EBook Gratuito

APPRENDIMENTO
html5-canvas

Free unaffiliated eBook created from
Stack Overflow contributors.

#html5-canvas

Sommario

Di

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

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

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

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

Capitolo 1: Iniziare con html5-canvas

Examples

Come aggiungere l'elemento Html5 Canvas a una pagina Web

Html5-Canvas ...

- È un elemento Html5.
- È supportato nella maggior parte dei browser moderni (Internet Explorer 9+).
- È un elemento visibile che è trasparente per impostazione predefinita
- Ha una larghezza predefinita di 300 px e un'altezza predefinita di 150 px.
- Richiede JavaScript perché tutto il contenuto deve essere aggiunto a livello di codice alla tela.

Esempio: crea un elemento Html5-Canvas usando sia markup Html5 che JavaScript:

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

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

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

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

</body>
</html>
```

Dimensioni e risoluzione della tela

La dimensione di una tela è l'area che occupa nella pagina ed è definita dalle proprietà di larghezza e altezza CSS.

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

```
   height : 1000px;
}
```

La risoluzione canvas definisce il numero di pixel che contiene. La risoluzione viene impostata impostando le proprietà di larghezza e altezza dell'elemento canvas. Se non specificato, il canvas ha un valore predefinito di 300 per 150 pixel.

La seguente tela utilizzerà la precedente dimensione CSS ma, poiché la `width` e l' `height` non sono specificate, la risoluzione sarà 300 per 150.

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

Ciò comporterà che ogni pixel viene allungato in modo non uniforme. L'aspetto dei pixel è 1: 2. Quando la tela viene tesa, il browser utilizzerà il filtro bilineare. Questo ha un effetto di sfocatura dei pixel che vengono allungati.

Per ottenere i migliori risultati quando si utilizza il canvas, assicurarsi che la risoluzione del canvas corrisponda alle dimensioni dello schermo.

Seguendo lo stile CSS sopra per abbinare le dimensioni dello schermo aggiungi la tela con la `width` e l' `height` impostate sullo stesso numero di pixel definito dallo stile.

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

Off screen canvas

Molte volte quando si lavora con la tela è necessario disporre di una tela per contenere alcuni dati intrum pixel. È facile creare una tela fuori schermo, ottenere un contesto 2D. Una tela fuori schermo utilizzerà anche l'hardware grafico disponibile per il rendering.

Il seguente codice crea semplicemente una tela e la riempie di pixel blu.

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

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

Molte volte la tela fuori schermo verrà utilizzata per molte attività e potresti avere molte tele. Per semplificare l'uso della tela è possibile allegare il contesto della tela alla tela.

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

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

Rilevamento della posizione del mouse sulla tela

Questo esempio mostrerà come ottenere la posizione del mouse rispetto alla tela, tale che `(0,0)` sarà l'angolo in alto a sinistra della tela HTML5. `e.clientX` ed `e.clientY` otterranno le posizioni del mouse rispetto alla parte superiore del documento, per cambiarlo basandosi sulla parte superiore della tela sottraiamo le posizioni `left` e `right` della tela dal client X e Y.

```
var canvas = document.getElementById("myCanvas");
var ctx = canvas.getContext("2d");
ctx.font = "16px Arial";

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

Esempio eseguibile

L'uso di `Math.round` è dovuto per garantire che le posizioni `x,y` siano numeri interi, poiché il rettangolo di delimitazione della tela potrebbe non avere posizioni di numeri interi.

Ciao mondo

HTML

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

Javascript

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

Risultato

Un indice delle capacità e degli usi della tela Html5

Funzionalità della tela

Canvas ti consente di disegnare in modo programmatico sulla tua pagina web:

- Immagini ,
- Testi ,
- Linee e curve .

I disegni su tela possono essere disegnati estensivamente:

- larghezza della corsa ,
- colore del tratto ,
- forma colore di riempimento ,
- opacità ,
- ombreggiato ,
- gradienti lineari e gradienti radiali ,
- carattere ,
- dimensione del carattere ,
- allineamento del testo ,
- il testo può essere accarezzato, riempito o sia accarezzato e riempito ,
- ridimensionamento delle immagini ,
- ritaglio di immagini ,
- compositing

Usi della tela

I disegni possono essere combinati e posizionati ovunque sulla tela in modo che possa essere utilizzato per creare:

- Applicazioni Paint / Sketch,
- Giochi interattivi veloci,
- I dati analizzano come grafici, grafici,
- Imaging simile a Photoshop,
- Pubblicità in stile Flash e contenuto Web appariscente.

Canvas ti consente di manipolare i colori dei componenti Rosso, Verde, Blu e Alfa delle immagini. Ciò consente al canvas di manipolare le immagini con risultati simili a Photoshop.

- Ricolora qualsiasi parte di un'immagine a livello di pixel (se usi HSL puoi anche ricolorare un'immagine conservando l'illuminazione e la saturazione importanti in modo che il risultato non assomigli a qualcuno che schiaffeggia l'immagine),
- "Elimina" lo sfondo attorno a una persona / elemento in un'immagine,
- Rileva e Floodfill parte di un'immagine (ad esempio, cambia il colore di un petalo selezionato dall'utente da verde a giallo - solo quel petalo cliccato!),
- Do Warping Perspective (ad es. Avvolgere un'immagine attorno alla curva di una tazza),
- Esaminare un'immagine per il contenuto (ad esempio riconoscimento facciale),
- Rispondere a domande su un'immagine: Esiste una macchina parcheggiata in questa immagine del mio parcheggio ?,
- Applicare filtri di immagine standard (scala di grigi, seppia, ecc.)
- Applica qualsiasi filtro di immagine esotico che puoi immaginare (Sobel Edge Detection),
- Combina immagini. Se la cara nonna Sue non riuscisse a raggiungere il ricongiungimento familiare, basta "fotografarla" nell'immagine della riunione. Non mi piace Cugino Phil - solo "photoshop fuori,
- Riproduci un video / Prendi una cornice da un video,
- Esportare il contenuto del canvas come .jpg | .png image (puoi anche ritagliare o annotare l'immagine e esportare il risultato come una nuova immagine),

Informazioni sullo spostamento e la modifica di disegni su tela (ad esempio per creare un gioco d'azione):

- Dopo che qualcosa è stato disegnato sulla tela, il disegno esistente non può essere spostato o modificato. Questo malinteso comune che i disegni su tela sono mobili vale la pena chiarire: i *disegni su tela esistenti non possono essere modificati o spostati!*
- La tela si disegna molto, molto velocemente. La tela può disegnare centinaia di immagini, testi, linee e curve in una frazione di secondo. Usa la GPU quando è disponibile per velocizzare il disegno.
- La tela crea l'illusione del movimento disegnando velocemente e ripetutamente qualcosa e quindi ridisegnandolo in una nuova posizione. Come la televisione, questo costante ridisegno dà allo sguardo l'illusione del movimento.

Ruotare

Il metodo `rotate(r)` del contesto 2D ruota la tela della quantità specificata `r` di *radianti* attorno all'origine.

HTML

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

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

Javascript

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

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

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

Demo dal vivo su JSfiddle

Salva la tela nel file immagine

È possibile salvare una tela su un file immagine utilizzando il metodo `canvas.toDataURL()` , che restituisce l' *URI di dati* per i dati dell'immagine della tela.

Il metodo può prendere due parametri opzionali `canvas.toDataURL(type, encoderOptions)` : `type` è il formato dell'immagine (se omesso il valore predefinito è `image/png`); `encoderOptions` è un numero compreso tra 0 e 1 che indica la qualità dell'immagine (il valore predefinito è 0.92).

Qui disegniamo una tela e colleghiamo l'URI dei dati della tela al link "Download to myImage.jpg".

HTML

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

Javascript

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

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

```
};
```

Demo dal vivo su JSfiddle.

Leggi Iniziare con html5-canvas online: https://riptutorial.com/it/html5-canvas/topic/1892/iniziare-con-html5-canvas

Capitolo 2: Animazione

Examples

Animazione semplice con contesto 2D e requestAnimationFrame

Questo esempio ti mostrerà come creare un'animazione semplice usando la tela e il contesto 2D. Si presume che tu sappia come creare e aggiungere una tela al DOM e ottenere il contesto

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


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

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

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

Una demo di questo esempio su jsfiddle

Animare a intervalli specificati (aggiungi un nuovo rettangolo ogni 1 secondo)

Questo esempio aggiunge un nuovo rettangolo al canvas ogni 1 secondo (== un intervallo di 1 secondo)

Codice annotato:

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

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

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

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

    // start the animation
    requestAnimationFrame(animate);

    function animate(currentTime){

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

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

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

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

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

Animare in un momento specifico (un orologio animato)

Questo esempio anima un orologio che mostra i secondi come un cuneo pieno

Codice annotato:

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

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

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

    // start the animation
    requestAnimationFrame(animate);

    function animate(currentTime){

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

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

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

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

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

Usa requestAnimationFrame () NOT setInterval () per i loop dell'animazione

`requestAnimationFrame` è simile a setInterval, ma ha questi importanti miglioramenti:

- Il codice dell'animazione è sincronizzato con gli aggiornamenti del display per l'efficienza Il codice clear + redraw è pianificato, ma non eseguito immediatamente. Il browser eseguirà il codice clear + redraw solo quando il display è pronto per l'aggiornamento. Questa sincronizzazione con il ciclo di aggiornamento aumenta le prestazioni dell'animazione fornendo al codice il tempo più disponibile per il completamento.
- Ogni ciclo è sempre completato prima che possa iniziare un altro ciclo. Questo impedisce "strappi", in cui l'utente vede una versione incompleta del disegno. L'occhio nota in particolare lacrimazione e si distrae quando si verifica la lacrimazione. In questo modo prevenire la lacrimazione rende l'animazione più omogenea e più coerente.
- L'animazione si interrompe automaticamente quando l'utente passa a una scheda del browser diversa. Ciò consente di risparmiare energia sui dispositivi mobili perché il dispositivo non spreca energia calcolando un'animazione che l'utente non può attualmente vedere.

I display dei dispositivi si aggiorneranno circa 60 volte al secondo, quindi requestAnimationFrame può essere ridisegnato continuamente a circa 60 "frame" al secondo. L'occhio vede il movimento a 20-30 fotogrammi al secondo, quindi requestAnimationFrame può facilmente creare l'illusione del movimento.

Si noti che requestAnimationFrame viene richiamato alla fine di ogni animateCircle. Questo perché ogni 'requestAnimatonFrame richiede una singola esecuzione della funzione di animazione.

Esempio: semplice `requestAnimationFrame

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

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

    // start the animation
    requestAnimationFrame(animate);

    function animate(currentTime){

        // draw a full randomly circle
```

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

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

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

Per illustrare i vantaggi di requestAnimationFrame questa domanda stackoverflow ha una demo live

Animare un'immagine attraverso la tela

Questo esempio carica, anima e immagini sulla tela

Suggerimento importante! Assicurati di dare il tempo necessario per caricare l'immagine utilizzando `image.onload` .

Codice annotato

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

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

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

    // Load a new image
```

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

    function animate(time){

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

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

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

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

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

Non disegnare animazioni nei gestori di eventi (una semplice app di schizzo)

Durante il `mousemove` vieni sommerso di 30 eventi del mouse al secondo. Potresti non essere in grado di ridisegnare i tuoi disegni a 30 volte al secondo. Anche se è possibile, si sta probabilmente sprecando potenza di calcolo disegnando quando il browser non è pronto per disegnare (sprecato == tra i cicli di aggiornamento della visualizzazione).

Pertanto ha senso separare gli eventi di input dell'utente (come mousemove) dal disegno delle animazioni.

- Nei gestori di eventi, salva tutte le variabili evento che controllano dove i disegni sono posizionati sulla tela. Ma in realtà non disegnare nulla.
- In un ciclo `requestAnimationFrame` , visualizza tutti i disegni sulla tela utilizzando le informazioni salvate.

Non disegnando i gestori di eventi, non si sta costringendo Canvas per provare ad aggiornare i disegni complessi alla velocità degli eventi del mouse.

Facendo tutto il disegno in `requestAnimationFrame` ottieni tutti i vantaggi descritti qui. Usa

'requestanimationFrame' non 'setInterval' per i loop dell'animazione .

Codice annotato:

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

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

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

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

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

    // start the  animation loop
    requestAnimationFrame(draw);

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


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

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

        // save the mouse position in the points[] array
        // but don't draw anything
        points.push({x:mouseX,y:mouseY});
```

```
    }

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

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

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

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

Facilitare usando le equazioni di Robert Penners

Un andamento fa sì che alcune **variabili** cambino in **modo non uniforme** per una **durata** .

"variabile" deve poter essere espressa come un numero e può rappresentare una notevole varietà di cose:

- una coordinata X,
- la larghezza di un rettangolo,
- un angolo di rotazione,
- il componente rosso di un colore R, G, B.
- tutto ciò che può essere espresso come un numero.

"durata" deve poter essere espressa come un numero e può anche essere una varietà di cose:

- un periodo di tempo,
- una distanza da percorrere,
- una quantità di loop di animazione da eseguire,
- tutto ciò che può essere espresso come

"In modo non uniforme" significa che la variabile progredisce dall'inizio alla fine in modo non uniforme:

- più veloce all'inizio e più lento alla fine - o viceversa,

- supera il finale ma torna alla fine alla fine della durata,
- avanza ripetutamente / si arresta elasticamente durante la durata,
- "rimbalza" sul finale mentre viene a riposare al termine della durata.

Attribuzione: Robert Penner ha creato il "gold standard" delle funzioni di andamento.

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

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

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

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

Esempio di utilizzo:

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

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

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

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

Imposta la frequenza fotogrammi utilizzando requestAnimationFrame

L'utilizzo di requestAnimationFrame può, su alcuni sistemi, aggiornarsi a più frame al secondo rispetto ai 60fps. 60fps è la frequenza predefinita se il rendering può tenere il passo. Alcuni sistemi funzioneranno a 120fps forse di più.

Se si utilizza il metodo seguente, è necessario utilizzare solo frequenze fotogrammi che sono divisioni intere di 60, in modo che `(60 / FRAMES_PER_SECOND) % 1 === 0` sia `true` o si ottengano frequenze fotogrammi incoerenti.

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

Animazione da [x0, y0] a [x1, y1]

Usa i vettori per calcolare incrementale [x, y] da [startX, startY] a [endX, endY]

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

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

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

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

Codice di esempio:

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

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

// start animation loop running
requestAnimationFrame(animate);

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

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

Leggi Animazione online: https://riptutorial.com/it/html5-canvas/topic/4822/animazione

Capitolo 3: Cancellare lo schermo

Sintassi

- void clearRect (x, y, width, height)
- ImageData createImageData (larghezza, altezza)

Osservazioni

Nessuno di questi metodi produrrà pixel trasparenti se il contesto è stato creato con il parametro `alpha: false` .

Examples

rettangoli

È possibile utilizzare il metodo `clearRect` per cancellare qualsiasi sezione rettangolare dell'area di `clearRect` .

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

> **Nota:** `clearRect` dipende dalla matrice di trasformazione.

Per far fronte a questo, è possibile ripristinare la matrice di trasformazione prima di cancellare la tela.

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

> **Nota:** `ctx.save` e `ctx.restore` vengono richiesti solo se si desidera mantenere lo stato del contesto 2D della tela. In alcune situazioni, il salvataggio e il ripristino possono essere lenti e in generale dovrebbero essere evitati se non richiesti.

Dati immagine grezza

È possibile scrivere direttamente sui dati dell'immagine renderizzata usando `putImageData` . Creando nuovi dati immagine e assegnandoli alla tela, si cancella l'intero schermo.

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

Nota: `putImageData` non è interessato da alcuna trasformazione applicata al contesto. Scriverà i dati

direttamente nella regione dei pixel sottoposti a rendering.

Forme complesse

È possibile cancellare regioni di forme complesse modificando la proprietà `globalCompositeOperation` .

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

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

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

Cancella la tela con il gradiente.

Piuttosto che usare `clearRect` che rende trasparenti tutti i pixel, potresti volere uno sfondo.

Per cancellare con un gradiente

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

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

Questo è circa la metà di `0.008ms` veloci come clearRect `0.004ms` ma i 4millions di secondo non dovrebbero avere un impatto negativo su alcuna animazione in tempo reale. (I tempi variano in base al dispositivo, alla risoluzione, al browser e alla configurazione del browser. I tempi sono solo per il confronto)

Cancella la tela usando l'operazione composita

Cancella la tela usando l'operazione di compositing. Ciò cancellerà la tela indipendentemente dalle trasformazioni ma non è veloce come `clearRect()` .

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

qualsiasi cosa successiva cancellerà il contenuto precedente.

Leggi Cancellare lo schermo online: https://riptutorial.com/it/html5-canvas/topic/5245/cancellare-

lo-schermo

Capitolo 4: Collisioni e intersezioni

Examples

Ci sono 2 cerchi in collisione?

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

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

I 2 rettangoli si scontrano?

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

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

Un cerchio e un rettangolo si scontrano?

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

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

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

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

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

Sono 2 i segmenti di linea che intercettano?

La funzione in questo esempio restituisce `true` se due segmenti di linea si intersecano e `false` caso contrario.

L'esempio è progettato per le prestazioni e utilizza la chiusura per contenere le variabili di lavoro

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

        var v1, v2, v3, cross, u1, u2;  // working variable are closed over so they do not
need creation
                                        // each time the function is called. This gives a
significant performance boost.
        v1 = {x : null, y : null}; // line p0, p1 as vector
        v2 = {x : null, y : null}; // line p2, p3 as vector
        v3 = {x : null, y : null}; // the line from p0 to p2 as vector

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

Esempio di utilizzo

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

L'esempio è facilmente modificato per restituire il punto di intercettazione. Sostituire il codice tra `code point A` e `A end` con

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

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

O se si desidera ottenere il punto di intercettazione sulle linee, ignorando i segmenti di linea iniziano e terminano sostituiscono il codice tra i `code point B` e `B end` con

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

Entrambe le modifiche restituiranno false se non ci sono intercettazioni o restituiscono il punto di intercettazione come `{x : xCoord, y : yCoord}`

Un segmento di linea e un cerchio entrano in collisione?

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

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

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

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

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

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

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

I segmenti di linea e i rettangoli si scontrano?

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

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

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

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

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

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

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

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

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

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

    return(isIntersecting);
}
```

I 2 poligoni convessi si scontrano?

Usa il Teorema dell'asse di separazione per determinare se 2 poligoni convessi si intersecano

I POLIGONI DEVONO ESSERE CONVESSI

Attribuzione: Markus Jarderot @ Come controllare l'intersezione tra 2 rettangoli ruotati?

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

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

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

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

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

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

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

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

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

```
};
```

Sono 2 i poligoni in collisione? (sono consentiti sia poligoni concavi che convessi)

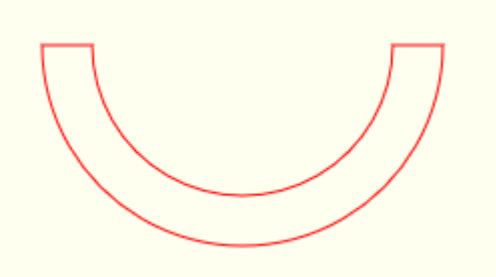
Verifica tutti i lati del poligono per le intersezioni per determinare se 2 poligoni sono in collisione.

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

function polygonsCollide(p1,p2){
    // turn vertices into line points
    var lines1=verticesToLinePoints(p1);
    var lines2=verticesToLinePoints(p2);
    // test each poly1 side vs each poly2 side for intersections
    for(i=0; i<lines1.length; i++){
    for(j=0; j<lines2.length; j++){
        // test if sides intersect
        var p0=lines1[i][0];
        var p1=lines1[i][1];
        var p2=lines2[j][0];
        var p3=lines2[j][1];
        // found an intersection -- polys do collide
        if(lineSegmentsCollide(p0,p1,p2,p3)){return(true);}
    }}
    // none of the sides intersect
    return(false);
}
// helper: turn vertices into line points
function verticesToLinePoints(p){
    // make sure polys are self-closing
    if(!(p[0].x==p[p.length-1].x && p[0].y==p[p.length-1].y)){
        p.push({x:p[0].x,y:p[0].y});
    }
    var lines=[];
    for(var i=1;i<p.length;i++){
        var p1=p[i-1];
        var p2=p[i];
        lines.push([
            {x:p1.x, y:p1.y},
            {x:p2.x, y:p2.y}
        ]);
    }
    return(lines);
}
// helper: test line intersections
// point object: {x:, y:}
// p0 & p1 form one segment, p2 & p3 form the second segment
// Get interseting point of 2 line segments (if any)
// Attribution: http://paulbourke.net/geometry/pointlineplane/
function lineSegmentsCollide(p0,p1,p2,p3) {
    var unknownA = (p3.x-p2.x) * (p0.y-p2.y) - (p3.y-p2.y) * (p0.x-p2.x);
    var unknownB = (p1.x-p0.x) * (p0.y-p2.y) - (p1.y-p0.y) * (p0.x-p2.x);
    var denominator  = (p3.y-p2.y) * (p1.x-p0.x) - (p3.x-p2.x) * (p1.y-p0.y);

    // Test if Coincident
```

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

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

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

    return(isIntersecting);
}
```

C'è un punto X, Y all'interno di un arco?

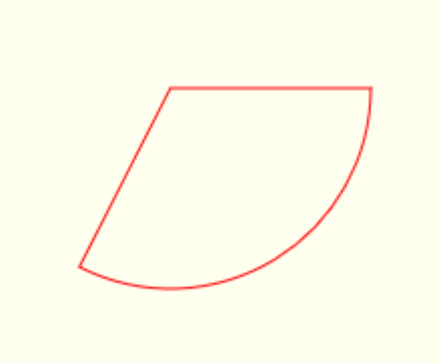
Verifica se il punto [x, y] si trova all'interno di un arco chiuso.



