño de la ventana, las coordenadas del evento de ratón en las que se basa el canvas probablemente cambien porque el cambio de tamaño hace que el canvas se desplace a una posición diferente con respecto a la ventana. Por lo tanto, el diseño responsivo requiere que la posición de desplazamiento del canvas sea recalculada cuando la ventana cambia de tamaño - y también recalculada cuando la ventana se desplaza.

Este código escucha los eventos de cambio de tamaño de la ventana y recalcula los desplazamientos utilizados en los manejadores de eventos del ratón:

```
// variables que contienen la posición actual de desplazamiento del canvas con respecto a la
ventana
var offsetX,offsetY;
// una función para recalcular los desplazamientos del canvas
function reOffset(){
      var BB=canvas.getBoundingClientRect();
      offsetX=BB.left;
      offsetY=BB.top;
}
// escuchar los eventos de redimensionamiento (y desplazamiento) de la ventana y recalcular los
desplazamientos del canvas
window.onscroll=function(e){ reOffset(); }
window.onresize=function(e){ reOffset(); }
// ejemplo de uso de los desplazamientos en un controlador de ratón
function handleMouseUp(e){
      // utiliza offsetX & offsetY para obtener la posición correcta del ratón
      mouseX=parseInt(e.clientX-offsetX);
      mouseY=parseInt(e.clientY-offsetY);
      // ...
}
```

Sección 13.3: Animaciones de canvas responsivos sin cambio de tamaño eventos

Los eventos de cambio de tamaño de la ventana pueden dispararse en respuesta al movimiento del dispositivo de entrada del usuario. Cuando redimensiona un canvas se borra automáticamente y te ves obligado a volver a renderizar el contenido. En el caso de las animaciones, esto se hace en cada fotograma a través de la función de bucle principal llamada por `requestAnimationFrame`, que hace todo lo posible para mantener el renderizado sincronizado con el hardware de la pantalla.

El problema con el evento de redimensionamiento es que cuando se utiliza el ratón para redimensionar la ventana los eventos pueden dispararse muchas veces más rápido que la tasa estándar de 60fps del navegador. Cuando el evento de redimensionamiento sale, el back buffer del canvas se presenta al DOM desincronizado con el dispositivo de visualización, lo que puede provocar "shearing" y otros efectos negativos. También hay un montón de asignación y liberación de memoria innecesaria que puede afectar aún más a la animación cuando GC limpia algún tiempo después.

Evento de redimensionamiento anulado

Una forma común de hacer frente a las altas tasas de disparo del evento de cambio de tamaño es anular el evento de cambio de tamaño.

```
// Asumir que el canvas está en el ámbito de aplicación
addEventListener.("resize", debouncedResize );
// tiempo de espera de anulacion
var debounceTimeoutHandle;
// El tiempo de anulacion en ms (1/1000 de segundo)
const DEBOUNCE_TIME = 100;
// Función de redimensionamiento
function debouncedResize () {
      clearTimeout(debounceTimeoutHandle); // Borra cualquier evento de anulacion pendiente
      // Programar un cambio de tamaño del lienzo
      debounceTimeoutHandle = setTimeout(resizeCanvas, DEBOUNCE_TIME);
}
// función de redimensionamiento del canvas
function resizeCanvas () { ... resize and redraw ... }
```

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

Todavía tiene algunos problemas, el más notable es el retraso entre el cambio de tamaño y ver el canvas redimensionado. Reduciendo el tiempo de anulación mejora esta situación, pero el cambio de tamaño sigue estando desincronizado con el dispositivo de visualización. También todavía tiene el bucle principal de animación renderizando en un canvas poco apropiado.

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

Simple y el mejor redimensionamiento

Después de probar muchas formas diferentes de suavizar el cambio de tamaño del canvas, desde las absurdamente complejas hasta simplemente ignorar el problema (¿a quién le importa de todos modos?), vuelves a recurrir a un amigo de confianza.

K.I.S.S. es algo que la mayoría de los programadores deberían conocer ((**K**eep **I**t **S**imple **S**tupid) El estúpido se refiere al programador por no haberlo pensado hace años) y resulta que la mejor solución es la más sencilla de todas.

Sólo tienes que redimensionar al canvas desde dentro del bucle de animación principal. Permanece sincronizado con el dispositivo de visualización, no hay renderizado innecesario y la gestión de recursos es la mínima posible, manteniendo la frecuencia de imagen máxima. Tampoco es necesario añadir un evento de redimensionamiento a la ventana ni ninguna función de redimensionamiento adicional.

Añade el redimensionamiento donde normalmente borrarías el canvas comprobando si el tamaño del canvas coincide con el de la ventana. Si no, redimensiónalo.

```
// Asume que el elemento canvas está en el ámbito canvas
// Función de bucle principal estándar de requestAnimationFrame
function mainLoop(time) {
     // Comprobar si el tamaño del canvas coincide con el tamaño de la ventana
     if (canvas.width !== innerWidth || canvas.height !== innerHeight) {
          canvas.width = innerWidth; // redimensionar canvas
          canvas.height = innerHeight; // también limpia el canvas
     } else {
          ctx.clearRect(0, 0, canvas.width, canvas.height); // borrar si no se redimensiona
     }
     // Código de animación normal.
     requestAnimationFrame(mainLoop);
}
```

Capítulo 14: Sombras

Sección 14.1: Efecto sticker con sombras

Este código añade sombras crecientes hacia el exterior a una imagen para crear una versión "sticker" de la misma.

Notas:

- Además de ser un `ImageObject`, el argumento "`img`" también puede ser un elemento `Canvas`. Esto permite pegar tus propios dibujos personalizados. Si dibuja texto en el argumento `Canvas`, también puede pegar stickers a ese texto.
- Las imágenes totalmente opacas no tendrán efecto de pegatina porque el efecto se dibuja alrededor de grupos de píxeles opacos 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;';
// Espera siempre a que las imágenes se carguen completamente antes de intentar dibujarlas.
var img=new Image();
img.onload=start;
// pon el img.src aquí...
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);
}
```

Sección 14.2: Cómo evitar que se produzcan más sombras

Una vez activado el sombreado, cada nuevo dibujo en el canvas será sombreado.

Desactive el sombreado adicional estableciendo `context.shadowColor` en un color transparente.

```
// empezar con sombreado
context.shadowColor='black';
... renderizar algunos dibujos con sombreados ...
// desactivar el sombreado.
context.shadowColor='rgba(0,0,0,0)';
```

Sección 14.3: La sombra es computacionalmente cara - Guarda esa sombra en la caché

¡Atención! Aplique las sombras con moderación.

La aplicación del sombreado es costosa y se multiplica si se aplica dentro de un bucle de animación.

En lugar de eso, guarda en caché una versión sombreada de tu imagen (u otro dibujo):

Al principio de tu aplicación, crea una versión sombreada de tu imagen en un segundo canvas de sólo memoria:
`var memoryCanvas = document.createElement('canvas') ...`

Siempre que necesites la versión sombreada, dibuja esa imagen pre-sombreada desde el canvas en memoria al canvas 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);
// Utiliza siempre "img.onload" para dar tiempo a que la imagen se cargue completamente antes de
dibujarla en el canvas.
var img=new Image();
img.onload=start;
// Pon aquí tu propio img.src
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);
}
```

Sección 14.4: Añade profundidad visual con las sombras

El uso tradicional de las sombras es dar a los dibujos bidimensionales la ilusión de profundidad tridimensional.