```
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);
}
```

C'è un punto X, Y all'interno di un cuneo?

Verifica se il punto [x, y] si trova all'interno di un cuneo.



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

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

C'è un punto X, Y all'interno di un cerchio?

Verifica se un punto [x, y] si trova all'interno di un cerchio.

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

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

È un punto X, Y all'interno di un rettangolo?

Verifica se un punto [x, y] si trova all'interno di un rettangolo.

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

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

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

Leggi Collisioni e intersezioni online: https://riptutorial.com/it/html5-canvas/topic/5017/collisioni-e-intersezioni

Capitolo 5: compositing

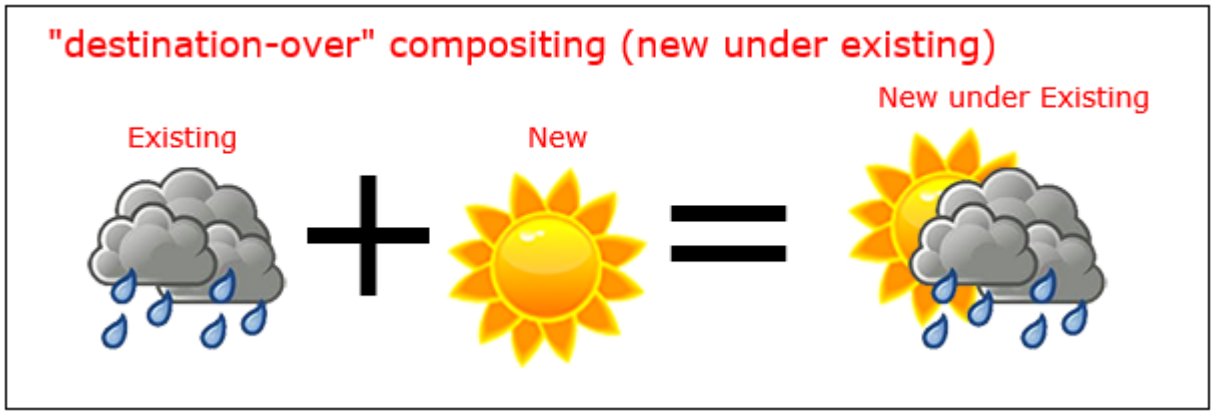
Examples

Disegna dietro le forme esistenti con "destination-over"

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

Il compositing "destination-over" posiziona il nuovo disegno *sotto i disegni esistenti* .

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



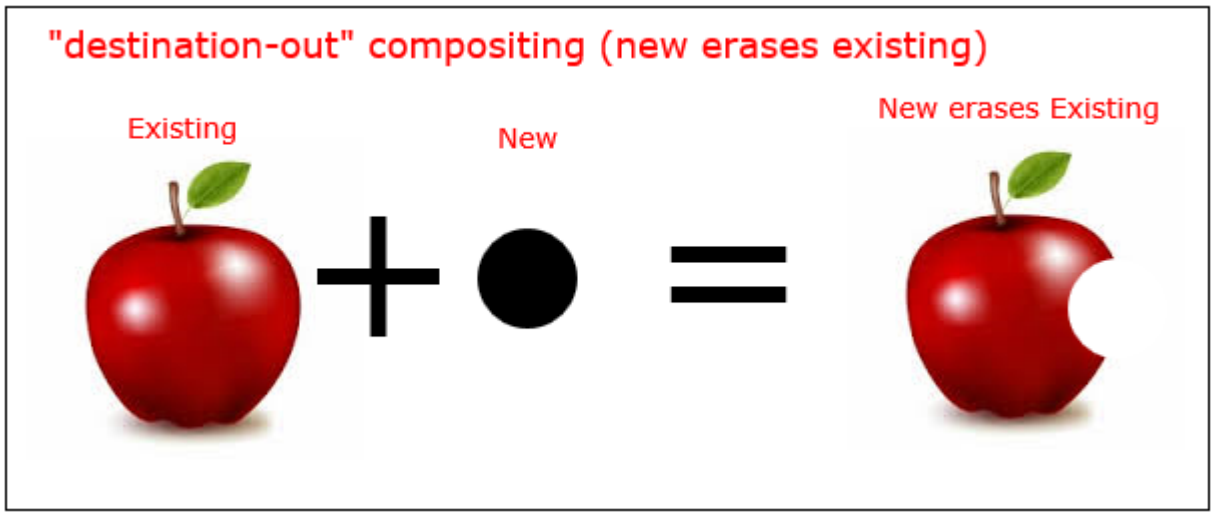
Cancellare le forme esistenti con "destinazione-out"

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

Il compositing "destination-out" utilizza nuove forme per cancellare i disegni esistenti.

La nuova forma non viene effettivamente disegnata, ma viene semplicemente utilizzata come "cookie-cutter" per cancellare i pixel esistenti.

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

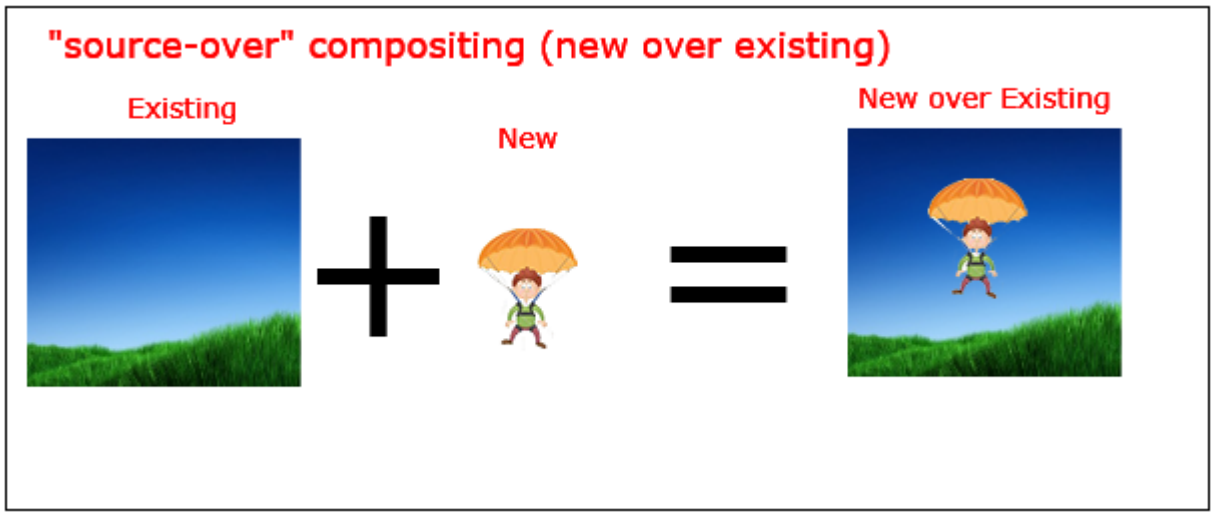


Compositing predefinito: nuove forme sono disegnate su forme esistenti

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

Compositing "source-over" **[predefinito]** , colloca tutti i nuovi disegni su qualsiasi disegno esistente.

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



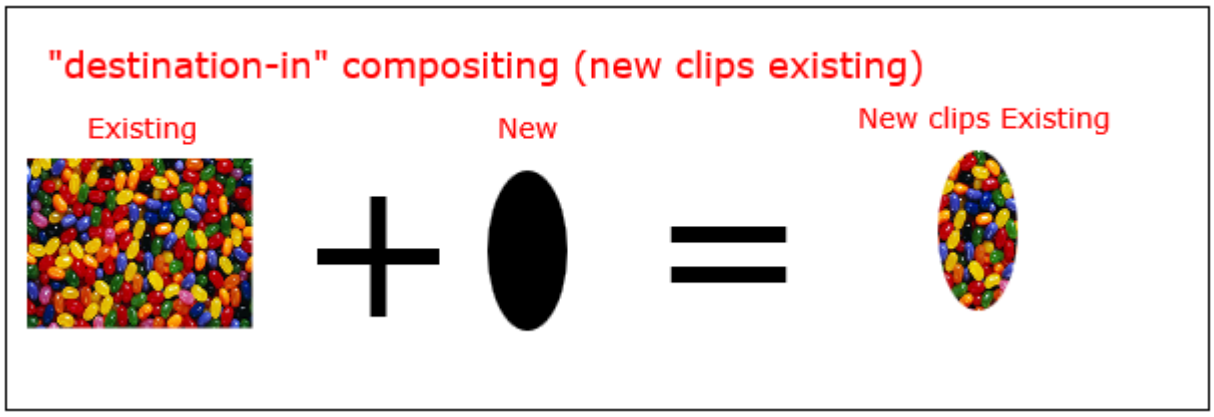
Ritaglia immagini all'interno di forme con "destinazione in entrata"

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

"destination-in" compositing clip disegni esistenti all'interno di una nuova forma.

Nota: qualsiasi parte del disegno esistente che cade all'esterno del nuovo disegno viene cancellata.

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



Ritaglia immagini all'interno di forme con "sorgente d'ingresso"

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

`source-in` compositing `source-in` ritaglia nuovi disegni all'interno di una forma esistente.

Nota: qualsiasi parte del nuovo disegno che cade all'esterno del disegno esistente viene cancellata.

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

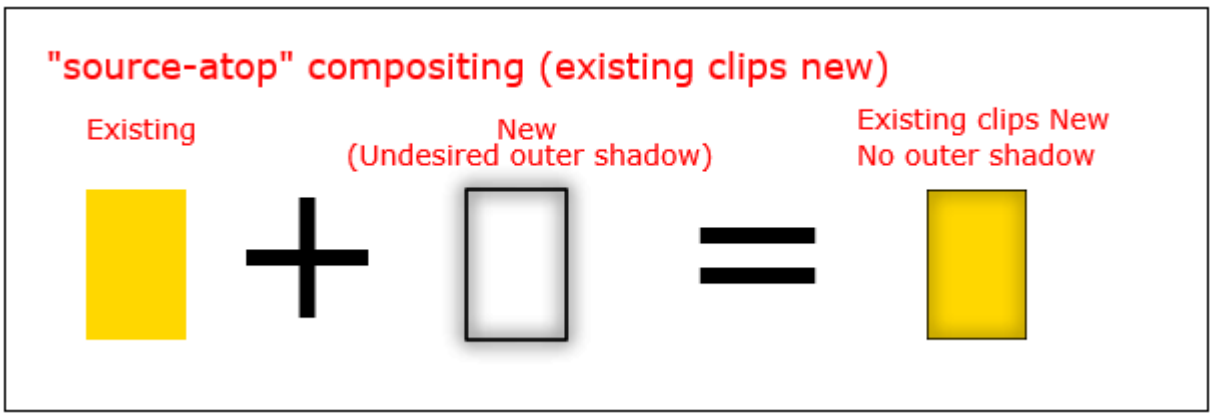
Ombre interne con "source-atop"

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

`source-atop` compositing clip nuova immagine all'interno di una forma esistente.

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

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



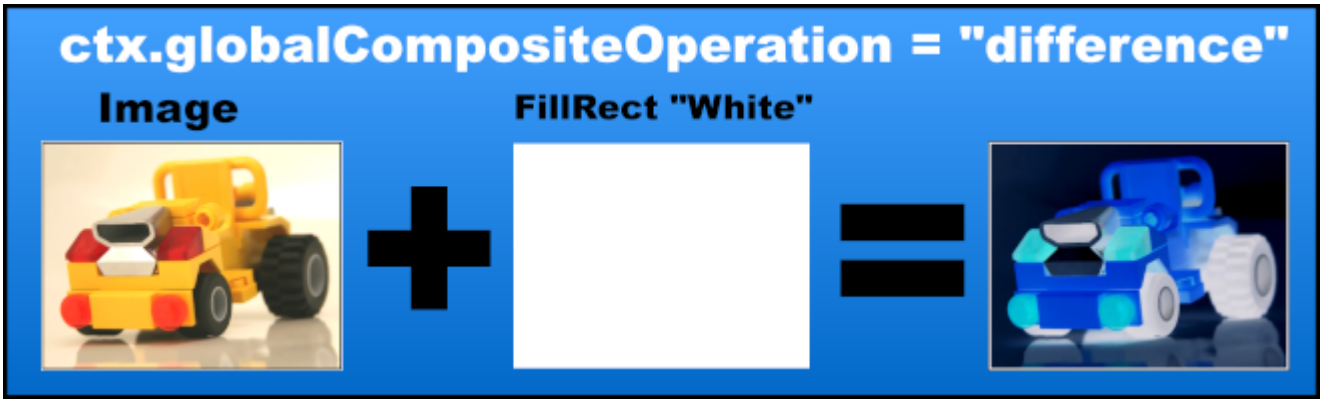
Inverti o nega l'immagine con "differenza"

Rendering di un rettangolo bianco su un'immagine con l'operazione composita

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

La quantità dell'effetto può essere controllata con l'impostazione alfa

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

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



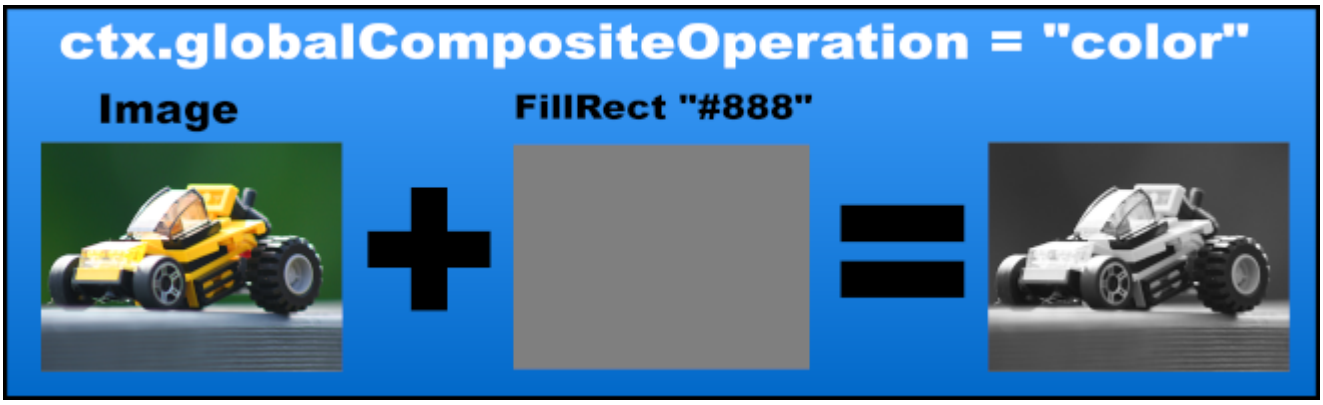
Bianco e nero con "colore"

Rimuovi il colore da un'immagine tramite

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

La quantità dell'effetto può essere controllata con l'impostazione alfa

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

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



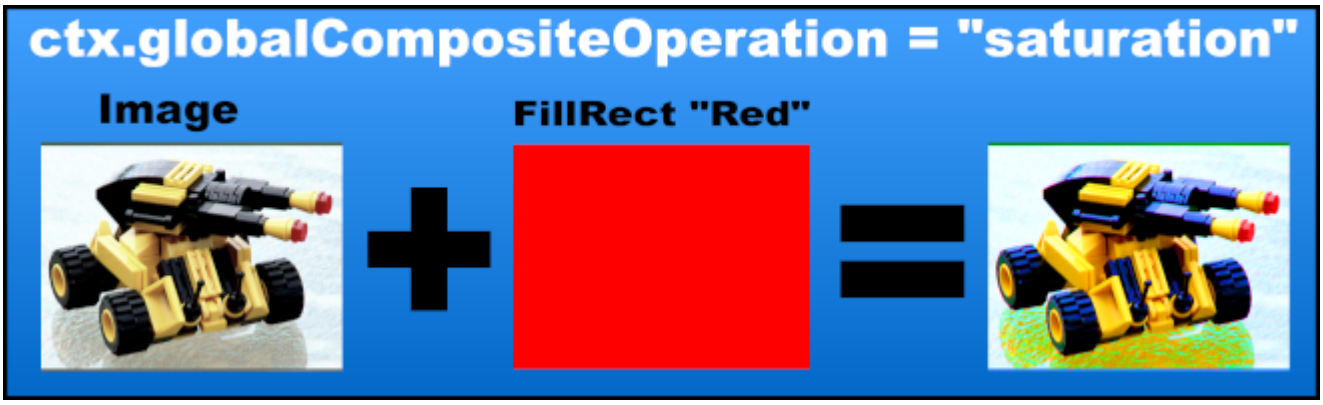
Aumentare il contrasto del colore con "saturazione"

Aumentare il livello di saturazione di un'immagine con

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

La quantità dell'effetto può essere controllata con l'impostazione alfa o la quantità di saturazione nella sovrapposizione di riempimento

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

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



Seppia FX con "luminosità"

Crea un seppia colorato con

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

In questo caso il colore seppia viene reso prima l'immagine.

La quantità dell'effetto può essere controllata con l'impostazione alfa o la quantità di saturazione nella sovrapposizione di riempimento

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

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

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

Cambia opacità con "globalAlpha"

```
context.globalAlpha=0.50
```

È possibile modificare l'opacità dei nuovi disegni impostando `globalAlpha` su un valore compreso tra 0,00 (completamente trasparente) e 1.00 (completamente opaco).

Il valore predefinito `globalAlpha` è 1.00 (completamente opaco).

I disegni esistenti non sono influenzati da `globalAlpha` .

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

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

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

Leggi compositing online: https://riptutorial.com/it/html5-canvas/topic/5547/compositing

Capitolo 6: Design reattivo

Examples

Creazione di una tela a pagina intera reattiva

Codice di avviamento per creare e rimuovere un'area di disegno pagina completa che risponde a ridimensionare gli eventi tramite javascript.

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

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

Se non hai più bisogno della tela, puoi rimuoverla chiamando `removeCanvas()`

Una demo di questo esempio su jsfiddle

Coordinate del mouse dopo il ridimensionamento (o lo scorrimento)

Le app su tela spesso si basano molto sull'interazione dell'utente con il mouse, ma quando la

finestra viene ridimensionata, le coordinate del mouse su cui si basa il canvas sono probabilmente modificate perché il ridimensionamento fa sì che l'area di disegno venga sfalsata in una posizione diversa rispetto alla finestra. Pertanto, la progettazione reattiva richiede che la posizione di offset della tela venga ricalcolata quando la finestra viene ridimensionata e anche ricalcolata quando si scorre la finestra.

Questo codice è in ascolto degli eventi di ridimensionamento delle finestre e ricalcola gli offset utilizzati nei gestori di eventi del mouse:

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

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

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

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

Animazioni su tela reattive senza ridimensionare gli eventi.

Gli eventi di ridimensionamento della finestra possono essere attivati in risposta al movimento del dispositivo di input dell'utente. Quando ridimensioni un quadro, questo viene automaticamente cancellato e sei costretto a ri-renderizzare il contenuto. Per le animazioni fai questo ogni fotogramma tramite la funzione del ciclo principale chiamata da `requestAnimationFrame` che fa del suo meglio per mantenere il rendering sincronizzato con l'hardware del display.

Il problema con l'evento di ridimensionamento è che quando il mouse viene utilizzato per ridimensionare la finestra, gli eventi possono essere attivati molte volte più velocemente rispetto alla velocità standard di 60 fps del browser. Quando l'evento di ridimensionamento termina, il buffer di back nell'area di disegno viene presentato al DOM non sincronizzato con il dispositivo di visualizzazione, che può causare cesoie e altri effetti negativi. C'è anche un sacco di allocazione e rilascio della memoria inutili che possono ulteriormente incidere sull'animazione quando GC si ripulisce qualche tempo dopo.

Evento di ridimensionamento rimbalzato

Un modo comune per gestire le alte velocità di attivazione dell'evento di ridimensionamento è quello di eliminare l'evento di ridimensionamento.

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

// debounce timeout handle
var debounceTimeoutHandle;

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

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

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

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

L'esempio precedente ritarda il ridimensionamento della tela fino a 100ms dopo l'evento di ridimensionamento. Se in quel momento vengono attivati ulteriori eventi di ridimensionamento, il timeout di ridimensionamento esistente viene annullato e ne viene pianificato uno nuovo. Questo effettivamente consuma la maggior parte degli eventi di ridimensionamento.

Ha ancora alcuni problemi, il più notevole è il ritardo tra il ridimensionamento e la visualizzazione della tela ridimensionata. La riduzione del tempo di rimbalzo migliora questo ma il ridimensionamento non è ancora sincronizzato con il dispositivo di visualizzazione. Hai anche il rendering dell'anello principale dell'animazione su una tela inadeguata.

Più codice può ridurre i problemi! Più codice crea anche i suoi nuovi problemi.

Semplice e il migliore ridimensionamento

Avendo provato molti modi diversi per appianare il ridimensionamento della tela, dall'assurdamente complesso, per ignorare il problema (chi se ne importa comunque?) Mi sono ricollegato ad un fidato amico.

KISS è qualcosa di cui la maggior parte dei programmatori dovrebbe essere a conoscenza ((**K** eep **I** t **S** imple **S** tupid) *Lo stupido si riferisce a me per non averci pensato anni fa.*) E si scopre che la soluzione migliore è la più semplice di tutte.

Ridimensiona la tela dall'interno del ciclo dell'animazione principale. Rimane sincronizzato con il dispositivo di visualizzazione, non c'è rendering inutile e la gestione delle risorse è al minimo possibile mantenendo la frequenza fotogrammi completa. Né è necessario aggiungere un evento di ridimensionamento alla finestra o altre funzioni di ridimensionamento aggiuntive.

Si aggiunge il ridimensionamento in cui normalmente si cancella la tela controllando se le

dimensioni della tela corrispondono alle dimensioni della finestra. Se non lo ridimensiona.

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

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

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

    // Animation code as normal.



    requestAnimationFrame(mainLoop);
}
```

Leggi Design reattivo online: https://riptutorial.com/it/html5-canvas/topic/5495/design-reattivo

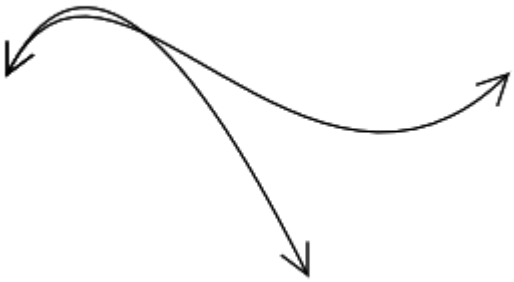
Capitolo 7: Grafici e diagrammi

Examples

Linea con punte di freccia

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

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

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

Curva di Bézier cubica e quadratica con punte di freccia

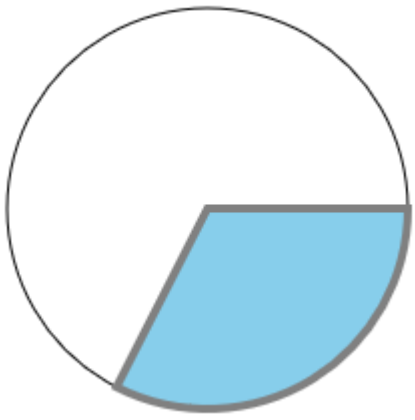


```
// Usage:
var p0={x:50,y:100};
var p1={x:100,y:0};
var p2={x:200,y:200};
var p3={x:300,y:100};

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

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

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

quadraticCurveArrowHeads(p0, p1, p2, 15);



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

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

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

    ctx.stroke();
}

function cubicCurveArrowHeads(p0, p1, p2, p3, arrowLength, hasStartArrow, hasEndArrow) {
    bezWithArrowheads(p0, p1, p2, p3, arrowLength, hasStartArrow, hasEndArrow);
}
function quadraticCurveArrowHeads(p0, p1, p2, arrowLength, hasStartArrow, hasEndArrow) {
    bezWithArrowheads(p0, p1, p2, undefined, arrowLength, hasStartArrow, hasEndArrow);
}
```

Cuneo

Il codice disegna solo il cuneo ... il cerchio disegnato qui solo per la prospettiva.



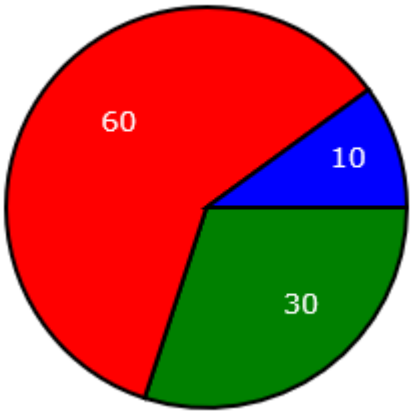
```
// Usage
var wedge={
    cx:150, cy:150,
    radius:100,
    startAngle:0,
    endAngle:Math.PI*.65
}
```

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

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

Arco con riempimento e corsa

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

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

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

Grafico a torta con demo




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

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

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

    pieChart(myData, myColor);

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

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

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

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

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

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

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

Leggi Grafici e diagrammi online: https://riptutorial.com/it/html5-canvas/topic/5492/grafici-e-diagrammi

Capitolo 8: immagini

Examples

Ritaglio di immagini con canvas

Questo esempio mostra una semplice funzione di ritaglio di immagini che prende un'immagine e ritaglia le coordinate e restituisce l'immagine ritagliata.

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

Usare

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

La tela Tained

Quando aggiungi contenuti da fonti esterne al tuo dominio o dal file system locale, il canvas viene contrassegnato come contaminato. Tentare di accedere ai dati dei pixel o convertirli in un dataURL genererà un errore di sicurezza.

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

Questo esempio è solo uno stub per attirare qualcuno con una comprensione dettagliata

elaborata.

"Context.drawImage" non visualizza l'immagine sulla tela?

Assicurati che il tuo oggetto immagine sia completamente caricato prima di provare a disegnarlo sulla tela con `context.drawImage` . In caso contrario, l'immagine non verrà visualizzata automaticamente.

In JavaScript, le immagini non vengono caricate immediatamente. Invece, le immagini vengono caricate in modo asincrono e durante il tempo necessario per caricare JavaScript continua ad eseguire qualsiasi codice che segue `image.src` . Ciò significa che `context.drawImage` può essere eseguito con un'immagine vuota e pertanto non visualizzerà nulla.

Esempio assicurandosi che l'immagine sia completamente caricata prima di provare a disegnarla con .drawImage

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

Esempio di caricamento di più immagini prima di tentare di disegnare con una di esse

Esistono più caricatori di immagini con funzioni complete, ma questo esempio illustra come farlo

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

Ridimensionamento dell'immagine per adattarsi o riempire.

Ridimensionamento per adattarsi

Significa che l'intera immagine sarà visibile ma potrebbe esserci dello spazio vuoto sui lati o superiore e inferiore se l'immagine non ha lo stesso aspetto della tela. L'esempio mostra l'immagine ridimensionata per adattarsi. Il blu sui lati è dovuto al fatto che l'immagine non ha lo stesso aspetto della tela.

Ridimensionamento per riempire

Significa che l'immagine viene ridimensionata in modo che tutti i pixel del canvas vengano coperti dall'immagine. Se l'aspetto dell'immagine non è lo stesso dell'area di disegno, alcune parti dell'immagine verranno ritagliate. L'esempio mostra l'immagine ridimensionata per riempire. Nota come la parte superiore e inferiore dell'immagine non sono più visibili.

Esempio Scala per adattarsi

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

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

```
    ctx.drawImage(img, x, y, img.width * scale, img.height * scale);
}
```

Esempio di scala da riempire

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

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

L'unica differenza tra le due funzioni è la scala. L'adattamento utilizza la scala di adattamento minima per il riempimento che utilizza la scala di adattamento massima.

Leggi immagini online: https://riptutorial.com/it/html5-canvas/topic/3210/immagini

Capitolo 9: Manipolazione pixel con "getImageData" e "putImageData"

Examples

Introduzione a "context.getImageData"

Html5 Canvas ti dà la possibilità di recuperare e cambiare il colore di qualsiasi pixel sulla tela.

Puoi utilizzare la manipolazione dei pixel di Canvas per:

- Crea un selettore di colori per un'immagine o seleziona un colore su una ruota dei colori.
- Crea filtri immagine complessi come sfocatura e rilevamento dei bordi.
- Ricolora qualsiasi parte di un'immagine a livello di pixel (se utilizzi HSL puoi anche ricolorare un'immagine pur conservando l'illuminazione e la saturazione importanti in modo che il risultato non assomigli a qualcuno che schiaffeggia l'immagine). Nota: Canvas ora dispone di Blend Compositing che può anche ricolorare un'immagine in alcuni casi.
- "Elimina" lo sfondo attorno a una persona / elemento in un'immagine,
- Crea uno strumento paint-bucket per rilevare e Floodfill parte di un'immagine (ad esempio, cambia il colore di un petalo selezionato dall'utente da verde a giallo).
- Esaminare un'immagine per contenuto (ad esempio riconoscimento facciale).

Problemi comuni:

- Per motivi di sicurezza, `getImageData` è disabilitato se hai disegnato un'immagine proveniente da un dominio diverso dalla stessa pagina web.
- `getImageData` è un metodo relativamente costoso perché crea una grande matrice di pixel-data e perché non utilizza la GPU per supportare i suoi sforzi. Nota: Canvas ora ha un mix compositing che può eseguire la stessa manipolazione dei pixel `getImageData` da `getImageData` .
- Per le immagini .png, `getImageData` potrebbe non riportare esattamente gli stessi colori del file .png originale perché al browser è consentito eseguire la correzione gamma e la premoltiplicazione alfa quando si disegnano immagini sul quadro.

Ottenere colori pixel

Usa `getImageData` per recuperare i colori dei pixel per tutto o parte del tuo contenuto su tela.

Il metodo `getImageData` restituisce un oggetto `imageData`

L'oggetto `imageData` ha una proprietà `.data` che contiene le informazioni sul colore dei pixel.

La proprietà `data` è un `Uint8ClampedArray` contenente i dati di colore Rosso, Verde, Blu e Alfa (opacità) per tutti i pixel richiesti.

```
// determine which pixels to fetch (this fetches all pixels on the canvas)
```

```
var x=0;
var y=0;
var width=canvas.width;
var height=canvas.height;

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

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

Puoi ottenere la posizione di qualsiasi [x, y] pixel all'interno di `data` array di `data` come questo:

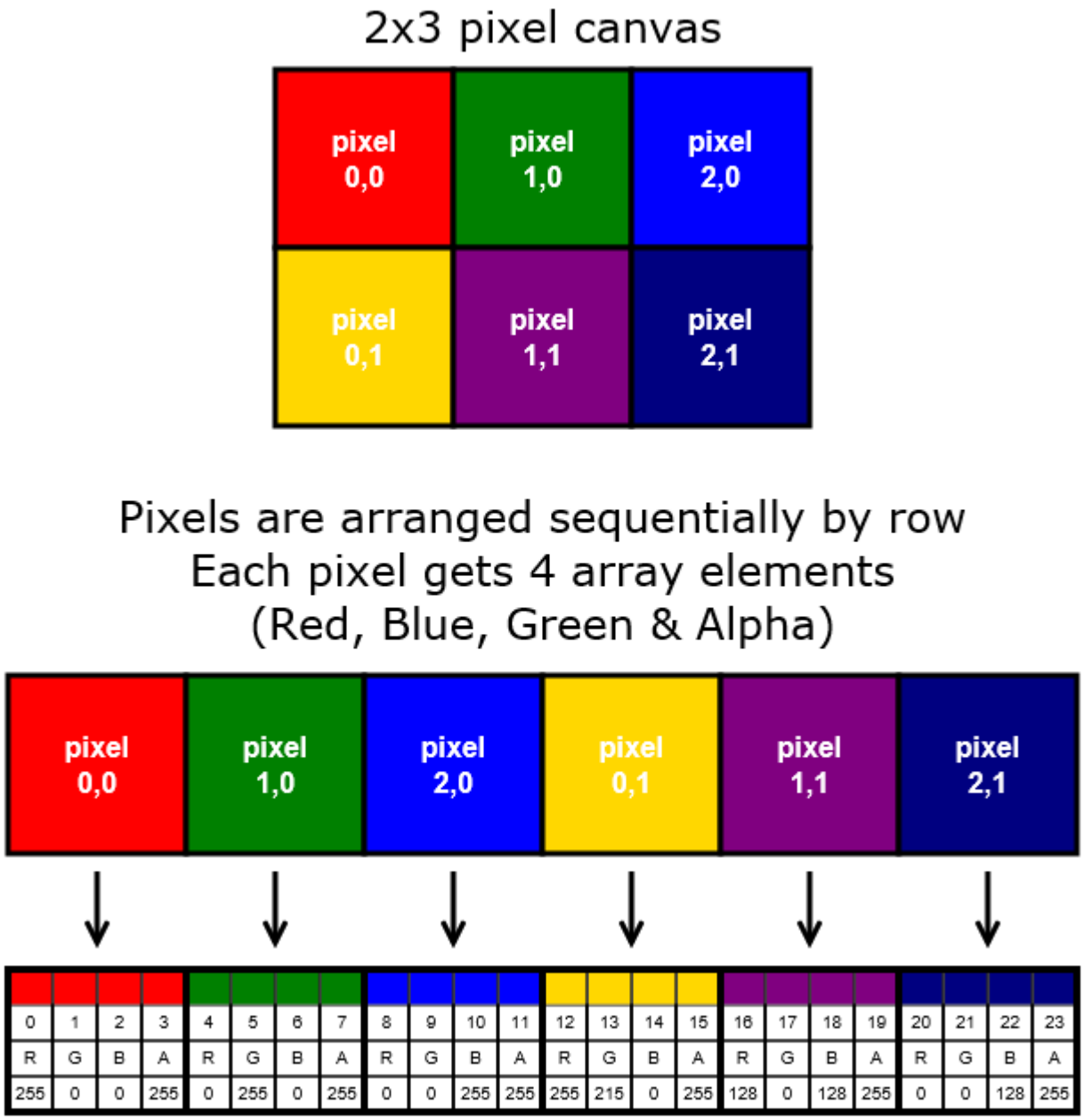
```
// the data[] array position for pixel [x,y]
var n = y * canvas.width + x;
```

E poi puoi recuperare i valori di rosso, verde, blu e alfa di quel pixel come questo:

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

Un'illustrazione che mostra come è strutturato l'array di dati del pixel

`context.getImageData` è illustrato di seguito per un piccolo canvas di dimensioni 2x3 pixel:



Leggi Manipolazione pixel con "getImageData" e "putImageData" online: https://riptutorial.com/it/html5-canvas/topic/5573/manipolazione-pixel-con--getimagedata--e--putimagedata-

Capitolo 10: Navigazione lungo un percorso

Examples

Trovare punti lungo una curva cubica di Bezier

Questo esempio trova una serie di punti approssimativamente equidistanti lungo una curva cubica di Bezier.

Decompone i segmenti Path creati con `context.bezierCurveTo` in punti lungo quella curva.

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

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

Trovare punti lungo una curva quadratica

Questo esempio trova una serie di punti approssimativamente equidistanti lungo una curva quadratica.

Scompone i segmenti Path creati con `context.quadraticCurveTo` in punti lungo quella curva.

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

Trovare punti lungo una linea

Questo esempio trova una serie di punti approssimativamente equidistanti lungo una linea.

Scompone i segmenti Path creati con `context.lineTo` in punti lungo quella linea.

```
// 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);
}
```

Trovare punti lungo un intero percorso contenente curve e linee

Questo esempio trova una serie di punti approssimativamente equidistanti lungo un intero percorso.

Decompone tutti i segmenti Path creati con `context.lineTo` , `context.quadraticCurveTo` e / o `context.bezierCurveTo` in punti lungo quel Path.

uso

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

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

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

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

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

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

Un plug-in che calcola automaticamente i punti lungo il percorso

Questo codice modifica i comandi di disegno di Canvas Context in modo che i comandi non solo disegnino la linea o la curva, ma creino anche una serie di punti lungo l'intero percorso:

- BeginPath,
- moveTo,
- lineTo,
- quadraticCurveTo,
- bezierCurveTo.

Nota importante!

Questo codice modifica le effettive funzioni di disegno del contesto, quindi quando si `stopPlottingPathCommands` di `stopPlottingPathCommands` lungo il percorso, è necessario chiamare `stopPlottingPathCommands` fornito per restituire le funzioni di disegno di contesto allo stato non modificato.

Lo scopo di questo contesto modificato è quello di consentire di "plug-in" il calcolo dell'array di punti nel codice esistente senza dover modificare i comandi di disegno Path esistenti. Ma non è necessario utilizzare questo Contesto modificato: è possibile chiamare separatamente le singole funzioni che decompongono una linea, una curva quadratica e una curva cubica di Bézier e quindi concatenare manualmente quei singoli array di punti in un singolo punto-array per l'intero percorso.

Si recupera una copia dell'array punti risultante utilizzando la funzione `getPathPoints` fornita.

Se disegni più percorsi con il contesto modificato, l'array punti conterrà un singolo insieme di punti

concatenato per tutti i percorsi multipli disegnati.

Se, invece, si desidera ottenere array di punti separati, è possibile recuperare l'array corrente con `getPathPoints` e quindi cancellare tali punti dall'array con la funzione `clearPathPoints` fornita.

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

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

Una demo completa:

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

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

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

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

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

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


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

```
////////////////////////////////////

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

```
        this.myBeginPath=undefined;
    }
}


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

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

// Return: an array of approximately evenly spaced points along a Quadratic curve
//
// Attribution: Stackoverflow's @Blindman67
```

```
// Cite: http://stackoverflow.com/questions/36637211/drawing-a-curved-line-in-css-or-canvas-
and-moving-circle-along-it/36827074#36827074
// As modified from the above citation
//
// ptCount: sample this many points at interval along the curve
// pxTolerance: approximate spacing allowed between points
// Ax,Ay,Bx,By,Cx,Cy: control points defining the curve
//
function plotQBez(ptCount,pxTolerance,Ax,Ay,Bx,By,Cx,Cy){
    var deltaBAx=Bx-Ax;
    var deltaCBx=Cx-Bx;
    var deltaBAy=By-Ay;
    var deltaCBy=Cy-By;
    var ax,ay;
    var lastX=-10000;
    var lastY=-10000;
    var pts=[{x:Ax,y:Ay}];
    for(var i=1;i<ptCount;i++){
        var t=i/ptCount;
        ax=Ax+deltaBAx*t;
        ay=Ay+deltaBAy*t;
        var x=ax+((Bx+deltaCBx*t)-ax)*t;
        var y=ay+((By+deltaCBy*t)-ay)*t;
        var dx=x-lastX;
        var dy=y-lastY;
        if(dx*dx+dy*dy>pxTolerance){
            pts.push({x:x,y:y});
            lastX=x;
            lastY=y;
        }
    }
    pts.push({x:Cx,y:Cy});
    return(pts);
}

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

```
}
```

Lunghezza di una curva quadratica

Dati i 3 punti di una curva quadratica, la seguente funzione restituisce la lunghezza.

```
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);
}
```

Derivato dalla funzione quadratica di bezier F (t) = a * (1 - t) ^{2} + 2 * b * (1 - t) * t + c * t ^{2}

Dividere le curve più bezier in posizione

Questo esempio divide in due le curve cubiche e più bezier.

La funzione `splitCurveAt` divide la curva nella `position` cui `0.0` = start, `0.5` = middle e `1` = end. Può dividere le curve quadratiche e cubiche. Il tipo di curva è determinato dall'ultimo x argomento `x4` . Se non è `undefined` o `null` allora assume che la curva sia cubica altrimenti la curva è una quadratica

Esempio di utilizzo

Dividere la curva quadratica di Bezier in due

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

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

```
ctx.moveTo(p[i++],p[i++]);
ctx.quadraticCurveTo(p[i++], p[i++], p[i++], p[i++]);
ctx.quadraticCurveTo(p[i++], p[i++], p[i++], p[i++]);
ctx.stroke();
```

Dividere la curva bezier cubica in due

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

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

La funzione di divisione

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

> **Nota: gli** argomenti all'interno di [x4, y4] sono facoltativi.
>
> **Nota:** la funzione ha un codice `/* */` commentato opzionale che si occupa di casi limite in cui le curve risultanti possono avere lunghezza zero o non rientrare all'inizio o alla fine della curva originale. Poiché sta tentando di dividere una curva al di fuori dell'intervallo valido per `position >= 0` o `position >= 1` verrà generato un errore di intervallo. Questo può essere rimosso e funzionerà bene, anche se potresti avere curve risultanti che hanno lunghezza zero.

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

    //
================================================================================================
    // you may remove this as the function will still work and resulting curves will still
render
    // but other curve functions may not like curves with 0 length
    //
================================================================================================
    if(position <= 0 || position >= 1){
```

```
        throw RangeError("spliteCurveAt requires position > 0 && position < 1");
    }

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

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

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


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

    if(quad){ // split quadratic bezier
        retPoints[i++] = (v1.x += (v2.x - v1.x) * c);  // new control point for first curve
```

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

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

Taglia la curva di Bezier.

Questo esempio mostra come tagliare un bezier.

La funzione trimBezier taglia le estremità dalla curva restituendo la curva da `fromPos` a `toPos` . `fromPos` e `toPos` sono `toPos` nell'intervallo compreso tra 0 e 1, possono tagliare le curve quadratiche e cubiche. Il tipo di curva è determinato dall'ultimo x argomento `x4` . Se non è `undefined` o `null` allora assume che la curva sia cubica altrimenti la curva è una quadratica

La curva tagliata viene restituita come una serie di punti. 6 punti per le curve quadratiche e 8 per le curve cubiche.

Esempio di utilizzo

Ritaglio di una curva quadratica.

```
var p1 = {x : 10 , y : 100};
```

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

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

Ritaglio di una curva cubica.

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

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

Esempio di funzione

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

> **Nota: gli** argomenti all'interno di [x4, y4] sono facoltativi.
>
> **Nota:** questa funzione richiede la funzione nell'esempio Split Bezier Curves At in questa sezione

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

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

    }
    i = 6;
    toPos = (toPos - fromPos) / (1 - fromPos);
    s = splitBezierAt(toPos, s[i++], s[i++], s[i++], s[i++], s[i++], s[i++], s[i++], s[i++]);
    i = 0;
    retBez = [s[i++], s[i++], s[i++], s[i++], s[i++], s[i++], s[i++], s[i++]];
    return retBez;
}
```

Lunghezza di una curva Bezier cubica (approssimazione ravvicinata)

Dati i 4 punti di una curva Bezier cubica, la funzione seguente restituisce la sua lunghezza.

Metodo: la lunghezza di una curva Bezier cubica non ha un calcolo matematico diretto. Questo metodo "forza bruta" trova un campionamento di punti lungo la curva e calcola la distanza totale misurata da quei punti.

Precisione: la lunghezza approssimativa è accurata del 99% rispetto alla dimensione di campionamento predefinita di 40.

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


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

```
        x:ax +(bx - ax) * t,
        y:ay +(by - ay) * t
    });
}
```

Trova il punto sulla curva

Questo esempio trova un punto su una curva bezier o cubica nella `position` cui la `position` è la distanza unitaria sulla curva 0 <= `position` <= 1. La posizione viene bloccata sull'intervallo, quindi se i valori <0 o> 1 vengono passati saranno impostare 0,1 rispettivamente.

Passa le coordinate della funzione 6 per il quadratico bezier o 8 per il cubico.

L'ultimo argomento facoltativo è il vettore restituito (punto). Se non viene dato, verrà creato.

Esempio di utilizzo

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

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

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

La funzione

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

> **Nota: gli** argomenti all'interno di [x4, y4] sono facoltativi.
>
> **Nota:** `x4` , `y4` se `null` o `undefined` significa che la curva è un bezier quadratico. `vec` è facoltativo e manterrà il punto restituito se fornito. Altrimenti verrà creato.

```
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;
}
```

Trovare l'estensione della curva quadratica

Quando è necessario trovare il rettangolo di delimitazione di una curva di bezier quadratica, è possibile utilizzare il seguente metodo performante.

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

```
    var brx, bx, x, bry, by, y, px, py;

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

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

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

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

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

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

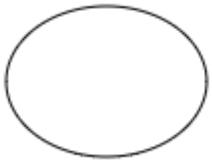
Per uno sguardo più dettagliato a risolvere per esteso vedi risposta Per ottenere l'estensione di un bezier quadratico che include demo eseguibili.

Leggi Navigazione lungo un percorso online: https://riptutorial.com/it/html5-canvas/topic/5281/navigazione-lungo-un-percorso

Capitolo 11: percorsi

Examples

Ellisse



Nota: i browser stanno aggiungendo un comando di disegno `context.ellipse` incorporato, ma questo comando non è universalmente adottato (in particolare non in IE). I metodi seguenti funzionano su tutti i browser.

Disegna un'ellisse data la sua coordinata in alto a sinistra desiderata:

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

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

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

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

Disegna un'ellisse in base alla coordinata del punto centrale desiderata:

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

    ctx.beginPath();
```

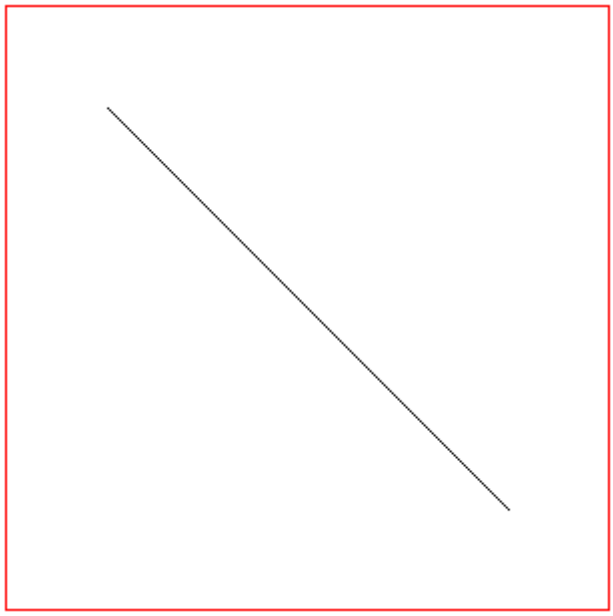
```
    var x = cx + radius * Math.cos(0);
    var y = cy - ratio * radius * Math.sin(0);
    ctx.lineTo(x,y);

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

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

Linea senza sfocature

Quando Canvas disegna una linea, aggiunge automaticamente l'anti-aliasing per curare visivamente la "frastagliatura". Il risultato è una linea meno frastagliata ma più sfocata.

Questa funzione disegna una linea tra 2 punti senza anti-aliasing usando l' algoritmo Bresenham's_line . Il risultato è una linea nitida senza la frastagliatura.

Nota importante: *questo metodo pixel per pixel è un metodo di disegno molto più lento di* `context.lineTo` *.*



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

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

```
    y = Math.floor(y);
    // 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; // error value
    var end = false;
    var x1 = x;
    var y1 = y;

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

Leggi percorsi online: https://riptutorial.com/it/html5-canvas/topic/5133/percorsi

Capitolo 12: Percorso (solo sintassi)

Sintassi

- context.beginPath ()
- context.moveTo (startx, starty)
- context.lineTo (EndX, Endy)
- context.arc (centerX, centerY, radius, startingRadianAngle, endingRadianAngle)
- context.quadraticCurveTo (controlX, controlY, EndX, Endy)
- context.bezierCurveTo (controlX1, controlY1, controlX2, controlY2, EndX, Endy)
- context.arcTo (puntoX1, puntoY1, puntoX2, puntoY2, raggio)
- context.rect (leftX, topY, width, height);
- context.closePath ()

Examples

Panoramica dei comandi di disegno del percorso di base: linee e curve

==========

TODO: collega ciascuno dei comandi di disegno sotto ai loro singoli esempi. Non so come farlo poiché i collegamenti ai singoli esempi puntano verso la cartella "bozza".

TODO: aggiungi esempi per questi comandi "azione" del percorso: stroke (), fill (), clip ()

==========

Sentiero

Un percorso definisce un insieme di linee e curve che possono essere disegnate visibilmente sulla tela.

Un tracciato non viene disegnato automaticamente sulla tela. Ma le linee e le curve del percorso possono essere disegnate sulla tela usando un tratto stilizzato. E la forma creata dalle linee e dalle curve può anche essere riempita con un riempimento stilizzato.

I percorsi hanno usi che vanno oltre il disegno sulla tela:

- Hit test se una coordinata x, y è all'interno della forma del percorso.
- Definire una regione di ritaglio in cui saranno visibili solo i disegni all'interno dell'area di ritaglio. Tutti i disegni all'esterno dell'area di ritaglio non verranno disegnati (== trasparente), in modo simile all'overflow CSS.

I comandi di base del disegno del percorso sono:

- BeginPath
- moveTo

- lineTo
- arco
- quadraticCurveTo
- bezierCurveTo
- arcTo
- rect
- closePath

Descrizione dei comandi di disegno di base:

BeginPath

```
context.beginPath()
```

Inizia l'assemblaggio di un nuovo set di comandi di percorso e scarta anche qualsiasi traccia precedentemente assemblato.

Lo scarto è un punto importante e spesso trascurato. Se non inizi un nuovo percorso, tutti i comandi di percorso precedentemente rilasciati verranno automaticamente ridisegnati.

Sposta anche la "penna" del disegno sull'origine in alto a sinistra della tela (== coordinate [0,0]).

moveTo

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

Sposta la posizione corrente della penna sulla coordinata [startX, inizioY].

Di default tutti i disegni di percorso sono collegati insieme. Quindi il punto finale di una linea o curva è il punto di partenza della linea o curva successiva. Ciò può causare l'estrazione di una linea inaspettata che collega due disegni adiacenti. Il comando `context.moveTo` fondamentalmente "preleva il pennino" e lo posiziona su una nuova coordinata in modo che la linea di collegamento automatica non venga disegnata.

lineTo

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

Disegna un segmento di linea dalla posizione corrente della penna per coordinare [endX, endY]

È possibile assemblare più comandi `.lineTo` per disegnare una polilinea. Ad esempio, è possibile assemblare 3 segmenti di linea per formare un triangolo.

arco

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

Disegna un arco circolare dato un punto centrale, raggio e angoli di inizio e fine. Gli angoli sono espressi come radianti. Per convertire i gradi in radianti puoi usare questa formula: `radians = degrees * Math.PI / 180; .`

L'angolo 0 è rivolto direttamente verso destra dal centro dell'arco. Per disegnare un cerchio completo puoi rendere endingAngle = startingAngle + 360 gradi (360 gradi == Math.PI *2):* `*context.arc (10,10,20,0, Math.PI* 2);

Per impostazione predefinita, l'arco viene disegnato in senso orario, Un parametro opzionale [true | false] indica l'arco da disegnare in senso antiorario: `context.arc(10,10,20,0,Math.PI*2,true)`

quadraticCurveTo

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

Disegna una curva quadratica che inizia dalla posizione corrente della penna fino a una determinata coordinata finale. Un'altra determinata coordinata di controllo determina la forma (curvatura) della curva.

bezierCurveTo

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

Disegna una curva Bezier cubica che inizia dalla posizione corrente della penna fino a una determinata coordinata finale. Altre 2 coordinate di controllo date determinano la forma (curvatura) della curva.

arcTo

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

Disegna un arco circolare con un raggio dato. L'arco viene disegnato in senso orario all'interno del cuneo formato dalla posizione attuale della penna e dato due punti: Punto1 e Punto2.

Una linea che collega la posizione corrente della penna e l'inizio dell'arco viene automaticamente disegnata prima dell'arco.

rect

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

Disegna un rettangolo dato un angolo in alto a sinistra e una larghezza e altezza.

`context.rect` è un comando di disegno univoco perché aggiunge rettangoli disconnessi. Questi rettangoli scollegati non sono automaticamente collegati da linee.

closePath

```
context.closePath()
```

Disegna una linea dalla posizione corrente della penna fino alla coordinata del percorso iniziale.

Ad esempio, se disegni 2 linee che formano 2 rami di un triangolo, `closePath` "chiude" il triangolo disegnando la terza gamba del triangolo dal punto finale della 2a tappa al punto iniziale della prima gamba.

Il nome di questo comando spesso lo rende incompreso. `context.closePath` NON è un delimitatore finale di `context.beginPath` . Di nuovo, il comando `closePath` disegna una linea - non "chiude" un `beginPath` .

lineTo (un comando di percorso)

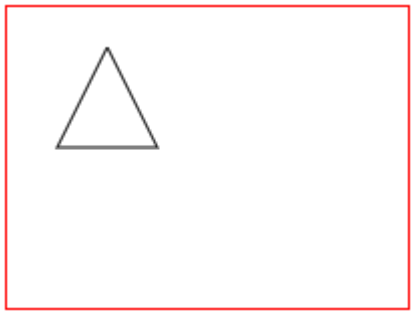
```
context.lineTo(endX, endY)
```

Disegna un segmento di linea dalla posizione corrente della penna per coordinare [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>
```

È possibile assemblare più comandi .line To per disegnare una polilinea. Ad esempio, è possibile assemblare 3 segmenti di linea per formare un triangolo.



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

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

    // arguments
    var topVertexX=50;
    var topVertexY=20;
    var rightVertexX=75;
    var rightVertexY=70;
    var leftVertexX=25;
    var leftVertexY=70;

    // A set of line segments drawn to form a triangle using
    //     "moveTo" and multiple "lineTo" commands
    ctx.beginPath();
    ctx.moveTo(topVertexX,topVertexY);
    ctx.lineTo(rightVertexX,rightVertexY);
    ctx.lineTo(leftVertexX,leftVertexY);
    ctx.lineTo(topVertexX,topVertexY);
    ctx.stroke();

}); // end window.onload
</script>
</head>
<body>
    <canvas id="canvas" width=200 height=150></canvas>
</body>
</html>
```

arco (un comando di percorso)

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

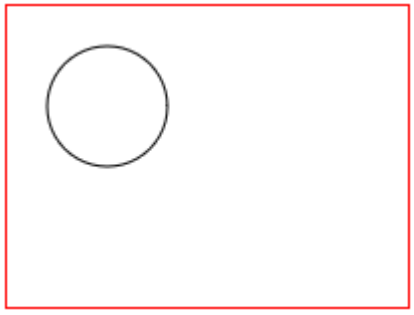
Disegna un arco circolare dato un punto centrale, raggio e angoli di inizio e fine. Gli angoli sono espressi come radianti. Per convertire i gradi in radianti puoi usare questa formula: `radians = degrees * Math.PI / 180;` .

L'angolo 0 è rivolto direttamente verso destra dal centro dell'arco.

Per impostazione predefinita, l'arco viene disegnato in senso orario, Un parametro opzionale [true | false] indica l'arco da disegnare in senso antiorario: `context.arc(10,10,20,0,Math.PI*2,true)`

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

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

    // arguments
    var centerX=50;
    var centerY=50;
    var radius=30;
    var startingRadianAngle=Math.PI*2*;  // start at 90 degrees == centerY+radius
    var endingRadianAngle=Math.PI*2*.75;  // end at 270 degrees (==PI*2*.75 in radians)

    // A partial circle (i.e. arc) drawn using the "arc" command
    ctx.beginPath();
    ctx.arc(centerX, centerY, radius,  startingRadianAngle, endingRadianAngle);
    ctx.stroke();

}); // end window.onload
</script>
</head>
<body>
    <canvas id="canvas" width=200 height=150></canvas>
</body>
</html>
```

Per disegnare un cerchio completo puoi rendere endingAngle = startingAngle + 360 gradi (360 gradi == Math.PI2).

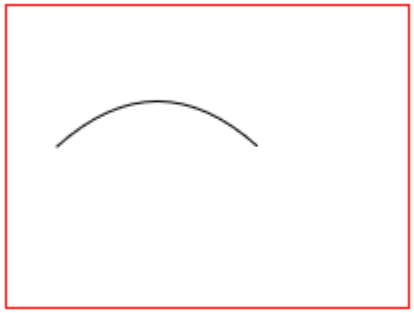


```
<!doctype html>
<html>
<head>
<style>
    body{ background-color:white; }
    #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

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

    // arguments
    var centerX=50;
    var centerY=50;
    var radius=30;
    var startingRadianAngle=0;       // start at 0 degrees
    var endingRadianAngle=Math.PI*2; // end at 360 degrees (==PI*2 in radians)

    // A complete circle drawn using the "arc" command
    ctx.beginPath();
    ctx.arc(centerX, centerY, radius,  startingRadianAngle, endingRadianAngle);
    ctx.stroke();

}); // end window.onload
</script>
</head>
<body>
    <canvas id="canvas" width=200 height=150></canvas>
</body>
</html>
```

quadraticCurveTo (un comando path)

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

Disegna una curva quadratica che inizia dalla posizione corrente della penna fino a una determinata coordinata finale. Un'altra determinata coordinata di controllo determina la forma (curvatura) della curva.