Este ejemplo muestra el mismo "botón" con y sin sombreado.

```
var canvas=document.createElement("canvas");
var ctx=canvas.getContext("2d");
document.body.appendChild(canvas);
ctx.fillStyle='skyblue';
ctx.strokeStyle='lightgray';
ctx.lineWidth=5;
// sin sombra
ctx.beginPath();
ctx.arc(60,60,30,0,Math.PI*2);
ctx.closePath();
ctx.fill();
ctx.stroke();
// con sombra
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();
// detener el sombreado
ctx.shadowColor='rgba(0,0,0,0)';
```

Sección 14.5: Sombras interiores

El canvas no tiene el `inner-shadow` de CSS.

- El canvas sombreará el exterior de una forma rellena.
- El canvas sombreará tanto el interior como el exterior de una forma trazada.

Pero es fácil crear sombras interiores utilizando la composición.

Trazos con sombra interior

Para crear trazos con `inner-shadow`, utilice la composición `destination-in`, que hace que el contenido existente permanezca sólo donde el contenido existente se superpone al nuevo contenido. Se borran los contenidos existentes que no se solapan con los nuevos.

1. **Trazar una forma con sombra.** La sombra se extenderá tanto hacia fuera como hacia dentro del trazo. Debemos deshacernos de la sombra exterior, dejando sólo la sombra interior deseada.
2. **Establecer la composición `destination-in`**, lo que mantiene la sombra trazada existente sólo donde se superpone con los nuevos dibujos.
3. **Rellenar la forma.** Esto hace que el trazo y la sombra interior permanezcan mientras que la sombra exterior se borra. *¡Pues no exactamente! Como un trazo está mitad dentro y mitad fuera de la forma rellenada, la mitad exterior del trazo también se borrará. La solución consiste en duplicar el ancho de línea de `context.lineWidth` para que la mitad del trazo de tamaño doble quede dentro de la forma rellenada.*

```
var canvas=document.createElement("canvas");
var ctx=canvas.getContext("2d");
document.body.appendChild(canvas);
// dibujar una forma opaca - aquí utilizamos un rectángulo redondeado
defineRoundedRect(30,30,100,75,10);
// establecer el sombreado
ctx.shadowColor='black';
ctx.shadowBlur=10;
// trazar el rectángulo redondeado con sombreado
ctx.lineWidth=4;
ctx.stroke();
// configurar la composición para borrar todo lo que está fuera del trazo
ctx.globalCompositeOperation='destination-in';
ctx.fill();
// limpiar siempre - volver a la composición por defecto
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();
}
```

Trazado con relleno con una sombra interior

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

4. **Establece la composición `destination-over`**, lo que hace que el relleno se dibuje **bajo** la sombra interior existente.
5. **Desactive el sombreado** estableciendo `context.shadowColor` en un color transparente.
6. **Rellene la forma** con el color deseado. La forma se rellenará por debajo de la sombra interior existente.

```
var canvas=document.createElement("canvas");
var ctx=canvas.getContext("2d");
document.body.appendChild(canvas);
// dibujar una forma opaca - aquí utilizamos un rectángulo redondeado
defineRoundedRect(30,30,100,75,10);
// establecer el sombreado
ctx.shadowColor='black';
ctx.shadowBlur=10;
// trazar el rectángulo redondeado con sombreado
ctx.lineWidth=4;
ctx.stroke();
// detener el sombreado
ctx.shadowColor='rgba(0,0,0,0)';
// configurar la composición para borrar todo lo que está fuera del trazo
ctx.globalCompositeOperation='destination-in';
ctx.fill();
// configurar la composición para borrar todo lo que está fuera del trazo
ctx.globalCompositeOperation='destination-over';
ctx.fillStyle='gold';
ctx.fill();
// limpiar siempre - volver a la composición por defecto
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();
}
```

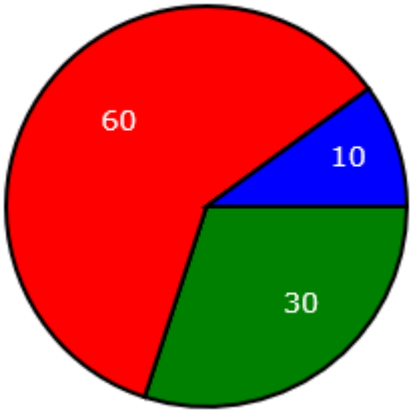
Rellenos sin trazados con una sombra interior

Para dibujar una forma rellena con una sombra interior, pero sin trazo, puede dibujar el trazo fuera del canvas y utilizar `shadowOffsetX` para devolver la sombra al canvas.

```
var canvas=document.createElement("canvas");
var ctx=canvas.getContext("2d");
document.body.appendChild(canvas);
// definir una forma opaca - aquí utilizamos un rectángulo redondeado
defineRoundedRect(30-500,30,100,75,10);
// establecer el sombreado
ctx.shadowColor='black';
ctx.shadowBlur=10;
ctx.shadowOffsetX=500;
// trazar el rectángulo redondeado con sombreado
ctx.lineWidth=4;
ctx.stroke();
// detener el sombreado
ctx.shadowColor='rgba(0,0,0,0)';
// redefinir una forma opaca - aquí utilizamos un rectángulo redondeado
defineRoundedRect(30,30,100,75,10);
// configurar la composición para borrar todo lo que está fuera del trazo
ctx.globalCompositeOperation='destination-in';
ctx.fill();
// configurar la composición para borrar todo lo que está fuera del trazo
ctx.globalCompositeOperation='destination-over';
ctx.fillStyle='gold';
ctx.fill();
// limpiar siempre - volver a la composición por defecto
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();
}
```

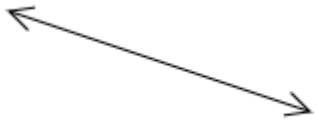
Capítulo 15: Gráficos y diagramas

Sección 15.1: Gráfico circular con demostración



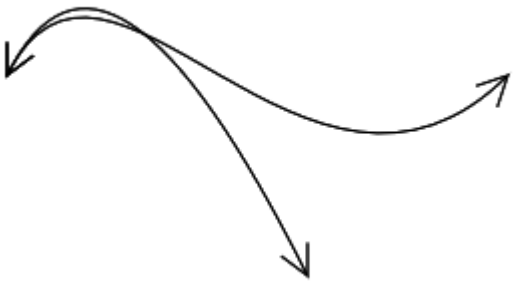
```
<!doctype html>
<html>
     <head>
          <style>
               body{ background-color:white; }
               #canvas{border:1px solid red; }
          </style>
          <script>
               window.onload=(function(){
                    var canvas = document.getElementById("canvas");
                    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) {
                         // dibujar la cuña
                         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();
                         // dibujar la etiqueta
                         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);
                    }
               }); // fin $(function(){});
          </script>
     </head>
     <body>
          <canvas id="canvas" width=512 height=512></canvas>
     </body>
</html>
```

Sección 15.2: Línea con puntas de flecha



```
// Uso:
drawLineWithArrows(50,50,150,50,5,8,true,true);
// x0,y0: el punto de partida de la línea
// x1,y1: el punto final de la línea
// width: la distancia que la punta de la flecha se extiende perpendicularmente fuera de la línea
// height: la distancia que la punta de la flecha se extiende hacia atrás desde el punto final
// arrowStart: true/false dirigir para dibujar la punta de flecha en el punto inicial de la línea
// arrowEnd: true/false dirigir para dibujar punta de flecha en el punto final de la línea
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);
}
```