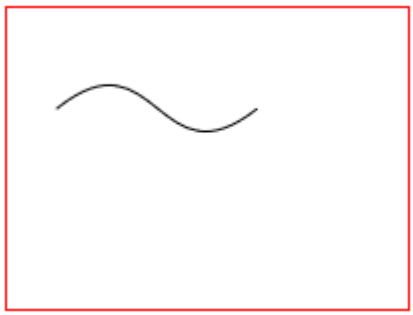


```
<!doctype html>
<html>
<head>
<style>
    body{ background-color:white; }
    #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

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

    // arguments
    var startX=25;
    var startY=70;
    var controlX=75;
    var controlY=25;
    var endX=125;
    var endY=70;

    // A quadratic curve drawn using "moveTo" and "quadraticCurveTo" commands
    ctx.beginPath();
    ctx.moveTo(startX,startY);
    ctx.quadraticCurveTo(controlX,controlY,endX,endY);
    ctx.stroke();


}); // end window.onload
</script>
</head>
<body>
    <canvas id="canvas" width=200 height=150></canvas>
</body>
</html>
```

bezierCurveTo (un comando path)

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

Disegna una curva Bezier cubica che inizia dalla posizione corrente della penna fino a una determinata coordinata finale. Altre 2 coordinate di controllo date determinano la forma (curvatura) della curva.



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

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

    // arguments
    var startX=25;
    var startY=50;
    var controlX1=75;
    var controlY1=10;
    var controlX2=75;
    var controlY2=90;
    var endX=125;
    var endY=50;

    // A cubic bezier curve drawn using "moveTo" and "bezierCurveTo" commands
    ctx.beginPath();
    ctx.moveTo(startX,startY);
    ctx.bezierCurveTo(controlX1,controlY1,controlX2,controlY2,endX,endY);
    ctx.stroke();

}); // end window.onload
</script>
</head>
<body>
    <canvas id="canvas" width=200 height=150></canvas>
</body>
</html>
```

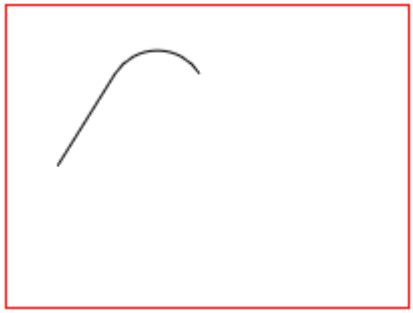
arcTo (un comando path)

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

Disegna un arco circolare con un raggio dato. L'arco viene disegnato in senso orario all'interno del cuneo formato dalla posizione attuale della penna e dato due punti: Punto1 e Punto2.

Una linea che collega la posizione corrente della penna e l'inizio dell'arco viene automaticamente

disegnata prima dell'arco.



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

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

    // arguments
    var pointX0=25;
    var pointY0=80;
    var pointX1=75;
    var pointY1=0;
    var pointX2=125;
    var pointY2=80;
    var radius=25;

    // A circular arc drawn using the "arcTo" command. The line is automatically drawn.
    ctx.beginPath();
    ctx.moveTo(pointX0,pointY0);
    ctx.arcTo(pointX1, pointY1, pointX2, pointY2, radius);
    ctx.stroke();

}); // end window.onload
</script>
</head>
<body>
    <canvas id="canvas" width=200 height=150></canvas>
</body>
</html>
```

rect (un comando di percorso)

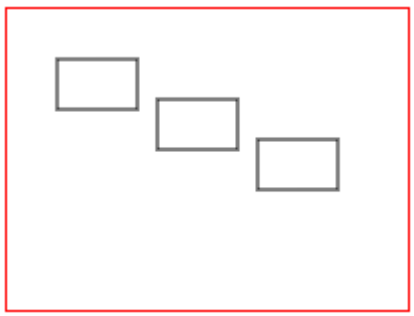
```
context.rect(leftX, topY, width, height)
```

Disegna un rettangolo dato un angolo in alto a sinistra e una larghezza e altezza.

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

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

    // arguments
    var leftX=25;
    var topY=25;
    var width=40;
    var height=25;

    // A rectangle drawn using the "rect" command.
    ctx.beginPath();
    ctx.rect(leftX, topY, width, height);
    ctx.stroke();

}); // end window.onload
</script>
</head>
<body>
    <canvas id="canvas" width=200 height=150></canvas>
</body>
</html>
```

`context.rect` è un comando di disegno univoco perché aggiunge rettangoli disconnessi.

Questi rettangoli scollegati non sono automaticamente collegati da linee.



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

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

    // arguments
    var leftX=25;
    var topY=25;
    var width=40;
    var height=25;

    // Multiple rectangles drawn using the "rect" command.
    ctx.beginPath();
    ctx.rect(leftX, topY, width, height);
    ctx.rect(leftX+50, topY+20, width, height);
    ctx.rect(leftX+100, topY+40, width, height);
    ctx.stroke();

}); // end window.onload
</script>
</head>
<body>
    <canvas id="canvas" width=200 height=150></canvas>
</body>
</html>
```

closePath (un comando path)

```
context.closePath()
```

Disegna una linea dalla posizione corrente della penna fino alla coordinata del percorso iniziale.

Ad esempio, se disegni 2 linee che formano 2 rami di un triangolo, closePath "chiude" il triangolo disegnando la terza gamba del triangolo dal punto finale della 2a tappa al punto iniziale della prima gamba.

Un equivoco spiegato!

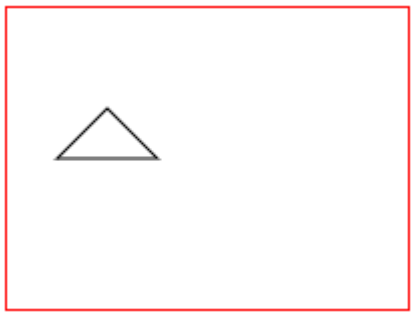
Il nome di questo comando spesso lo rende incompreso.

`context.closePath` NON è un delimitatore finale di `context.beginPath` .

Di nuovo, il comando closePath disegna una linea - non "chiude" un beginPath.

Questo esempio disegna 2 gambe di un triangolo e usa `closePath` per completare (chiudere ?!) il triangolo disegnando la terza gamba. Quello che `closePath` sta effettivamente facendo è tracciare

una linea dall'endpoint della seconda gamba fino al punto di partenza della prima gamba.



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

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

    // arguments
    var topVertexX=50;
    var topVertexY=50;
    var rightVertexX=75;
    var rightVertexY=75;
    var leftVertexX=25;
    var leftVertexY=75;

    // A set of line segments drawn to form a triangle using
    //     "moveTo" and multiple "lineTo" commands
    ctx.beginPath();
    ctx.moveTo(topVertexX,topVertexY);
    ctx.lineTo(rightVertexX,rightVertexY);
    ctx.lineTo(leftVertexX,leftVertexY);

    // closePath draws the 3rd leg of the triangle
    ctx.closePath()

    ctx.stroke();

}); // end window.onload
</script>
</head>
<body>
    <canvas id="canvas" width=200 height=150></canvas>
</body>
</html>
```

beginPath (un comando path)

```
context.beginPath()
```

Inizia l'assemblaggio di un nuovo set di comandi di percorso e scarta anche qualsiasi traccia precedentemente assemblato.

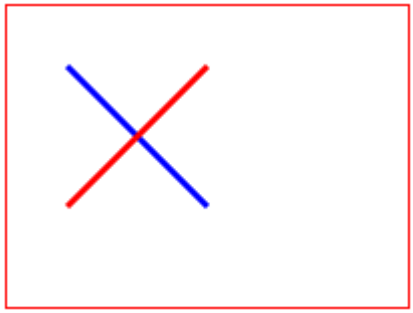
Sposta anche la "penna" del disegno sull'origine in alto a sinistra della tela (== coordinate [0,0]).

Sebbene facoltativo, dovresti SEMPRE avviare un percorso con `beginPath`

Lo scarto è un punto importante e spesso trascurato. Se non inizi un nuovo percorso con `beginPath` , qualsiasi comando di percorso precedentemente rilasciato verrà automaticamente ridisegnato.

Questi 2 demo tentano entrambi di disegnare una "X" con un tratto rosso e un tratto blu.

Questa prima demo utilizza correttamente `beginPath` per avviare il secondo tratto rosso. Il risultato è che la "X" ha correttamente sia una traccia rossa che una blu.



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

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

    // draw a blue line
    ctx.beginPath();
    ctx.moveTo(30,30);
    ctx.lineTo(100,100);
    ctx.strokeStyle='blue';
    ctx.lineWidth=3;
    ctx.stroke();

    // draw a red line
    ctx.beginPath();        // Important to begin a new path!
    ctx.moveTo(100,30);
    ctx.lineTo(30,100);
    ctx.strokeStyle='red';
    ctx.lineWidth=3;
    ctx.stroke();

}); // end window.onload
```

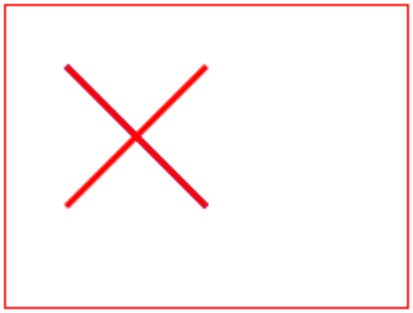
```
</script>
</head>
<body>
    <canvas id="canvas" width=200 height=150></canvas>
</body>
</html>
```

Questa seconda demo lascia erroneamente `beginPath` al secondo colpo. Il risultato è che la "X" ha erroneamente entrambi i tratti rossi.

Il secondo `stroke()` disegna il secondo tratto rosso.

Ma senza un secondo `beginPath` , quello stesso secondo `stroke()` inoltre **ridisegna in** modo errato il primo tratto.

Poiché il secondo `stroke()` è ora stilizzato in rosso, il primo tratto blu viene **sovrascritto** da un tratto rosso colorato in modo errato.



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

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

    // draw a blue line
    ctx.beginPath();
    ctx.moveTo(30,30);
    ctx.lineTo(100,100);
    ctx.strokeStyle='blue';
    ctx.lineWidth=3;
    ctx.stroke();

    // draw a red line
    // Note: The necessary 'beginPath' is missing!
    ctx.moveTo(100,30);
    ctx.lineTo(30,100);
    ctx.strokeStyle='red';
    ctx.lineWidth=3;
    ctx.stroke();
```

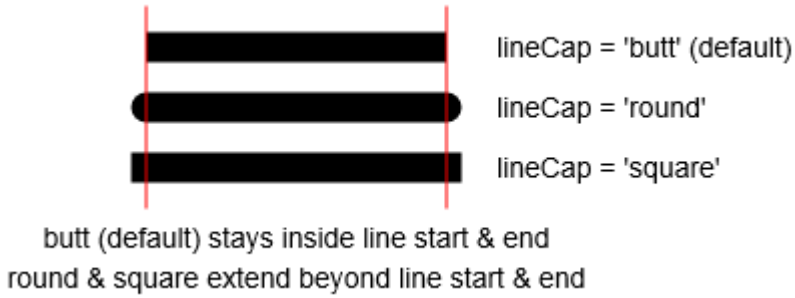
```
}); // end window.onload
</script>
</head>
<body>
    <canvas id="canvas" width=200 height=150></canvas>
</body>
</html>
```

lineCap (un attributo styling del percorso)

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

Imposta lo stile del cappuccio dei punti iniziali e finali della linea.

- **butt** , lo stile lineCap predefinito, mostra i cappucci quadrati che non si estendono oltre i punti iniziale e finale della linea.
- **round** , mostra tappi arrotondati che si estendono oltre i punti di inizio e fine della linea.
- **quadrato** , mostra i cappucci quadrati che si estendono oltre i punti di inizio e fine della linea.




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

    // canvas related variables
    var canvas=document.getElementById("canvas");
    var ctx=canvas.getContext("2d");
    ctx.lineWidth=15;

    // lineCap default: butt
    ctx.lineCap='butt';
    drawLine(50,40,200,40);

    // lineCap: round
    ctx.lineCap='round';
    drawLine(50,70,200,70);
```

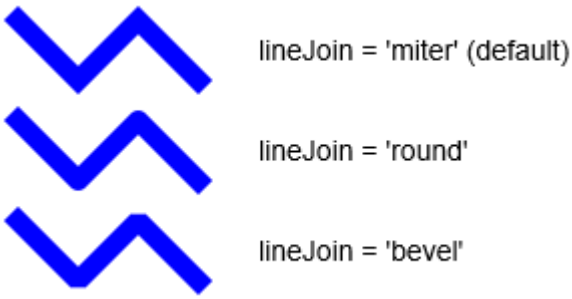
```
    // lineCap: square
    ctx.lineCap='square';
    drawLine(50,100,200,100);

    // utility function to draw a line
    function drawLine(startX,startY,endX,endY){
        ctx.beginPath();
        ctx.moveTo(startX,startY);
        ctx.lineTo(endX,endY);
        ctx.stroke();
    }

    // For demo only,
    // Rulers to show which lineCaps extend beyond endpoints
    ctx.lineWidth=1;
    ctx.strokeStyle='red';
    drawLine(50,20,50,120);
    drawLine(200,20,200,120);

}); // end window.onload
</script>
</head>
<body>
    <canvas id="canvas" width=300 height=200></canvas>
</body>
</html>
```

lineJoin (un attributo styling del percorso)

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

Imposta lo stile utilizzato per collegare segmenti di linea contigui.

- **mitra** , l'impostazione predefinita, unisce i segmenti di linea con un giunto appuntito.
- **rotondo** , unisce segmenti di linea con un giunto arrotondato.
- **smusso** , unisce segmenti di linea con un giunto **smussato** .




```
<!doctype html>
<html>
<head>
<style>
    body{ background-color:white; }
    #canvas{border:1px solid red; }
</style>
```

```
<script>
window.onload=(function(){

    // canvas related variables
    var canvas=document.getElementById("canvas");
    var ctx=canvas.getContext("2d");
    ctx.lineWidth=15;

    // lineJoin: miter (default)
    ctx.lineJoin='miter';
    drawPolyline(50,30);

    // lineJoin: round
    ctx.lineJoin='round';
    drawPolyline(50,80);

    // lineJoin: bevel
    ctx.lineJoin='bevel';
    drawPolyline(50,130);

    // utility to draw polyline
    function drawPolyline(x,y){
        ctx.beginPath();
        ctx.moveTo(x,y);
        ctx.lineTo(x+30,y+30);
        ctx.lineTo(x+60,y);
        ctx.lineTo(x+90,y+30);
        ctx.stroke();
    }

}); // end window.onload
</script>
</head>
<body>
    <canvas id="canvas" width=300 height=200></canvas>
</body>
</html>
```

strokeStyle (un attributo styling del percorso)

```
context.strokeStyle=color
```

Imposta il colore che verrà utilizzato per tracciare il contorno del percorso corrente.

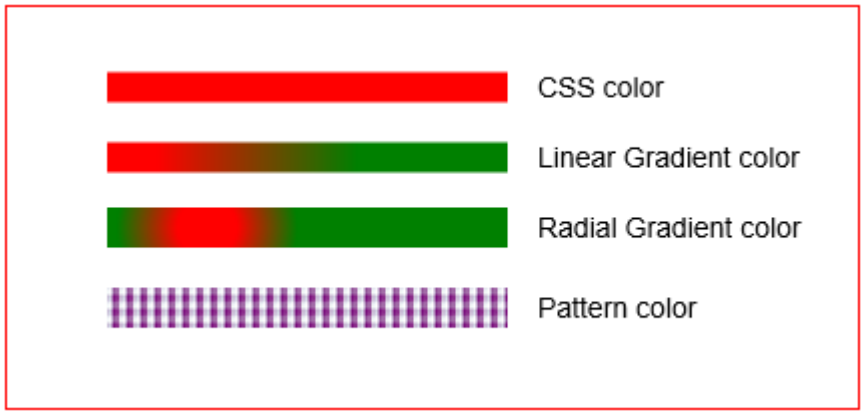
Queste sono opzioni di `color` (queste devono essere citate):

- **Un colore denominato CSS** , ad esempio `context.strokeStyle='red'`
- **Un colore esadecimale** , ad esempio `context.strokeStyle='#FF0000'`
- **Un colore RGB** , ad esempio `context.strokeStyle='rgb(red,green,blue)'` dove rosso, verde e blu sono numeri interi 0-255 che indicano la forza di ciascun colore del componente.
- **Un colore HSL** , ad esempio `context.strokeStyle='hsl(hue,saturation,lightness)'` dove hue è un numero intero 0-360 sulla ruota dei colori e saturazione e luminosità sono percentuali (0-100%) che indicano la forza di ciascun componente .

- **Un colore HSLA** , ad esempio `context.strokeStyle='hsl(hue,saturation,lightness,alpha)'` dove hue è un numero intero 0-360 sulla ruota dei colori e saturazione e luminosità sono percentuali (0-100%) che indicano la forza di ogni componente e alfa è un valore decimale 0.00-1.00 che indica l'opacità.

Puoi anche specificare queste opzioni colore (queste opzioni sono oggetti creati dal contesto):

- **Un gradiente lineare** che è un oggetto gradiente lineare creato con `context.createLinearGradient`
- **Un gradiente radiale** che è un oggetto gradiente radiale creato con `context.createRadialGradient`
- **Un motivo** che è un oggetto modello creato con `context.createPattern`




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

    // canvas related variables
    var canvas=document.getElementById("canvas");
    var ctx=canvas.getContext("2d");
    ctx.lineWidth=15;

    // stroke using a CSS color: named, RGB, HSL, etc
    ctx.strokeStyle='red';
    drawLine(50,40,250,40);

    // stroke using a linear gradient
    var gradient = ctx.createLinearGradient(75,75,175,75);
    gradient.addColorStop(0,'red');
    gradient.addColorStop(1,'green');
    ctx.strokeStyle=gradient;
    drawLine(50,75,250,75);

    // stroke using a radial gradient
    var gradient = ctx.createRadialGradient(100,110,15,100,110,45);
    gradient.addColorStop(0,'red');
```

```
    gradient.addColorStop(1,'green');
    ctx.strokeStyle=gradient;
    ctx.lineWidth=20;
    drawLine(50,110,250,110);

    // stroke using a pattern
    var patternImage=new Image();
    patternImage.onload=function(){
        var pattern = ctx.createPattern(patternImage,'repeat');
        ctx.strokeStyle=pattern;
        drawLine(50,150,250,150);
    }
    patternImage.src='https://dl.dropboxusercontent.com/u/139992952/stackoverflow/BooMu1.png';

    // for demo only, draw labels by each stroke
    ctx.textBaseline='middle';
    ctx.font='14px arial';
    ctx.fillText('CSS color',265,40);
    ctx.fillText('Linear Gradient color',265,75);
    ctx.fillText('Radial Gradient color',265,110);
    ctx.fillText('Pattern color',265,150);

    // utility to draw a line
    function drawLine(startX,startY,endX,endY){
        ctx.beginPath();
        ctx.moveTo(startX,startY);
        ctx.lineTo(endX,endY);
        ctx.stroke();
    }

}); // end window.onload
</script>
</head>
<body>
    <canvas id="canvas" width=425 height=200></canvas>
</body>
</html>
```

fillStyle (un attributo styling del percorso)

```
context.fillStyle=color
```

Imposta il colore che verrà utilizzato per riempire l'interno del percorso corrente.

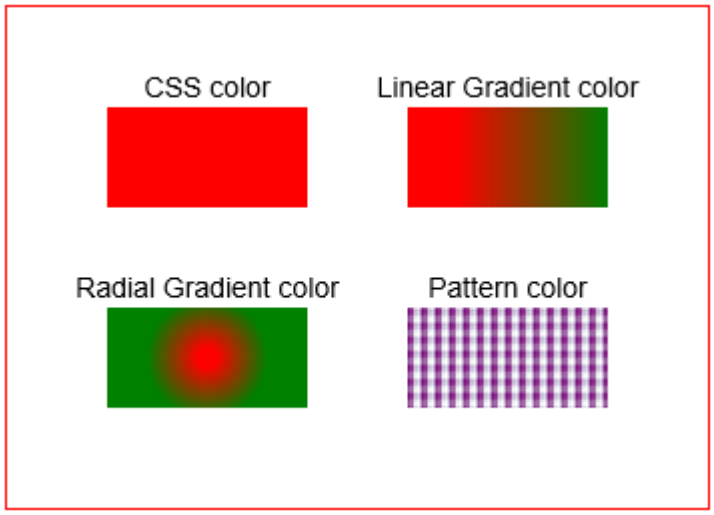
Queste sono opzioni di colore (queste devono essere citate):

- **Un colore denominato CSS** , ad esempio `context.fillStyle='red'`
- **Un colore esadecimale** , ad esempio `context.fillStyle='#FF0000'`
- **Un colore RGB** , ad esempio `context.fillStyle='rgb(red,green,blue)'` dove rosso, verde e blu sono numeri interi 0-255 che indicano la forza di ciascun colore del componente.
- **Un colore HSL** , ad esempio `context.fillStyle='hsl(hue,saturation,lightness)'` dove hue è un numero intero 0-360 sulla ruota dei colori e saturazione e luminosità sono percentuali (0-100%) che indicano la forza di ciascun componente .

- **Un colore HSLA** , ad esempio `context.fillStyle='hsl(hue,saturation,lightness,alpha)'` dove hue è un numero intero 0-360 sulla ruota dei colori e saturazione e luminosità sono percentuali (0-100%) che indicano la forza di ogni componente e alfa è un valore decimale 0.00-1.00 che indica l'opacità.

Puoi anche specificare queste opzioni colore (queste opzioni sono oggetti creati dal contesto):

- **Un gradiente lineare** che è un oggetto gradiente lineare creato con `context.createLinearGradient`
- **Un gradiente radiale** che è un oggetto gradiente radiale creato con `context.createRadialGradient`
- **Un motivo** che è un oggetto modello creato con `context.createPattern`



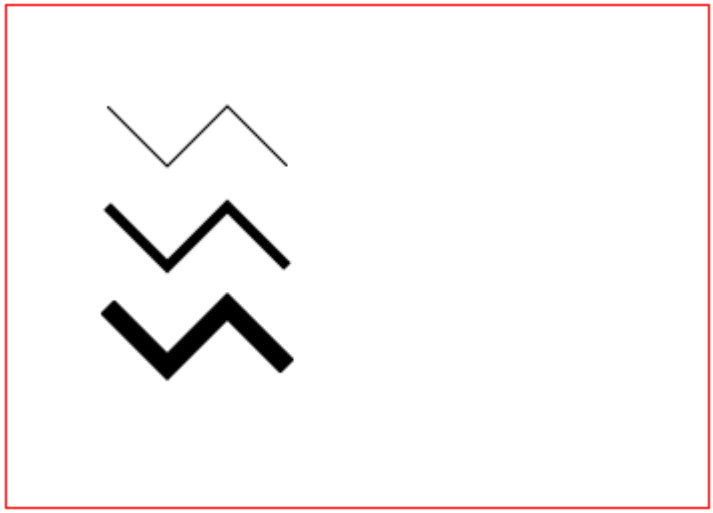

```
<!doctype html>
<html>
<head>
<style>
    body{ background-color:white; }
    #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

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

    // stroke using a CSS color: named, RGB, HSL, etc
    ctx.fillStyle='red';
    ctx.fillRect(50,50,100,50);

    // stroke using a linear gradient
    var gradient = ctx.createLinearGradient(225,50,300,50);
    gradient.addColorStop(0,'red');
    gradient.addColorStop(1,'green');
    ctx.fillStyle=gradient;
    ctx.fillRect(200,50,100,50);

    // stroke using a radial gradient
```

```
    var gradient = ctx.createRadialGradient(100,175,5,100,175,30);
    gradient.addColorStop(0,'red');
    gradient.addColorStop(1,'green');
    ctx.fillStyle=gradient;
    ctx.fillRect(50,150,100,50);

    // stroke using a pattern
    var patternImage=new Image();
    patternImage.onload=function(){
        var pattern = ctx.createPattern(patternImage,'repeat');
        ctx.fillStyle=pattern;
        ctx.fillRect(200,150,100,50);
    }
    patternImage.src='http://i.stack.imgur.com/ixrWe.png';

    // for demo only, draw labels by each stroke
    ctx.fillStyle='black';
    ctx.textAlign='center';
    ctx.textBaseline='middle';
    ctx.font='14px arial';
    ctx.fillText('CSS color',100,40);
    ctx.fillText('Linear Gradient color',250,40);
    ctx.fillText('Radial Gradient color',100,140);
    ctx.fillText('Pattern color',250,140);

}); // end window.onload
</script>
</head>
<body>
    <canvas id="canvas" width=350 height=250></canvas>
</body>
</html>
```

lineWidth (attributo di stile del percorso)

```
context.lineWidth=lineWidth
```

Imposta la larghezza della linea che traccerà il contorno del percorso



```
<!doctype html>
<html>
```

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

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

    ctx.lineWidth=1;
    drawPolyline(50,50);

    ctx.lineWidth=5;
    drawPolyline(50,100);

    ctx.lineWidth=10;
    drawPolyline(50,150);


    // utility to draw a polyline
    function drawPolyline(x,y){
        ctx.beginPath();
        ctx.moveTo(x,y);
        ctx.lineTo(x+30,y+30);
        ctx.lineTo(x+60,y);
        ctx.lineTo(x+90,y+30);
        ctx.stroke();
    }

}); // end window.onload
</script>
</head>
<body>
    <canvas id="canvas" width=350 height=250></canvas>
</body>
</html>
```

shadowColor, shadowBlur, shadowOffsetX, shadowOffsetY (attributi di styling del percorso)

```
shadowColor = color        // CSS color
shadowBlur =  width        // integer blur width
shadowOffsetX = distance   // shadow is moved horizontally by this offset
shadowOffsetY = distance   // shadow is moved vertically by this offset
```

Questo insieme di attributi aggiungerà un'ombra attorno a un tracciato.

Sia i percorsi riempiti che i percorsi tracciati possono avere un'ombra.

L'ombra è più scura (opaca) sul perimetro del percorso e diventa gradualmente più leggera mentre si allontana dal perimetro del percorso.

- **shadowColor** indica quale colore CSS verrà utilizzato per creare l'ombra.
- **shadowBlur** è la distanza su cui l'ombra si estende verso l'esterno dal percorso.

- **shadowOffsetX** è una distanza per cui l'ombra viene spostata orizzontalmente dal percorso. Una distanza positiva sposta l'ombra verso destra, una distanza negativa sposta l'ombra verso sinistra.
- **shadowOffsetY** è una distanza per cui l'ombra viene spostata verticalmente lontano dal percorso. Una distanza positiva sposta l'ombra verso il basso, una distanza negativa sposta l'ombra verso l'alto.

Informazioni su shadowOffsetX e shadowOffsetY

È importante notare che *l'intera ombra è spostata nella sua interezza* . Ciò farà sì che parte dell'ombra si sposti al di sotto dei percorsi riempiti e quindi parte dell'ombra non sarà visibile.

Informazioni sui tratti ombreggiati

Quando si ombreggia un tratto, l'interno e l'esterno del tratto sono ombreggiati. L'ombra è più scura nel tratto e si illumina quando l'ombra si estende verso l'esterno in entrambe le direzioni rispetto al tratto.

Disattivazione dell'ombreggiamento al termine

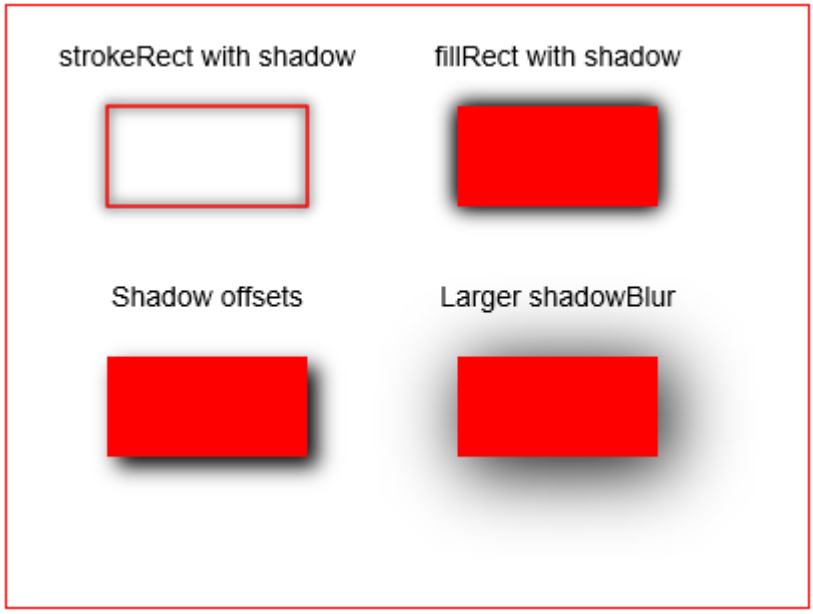
Dopo aver disegnato le ombre, potresti voler disattivare lo shadowing per disegnare più percorsi. Per disattivare l'ombreggiatura, imposta `shadowColor` su trasparente.

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

Considerazioni sulle prestazioni

Le ombre (come i gradienti) richiedono calcoli estesi e quindi è necessario utilizzare le ombre con parsimonia.

Prestare particolare attenzione durante l'animazione, poiché disegnare ombre molte volte al secondo avrà un impatto notevole sulle prestazioni. Una soluzione alternativa se è necessario animare i percorsi ombreggiati consiste nel pre-creare il percorso ombreggiato su un secondo "riquadro ombra". La tela ombra è una tela normale che viene creata in memoria con `document.createElement` : non viene aggiunta al DOM (è solo una tela di staging). Quindi disegna la tela ombra sulla tela principale. Questo è molto più veloce perché i calcoli dell'ombra non devono essere fatti molte volte al secondo. Tutto quello che stai facendo è copiare una tela prefabbricata sulla tela visibile.




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

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

    // shadowed stroke
    ctx.shadowColor='black';
    ctx.shadowBlur=6;
    ctx.strokeStyle='red';
    ctx.strokeRect(50,50,100,50);
    // darken the shadow by stroking a second time
    ctx.strokeRect(50,50,100,50);

    // shadowed fill
    ctx.shadowColor='black';
    ctx.shadowBlur=10;
    ctx.fillStyle='red';
    ctx.fillRect(225,50,100,50);
    // darken the shadow by stroking a second time
    ctx.fillRect(225,50,100,50);

    // the shadow offset rightward and downward
    ctx.shadowColor='black';
    ctx.shadowBlur=10;
    ctx.shadowOffsetX=5;
    ctx.shadowOffsetY=5;
    ctx.fillStyle='red';
    ctx.fillRect(50,175,100,50);

    // a wider blur (==extends further from the path)
    ctx.shadowColor='black';
    ctx.shadowBlur=35;
```

```
    ctx.fillStyle='red';
    ctx.fillRect(225,175,100,50);

    // always clean up! Turn off shadowing
    ctx.shadowColor='rgba(0,0,0,0)';

}); // end window.onload
</script>
</head>
<body>
    <canvas id="canvas" width=400 height=300></canvas>
</body>
</html>
```

createLinearGradient (crea un oggetto di stile del percorso)

```
var gradient = createLinearGradient( startX, startY, endX, endY )
gradient.addColorStop(gradientPercentPosition, CssColor)
gradient.addColorStop(gradientPercentPosition, CssColor)
[optionally add more color stops to add to the variety of the gradient]
```

Crea un gradiente lineare riutilizzabile (oggetto).

L'oggetto può essere assegnato a qualsiasi `strokeStyle` e / o `fillStyle` .

Quindi il tratto () o il riempimento () coloreranno il tracciato con i colori sfumati dell'oggetto.

La creazione di un oggetto sfumato è una procedura in due passaggi:

1. Crea l'oggetto gradiente stesso. Durante la creazione, definisci una linea sulla tela in cui inizia e termina la sfumatura. L'oggetto gradiente viene creato con `var gradient = context.createLinearGradient` .
2. Quindi aggiungi 2 (o più) colori che compongono il gradiente. Questo viene fatto aggiungendo più stop di colore all'oggetto `gradient.addColorStop` con `gradient.addColorStop` .

Argomenti:

- **startX, startY** è la coordinata del canvas in cui inizia la sfumatura. Al punto di partenza (e prima) la tela è solidamente il colore della più bassa `gradientPercentPosition` .
- **endX, endY** è la coordinata del canvas in cui termina il gradiente. Al punto finale (e dopo) la tela è solidamente il colore della più alta `gradientPercentPosition` .
- **gradientPercentPosition** è un numero float compreso tra 0,00 e 1,00 assegnato a un arresto colore. È fondamentalmente un waypoint in percentuale lungo la linea in cui si applica questo particolare arresto del colore.
    - Il gradiente inizia alla percentuale 0,00 che è [startX, startY] sulla tela.
    - Il gradiente termina con la percentuale 1,00 che è [endX, endY] sulla tela.
    - *Nota tecnica:* il termine "percentuale" non è tecnicamente corretto poiché i valori vanno da 0,00 a 1,00 anziché dallo 0% al 100%.

- **CssColor** è un colore CSS assegnato a questo particolare arresto del colore.

L'oggetto gradiente è un oggetto che puoi usare (e riutilizzare!) Per fare in modo che i tratti e i riempimenti del tuo percorso diventino sfumati.

Nota a margine: l'oggetto sfumato non è interno all'elemento Canvas né è Context. È un oggetto JavaScript separato e riutilizzabile che puoi assegnare a qualsiasi percorso che desideri. Puoi persino usare questo oggetto per colorare un percorso su un elemento Canvas diverso (!)

Le interruzioni di colore sono (percentuali) waypoint lungo la linea del gradiente. Ad ogni fermo del colore, il gradiente è completamente (== opaco) colorato con il colore assegnato. I punti intermedi lungo la linea del gradiente tra i fermi colore sono colorati come gradienti di questo e del colore precedente.

Suggerimento importante sui gradienti della tela!

Quando crei un oggetto sfumato, l'intera tela è "invisibilmente" piena di quel gradiente.

Quando si percorre `stroke()` o si `fill()` un tracciato, il gradiente invisibile viene rivelato, ma solo rivelato su quel percorso che viene tracciato o riempito.

1. Se crei un gradiente lineare da rosso a magenta come questo:

```
// create a linearGradient
var gradient=ctx.createLinearGradient(100,0,canvas.width-100,0);
gradient.addColorStop(0,'red');
gradient.addColorStop(1,'magenta');
ctx.fillStyle=gradient;
```

2. Quindi Canvas visualizzerà "in modo invisibile" la creazione della sfumatura in questo modo:

3. Ma fino a quando non si `stroke()` o si `fill()` con il gradiente, non vedrai nessuno dei gradienti sulla tela.

4. Infine, se si accarezza o si riempie un percorso usando la sfumatura, la sfumatura "invisibile" diventa visibile sulla tela ... ma solo dove viene disegnato il tracciato.

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

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

    // Create a linearGradient
    // Note: Nothing visually appears during this process
    var gradient=ctx.createLinearGradient(100,0,canvas.width-100,0);
    gradient.addColorStop(0,'red');
    gradient.addColorStop(1,'magenta');

    // Create a polyline path
    // Note: Nothing visually appears during this process
    var x=20;
    var y=40;
    ctx.lineCap='round';
    ctx.lineJoin='round';
    ctx.lineWidth=15;
    ctx.beginPath();
    ctx.moveTo(x,y);
    ctx.lineTo(x+30,y+50);
    ctx.lineTo(x+60,y);
    ctx.lineTo(x+90,y+50);
    ctx.lineTo(x+120,y);
    ctx.lineTo(x+150,y+50);
    ctx.lineTo(x+180,y);
    ctx.lineTo(x+210,y+50);
    ctx.lineTo(x+240,y);
    ctx.lineTo(x+270,y+50);
    ctx.lineTo(x+300,y);
    ctx.lineTo(x+330,y+50);
    ctx.lineTo(x+360,y);

    // Set the stroke style to be the gradient
    // Note: Nothing visually appears during this process
    ctx.strokeStyle=gradient;

    // stroke the path
    // FINALLY! The gradient-stroked path is visible on the canvas
    ctx.stroke();
```

```
}); // end window.onload
</script>
</head>
<body>
    <canvas id="canvas" width=400 height=150></canvas>
</body>
</html>
```

createRadialGradient (crea un oggetto di stile del percorso)

```
var gradient = createRadialGradient(
      centerX1, centerY1, radius1,     // this is the "display' circle
      centerX2, centerY2, radius2      // this is the "light casting" circle
)
gradient.addColorStop(gradientPercentPosition, CssColor)
gradient.addColorStop(gradientPercentPosition, CssColor)
[optionally add more color stops to add to the variety of the gradient]
```

Crea un gradiente radiale riutilizzabile (oggetto). L'oggetto gradiente è un oggetto che puoi usare (e riutilizzare!) Per fare in modo che i tratti e i riempimenti del tuo percorso diventino sfumati.

Di...

Il gradiente radiale Canvas è *estremamente diverso* dai tradizionali gradienti radiali.

La definizione "ufficiale" (quasi indecifrabile!) Del gradiente radiale di Canvas è in fondo a questo post. Non guardarlo se hai una debole predisposizione !!

In termini (quasi comprensibili):

- Il gradiente radiale ha 2 cerchi: un cerchio "casting" e un cerchio "display".
- Il cerchio di lancio proietta la luce nel cerchio del display.
- Quella luce è il gradiente.
- La forma di quella luce sfumata è determinata dalla dimensione relativa e dalla posizione di entrambi i cerchi.

La creazione di un oggetto sfumato è una procedura in due passaggi:

1. Crea l'oggetto gradiente stesso. Durante la creazione, definisci una linea sulla tela in cui inizia e termina la sfumatura. L'oggetto gradiente viene creato con `var gradient = context.radialLinearGradient` .
2. Quindi aggiungi 2 (o più) colori che compongono il gradiente. Questo viene fatto aggiungendo più stop di colore all'oggetto `gradient.addColorStop` con `gradient.addColorStop` .

Argomenti:

- **centerX1, centerY1, raggio1** definisce un primo cerchio in cui verrà visualizzato il gradiente.
- **centerX2, centerY2, raggio2** definisce un secondo cerchio che lancia la luce sfumata nel primo cerchio.

- **gradientPercentPosition** è un numero float compreso tra 0,00 e 1,00 assegnato a un arresto colore. È fondamentalmente un punto di riferimento in percentuale che definisce dove questo particolare arresto del colore si applica lungo il gradiente.
    - Il gradiente inizia in percentuale 0,00.
    - Il gradiente termina in percentuale 1,00.
    - *Nota tecnica:* il termine "percentuale" non è tecnicamente corretto poiché i valori vanno da 0,00 a 1,00 anziché dallo 0% al 100%.
- **CssColor** è un colore CSS assegnato a questo particolare arresto del colore.

Nota a margine: l'oggetto sfumato non è interno all'elemento Canvas né è Context. È un oggetto JavaScript separato e riutilizzabile che puoi assegnare a qualsiasi percorso che desideri. Puoi persino usare questo oggetto per colorare un percorso su un elemento Canvas diverso (!)

Le interruzioni di colore sono (percentuali) waypoint lungo la linea del gradiente. Ad ogni fermo del colore, il gradiente è completamente (== opaco) colorato con il colore assegnato. I punti intermedi lungo la linea del gradiente tra i fermi colore sono colorati come gradienti di questo e del colore precedente.

Suggerimento importante sui gradienti della tela!

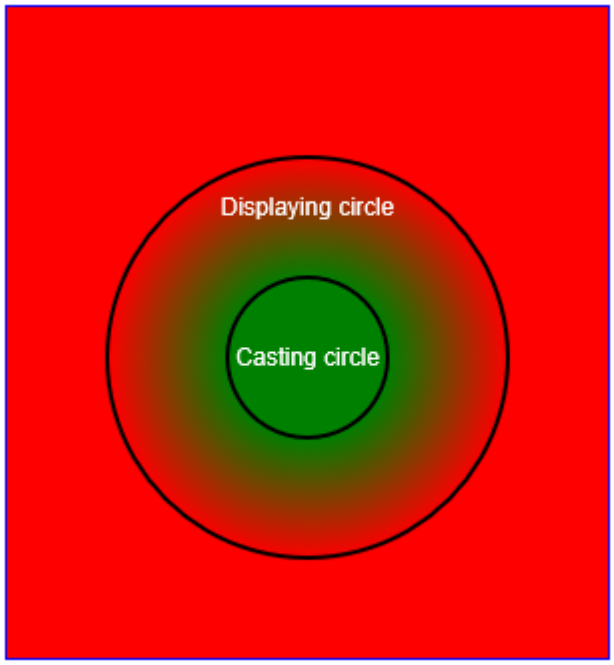
Quando crei un oggetto sfumato, l'intera sfumatura radiale viene proiettata "in modo invisibile" sulla tela.

Quando si percorre `stroke()` o si `fill()` un tracciato, il gradiente invisibile viene rivelato, ma solo rivelato su quel percorso che viene tracciato o riempito.

1. Se crei un gradiente radiale da verde a rosso come questo:

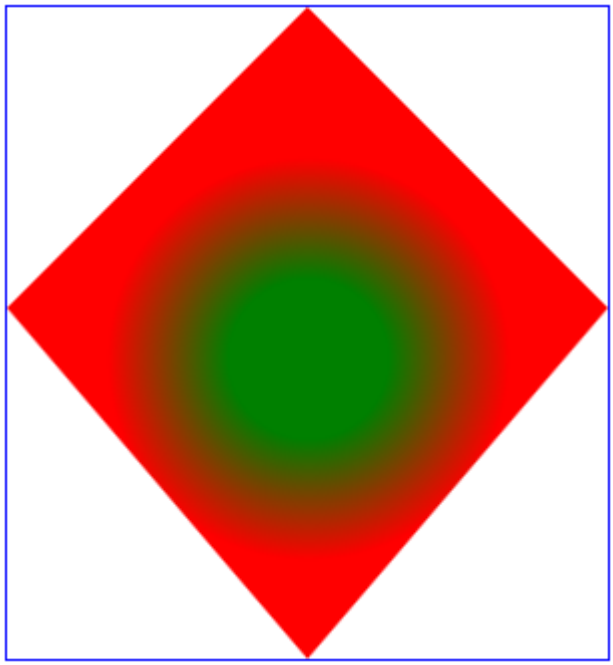
```
// create a radialGradient
var x1=150;
var y1=150;
var x2=280;
var y2=150;
var r1=100;
var r2=120;
var gradient=ctx.createRadialGradient(x1,y1,r1,x2,y2,r2);
gradient.addColorStop(0,'red');
gradient.addColorStop(1,'green');
ctx.fillStyle=gradient;
```

2. Quindi Canvas visualizzerà "in modo invisibile" la creazione della sfumatura in questo modo:



3. Ma fino a quando non si `stroke()` o si `fill()` con il gradiente, non vedrai nessuno dei gradienti sulla tela.

4. Infine, se si accarezza o si riempie un percorso usando la sfumatura, la sfumatura "invisibile" diventa visibile sulla tela ... ma solo dove viene disegnato il tracciato.



```
<!doctype html>
<html>
<head>
<style>
    body{ background-color:white; padding:10px; }
    #canvas{border:1px solid blue; }
</style>
<script>
window.onload=(function(){
```

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

    // create a radial gradient
    var x1=150;
    var y1=175;
    var x2=350;
    var y2=175;
    var r1=100;
    var r2=40;
    x2=x1;
    var gradient=ctx.createRadialGradient(x1,y1,r1,x2,y2,r2);
    gradient.addColorStop(0,'red');
    gradient.addColorStop(1,'green');
    ctx.fillStyle=gradient;

    // fill a path with the gradient
    ctx.beginPath();
    ctx.moveTo(150,0);
    ctx.lineTo(300,150);
    ctx.lineTo(150,325);
    ctx.lineTo(0,150);
    ctx.lineTo(150,0);
    ctx.fill();

}); // end window.onload
</script>
</head>
<body>
    <canvas id="canvas" width=300 height=325></canvas>
</body>
</html>
```

I dettagli ufficiali spaventosi

Chi decide cosa `creaRadialGradient?

Il W3C rilascia le specifiche ufficiali raccomandate che i browser usano per costruire l'elemento Html5 Canvas.

Le specifiche W3C per `createRadialGradient` leggono in modo criptico come questo:

Cosa crea `createRadialGradient`

> `createRadialGradient` ... crea effettivamente un cono, toccato dai due cerchi definiti nella creazione del gradiente, con la parte del cono prima del cerchio iniziale (0.0) utilizzando il colore del primo offset, la parte del cono dopo il cerchio finale (1.0) utilizzando il colore dell'ultimo offset e le aree esterne al cono non toccate dal gradiente (nero trasparente).

Come funziona internamente

> Il `createRadialGradient(x0, y0, r0, x1, y1, r1)` accetta sei argomenti, i primi tre

rappresentano il cerchio iniziale con origine (x0, y0) e raggio r0 e gli ultimi tre rappresentano il cerchio finale con origine (x1 , y1) e raggio r1. I valori sono in unità spaziali coordinate. Se uno dei file r0 o r1 è negativo, deve essere generata un'eccezione IndexSizeError. In caso contrario, il metodo deve restituire un CanvasGradient radiale inizializzato con i due cerchi specificati.

I gradienti radiali devono essere visualizzati seguendo questi passaggi:

1. Se x0 = x1 e y0 = y1 e r0 = r1, allora il gradiente radiale non deve dipingere nulla. Annullare questi passaggi.
2. Sia x (ω) = (x1-x0) ω + x0; Sia y (ω) = (y1-y0) ω + y0; Sia r (ω) = (r1-r0) ω + r0 Lascia che il colore in ω sia il colore in quella posizione sul gradiente (con i colori che provengono dall'interpolazione e dall'estrapolazione sopra descritti).
3. Per tutti i valori di ω dove r (ω)> 0, partendo dal valore di ω più vicino all'infinito positivo e finendo con il valore di ω più vicino all'infinito negativo, traccia la circonferenza del cerchio con raggio r (ω) in posizione (x (ω), y (ω)), con il colore in ω, ma solo la pittura sulle parti della tela che non sono state ancora dipinte da cerchi precedenti in questo passaggio per questo rendering del gradiente.

createPattern (crea un oggetto di stile del percorso)

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

Crea un modello riutilizzabile (oggetto).

L'oggetto può essere assegnato a qualsiasi `strokeStyle` e / o `fillStyle` .

Quindi stroke () o fill () dipingerà il Path con il pattern dell'oggetto.

Argomenti:

- **imageObject** è un'immagine che verrà utilizzata come pattern. La fonte dell'immagine può essere:
    - HTMLImageElement --- un elemento img o una nuova immagine (),
    - HTMLCanvasElement --- un elemento canvas,
    - HTMLVideoElement --- un elemento video (catturerà il fotogramma video corrente)
    - ImageBitmap,
    - Blob.
- **repeat** determina come l'imageObject verrà ripetuto sull'area di disegno (proprio come in uno sfondo CSS). Questo argomento deve essere delimitato da virgolette e i valori validi sono:
    - "repeat" --- il pattern riempie orizzontalmente e verticalmente la tela
    - "repeat-x" --- il pattern si ripeterà solo orizzontalmente (1 riga orizzontale)
    - "repeat-y" --- il pattern si ripeterà solo verticalmente (1 riga verticale)
    - "repeat none" --- il pattern appare solo una volta (in alto a sinistra)

L'oggetto modello è un oggetto che puoi usare (e riutilizzare!) Per fare in modo che i tratti e i riempimenti del tuo percorso vengano modellati.

Nota a margine: l'oggetto modello non è interno all'elemento Canvas né è Context. È un oggetto JavaScript separato e riutilizzabile che puoi assegnare a qualsiasi percorso che desideri. Puoi persino usare questo oggetto per applicare il modello a un tracciato su un elemento Canvas diverso (!)

Suggerimento importante sui modelli di tela!

Quando crei un oggetto motivo, l'intera tela è "invisibilmente" piena di quel motivo (soggetto all'argomento `repeat`).

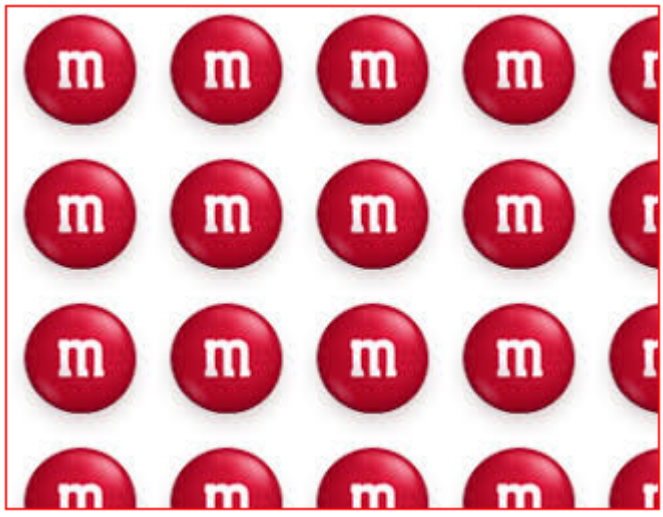
Quando si percorre `stroke()` o si `fill()` un percorso, viene rivelato il motivo invisibile, ma solo rivelato su quel percorso che viene tracciato o riempito.

1. Inizia con un'immagine che desideri utilizzare come modello. Importante (!): Assicurati che l'immagine sia completamente caricata (utilizzando `patternimage.onload`) prima di provare a utilizzarla per creare il tuo pattern.

2. Crei uno schema come questo:

```
// create a pattern
var pattern = ctx.createPattern(patternImage,'repeat');
ctx.fillStyle=pattern;
```

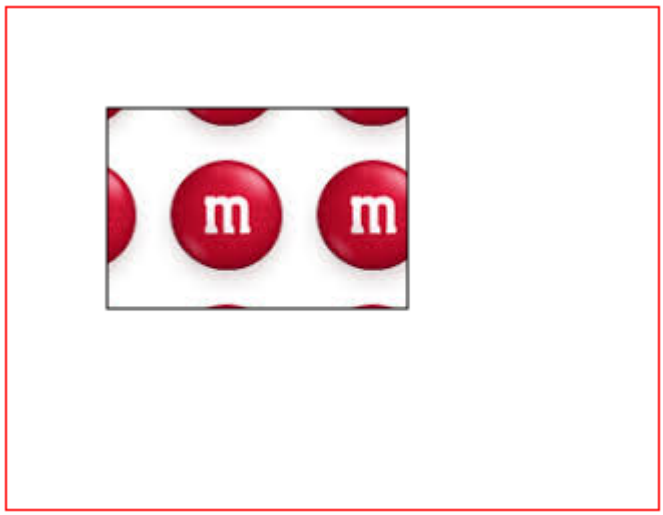
3. Quindi Canvas visualizzerà "invisibilmente" la creazione del tuo pattern in questo modo:



4. Ma fino a quando non si `stroke()` o si `fill()` con il motivo, non vedrai nessuno dei motivi sulla tela.

5. Infine, se si fa un tratto o si riempie un percorso usando il modello, il motivo "invisibile"

diventa visibile sulla tela ... ma solo dove viene disegnato il tracciato.



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

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

    // fill using a pattern
    var patternImage=new Image();
    // IMPORTANT!
    // Always use .onload to be sure the image has
    //     fully loaded before using it in .createPattern
    patternImage.onload=function(){
        // create a pattern object
        var pattern = ctx.createPattern(patternImage,'repeat');
        // set the fillstyle to that pattern
        ctx.fillStyle=pattern;
        // fill a rectangle with the pattern
        ctx.fillRect(50,50,150,100);
        // demo only, stroke the rect for clarity
        ctx.strokeRect(50,50,150,100);
    }
    patternImage.src='http://i.stack.imgur.com/K9EZl.png';

}); // end window.onload
</script>
</head>
<body>
    <canvas id="canvas" width=325 height=250></canvas>
</body>
</html>
```

tratto (un comando di percorso)

```
context.stroke()
```

Fa sì che il perimetro del tracciato venga tracciato in base al `context.strokeStyle` corrente.strokeStyle e il tracciato tratteggiato viene visivamente disegnato sull'area di disegno.