Sección 15.3: Curva de Bézier cúbica y cuadrática con puntas de flecha

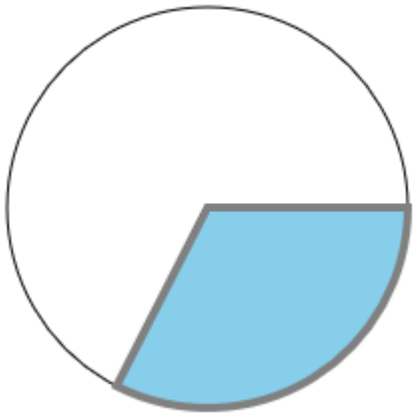


```
// Uso:
var p0={x:50,y:100};
var p1={x:100,y:0};
var p2={x:200,y:200};
var p3={x:300,y:100};
cubicCurveArrowHeads(p0, p1, p2, p3, 15, true, true);
quadraticCurveArrowHeads(p0, p1, p2, 15, true, true);
// o utilizar por defecto true para ambos extremos con puntas de flecha
cubicCurveArrowHeads(p0, p1, p2, p3, 15);
quadraticCurveArrowHeads(p0, p1, p2, 15);
// dibuja bezier cúbicos y cuadráticos
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 = {};
      // por defecto true para ambas flechas si no se incluyen argumentos
      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; // obtener punto final
            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; // obtener punto final
            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);
}
```

Sección 15.4: Cuña

El código dibuja sólo la cuña ... del círculo dibujado aquí sólo para la perspectiva.



```
// Uso
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();
}
```

Sección 15.5: Arco con relleno y trazo

```
// Uso:
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();
}
```

Capítulo 16: Transformaciones

Sección 16.1: Rotar una Imagen o Ruta alrededor de su punto central

Los pasos#1-5 permiten mover cualquier imagen o trazado a cualquier lugar del canvas y rotarla en cualquier ángulo sin cambiar ninguna de las coordenadas originales de la imagen o trazado.

1. Mover el origen del canvas `[0,0]` al punto central de la forma.

```
context.translate( shapeCenterX, shapeCenterY );
```

2. Girar el canvas el ángulo deseado (en radianes).

```
context.rotate( radianAngle );
```

3. Desplazar el origen del canvas a la esquina superior izquierda.

```
context.translate( -shapeCenterX, -shapeCenterY );
```

4. Dibujar la imagen o ruta utilizando sus coordenadas originales.

```
context.fillRect( shapeX, shapeY, shapeWidth, shapeHeight );
```

5. ¡Siempre limpiar! Devuelve el estado de transformación a donde estaba antes de #1.

- *Paso#5, Opción#1:* Deshacer todas las transformaciones en orden inverso.

```
// deshacer #3
context.translate( shapeCenterX, shapeCenterY );
// deshacer #2
context.rotate( -radianAngle );
// deshacer #1
context.translate( -shapeCenterX, shapeCenterY );
```

- *Paso#5, Opción#2:* Si el canvas estaba en un estado sin transformar (el predeterminado) antes de comenzar el paso#1, puede deshacer los efectos de los pasos#1-3 restableciendo todas las transformaciones a su estado por defecto

```
// poner la transformación en el estado por defecto (==sin transformación aplicada)
context.setTransform(1,0,0,1,0,0)
```

Ejemplo de código de demostración

```
// referencias y estilo del canvas
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;
// definir un rectángulo para rotar
var rect={ x:100, y:100, width:175, height:50 };
// dibujar el rectángulo sin rotar
ctx.fillRect( rect.x, rect.y, rect.width, rect.height );
// dibujar el rectángulo girado 45 grados (==PI/4 radianes)
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 );
```

Sección 16.2: Dibujar muchas imágenes trasladadas, escaladas y rotadas rápidamente

Hay muchas situaciones en las que se desea dibujar una imagen girada, escalada y trasladada. La rotación debe producirse alrededor del centro de la imagen. Esta es la forma más rápida de hacerlo en el canvas 2D. Estas funciones son muy adecuadas para juegos 2D en los que se espera renderizar varios cientos de imágenes, incluso más de 1.000, cada 60 segundos. (dependiendo del hardware)

```
// asume que el contexto canvas está en ctx y en scope
function drawImageRST(image, x, y, scale, rotation){
ctx.setTransform(scale, 0, 0, scale, x, y); // establecer la escala y la traslación
ctx.rotate(rotation); // añadir la rotación
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); // dibujar la imagen desplazada la
mitad de su anchura y altura
}
```

Es importante señalar que ambas funciones dejan el contexto del lienzo en un estado aleatorio. Aunque las funciones no se verán afectadas, sí lo harán otras prestaciones. Cuando termine de renderizar imágenes puede que necesite restaurar la transformación por defecto.

```
ctx.setTransform(1, 0, 0, 1, 0, 0); // restablecer la transformación de contexto predeterminada
```

Si utilizas la versión alfa (segundo ejemplo) y luego la versión estándar, tendrás que asegurarte de que se restablece el estado alfa global.

```
ctx.globalAlpha = 1;
```

Un ejemplo de uso de las funciones anteriores para renderizar algunas partículas y algunas imágenes.

```
// asumir que las partículas contienen un array de partículas
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 es necesario dejar el alfa en el bucle
}
// es necesario restablecer el alfa, ya que puede ser cualquier valor
ctx.globalAlpha = 1;
drawImageRST(myImage, 100, 100, 1, 0.5); // dibujar una imagen a 100,100
// no es necesario restablecer la transformación
drawImageRST(myImage, 200, 200, 1, -0.5); // dibujar una imagen a 200,200
ctx.setTransform(1,0,0,1,0,0); // restablecer la transformación
```

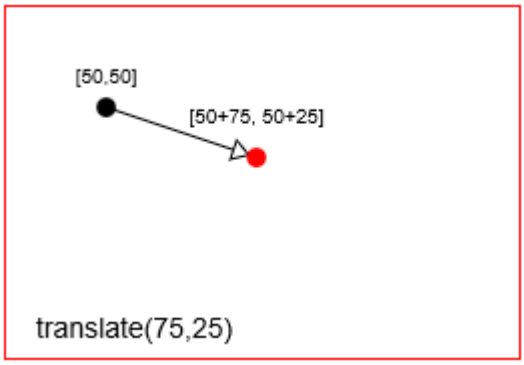
Sección 16.3: Introducción a las transformaciones

Las transformaciones modifican la posición inicial de un punto determinado moviéndolo, rotándolo y escalándolo.

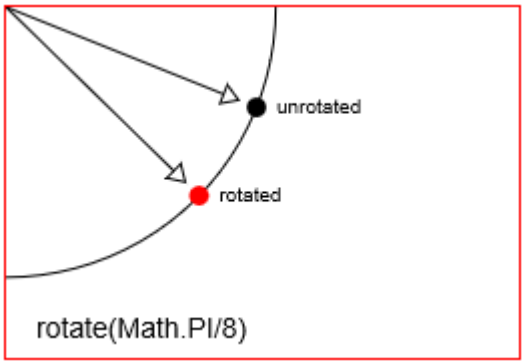
- **Traslación:** Desplaza un punto una `distanceX` y una `distanceY`.
- **Rotación:** Rota un punto un `angulo de un radián` alrededor de su punto de rotación. El punto de rotación por defecto en HTML Canvas es el origen superior izquierdo `[x=0,y=0]` del Canvas. Pero puede reposicionar el punto de rotación utilizando traslaciones.
- **Escala:** Escala la posición de un punto mediante un `factorEscalaX` y un `factorEscalaY` desde su punto de escala. El punto de escala por defecto en el HTML Canvas es el origen superior izquierdo `[x=0,y=0]` del Canvas. Pero puede reposicionar el punto de escala mediante traslaciones.