Prima di eseguire `context.stroke` (o `context.fill`) il Path esiste in memoria e non è ancora disegnato visivamente sulla tela.

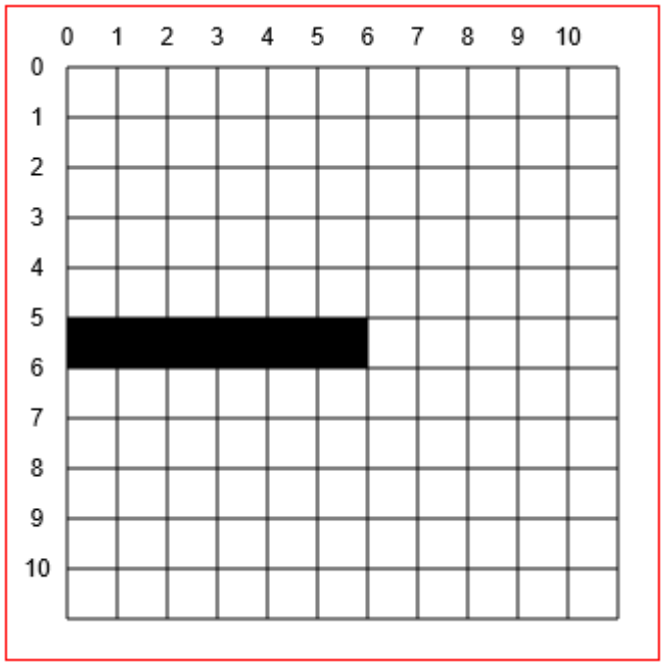
I tratti inusuali sono disegnati

Considera questo esempio Path che disegna una linea nera da 1 pixel da `[0,5]` a `[5,5]` :

```
// draw a 1 pixel black line from [0,5] to [5,5]
context.strokeStyle='black';
context.lineWidth=1;
context.beginPath();
context.moveTo(0,5);
context.lineTo(5,5);
context.stroke();
```

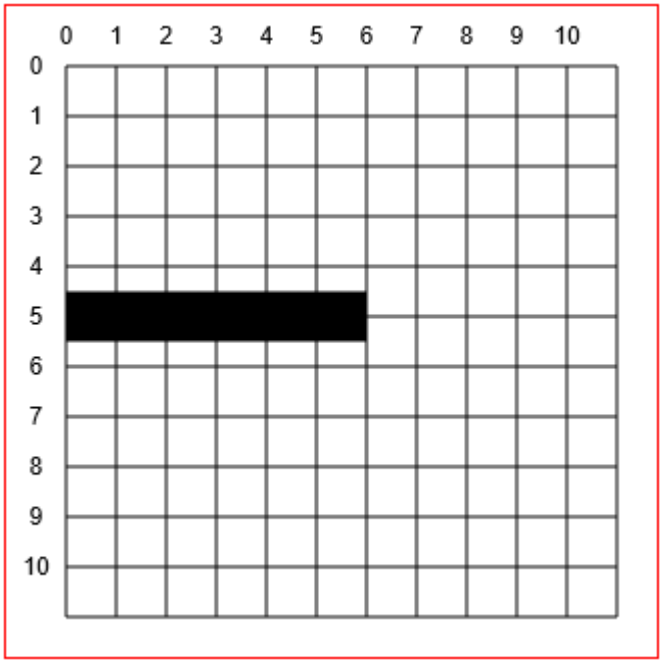
Domanda: che cosa il browser effettivamente disegna sulla tela?

Probabilmente ti aspetteresti di ottenere 6 pixel neri su y = 5



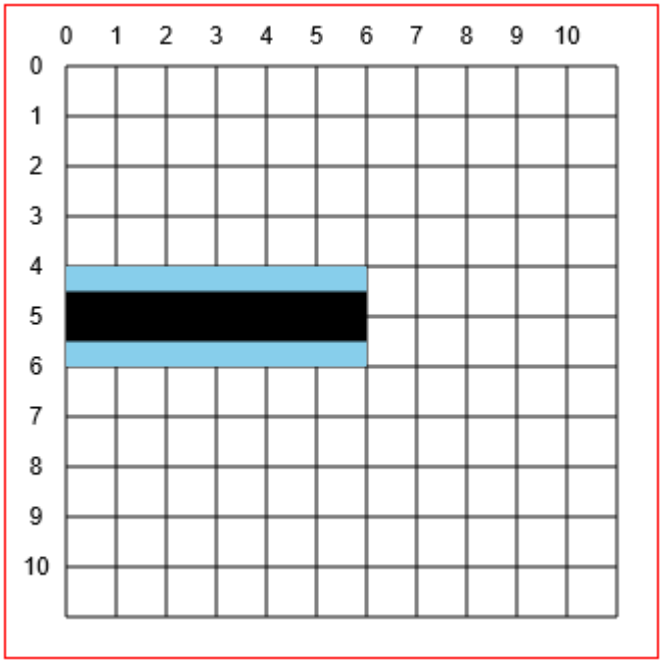
Ma (!) ... La tela disegna sempre tratti a metà strada su entrambi i lati del percorso definito!

Quindi dal momento che la linea è definita in `y==5.0` Canvas vuole tracciare la linea tra `y==4.5` e `y==5.5`

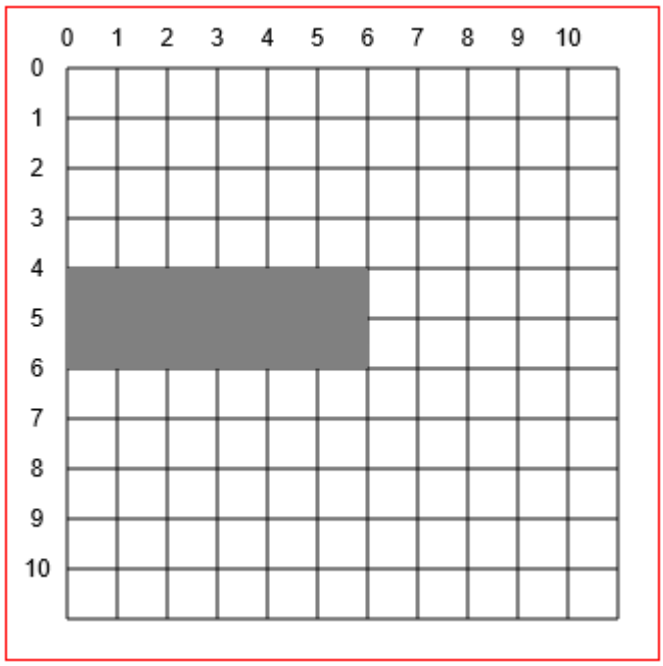


Ma, ancora una volta (!) ... Il display del computer non può disegnare mezzo pixel!

Quindi cosa si deve fare con i semi-pixel indesiderati (mostrati in blu sotto)?



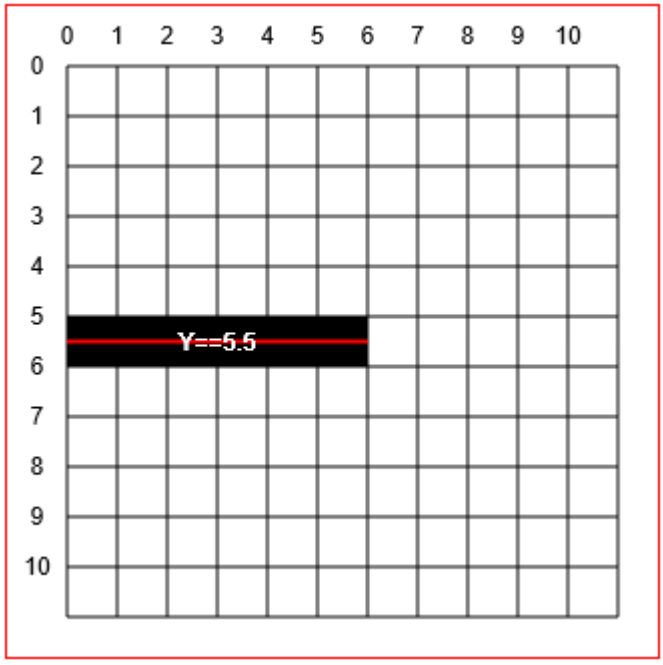
La risposta è che Canvas in effetti ordina al display di disegnare una linea larga 2 pixel dalla `4.0` alla `6.0` . Inoltre, colora la linea più chiara del `black` definito. Questo strano comportamento di disegno è "anti-aliasing" e aiuta Canvas a evitare di disegnare tratti che sembrano frastagliati.



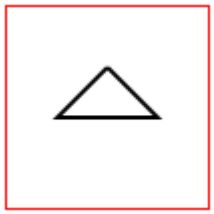
Un trucco di regolazione che funziona SOLO per i tratti orizzontali e verticali

Puoi ottenere una linea nera piena di 1 pixel specificando la linea da disegnare sul mezzo pixel:

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



Esempio di codice che utilizza `context.stroke()` per tracciare un tracciato tracciato sulla tela:



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

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

    ctx.beginPath();
    ctx.moveTo(50,30);
    ctx.lineTo(75,55);
    ctx.lineTo(25,55);
    ctx.lineTo(50,30);
    ctx.lineWidth=2;
    ctx.stroke();

}); // end window.onload
</script>
</head>
<body>
    <canvas id="canvas" width=100 height=100></canvas>
</body>
</html>
```

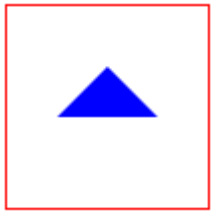
riempire (un comando di percorso)

```
context.fill()
```

Fa sì che l'interno del percorso venga riempito in base al `context.fillStyle` corrente.fillStyle e il percorso riempito viene visivamente disegnato sulla tela.

Prima di eseguire `context.fill` (o `context.stroke`) il Path esiste in memoria e non è ancora disegnato visivamente sulla tela.

Esempio di codice utilizzando `context.fill()` per disegnare un percorso pieno `context.fill()` di disegno:



```
<!doctype html>
<html>
<head>
<style>
    body{ background-color:white; }
```

```
    #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

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

    ctx.beginPath();
    ctx.moveTo(50,30);
    ctx.lineTo(75,55);
    ctx.lineTo(25,55);
    ctx.lineTo(50,30);
    ctx.fillStyle='blue';
    ctx.fill();

}); // end window.onload
</script>
</head>
<body>
    <canvas id="canvas" width=100 height=100></canvas>
</body>
</html>
```

clip (un comando di percorso)

```
context.clip
```

Limita eventuali disegni futuri da visualizzare solo all'interno del percorso corrente.

Esempio: ritaglia questa immagine in un percorso triangolare

```
<!doctype html>
<html>
<head>
<style>
```

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

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

    var img=new Image();
    img.onload=start;
    img.src='http://i.stack.imgur.com/1CqWf.jpg'

    function start(){
        // draw a triangle path
        ctx.beginPath();
        ctx.moveTo(75,50);
        ctx.lineTo(125,100);
        ctx.lineTo(25,100);
        ctx.lineTo(75,50);

        // clip future drawings to appear only in the triangle
        ctx.clip();

        // draw an image
        ctx.drawImage(img,0,0);
    }

}); // end window.onload
</script>
</head>
<body>
    <canvas id="canvas" width=150 height=150></canvas>
</body>
</html>
```

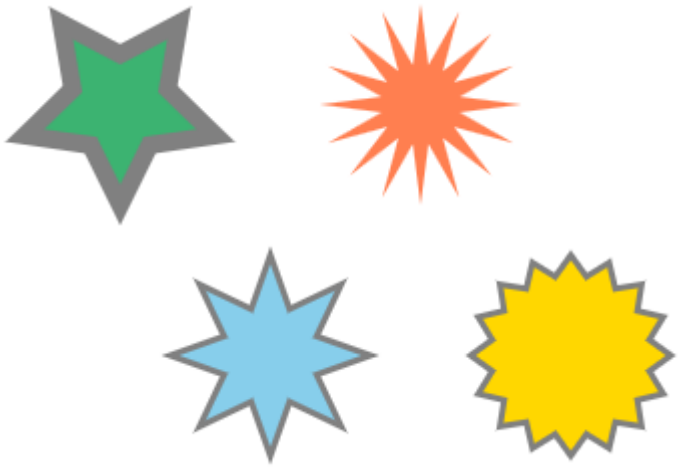
Leggi Percorso (solo sintassi) online: https://riptutorial.com/it/html5-canvas/topic/3241/percorso--solo-sintassi-

Capitolo 13: poligoni

Examples

Stelle

Disegna stelle con uno stile flessibile (dimensioni, colori, numero di punti).



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

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

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

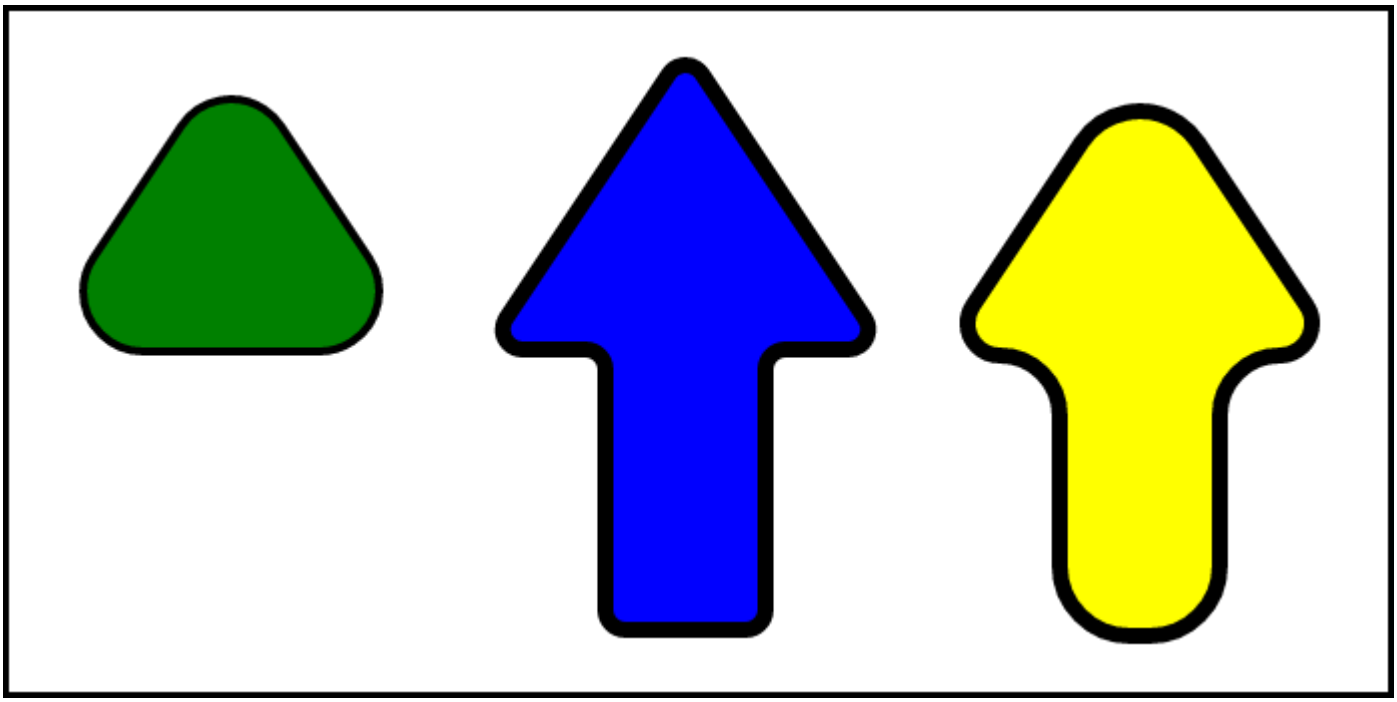
Poligono regolare

Un poligono regolare ha tutti i lati di uguale lunghezza.

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

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

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

Rendi un poligono arrotondato.

Crea un percorso da un insieme di punti `[{x:?,y:?},{x:?,y:?},...,{x:?,y:?}]` con angoli arrotondati del raggio. Se l'angolo dell'angolo è troppo piccolo per adattarsi al raggio o la distanza tra gli angoli non consente di ridurre il raggio degli angoli al miglior adattamento.



Esempio di utilizzo

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

Funzione di rendering

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

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

Leggi poligoni online: https://riptutorial.com/it/html5-canvas/topic/5493/poligoni

Capitolo 14: Shadows

Examples

Effetto adesivo usando le ombre

Questo codice aggiunge esternamente ombre a un'immagine per creare una versione "adesiva" dell'immagine.

Gli appunti:

- Oltre ad essere un oggetto ImageObject, l'argomento "img" può anche essere un elemento Canvas. Ciò ti consente di auto-adesivizzare i tuoi disegni personalizzati. Se si disegna un testo sull'argomento Canvas, è anche possibile applicare un adesivo a tale testo.
- Le immagini completamente opache non avranno effetto adesivo perché l'effetto è disegnato attorno a gruppi di pixel opachi che sono bordati da pixel trasparenti.

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

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

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

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

Come smettere di ombreggiare ulteriormente

Una volta attivato lo shadowing, ogni nuovo disegno sulla tela verrà ombreggiato.

Disattiva ulteriormente l'ombreggiatura impostando `context.shadowColor` su un colore trasparente.

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

... render some shadowed drawings ...

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

Shadowing è computazionalmente costoso - Cache that shadow!

Avvertimento! Applica le ombre con parsimonia!

Applicare lo shadowing è costoso ed è molto costoso se si applica lo shadowing all'interno di un ciclo di animazione.

Invece, memorizza nella cache una versione ombreggiata della tua immagine (o di un altro disegno):

- All'inizio della tua app, crea una versione ombreggiata della tua immagine in un secondo Canvas in memoria: `var memoryCanvas = document.createElement('canvas') ...`
- Ogni volta che hai bisogno della versione in ombra, disegna l'immagine pre-ombreggiata dalla tela in memoria alla tela visibile: `context.drawImage(memoryCanvas,x,y)`

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

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

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

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

Aggiungi profondità visiva con le ombre

L'uso tradizionale di shadowing è di dare ai disegni bidimensionali l'illusione della profondità 3D.

Questo esempio mostra lo stesso "pulsante" con e senza ombra

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

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

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

// with shadow
```

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

Ombre interne

Canvas non ha l' `inner-shadow` del CSS.

- La tela ombreggia l'esterno di una forma piena.
- La tela ombreggia sia all'interno che all'esterno di una forma accarezzata.

Ma è facile creare ombre interne usando il compositing.

Colpi con un'ombra interiore

Per creare tratti con un'ombra interiore, utilizzare la composizione di `destination-in` cui il contenuto esistente rimane solo dove il contenuto esistente viene sovrapposto al nuovo contenuto. I contenuti esistenti che non sono sovrapposti da nuovi contenuti vengono cancellati.

1. **Traccia una forma con un'ombra.** L'ombra si estenderà sia verso l'esterno che verso l'interno dal tratto. Dobbiamo sbarazzarci dell'ombra esterna, lasciando solo l'ombra interiore desiderata.
2. **Imposta il compositing alla `destination-in`** cui mantiene l'ombra tratteggiata esistente solo dove è sovrapposta a eventuali nuovi disegni.
3. **Riempi la forma.** Questo fa sì che il tratto e l'ombra interiore rimangano mentre l'ombra esterna viene cancellata. *Bene, non esattamente! Poiché un tratto è metà dentro e metà all'esterno della forma piena, anche la metà esterna del tratto verrà cancellata. La correzione è raddoppiare il `context.lineWidth` modo che metà del tratto a doppia dimensione sia ancora all'interno della forma riempita.*

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

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

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

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

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

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


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

Riempie riempito con un'ombra interiore

Per creare riempimenti con un'ombra interiore, seguire i passaggi da 1 a 3 sopra, ma utilizzare ulteriormente `destination-over` compositing di `destination-over` che consente di disegnare nuovi contenuti **sotto il contenuto esistente** .

4. **Imposta il compositing su `destination-over`** che fa disegnare il riempimento **sotto** l'ombra interna esistente.
5. **Disattiva lo shadowing** impostando `context.shadowColor` su un colore trasparente.
6. **Riempi la forma** con il colore desiderato. La forma sarà riempita sotto l'ombra interiore esistente.

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

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

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

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

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

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

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

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

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

Riempimenti non a tratti con un'ombra interiore

Per disegnare una forma piena con un'ombra interiore, ma senza tratto, puoi disegnare il tratto fuori campo e usare `shadowOffsetX` per `shadowOffsetX` l'ombra `shadowOffsetX` di `shadowOffsetX` .

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

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

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

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

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

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

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

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

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

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

Leggi Shadows online: https://riptutorial.com/it/html5-canvas/topic/5322/shadows

Capitolo 15: Testo

Examples

Disegno di testo

Disegnare su tela non è solo limitato a forme e immagini. Puoi anche disegnare del testo sulla tela.

Per disegnare del testo sulla tela, ottenere un riferimento alla tela e quindi chiamare il metodo `fillText` sul contesto.

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

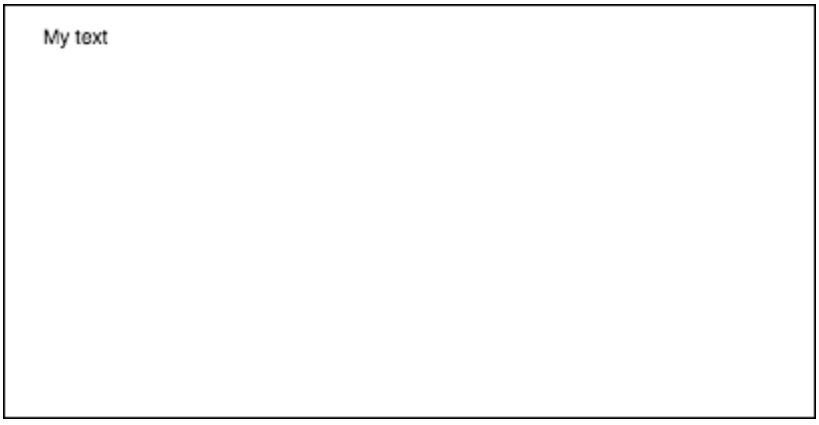
I tre argomenti **obbligatori** passati in `fillText` sono:

1. Il testo che vorresti mostrare
2. La posizione orizzontale (asse x)
3. La posizione verticale (asse y)

Inoltre, esiste un quarto argomento **facoltativo** , che è possibile utilizzare per specificare la larghezza massima del testo in pixel. Nell'esempio sotto il valore di `200` limita la larghezza massima del testo a 200px:

```
ctx.fillText("My text", 0, 0, 200);
```

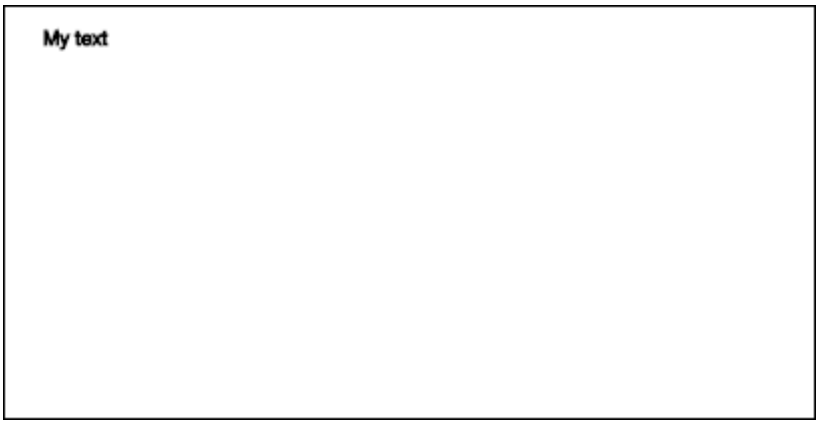
Risultato:



Puoi anche disegnare il testo senza un riempimento, e solo un contorno, usando il metodo `strokeText` :

```
ctx.strokeText("My text", 0, 0);
```

Risultato:



Senza le proprietà di formattazione dei caratteri applicate, la tela esegue il rendering del testo a 10px in sans-serif per impostazione predefinita, rendendo difficile vedere la differenza tra il risultato dei metodi `fillText` e `strokeText` . Vedere l' esempio di formattazione del testo per i dettagli su come aumentare le dimensioni del testo e applicare altre modifiche estetiche al testo.

Formattare il testo

La formattazione predefinita dei font fornita dai metodi `fillText` e `strokeText` non è molto esteticamente attraente. Fortunatamente l'API canvas fornisce proprietà per la formattazione del testo.

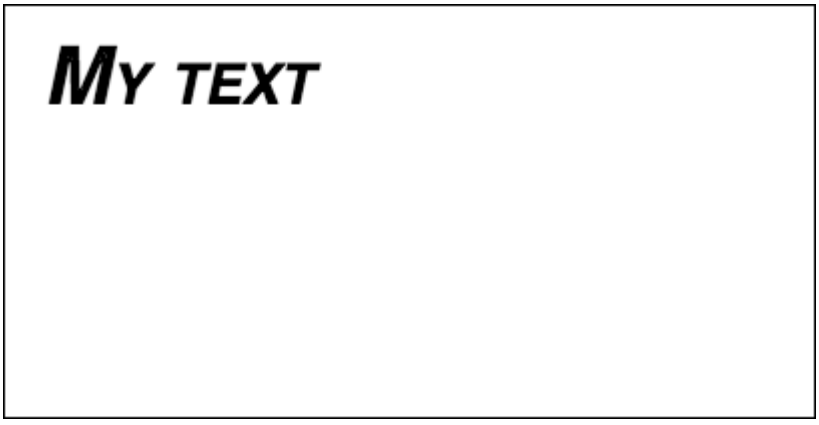
Usando la proprietà `font` puoi specificare:

- stile del font
- font-variant
- font-weight
- font-size / line-height
- famiglia di font

Per esempio:

```
ctx.font = "italic small-caps bold 40px Helvetica, Arial, sans-serif";
ctx.fillText("My text", 20, 50);
```

Risultato:



Usando la proprietà `textAlign` puoi anche cambiare l'allineamento del testo in entrambi:

- sinistra
- centro
- destra
- fine (uguale a destra)
- inizio (come a sinistra)

Per esempio:

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

Disporre il testo in paragrafi

L'API Native Canvas non ha un metodo per avvolgere il testo sulla riga successiva quando viene raggiunta la larghezza massima desiderata. Questo esempio avvolge il testo in paragrafi.