También puede realizar transformaciones menos comunes, como el shearing (sesgado), estableciendo directamente la matriz de transformación del canvas mediante `context.transform`.

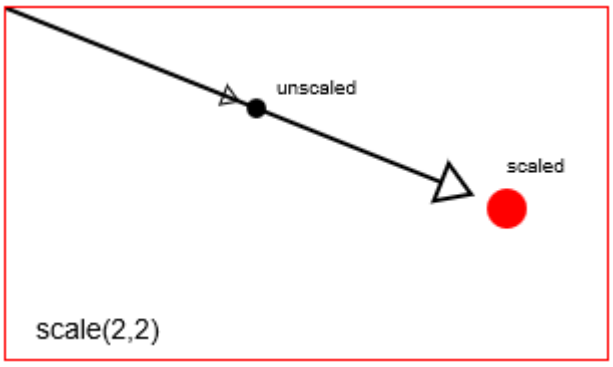
Trasladar (==mover) un punto con `context.translate(75,25)`



Girar un punto con `context.rotate(Math.PI/8)`



Escala un punto con `context.scale(2,2)`



En realidad, el canvas se transforma alterando todo su sistema de coordenadas.

`context.translate` moverá el origen `[0,0]` del canvas desde la esquina superior izquierda a una nueva ubicación.

`context.rotate` rotará todo el sistema de coordenadas del canvas alrededor del origen.

`context.scale` escalará todo el sistema de coordenadas del canvas alrededor del origen. Es como aumentar el tamaño de cada x,y en el canvas: `cada x*=escalaX y cada y*=escalaY`

Las transformaciones del canvas son persistentes. Todos los dibujos nuevos seguirán transformándose hasta que restablezca la transformación del canvas a su estado por defecto (==totalmente sin transformar). Puede restablecer el valor predeterminado con:

```
// restablecer las transformaciones de contexto al estado por defecto (sin transformar)
context.setTransform(1,0,0,1,0,0);
```

Sección 16.4: Una matriz de transformación para realizar un seguimiento de las formas trasladadas, rotar y escalar formas

Canvas te permite `context.translate`, `context.rotate` y `context.scale` para dibujar tu forma en la posición y tamaño que necesites.

El propio canvas utiliza una matriz de transformación para realizar un seguimiento eficaz de las transformaciones.

Puedes cambiar la matriz de Canvas con `context.transform`,

Puedes cambiar la matriz del canvas con comandos individuales de `translate`, `rotate` y `scale`,

Puedes sobrescribir completamente la matriz de Canvas con `context.setTransform`,

Pero no puedes leer la matriz de transformación interna de Canvas - es sólo de escritura.

¿Por qué utilizar una matriz de transformación?

Una matriz de transformación permite agregar muchas traslaciones, rotaciones y escalados individuales en una única matriz de fácil reaplicación.

Durante animaciones complejas, puede aplicar docenas (o cientos) de transformaciones a una forma. Utilizando una matriz de transformación, puede volver a aplicar (casi) instantáneamente esas docenas de transformaciones con una sola línea de código.

Algunos ejemplos de uso:

- **Comprueba si el ratón está dentro de una forma que has trasladado, rotado y escalado.**

Existe un test integrado `context.isPointInPath` que comprueba si un punto (por ejemplo, el ratón) se encuentra dentro de una ruta, pero este test integrado es muy lento en comparación con la comprobación mediante una matriz.

Para comprobar eficazmente si el ratón está dentro de una forma, hay que tomar la posición del ratón indicada por el navegador y transformarla del mismo modo que se transformó la forma. A continuación, puede aplicar la prueba de acierto como si la forma no se hubiera transformado.

- **Volver a dibujar una forma que se ha trasladado, rotado y escalado ampliamente.**

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.

- **Prueba de colisión de formas trasladadas, giradas y escaladas**

Puedes utilizar la geometría y la trigonometría para calcular los puntos que forman las formas transformadas, pero es más rápido utilizar una matriz de transformación para calcular esos puntos.

Una "clase" de matriz de transformación

Este código refleja los comandos de transformación nativos `context.translate`, `context.rotate`, `context.scale`. A diferencia de la matriz nativa del canvas, esta matriz es legible y reutilizable.

Métodos:

- `translate`, `rotate`, `scale` reflejan los comandos de transformación del contexto y permiten introducir transformaciones en la matriz. La matriz contiene eficazmente las transformaciones agregadas.
- `setContextTransform` toma un contexto y establece la matriz de ese contexto igual a esta matriz de transformación. Esto reaplica de forma eficiente todas las transformaciones almacenadas en esta matriz al contexto.
- `resetContextTransform` restablece la transformación del contexto a su estado por defecto (==sin transformar).
- `getTransformedPoint` toma un punto de coordenadas no transformado 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.

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]) ];
          // punto
          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;
     }
     // metodos compartidos
     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);
     }
     // devolver public
     return(TransformationMatrix);
})();
```

Demostración:

Esta demostración utiliza la Matriz de Transformación "Clase" anterior para:

- Rastrear (==guardar) la matriz de transformación de un rectángulo.
- Vuelve a dibujar el rectángulo transformado sin utilizar comandos de transformación contextual.
- Comprueba 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(); }
                    // Matriz de transformación "Clase"
                    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])];
                              // punto
                              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;
                         }
                         // métodos compartidos
                         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,sc
        reenY){
            return(transformedPoint(screenX,screenY));
        }
        TransformationMatrix.prototype.getScreenPoint=function(transformedX,tr
        ansformedY){
            return(screenPoint(transformedX,transformedY));
        }
        TransformationMatrix.prototype.getMatrix=function(){
            var clone=[m[0],m[1],m[2],m[3],m[4],m[5]];
            return(clone);
        }
        // devuelve public
        return(TransformationMatrix);
})();
// DEMOSTRACIÓN comienza aquí creando un rectángulo y añadiendo una matriz de
transformación para seguir sus traslaciones, rotaciones y escalados.
var rect={x:30,y:30,w:50,h:35,matrix:new TransformationMatrix()};
// dibujar el rectángulo no transformado en negro
ctx.strokeRect(rect.x, rect.y, rect.w, rect.h);
// Demostración: etiqueta
ctx.font='11px arial';
ctx.fillText('Untransformed Rect',rect.x,rect.y-10);
// transforma el canvas y dibuja el rectángulo transformado en rojo
ctx.translate(100,0);
ctx.scale(2,2);
ctx.rotate(Math.PI/12);
// dibujar el rectángulo transformado
ctx.strokeStyle='red';
ctx.strokeRect(rect.x, rect.y, rect.w, rect.h);
ctx.font='6px arial';
// Demostración: etiqueta
ctx.fillText('Same Rect: Translated, rotated & scaled',rect.x,rect.y-6);
// restablecer el contexto al estado sin transformar
ctx.setTransform(1,0,0,1,0,0);
// registrar las transformaciones en la matriz
var m=rect.matrix;
m.translate(100,0);
```

```
                m.scale(2,2);
                m.rotate(Math.PI/12);
                // utilizar la matriz de transformación guardada del rectángulo para
                reposicionarlo, redimensionarlo y redibujarlo
                ctx.strokeStyle='blue';
                drawTransformedRect(rect);
                // Demostración: instrucciones
                ctx.font='14px arial';
                ctx.fillText('Demo: click inside the blue rect',30,200);
                // redibujar un rectángulo basándose en su matriz de transformación guardada
                function drawTransformedRect(r){
                     // establece la matriz de transformación del contexto utilizando la
                     matriz guardada del rectángulo
                     m.setContextTransform(ctx);
                     // dibujar el rectángulo (¡no es necesario cambiar la posición ni el
                     tamaño!)
                     ctx.strokeRect( r.x, r.y, r.w, r.h );
                     // restablecer la transformación del contexto a su valor por defecto
                     (==sin transformar);
                     m.resetContextTransform(ctx);
                }
                // ¿es el punto del rectángulo transformado?
                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);
                }
                // escucha los eventos mousedown
                canvas.onmousedown=handleMouseDown;
                function handleMouseDown(e){
                     // indica al navegador que estamos gestionando este evento
                     e.preventDefault();
                     e.stopPropagation();
                     // obtener la posición del ratón
                     mouseX=parseInt(e.clientX-offsetX);
                     mouseY=parseInt(e.clientY-offsetY);
                     // ¿está el ratón dentro del rectángulo transformado?
                     if(isPointInTransformedRect(rect,mouseX,mouseY)){
                          alert('You clicked in the transformed Rect');
                     }
                }
                // Demostración: redibujar un rectángulo transformado sin utilizar comandos
                de transformación contextual
                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();
                }
           }); // fin window.onload
     </script>
</head>
<body>
     <canvas id="canvas" width=512 height=250></canvas>
```

```
    </body>
</html>
```

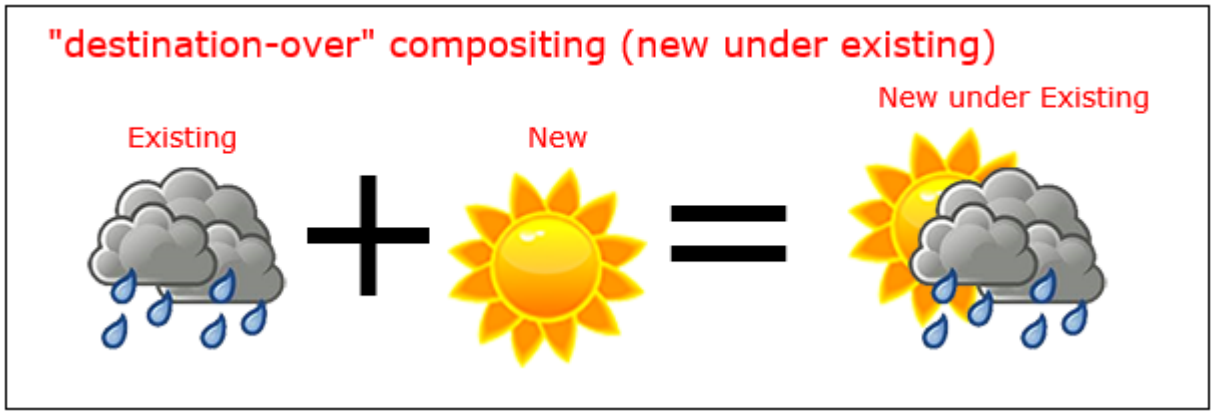
Capítulo 17: Composición

Sección 17.1: Dibujar detrás de formas existentes con “destination-over”

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

La composición "`destination-over`" coloca el nuevo dibujo debajo de los dibujos existentes.

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



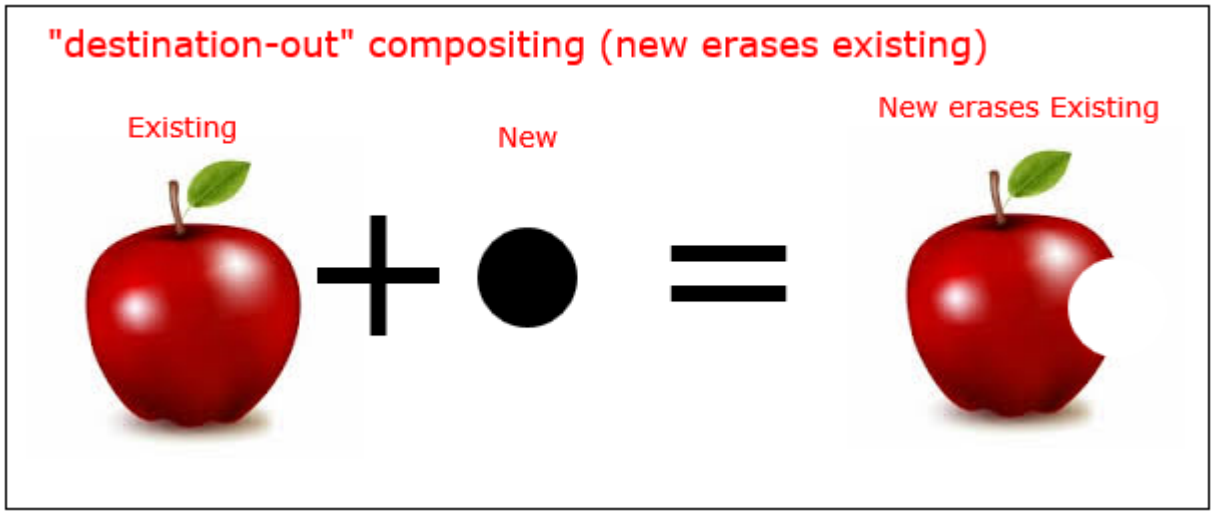
Sección 17.2: Borrar formas existentes con “destination-out”

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

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

La nueva forma no se dibuja realmente, sólo se utiliza como "cookie-cutter" para borrar los píxeles existentes.

```
context.drawImage(apple,0,0);
context.globalCompositeOperation = 'destination-out'; // borrado de marcas de bits
context.drawImage(bitemark,100,40);
```