```
function wrapText(text, x, y, maxWidth, fontSize, fontFace){
  var firstY=y;
  var words = text.split(' ');
  var line = '';
  var lineHeight=fontSize*1.286; // a good approx for 10-18px sizes

  ctx.font=fontSize+" "+fontFace;
  ctx.textBaseline='top';

  for(var n = 0; n < words.length; n++) {
    var testLine = line + words[n] + ' ';
    var metrics = ctx.measureText(testLine);
    var testWidth = metrics.width;
    if(testWidth > maxWidth) {
      ctx.fillText(line, x, y);
      if(n<words.length-1){
          line = words[n] + ' ';
          y += lineHeight;
      }
    }
    else {
      line = testLine;
    }
  }
  ctx.fillText(line, x, y);
}
```

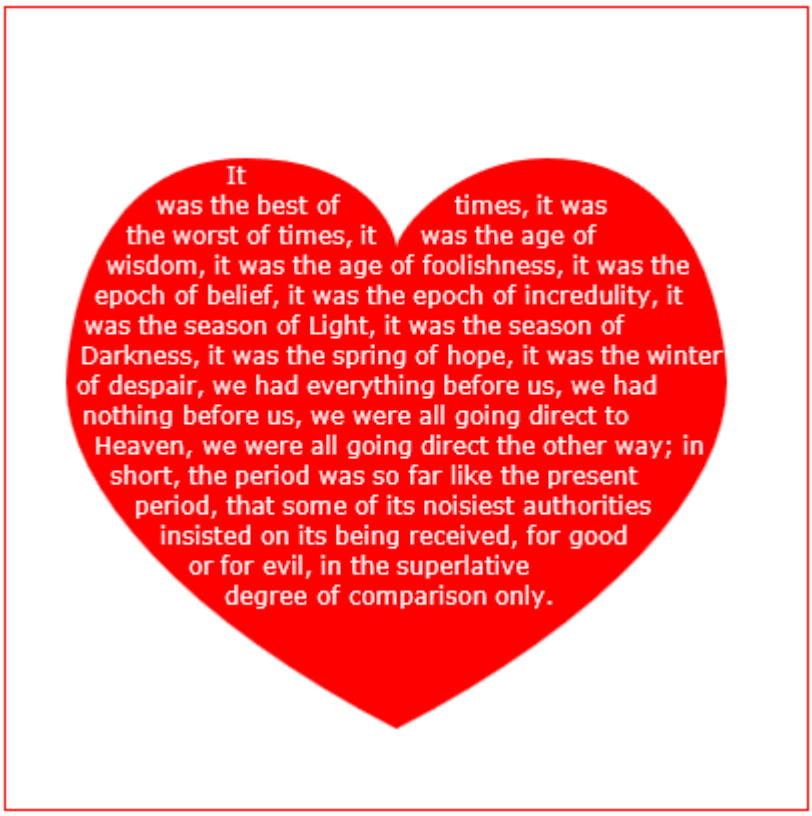
Disegna i paragrafi di testo in forme irregolari

Questo esempio disegna i paragrafi di testo in qualsiasi parte dell'area di disegno che presenta pixel opachi.

Funziona trovando il prossimo blocco di pixel opachi che è abbastanza grande da contenere la successiva parola specificata e riempire quel blocco con la parola specificata.

I pixel opachi possono provenire da qualsiasi fonte: comandi di disegno del percorso e / o

immagini.




```
<!doctype html>
<html>
<head>
<style>
    body{ background-color:white; padding:10px; }
    #canvas{border:1px solid red;}
</style>
<script>
window.onload=(function(){

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

    var fontsize=12;
    var fontface='verdana';
    var lineHeight=parseInt(fontsize*1.286);

    var text='It was the best of times, it was the worst of times, it was the age of wisdom,
it was the age of foolishness, it was the epoch of belief, it was the epoch of incredulity, it
was the season of Light, it was the season of Darkness, it was the spring of hope, it was the
winter of despair, we had everything before us, we had nothing before us, we were all going
direct to Heaven, we were all going direct the other way; in short, the period was so far like
the present period, that some of its noisiest authorities insisted on its being received, for
good or for evil, in the superlative degree of comparison only.';
    var words=text.split(' ');
    var wordWidths=[];
    ctx.font=fontsize+'px '+fontface;
    for(var i=0;i<words.length;i++){ wordWidths.push(ctx.measureText(words[i]).width); }
    var spaceWidth=ctx.measureText(' ').width;
    var wordIndex=0
```

```
var data=[];

// Demo: draw Heart
// Note: the shape can be ANY opaque drawing -- even an image
ctx.scale(3,3);
ctx.beginPath();
ctx.moveTo(75,40);
ctx.bezierCurveTo(75,37,70,25,50,25);
ctx.bezierCurveTo(20,25,20,62.5,20,62.5);
ctx.bezierCurveTo(20,80,40,102,75,120);
ctx.bezierCurveTo(110,102,130,80,130,62.5);
ctx.bezierCurveTo(130,62.5,130,25,100,25);
ctx.bezierCurveTo(85,25,75,37,75,40);
ctx.fillStyle='red';
ctx.fill();
ctx.setTransform(1,0,0,1,0,0);

// fill heart with text
ctx.fillStyle='white';
var imgDataData=ctx.getImageData(0,0,cw,ch).data;
for(var i=0;i<imgDataData.length;i+=4){
    data.push(imgDataData[i+3]);
}
placeWords();

// draw words sequentially into next available block of
//    available opaque pixels
function placeWords(){
    var sx=0;
    var sy=0;
    var y=0;
    var wordIndex=0;
    ctx.textBaseline='top';
    while(y<ch && wordIndex<words.length){
        sx=0;
        sy=y;
        var startingIndex=wordIndex;
        while(sx<cw && wordIndex<words.length){
            var x=getRect(sx,sy,lineHeight);
            var available=x-sx;
            var spacer=spaceWidth;  // spacer=0 to have no left margin
            var w=spacer+wordWidths[wordIndex];
            while(available>=w){
                ctx.fillText(words[wordIndex],spacer+sx,sy);
                sx+=w;
                available-=w;
                spacer=spaceWidth;
                wordIndex++;
                w=spacer+wordWidths[wordIndex];
            }
            sx=x+1;
        }
        y=(wordIndex>startingIndex)?y+lineHeight:y+1;
    }
}

// find a rectangular block of opaque pixels
function getRect(sx,sy,height){
    var x=sx;
    var y=sy;
    var ok=true;
```

```
        while(ok){
            if(data[y*cw+x]<250){ok=false;}
            y++;
            if(y>=sy+height){
                y=sy;
                x++;
                if(x>=cw){ok=false;}
            }
        }
        return(x);
    }

}); // end $(function(){});
</script>
</head>
<body>
    <h4>Note: the shape must be closed and alpha>=250 inside</h4>
    <canvas id="canvas" width=400 height=400></canvas>
</body>
</html>
```

Riempi il testo con un'immagine

Questo esempio riempie il testo con un'immagine specificata.

Importante! L'immagine specificata deve essere completamente caricata prima di chiamare questa funzione o il disegno fallirà. Usa `image.onload` per essere sicuro che l'immagine sia completamente caricata.

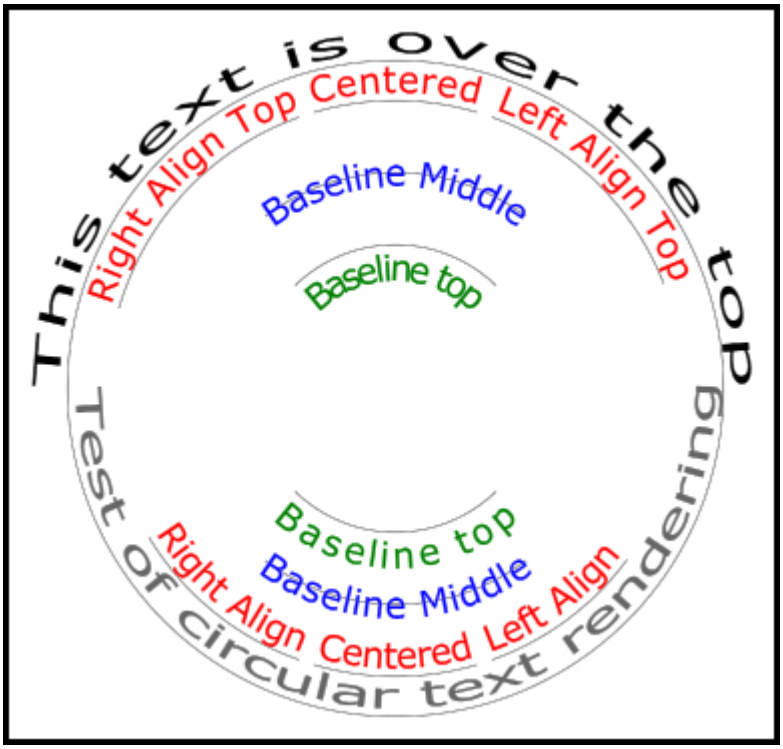

```
function drawImageInsideText(canvas,x,y,img,text,font){
    var c=canvas.cloneNode();
    var ctx=c.getContext('2d');
    ctx.font=font;
    ctx.fillText(text,x,y);
    ctx.globalCompositeOperation='source-atop';
    ctx.drawImage(img,0,0);
    canvas.getContext('2d').drawImage(c,0,0);
}
```

Rendering del testo lungo un arco.

Questo esempio mostra come eseguire il rendering del testo lungo un arco. Include come aggiungere funzionalità a `CanvasRenderingContext2D` estendendo il suo prototipo.

Questo esempio è derivato dal testo circolare risposta StackOverflow.

Esempio di rendering



Codice di esempio

L'esempio aggiunge 3 nuove funzioni di rendering del testo al prototipo del contesto 2D.

- **ctx.fillCircleText (testo, x, y, raggio, inizio, fine, avanti);**
- **ctx.strokeCircleText (testo, x, y, raggio, inizio, fine, avanti);**
- **ctx.measureCircleText (testo, raggio);**

```
(function(){
    const FILL = 0;        // const to indicate filltext render
    const STROKE = 1;
    var renderType = FILL; // used internal to set fill or stroke text
    const multiplyCurrentTransform = true; // if true Use current transform when rendering
                                           // if false use absolute coordinates which is a
little quicker
                                           // after render the currentTransform is restored to
default transform



    // measure circle text
    // ctx: canvas context
    // text: string of text to measure
    // r: radius in pixels
    //
    // returns the size metrics of the text
    //
    // width: Pixel width of text
    // angularWidth : angular width of text in radians
```

```
    // pixelAngularSize : angular width of a pixel in radians
    var measure = function(ctx, text, radius){
        var textWidth = ctx.measureText(text).width; // get the width of all the text
        return {
            width               : textWidth,
            angularWidth        : (1 / radius) * textWidth,
            pixelAngularSize    : 1 / radius
        };
    }

    // displays text along a circle
    // ctx: canvas context
    // text: string of text to measure
    // x,y: position of circle center
    // r: radius of circle in pixels
    // start: angle in radians to start.
    // [end]: optional. If included text align is ignored and the text is
    //        scaled to fit between start and end;
    // [forward]: optional default true. if true text direction is forwards, if false
direction is backward
    var circleText = function (ctx, text, x, y, radius, start, end, forward) {
        var i, textWidth, pA, pAS, a, aw, wScale, aligned, dir, fontSize;
        if(text.trim() === "" || ctx.globalAlpha === 0){ // dont render empty string or
transparent
            return;
        }
        if(isNaN(x) || isNaN(y) || isNaN(radius) || isNaN(start) || (end !== undefined && end
!== null && isNaN(end))){ //
            throw TypeError("circle text arguments requires a number for x,y, radius, start,
and end.")
        }
        aligned = ctx.textAlign;        // save the current textAlign so that it can be
restored at end
        dir = forward ? 1 : forward === false ? -1 : 1;  // set dir if not true or false set
forward as true
        pAS = 1 / radius;               // get the angular size of a pixel in radians
        textWidth = ctx.measureText(text).width; // get the width of all the text
        if (end !== undefined && end !== null) { // if end is supplied then fit text between
start and end
            pA = ((end - start) / textWidth) * dir;
            wScale = (pA / pAS) * dir;
        } else {                 // if no end is supplied correct start and end for alignment
            // if forward is not given then swap top of circle text to read the correct
direction
            if(forward === null || forward === undefined){
                if(((start % (Math.PI * 2)) + Math.PI * 2) % (Math.PI * 2) > Math.PI){
                    dir = -1;
                }
            }
            pA = -pAS * dir ;
            wScale = -1 * dir;
            switch (aligned) {
            case "center":       // if centered move around half width
                start -= (pA * textWidth )/2;
                end = start + pA * textWidth;
                break;
            case "right":// intentionally falls through to case "end"
            case "end":
                end = start;
                start -= pA * textWidth;
                break;
```

```
            case "left":  // intentionally falls through to case "start"
            case "start":
                end = start + pA * textWidth;
            }
        }

        ctx.textAlign = "center";                       // align for rendering
        a = start;                                      // set the start angle
        for (var i = 0; i < text.length; i += 1) {    // for each character
            aw = ctx.measureText(text[i]).width * pA; // get the angular width of the text
            var xDx = Math.cos(a + aw / 2);           // get the yAxies vector from the center
x,y out
            var xDy = Math.sin(a + aw / 2);
            if(multiplyCurrentTransform){ // transform multiplying current transform
                ctx.save();
                if (xDy < 0) { // is the text upside down. If it is flip it
                    ctx.transform(-xDy * wScale, xDx * wScale, -xDx, -xDy, xDx * radius + x,
xDy * radius + y);
                } else {
                    ctx.transform(-xDy * wScale, xDx * wScale, xDx, xDy, xDx * radius + x, xDy
* radius + y);
                }
            }else{
                if (xDy < 0) { // is the text upside down. If it is flip it
                    ctx.setTransform(-xDy * wScale, xDx * wScale, -xDx, -xDy, xDx * radius +
x, xDy * radius + y);
                } else {
                    ctx.setTransform(-xDy * wScale, xDx * wScale, xDx, xDy, xDx * radius + x,
xDy * radius + y);
                }
            }
            if(renderType === FILL){
                ctx.fillText(text[i], 0, 0);    // render the character
            }else{
                ctx.strokeText(text[i], 0, 0);  // render the character
            }
            if(multiplyCurrentTransform){  // restore current transform
                ctx.restore();
            }
            a += aw;                      // step to the next angle
        }
        // all done clean up.
        if(!multiplyCurrentTransform){
            ctx.setTransform(1, 0, 0, 1, 0, 0); // restore the transform
        }
        ctx.textAlign = aligned;            // restore the text alignment
    }
    // define fill text
    var fillCircleText = function(text, x, y, radius, start, end, forward){
        renderType = FILL;
        circleText(this, text, x, y, radius, start, end, forward);
    }
    // define stroke text
    var strokeCircleText = function(text, x, y, radius, start, end, forward){
        renderType = STROKE;
        circleText(this, text, x, y, radius, start, end, forward);
    }
    // define measure text
    var measureCircleTextExt = function(text,radius){
        return measure(this, text, radius);
    }
```

```
    // set the prototypes
    CanvasRenderingContext2D.prototype.fillCircleText = fillCircleText;
    CanvasRenderingContext2D.prototype.strokeCircleText = strokeCircleText;
    CanvasRenderingContext2D.prototype.measureCircleText = measureCircleTextExt;
})();
```

Descrizioni delle funzioni

Questo esempio aggiunge 3 funzioni al `CanvasRenderingContext2D prototype` . `fillCircleText` , `strokeCircleText` e `measureCircleText`

CanvasRenderingContext2D.fillCircleText (testo, x, y, raggio, inizio, [fine, [avanti]]);

CanvasRenderingContext2D.strokeCircleText (testo, x, y, raggio, inizio, [fine, [avanti]]);

- **testo:** testo da rendere come stringa.
- **x** , **y** : posizione del centro del cerchio come numeri.
- **raggio:** raggio del cerchio in pixel
- **inizio:** angolo in radianti per iniziare.
- **[fine]:** facoltativo. Se incluso `ctx.textAlign` viene ignorato e il testo viene ridimensionato per adattarsi tra inizio e fine.
- **[avanti]:** facoltativo predefinito 'true'. se la direzione del testo vero è in avanti, se la direzione 'falso' è indietro.

Entrambe le funzioni utilizzano la barra di testo per posizionare il testo verticalmente intorno al raggio. Per i migliori risultati usa `ctx.TextBaseline` .

Le funzioni generano un `TypeError` uno qualsiasi degli argomenti numerici come NaN.

Se l'argomento di `text` taglia su una stringa vuota o `ctx.globalAlpha = 0` la funzione passa e non fa nulla.

CanvasRenderingContext2D.measureCircleText (testo, raggio);

```
- **text:** String of text to measure.
- **radius:** radius of circle in pixels.
```

Restituisce un oggetto contenente metriche di varie dimensioni per il rendering di testo circolare

```
   - **width:** Pixel width of text as it would normaly be rendered
   - **angularWidth:** angular width of text in radians.
   - **pixelAngularSize:** angular width of a pixel in radians.
```

Esempi di utilizzo

```
const rad = canvas.height * 0.4;
const text = "Hello circle TEXT!";
const fontSize = 40;
const centX = canvas.width / 2;
const centY = canvas.height / 2;
ctx.clearRect(0,0,canvas.width,canvas.height)

ctx.font = fontSize + "px verdana";
ctx.textAlign = "center";
ctx.textBaseline = "bottom";
ctx.fillStyle = "#000";
ctx.strokeStyle = "#666";

// Text under stretched from Math.PI to 0 (180 - 0 deg)
ctx.fillCircleText(text, centX, centY, rad, Math.PI, 0);

// text over top centered at Math.PI * 1.5 ( 270 deg)
ctx.fillCircleText(text, centX, centY, rad, Math.PI * 1.5);

// text under top centered at Math.PI * 1.5 ( 270 deg)
ctx.textBaseline = "top";
ctx.fillCircleText(text, centX, centY, rad, Math.PI * 1.5);


// text over top centered at Math.PI * 1.5 ( 270 deg)
ctx.textBaseline = "middle";
ctx.fillCircleText(text, centX, centY, rad, Math.PI * 1.5);


// Use measureCircleText to get angular size
var circleTextMetric = ctx.measureCircleText("Text to measure", rad);
console.log(circleTextMetric.width);            // width of text if rendered normally
console.log(circleTextMetric.angularWidth);     // angular width of text
console.log(circleTextMetric.pixelAngularSize); // angular size of a pixel



// Use measure text to draw a arc around the text
ctx.textBaseline = "middle";
var width = ctx.measureCircleText(text, rad).angularWidth;
ctx.fillCircleText(text, centX, centY, rad, Math.PI * 1.5);

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

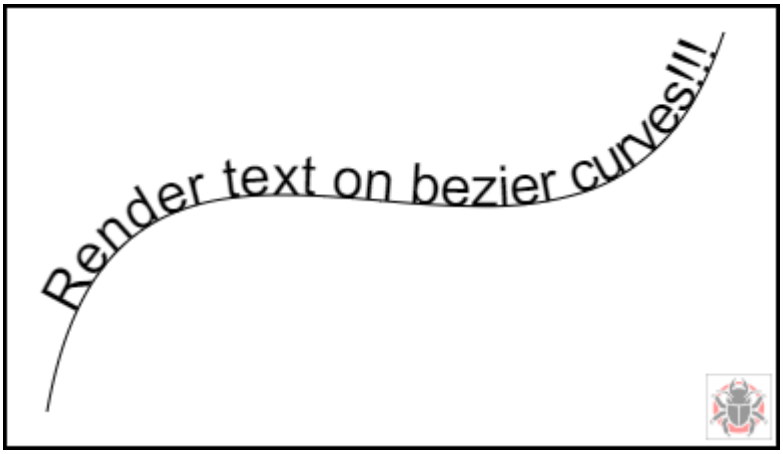
```
ctx.stroke();
```

NOTA: il testo visualizzato è solo un'approssimazione del testo circolare. Ad esempio, se due sono resi le due linee non saranno parallele, ma se si esegue il rendering di una "H" i due bordi saranno paralleli. Questo perché ogni personaggio è reso il più vicino possibile alla direzione richiesta, piuttosto che ogni pixel viene trasformato correttamente per creare testo circolare.

NOTA: `const multiplyCurrentTransform = true;` definito in questo esempio viene utilizzato per impostare il metodo di trasformazione utilizzato. Se `false` la trasformazione per il rendering circolare del testo è assoluta e non dipende dallo stato corrente della trasformazione. Il testo non sarà influenzato da precedenti scale, rotazioni o traduzioni di trasformazioni. Ciò aumenterà le prestazioni della funzione di rendering, dopo che la funzione è stata chiamata la trasformazione verrà impostata sul `setTransform(1,0,0,1,0,0)` predefinito `setTransform(1,0,0,1,0,0)`

Se `multiplyCurrentTransform = true` (impostato come predefinito in questo esempio) il testo utilizzerà la trasformazione corrente in modo che il testo possa essere ridimensionato, traslato, inclinato, ruotato, ecc. Ma modificando la trasformazione corrente prima di chiamare le funzioni `fillCircleText` e `strokeCircleText` . A seconda dello stato corrente del contesto 2D, questo potrebbe essere un po 'più lento di quello di `multiplyCurrentTransform = false`

Testo su beziers a curva, cubici e quadratici



TextOnCurve (testo, offset, x1, y1, x2, y2, x3, y3, x4, Y4)

Rende il testo su curve quadratiche e cubiche.

- `text` è il testo da rappresentare
- distanza di `offset` dall'inizio della curva al testo> = 0
- `x1,y1` - `x3,y3` punti di curva quadratica o
- `x1,y1` - `x4,y4` punti di curva cubica o

Esempio di utilizzo:

```
textOnCurve("Hello world!",50,100,100,200,200,300,100); // draws text on quadratic curve
                                                        // 50 pixels from start of curve


textOnCurve("Hello world!",50,100,100,200,200,300,100,400,200);
                                                        // draws text on cubic curve
                                                        // 50 pixels from start of curve
```

Funzione Function e curver helper

```
// pass 8 values for cubic bezier
// pass 6 values for quadratic
// Renders text from start of curve
var textOnCurve = function(text,offset,x1,y1,x2,y2,x3,y3,x4,y4){
    ctx.save();
    ctx.textAlign = "center";
    var widths = [];
    for(var i = 0; i < text.length; i ++){
        widths[widths.length] = ctx.measureText(text[i]).width;
    }
    var ch = curveHelper(x1,y1,x2,y2,x3,y3,x4,y4);
    var pos = offset;
    var cpos = 0;

    for(var i = 0; i < text.length; i ++){
        pos += widths[i] / 2;
        cpos = ch.forward(pos);
        ch.tangent(cpos);
        ctx.setTransform(ch.vect.x, ch.vect.y, -ch.vect.y, ch.vect.x, ch.vec.x, ch.vec.y);
        ctx.fillText(text[i],0,0);

        pos += widths[i] / 2;
    }
    ctx.restore();
}
```

La funzione di helper della curva è progettata per aumentare le prestazioni nel trovare punti su Bezier.

```
// helper function locates points on bezier curves.
function curveHelper(x1, y1, x2, y2, x3, y3, x4, y4){
    var tx1, ty1, tx2, ty2, tx3, ty3, tx4, ty4;
    var a,b,c,u;
    var vec,currentPos,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){
```

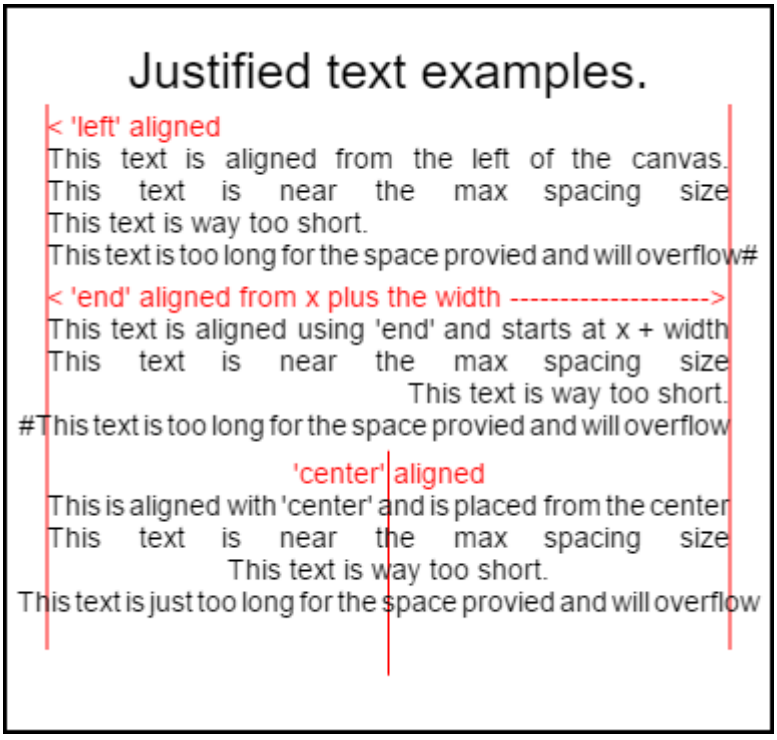
```
        tx1 = x1; ty1 = y1;
        tx2 = x2; ty2 = y2;
        tx3 = x3; ty3 = y3;
        tx1 += (tx2 - tx1) * c;
        ty1 += (ty2 - ty1) * c;
        tx2 += (tx3 - tx2) * c;
        ty2 += (ty3 - ty2) * c;
        tx3 += (x4 - tx3) * c;
        ty3 += (y4 - ty3) * c;
        tx1 += (tx2 - tx1) * 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
}
```

Testo giustificato

Questo esempio rende il testo giustificato. Aggiunge funzionalità extra a `CanvasRenderingContext2D` estendendo il suo prototipo o come oggetto globale `justifiedText` (opzionale vedi Nota A).