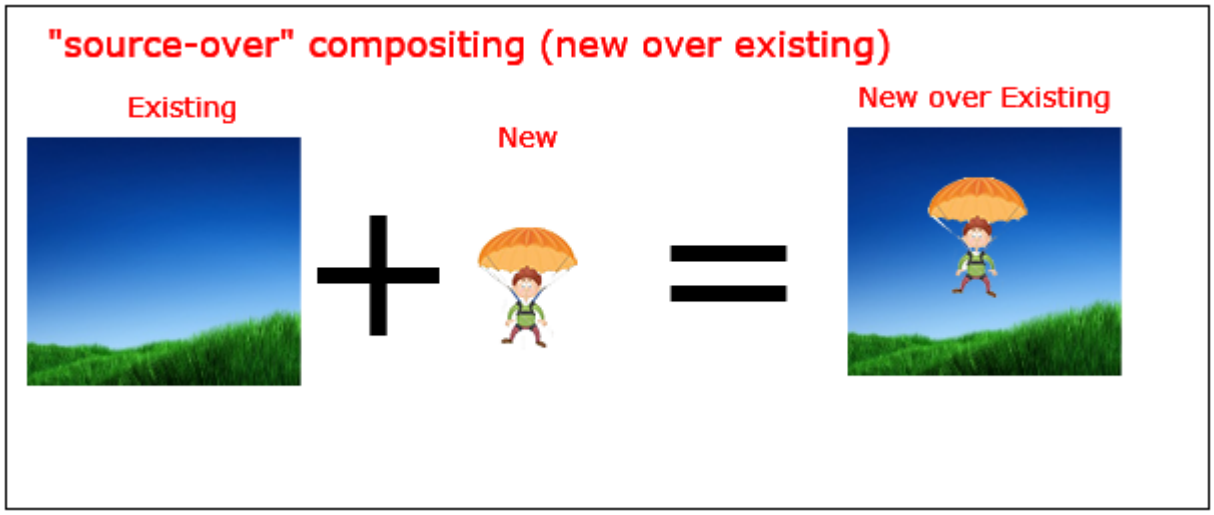


Sección 17.3: Composición por defecto: Las nuevas formas se dibujan sobre formas existentes

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

Composición "`source-over`" [por defecto], coloca todos los dibujos nuevos sobre cualquier dibujo existente.

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



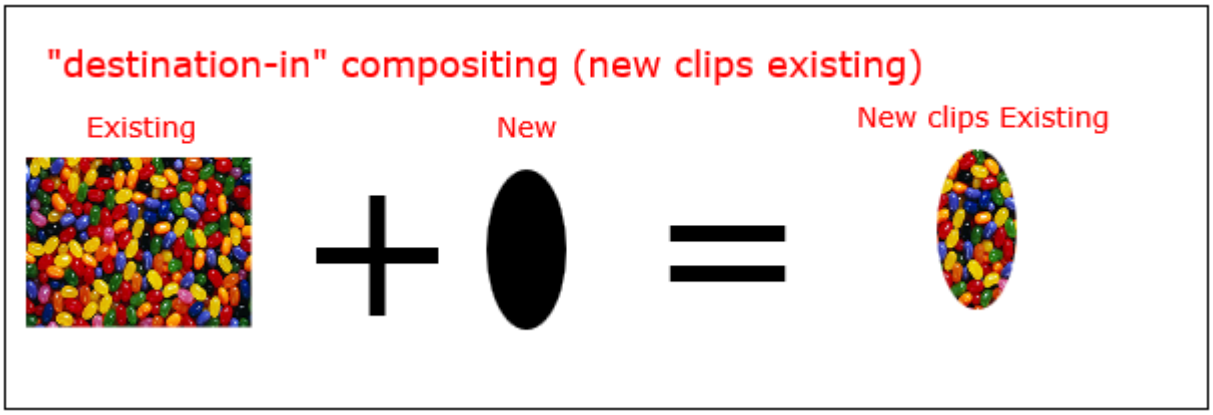
Sección 17.4: Recortar imágenes dentro de formas con “destination-in”

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

La composición "`destination-in`" recorta los dibujos existentes dentro de una nueva forma.

Nota: Cualquier parte del dibujo existente que quede fuera del nuevo dibujo se borra.

```
context.drawImage(picture,0,0);
context.globalCompositeOperation='destination-in'; // imagen recortada dentro del óvalo
context.drawImage(oval,0,0);
```



Sección 17.5: Recortar imágenes dentro de formas con “source-in”

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

La composición “`source-in`” recorta nuevos dibujos dentro de una forma existente.

Nota: Cualquier parte del nuevo dibujo que quede fuera del dibujo existente se borra.

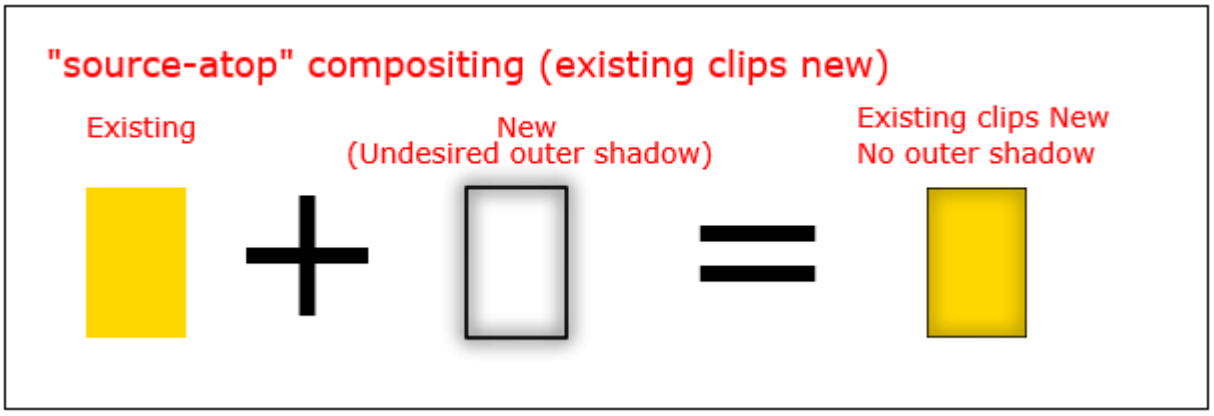
```
context.drawImage(oval,0,0);
context.globalCompositeOperation='source-in'; // imagen recortada dentro del óvalo
context.drawImage(picture,0,0);
```

Sección 17.6: Sombras interiores con “source-atop”

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

La composición “`source-atop`” de clips nueva imagen dentro de una forma existente.

```
// rectangulo relleno de oro
ctx.fillStyle='gold';
ctx.fillRect(100,100,100,75);
// sombra
ctx.shadowColor='black';
ctx.shadowBlur=10;
// restringir el dibujo nuevo para cubrir los píxeles existentes
ctx.globalCompositeOperation='source-atop';
// trazo sombreado
// "source-atop" recorta la sombra exterior no deseada
ctx.strokeRect(100,100,100,75);
ctx.strokeRect(100,100,100,75);
```



Sección 17.7: Cambiar la opacidad con “globalAlpha”

```
context.globalAlpha=0.50
```

Puede cambiar la opacidad de los nuevos dibujos estableciendo el `globalAlpha` en un valor comprendido entre 0,00 (totalmente transparente) y 1,00 (totalmente opaco).

El `globalAlpha` por defecto es 1.00 (totalmente opaco).

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

```
// dibujar un rectángulo opaco
context.fillRect(10,10,50,50);
// cambiar alfa a 50% - todos los dibujos nuevos tendrán una opacidad del 50%.
context.globalAlpha=0.50;
// dibujar un rectángulo semitransparente
context.fillRect(100,10,50,50);
```

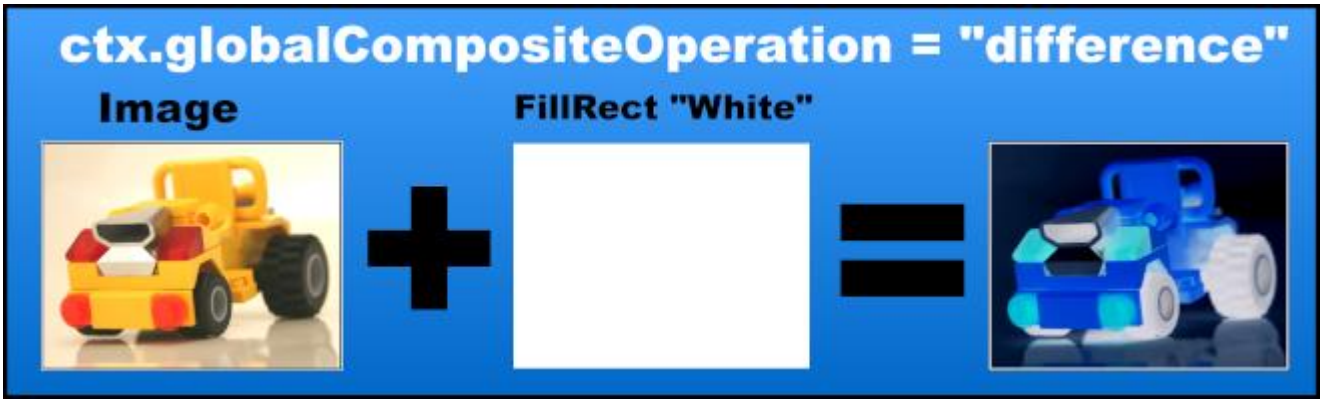
Sección 17.8: Invertir o negar la imagen con “difference”.

Renderizar un rectángulo blanco sobre una imagen con la operación de composición.

```
ctx.globalCompositeOperation = 'difference';
```

La cantidad del efecto se puede controlar con el ajuste alfa.

```
// Renderizar la imagen
ctx.globalCompositeOperation='source-atop';
ctx.drawImage(image, 0, 0);
// establecer la operación compuesta
ctx.globalCompositeOperation='difference';
ctx.fillStyle = "white";
ctx.globalAlpha = alpha; // alfa 0 = sin efecto 1 = efecto total
ctx.fillRect(0, 0, image.width, image.height);
```



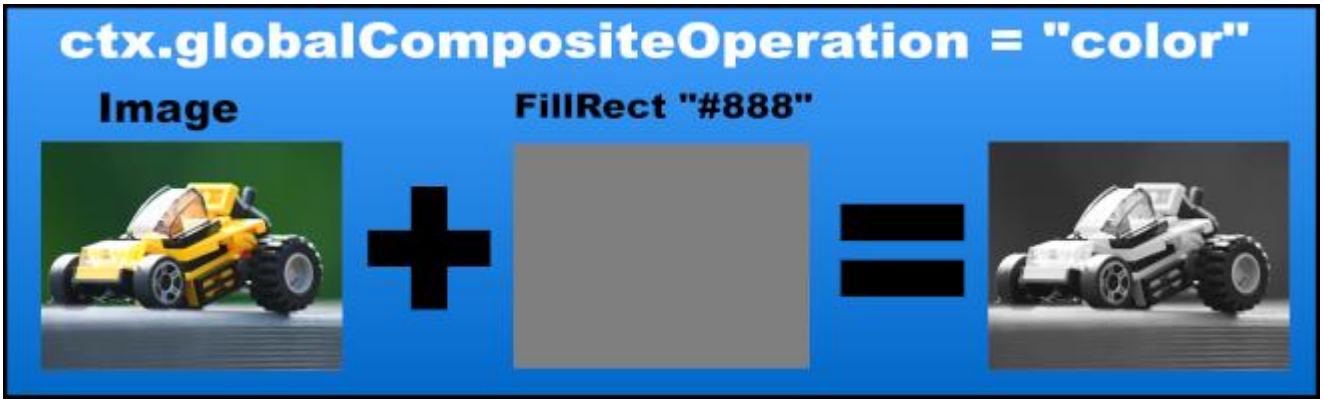
Sección 17.9: Blanco y negro con “color”

Eliminar el color de una imagen mediante

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

La cantidad del efecto se puede controlar con el ajuste alfa.

```
// Renderizar la imagen
ctx.globalCompositeOperation='source-atop';
ctx.drawImage(image, 0, 0);
// establecer la operación compuesta
ctx.globalCompositeOperation='color';
ctx.fillStyle = "white";
ctx.globalAlpha = alpha; // alfa 0 = sin efecto 1 = efecto total
ctx.fillRect(0, 0, image.width, image.height);
```



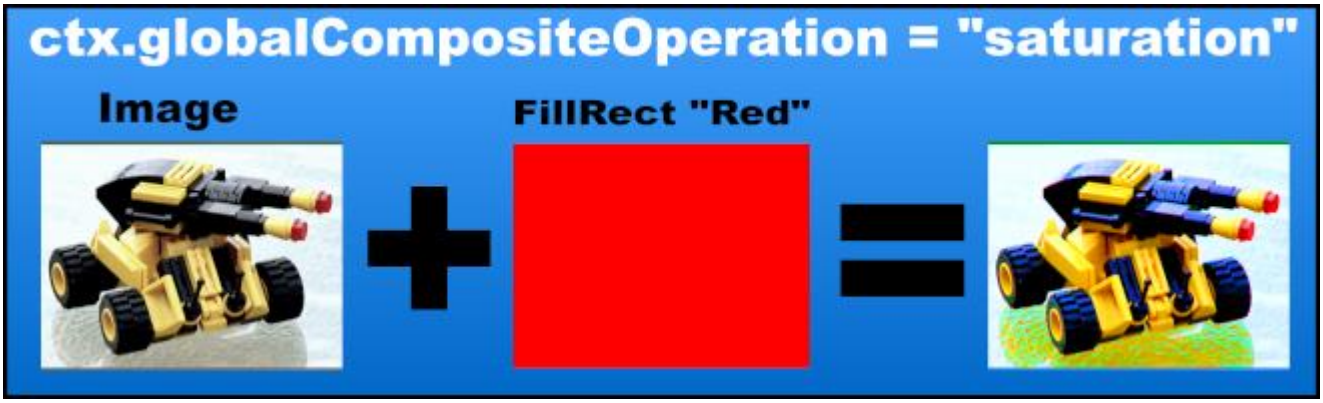
Sección 17.10: Aumentar el contraste de color con “saturation”

Aumenta el nivel de saturación de una imagen con

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

La cantidad de efecto puede controlarse con el ajuste alfa o la cantidad de saturación en la superposición de relleno.

```
// Renderizar la imagen
ctx.globalCompositeOperation='source-atop';
ctx.drawImage(image, 0, 0);
// establecer la operación compuesta
ctx.globalCompositeOperation ='saturation';
ctx.fillStyle = "red";
ctx.globalAlpha = alpha; // alfa 0 = sin efecto 1 = efecto total
ctx.fillRect(0, 0, image.width, image.height);
```



Sección 17.11: Sepia FX con “luminosity”

Cree un efecto sepia con

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

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

La cantidad de efecto puede controlarse con el ajuste alfa o la cantidad de saturación en la superposición de relleno.

```
// Renderizar la imagen
ctx.globalCompositeOperation='source-atop';
ctx.fillStyle = "#F80"; // el color del efecto sepia
ctx.fillRect(0, 0, image.width, image.height);
// establecer la operación compuesta
ctx.globalCompositeOperation ='luminosity';
ctx.globalAlpha = alpha; // alfa 0 = sin efecto 1 = efecto total
ctx.drawImage(image, 0, 0);
```

Capítulo 18: Manipulación de píxeles con “getImageData” y “putImageData”

Sección 18.1: Introducción a “context.getImageData”

HTML5 Canvas le da la capacidad de obtener y cambiar el color de cualquier píxel en el canvas.

Puedes utilizar la manipulación de píxeles de Canvas para:

- Cree un selector de color para una imagen o seleccione un color en una rueda de color.
- Crea filtros de imagen complejos como el desenfoque y la detección de bordes.
- Recolorear cualquier parte de una imagen a nivel de píxel (si utilizas HSL puedes incluso recolorear una imagen conservando la iluminación y la saturación importantes para que el resultado no parezca que alguien ha pintado la imagen). Nota: Canvas ahora tiene Blend Compositing (composición de mezcla) que también puede volver a colorear una imagen en algunos casos.
- Un "Knockout" alrededor del fondo de una persona / elemento en una imagen.
- Crear una herramienta de pintura para detectar y rellenar parte de una imagen (por ejemplo, cambiar el color de un pétalo de flor pulsado por el usuario de verde a amarillo).
- Examinar el contenido de una imagen (por ejemplo, reconocimiento facial).

Problemas comunes:

- Por razones de seguridad, `getImageData` está desactivado si ha dibujado una imagen originada en un dominio diferente que la propia página web.
- `getImageData` es un método relativamente caro porque crea un gran array de datos de píxeles y porque no utiliza la GPU para ayudar a sus esfuerzos. Nota: Canvas ahora tiene composición de mezcla que puede hacer algunas de las mismas manipulaciones de píxeles que hace `getImageData`.
- En el caso de las imágenes .png, es posible que `getImageData` no muestre exactamente los mismos colores que el archivo .png original, ya que el navegador puede realizar la corrección gamma y la premultiplicación alfa al dibujar las imágenes en el canvas.

Obtener los colores de los píxeles

Utiliza `getImageData` para obtener los colores de los píxeles de todo o parte del contenido del canvas.

El método `getImageData` devuelve un objeto `imageData`.

El objeto `imageData` tiene una propiedad `.data` que contiene la información del color de los píxeles.

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

```
// determinar los píxeles que se van a recuperar (se recuperan todos los píxeles del canvas)
var x=0;
var y=0;
var width=canvas.width;
var height=canvas.height;
// Obtener el objeto imageData
var imageData = context.getImageData(x,y,width,height);
// Extrae el array de datos de color de los píxeles del objeto imageData
var pixelDataArray = imageData.data;
```

Puede obtener la posición de cualquier píxel `[x,y]` dentro del array de datos de la siguiente manera:

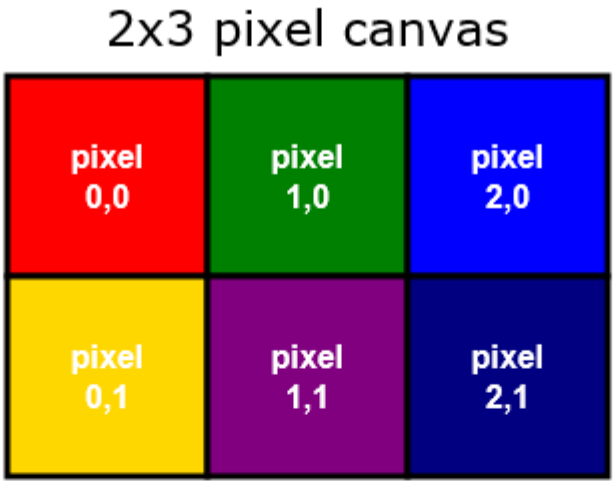
```
// la posición del array data[] para el píxel [x,y]
var n = y * canvas.width + x;
```

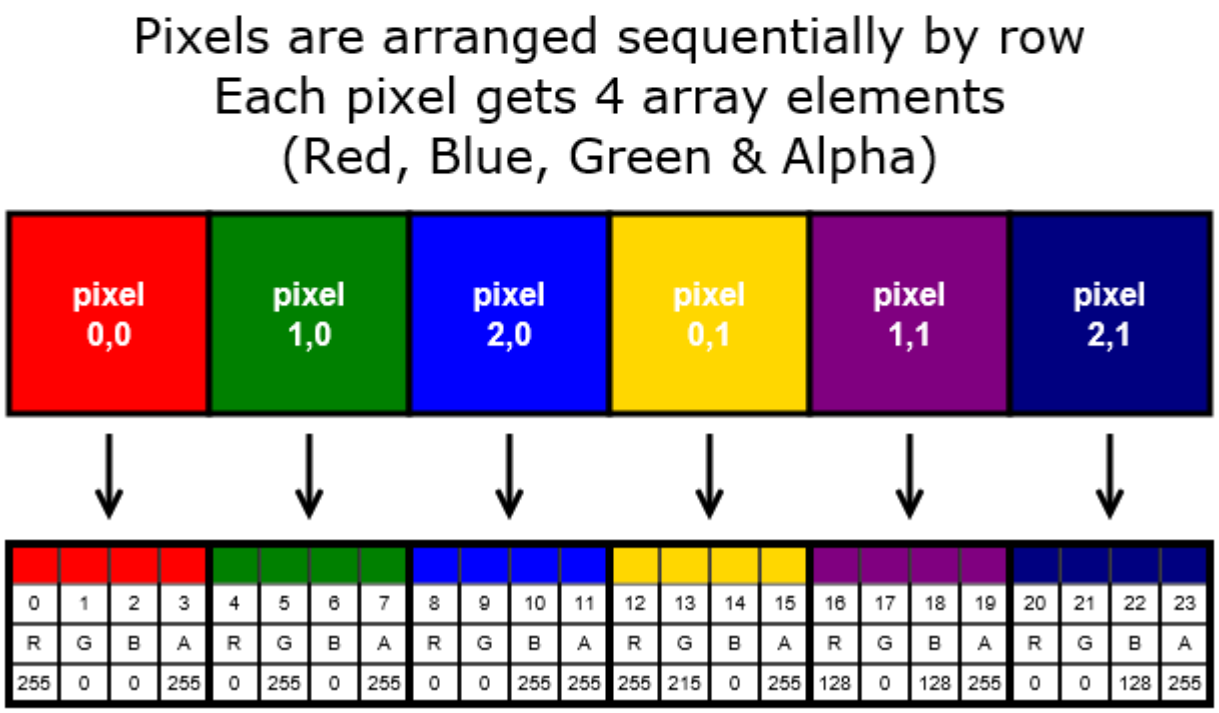
Y luego puedes obtener los valores rojo, verde, azul y alfa de ese píxel así:

```
// la información RGBA para el píxel [x,y]
var red=data[n];
var green=data[n+1];
var blue=data[n+2];
var alpha=data[n+3];
```

Ilustración de la estructura del array de datos de píxeles

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





Créditos

Muchas gracias a todas las personas de Stack Overflow Documentation que ayudaron a proporcionar este contenido, más cambios pueden ser enviados a web@petercv.com para que el nuevo contenido sea publicado o actualizado.

Traductor al español

rortegag

almcd	Capítulo 2
bjanes	Capítulo 12
Blindman67	Capítulos 1, 2, 3, 4, 6, 7, 9, 10, 11, 12, 13, 15, 16 y 17
Kaiido	Capítulos 1, 4, 9 y 12
markE	Capítulos 1, 2, 3, 4, 5, 6, 7, 8, 10, 11, 13, 14, 15, 16, 17 y 18
Mike C	Capítulo 12
Ronen Ness	Capítulo 12
Spencer Wieczorek	Capítulo 1
user2314737	Capítulo 1