Esempio di rendering.



Il codice per il rendering di questa immagine è negli esempi di utilizzo in basso .

L'esempio

La funzione come funzione anonima invocata immediatamente.

```
(function(){
    const FILL = 0;        // const to indicate filltext render
    const STROKE = 1;
    const MEASURE = 2;
    var renderType = FILL; // used internal to set fill or stroke text

    var maxSpaceSize = 3; // Multiplier for max space size. If greater then no justificatoin
applied
    var minSpaceSize = 0.5; // Multiplier for minimum space size
    var renderTextJustified = function(ctx,text,x,y,width){
        var words, wordsWidth, count, spaces, spaceWidth, adjSpace, renderer, i, textAlign,
useSize, totalWidth;
        textAlign = ctx.textAlign; // get current align settings
        ctx.textAlign = "left";
        wordsWidth = 0;
        words = text.split(" ").map(word => {
            var w = ctx.measureText(word).width;
            wordsWidth += w;
            return {
                width : w,
                word : word,
            };
        });
        // count = num words, spaces = number spaces, spaceWidth normal space size
        // adjSpace new space size >= min size. useSize Resulting space size used to render
        count = words.length;
        spaces = count - 1;
        spaceWidth = ctx.measureText(" ").width;
        adjSpace = Math.max(spaceWidth * minSpaceSize, (width - wordsWidth) / spaces);
        useSize = adjSpace > spaceWidth * maxSpaceSize ? spaceWidth : adjSpace;
        totalWidth = wordsWidth + useSize * spaces
        if(renderType === MEASURE){ // if measuring return size
            ctx.textAlign = textAlign;
            return totalWidth;
        }
        renderer = renderType === FILL ? ctx.fillText.bind(ctx) : ctx.strokeText.bind(ctx); //
fill or stroke
        switch(textAlign){
            case "right":
                x -= totalWidth;
                break;
            case "end":
                x += width - totalWidth;
                break;
            case "center": // intentional fall through to default
                x -= totalWidth / 2;
            default:
        }
        if(useSize === spaceWidth){ // if space size unchanged
            renderer(text,x,y);
        } else {
            for(i = 0; i < count; i += 1){
                renderer(words[i].word,x,y);
                x += words[i].width;
                x += useSize;
```

```
            }
        }
        ctx.textAlign = textAlign;
    }
    // Parse vet and set settings object.
    var justifiedTextSettings = function(settings){
        var min,max;
        var vetNumber = (num, defaultNum) => {
            num = num !== null && num !== null && !isNaN(num) ? num : defaultNum;
            if(num < 0){
                num = defaultNum;
            }
            return num;
        }
        if(settings === undefined || settings === null){
            return;
        }
        max = vetNumber(settings.maxSpaceSize, maxSpaceSize);
        min = vetNumber(settings.minSpaceSize, minSpaceSize);
        if(min > max){
            return;
        }
        minSpaceSize = min;
        maxSpaceSize = max;
    }
    // define fill text
    var fillJustifyText = function(text, x, y, width, settings){
        justifiedTextSettings(settings);
        renderType = FILL;
        renderTextJustified(this, text, x, y, width);
    }
    // define stroke text
    var strokeJustifyText = function(text, x, y, width, settings){
        justifiedTextSettings(settings);
        renderType = STROKE;
        renderTextJustified(this, text, x, y, width);
    }
    // define measure text
    var measureJustifiedText = function(text, width, settings){
        justifiedTextSettings(settings);
        renderType = MEASURE;
        return renderTextJustified(this, text, 0, 0, width);
    }
    // code point A
    // set the prototypes
    CanvasRenderingContext2D.prototype.fillJustifyText = fillJustifyText;
    CanvasRenderingContext2D.prototype.strokeJustifyText = strokeJustifyText;
    CanvasRenderingContext2D.prototype.measureJustifiedText = measureJustifiedText;
    // code point B

    // optional code if you do not wish to extend the CanvasRenderingContext2D prototype
    /* Uncomment from here to the closing comment
    window.justifiedText = {
        fill : function(ctx, text, x, y, width, settings){
            justifiedTextSettings(settings);
            renderType = FILL;
            renderTextJustified(ctx, text, x, y, width);
        },
        stroke : function(ctx, text, x, y, width, settings){
            justifiedTextSettings(settings);
            renderType = STROKE;
```

```
            renderTextJustified(ctx, text, x, y, width);
        },
        measure : function(ctx, text, width, settings){
            justifiedTextSettings(settings);
            renderType = MEASURE;
            return renderTextJustified(ctx, text, 0, 0, width);
        }
    }
    to here*/
})();
```

> **Nota A:** Se non desideri estendere il prototipo `CanvasRenderingContext2D` Rimuovi dall'esempio tutto il codice tra `// code point A` e `// code point B` e decommenta il codice contrassegnato `/* Uncomment from here to the closing comment`

Come usare

Tre funzioni vengono aggiunte a `CanvasRenderingContext2D` e sono disponibili per tutti gli oggetti di contesto 2D creati.

- ctx.fillJustifyText (testo, x, y, larghezza, [impostazioni]);
- ctx.strokeJustifyText (testo, x, y, larghezza, [impostazioni]);
- ctx.measureJustifiedText (testo, larghezza, [impostazioni]);

Riempi e tratti la funzione di testo, riempi o tratti il testo e utilizza gli stessi argomenti. `measureJustifiedText` restituirà la larghezza effettiva in cui il testo verrà visualizzato. Questo può essere uguale, inferiore o superiore alla `width` dell'argomento a seconda delle impostazioni correnti.

> **Nota: gli** argomenti all'interno di `[` e `]` sono facoltativi.

Argomenti di funzione

- **testo:** stringa contenente il testo da rendere.
- **x, y:** coordinate per rendere il testo a.
- **larghezza:** larghezza del testo giustificato. Il testo aumenterà / diminuirà gli spazi tra le parole per adattarsi alla larghezza. Se lo spazio tra le parole è maggiore di `maxSpaceSize` (default = 6), verrà utilizzata la spaziatura normale e il testo non riempirà la larghezza richiesta. Se la spaziatura è inferiore alla spaziatura `minSpaceSize` di `minSpaceSize` (default = 0.5), allora viene utilizzata la dimensione dello spazio min e il testo supererà la larghezza richiesta
- **impostazioni:** Opzionale. Oggetto contenente dimensioni minime e massime.

L'argomento `settings` è facoltativo e, se non incluso, il rendering del testo utilizzerà l'ultima impostazione definita o quella predefinita (mostrata sotto).

Sia min che max sono le dimensioni min e max per il carattere [spazio] che separa le parole. Il valore predefinito `maxSpaceSize = 6` indica che quando lo spazio tra i caratteri è> 63 * ctx.measureText (""). Il testo della larghezza non sarà giustificato. Se il testo da giustificare ha spazi inferiori a `minSpaceSize = 0.5` (valore predefinito 0.5) * `ctx.measureText(" ").width` la spaziatura verrà impostata su `minSpaceSize * ctx.measureText(" ").width` e il testo risultante verrà sovraccaricato la larghezza giustificante.

Le seguenti regole sono applicate, min e max devono essere numeri. In caso contrario, i valori associati non verranno modificati. Se `minSpaceSize` è maggiore di `maxSpaceSize` entrambe le impostazioni di input non sono valide e il valore min max non verrà modificato.

Esempio di impostazione dell'oggetto con valori predefiniti

```
settings = {
    maxSpaceSize : 6;   // Multiplier for max space size.
    minSpaceSize : 0.5; // Multiplier for minimum space size
};
```

> **NOTA:** Queste funzioni di testo introducono un sottile cambiamento di comportamento per la proprietà `textAlign` del contesto 2D. 'Left', 'right', 'center' e 'start' si comportano come previsto ma 'end' non si allinea dalla destra dell'argomento function `x` ma piuttosto dalla destra di `x + width`
>
> **Nota: le** impostazioni (dimensioni spazio min e max) sono globali per tutti gli oggetti contesto 2D.

Esempi di utilizzo

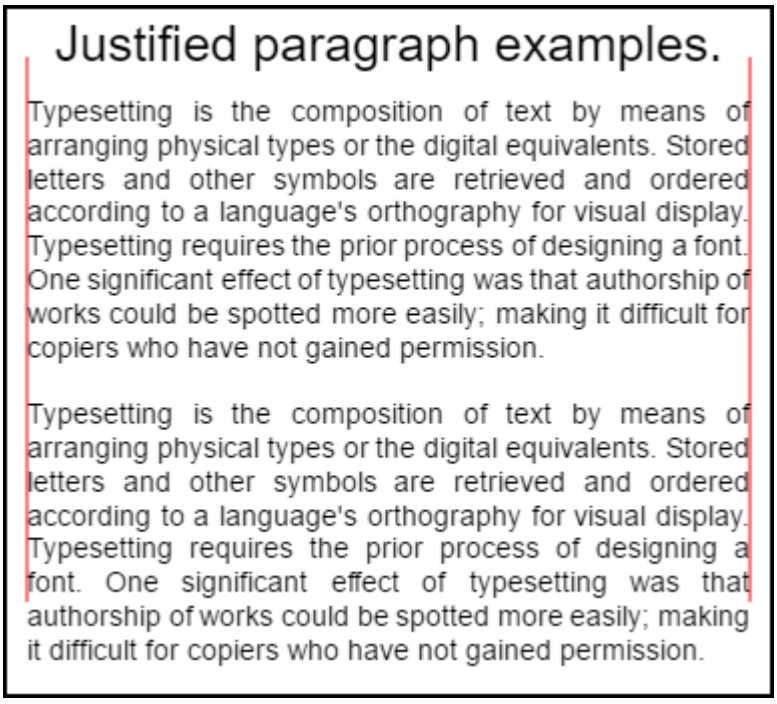
```
var i = 0;
text[i++] = "This text is aligned from the left of the canvas.";
text[i++] = "This text is near the max spacing size";
text[i++] = "This text is way too short.";
text[i++] = "This text is too long for the space provied and will overflow#";
text[i++] = "This text is aligned using 'end' and starts at x + width";
text[i++] = "This text is near the max spacing size";
text[i++] = "This text is way too short.";
text[i++] = "#This text is too long for the space provied and will overflow";
text[i++] = "This is aligned with 'center' and is placed from the center";
text[i++] = "This text is near the max spacing size";
text[i++] = "This text is way too short.";
text[i++] = "This text is just too long for the space provied and will overflow";

// ctx is the 2d context
// canvas is the canvas

ctx.clearRect(0,0,w,h);
ctx.font = "25px arial";
ctx.textAlign = "center"
var left = 20;
var center = canvas.width / 2;
var width = canvas.width-left*2;
var y = 40;
```

```
var size = 16;
var i = 0;
ctx.fillText("Justified text examples.",center,y);
y+= 40;
ctx.font = "14px arial";
ctx.textAlign = "left"
var ww = ctx.measureJustifiedText(text[0], width);
var setting = {
    maxSpaceSize : 6,
    minSpaceSize : 0.5
}
ctx.strokeStyle = "red"
ctx.beginPath();
ctx.moveTo(left,y - size * 2);
ctx.lineTo(left, y + size * 15);
ctx.moveTo(canvas.width - left,y - size * 2);
ctx.lineTo(canvas.width - left, y + size * 15);
ctx.stroke();
ctx.textAlign = "left";
ctx.fillStyle = "red";
ctx.fillText("< 'left' aligned",left,y - size)
ctx.fillStyle = "black";
ctx.fillJustifyText(text[i++], left, y, width, setting);  // settings is remembered
ctx.fillJustifyText(text[i++], left, y+=size, width);
ctx.fillJustifyText(text[i++], left, y+=size, width);
ctx.fillJustifyText(text[i++], left, y+=size, width);
y += 2.3*size;
ctx.fillStyle = "red";
ctx.fillText("< 'end' aligned from x plus the width ---------------------->",left,y - size)
ctx.fillStyle = "black";
ctx.textAlign = "end";
ctx.fillJustifyText(text[i++], left, y, width);
ctx.fillJustifyText(text[i++], left, y+=size, width);
ctx.fillJustifyText(text[i++], left, y+=size, width);
ctx.fillJustifyText(text[i++], left, y+=size, width);

y += 40;
ctx.strokeStyle = "red"
ctx.beginPath();
ctx.moveTo(center,y - size * 2);
ctx.lineTo(center, y + size * 5);
ctx.stroke();
ctx.textAlign = "center";
ctx.fillStyle = "red";
ctx.fillText("'center' aligned",center,y - size)
ctx.fillStyle = "black";
ctx.fillJustifyText(text[i++], center, y, width);
ctx.fillJustifyText(text[i++], center, y+=size, width);
ctx.fillJustifyText(text[i++], center, y+=size, width);
ctx.fillJustifyText(text[i++], center, y+=size, width);
```

Paragrafi giustificati.

Rende il testo come paragrafi giustificati. **RICHIEDE** l'esempio di **testo giustificato**

Esempio di rendering



Il paragrafo in alto ha **setting.compact = true** e bottom **false** e l'interlinea è **1.2** invece del default **1.5** . Resi dall'esempio di utilizzo del codice in fondo a questo esempio.

Codice di esempio

```
 // Requires justified text extensions
(function(){
    // code point A
    if(typeof CanvasRenderingContext2D.prototype.fillJustifyText !== "function"){
        throw new ReferenceError("Justified Paragraph extension missing requiered
CanvasRenderingContext2D justified text extension");
    }
    var maxSpaceSize = 3; // Multiplier for max space size. If greater then no justificatoin
applied
    var minSpaceSize = 0.5; // Multiplier for minimum space size
    var compact = true; // if true then try and fit as many words as possible. If false then
try to get the spacing as close as possible to normal
    var lineSpacing = 1.5; // space between lines
    const noJustifySetting = {  // This setting forces justified text off. Used to render last
line of paragraph.
        minSpaceSize : 1,
        maxSpaceSize : 1,
    }

    // Parse vet and set settings object.
    var justifiedTextSettings = function(settings){
        var min, max;
        var vetNumber = (num, defaultNum) => {
            num = num !== null && num !== null && !isNaN(num) ? num : defaultNum;
            return num < 0 ? defaultNum : num;
        }
        if(settings === undefined || settings === null){ return; }
        compact = settings.compact === true ? true : settings.compact === false ? false :
compact;
        max = vetNumber(settings.maxSpaceSize, maxSpaceSize);
        min = vetNumber(settings.minSpaceSize, minSpaceSize);
```

```
        lineSpacing = vetNumber(settings.lineSpacing, lineSpacing);
        if(min > max){ return; }
        minSpaceSize = min;
        maxSpaceSize = max;
    }
    var getFontSize = function(font){  // get the font size.
        var numFind = /[0-9]+/;
        var number = numFind.exec(font)[0];
        if(isNaN(number)){
            throw new ReferenceError("justifiedPar Cant find font size");
        }
        return Number(number);
    }
    function justifiedPar(ctx, text, x, y, width, settings, stroke){
        var spaceWidth, minS, maxS, words, count, lines, lineWidth, lastLineWidth, lastSize,
i, renderer, fontSize, adjSpace, spaces, word, lineWords, lineFound;
        spaceWidth = ctx.measureText(" ").width;
        minS = spaceWidth * minSpaceSize;
        maxS = spaceWidth * maxSpaceSize;
        words = text.split(" ").map(word => {  // measure all words.
            var w = ctx.measureText(word).width;
            return {
                width : w,
                word : word,
            };
        });
        // count = num words, spaces = number spaces, spaceWidth normal space size
        // adjSpace new space size >= min size. useSize Resulting space size used to render
        count = 0;
        lines = [];
        // create lines by shifting words from the words array until the spacing is optimal.
If compact
        // true then will true and fit as many words as possible. Else it will try and get the
spacing as
        // close as possible to the normal spacing
        while(words.length > 0){
            lastLineWidth = 0;
            lastSize = -1;
            lineFound = false;
            // each line must have at least one word.
            word = words.shift();
            lineWidth = word.width;
            lineWords = [word.word];
            count = 0;
            while(lineWidth < width && words.length > 0){ // Add words to line
                word = words.shift();
                lineWidth += word.width;
                lineWords.push(word.word);
                count += 1;
                spaces = count - 1;
                adjSpace =  (width - lineWidth) / spaces;
                if(minS > adjSpace){  // if spacing less than min remove last word and finish
line
                    lineFound = true;
                    words.unshift(word);
                    lineWords.pop();
                }else{
                    if(!compact){ // if compact mode
                        if(adjSpace < spaceWidth){ // if less than normal space width
                            if(lastSize === -1){
                                lastSize = adjSpace;
```

```
                            }
                            // check if with last word on if its closer to space width
                            if(Math.abs(spaceWidth - adjSpace) < Math.abs(spaceWidth -
lastSize)){
                                lineFound = true; // yes keep it
                            }else{
                                words.unshift(word);  // no better fit if last word removes
                                lineWords.pop();
                                lineFound = true;
                            }
                        }
                    }
                }
                lastSize = adjSpace; // remember spacing
            }
            lines.push(lineWords.join(" ")); // and the line
        }
        // lines have been worked out get font size, render, and render all the lines. last
        // line may need to be rendered as normal so it is outside the loop.
        fontSize = getFontSize(ctx.font);
        renderer = stroke === true ? ctx.strokeJustifyText.bind(ctx) :
ctx.fillJustifyText.bind(ctx);
        for(i = 0; i < lines.length - 1; i ++){
            renderer(lines[i], x, y, width, settings);
            y += lineSpacing * fontSize;
        }
        if(lines.length > 0){ // last line if left or start aligned for no justify
            if(ctx.textAlign === "left" || ctx.textAlign === "start"){
                renderer(lines[lines.length - 1], x, y, width, noJustifySetting);
                ctx.measureJustifiedText("", width, settings);
            }else{
                renderer(lines[lines.length - 1], x, y, width);
            }
        }
        // return details about the paragraph.
        y += lineSpacing * fontSize;
        return {
            nextLine : y,
            fontSize : fontSize,
            lineHeight : lineSpacing * fontSize,
        };
    }
    // define fill
    var fillParagraphText = function(text, x, y, width, settings){
        justifiedTextSettings(settings);
        settings = {
            minSpaceSize : minSpaceSize,
            maxSpaceSize : maxSpaceSize,
        };
        return justifiedPar(this, text, x, y, width, settings);
    }
    // define stroke
    var strokeParagraphText = function(text, x, y, width, settings){
        justifiedTextSettings(settings);
        settings = {
            minSpaceSize : minSpaceSize,
            maxSpaceSize : maxSpaceSize,
        };
        return justifiedPar(this, text, x, y, width, settings,true);
    }
    CanvasRenderingContext2D.prototype.fillParaText = fillParagraphText;
```

```
    CanvasRenderingContext2D.prototype.strokeParaText = strokeParagraphText;
})();
```

> **NOTA** questo estende il prototipo `CanvasRenderingContext2D` . Se non si desidera che ciò accada, utilizzare il **testo Giustificato di** esempio per capire come modificare questo esempio per far parte dello spazio dei nomi globale.
>
> **NOTA** `CanvasRenderingContext2D.prototype.fillJustifyText` un ReferenceError se questo esempio non riesce a trovare la funzione `CanvasRenderingContext2D.prototype.fillJustifyText`

Come usare

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

Vedi il **testo giustificato** per i dettagli sugli argomenti. Gli argomenti tra `[` e `]` sono opzionali.

L'argomento `settings` ha due proprietà aggiuntive.

- **compatto:** predefinito `true` . Se true cerca di comprimere più parole possibili per riga. Se falso, cerca di far spaziare la parola il più vicino possibile alla normale spaziatura.
- **lineSpacing** Predefinito `1.5` . Spazio per linea predefinito `1.5` la distanza da on line a next in termini di dimensione del font

Le proprietà mancanti dall'oggetto delle impostazioni imposteranno automaticamente i loro valori predefiniti o gli ultimi valori validi. Le proprietà saranno modificate solo se i nuovi valori sono validi. Per valori validi `compact` sono solo booleans `true` o `false` I valori di Truthy non sono considerati validi.

Restituisci oggetto

Le due funzioni restituiscono un oggetto contenente informazioni per aiutarti a posizionare il paragrafo successivo. L'oggetto contiene le seguenti proprietà.

- **nextLine** Posizione della riga successiva dopo i pixel del paragrafo.
- **fontSize** Dimensione del carattere. (si prega di notare solo utilizzare i caratteri definiti in pixel, ad es. `14px arial`)
- **lineHeight** Distanza in pixel da una linea a quella successiva

Questo esempio utilizza un semplice algoritmo che lavora una riga alla volta per trovare la soluzione migliore per un paragrafo. Ciò non significa che sia la soluzione migliore (piuttosto il migliore dell'algoritmo). Potresti voler migliorare l'algoritmo creando un algoritmo a più passate sulle linee generate. Spostare le parole dalla fine di una riga all'inizio della successiva o dall'inizio alla fine. Il miglior aspetto si ottiene quando la spaziatura su tutto il paragrafo ha la variazione più piccola ed è la più vicina alla normale spaziatura del testo.

Poiché questo esempio dipende dall'esempio di **testo giustificato,** il codice è molto simile. Potresti voler spostare i due in una funzione. Sostituisci la funzione `justifiedTextSettings` nell'altro esempio con quello usato in questo esempio. Quindi copiare tutto il resto del codice da questo esempio nel corpo della funzione anonima dell'esempio di **testo giustificato** . Non sarà più necessario verificare le dipendenze trovate in `// Code point A` Può essere rimosso.

Esempio di utilizzo

```
ctx.font = "25px arial";
ctx.textAlign = "center"

var left = 10;
var center = canvas.width / 2;
var width = canvas.width-left*2;
var y = 20;
var size = 16;
var i = 0;
ctx.fillText("Justified paragraph examples.",center,y);
y+= 30;
ctx.font = "14px arial";
ctx.textAlign = "left"
// set para settings
var setting = {
    maxSpaceSize : 6,
    minSpaceSize : 0.5,
    lineSpacing : 1.2,
    compact : true,
}
// Show the left and right bounds.
ctx.strokeStyle = "red"
ctx.beginPath();
ctx.moveTo(left,y - size * 2);
ctx.lineTo(left, y + size * 15);
ctx.moveTo(canvas.width - left,y - size * 2);
ctx.lineTo(canvas.width - left, y + size * 15);
ctx.stroke();
ctx.textAlign = "left";
ctx.fillStyle = "black";

// Draw paragraph
var line = ctx.fillParaText(para, left, y, width, setting);  // settings is remembered

// Next paragraph
y = line.nextLine + line.lineHeight;
setting.compact = false;
ctx.fillParaText(para, left, y, width, setting);
```

> **Nota:** per il testo allineato a `left` o l' `start` dell'ultima riga di quel paragrafo avrà sempre una spaziatura normale. Per tutti gli altri allineamenti l'ultima riga viene trattata come tutte le altre.
>
> **Nota:** puoi inserire l'inizio del paragrafo con spazi. Anche se questo potrebbe non essere coerente da paragrafo a paragrafo. È sempre una buona cosa imparare cosa sta facendo una funzione e modificarla. Un esercizio sarebbe aggiungere

un'impostazione alle impostazioni che rientrano nella prima riga di una quantità fissa. Indica che il ciclo while dovrà rendere temporaneamente la prima parola più grande (+ indent) `words[0].width += ?` e poi quando le linee di rendering rientrano nella prima riga.

Leggi Testo online: https://riptutorial.com/it/html5-canvas/topic/5235/testo

Capitolo 16: Tipi di media e tela

Osservazioni

Questo argomento tratta dei vari tipi di media e di come possono essere utilizzati con il canvas nell'interfaccia 2D.

I tipi di media hanno categorie generiche e di formato specifico

Tipi di media

- animazioni
- video
- immagini
- Immagini HD
- Immagine vettoriale
- Immagini animate

Formati multimediali

- Jpg / Jpeg
- png
- gif
- SVG
- M-JPEG
- webm
- WebP

---

immagini

Esistono numerosi formati di immagini supportati dai browser, sebbene nessun browser li supporti tutti. Se si dispone di formati di immagine particolari che si desidera utilizzare Browser Wiki e formati di immagine supportati fornisce una buona panoramica.

Il miglior supporto è per i 3 formati principali, "jpeg", "png" e "gif" con tutti i principali browser che forniscono supporto.

JPEG

Le immagini JPEG sono più adatte a foto e immagini simili a foto. Non si prestano bene a grafici, diagrammi e testi. Le immagini JPEG non supportano la trasparenza.

Canvas può emettere immagini JPEG tramite `canvas.toDataURL` e `canvas.toBlob` e fornisce un'impostazione di qualità. Dato che JPEG non supporta la trasparenza, tutti i pixel trasparenti verranno mescolati con il nero per l'output finale JPG. L'immagine risultante non sarà una copia perfetta della tela.

JPEG su wikipedia

PNG

Le immagini PNG sono immagini della massima qualità e possono anche includere un canale alfa per i pixel trasparenti. I dati dell'immagine sono compressi ma non producono artefatti come le immagini JPG.

A causa della compressione lossless e del supporto del canale alfa, i PNG vengono utilizzati per i giochi, le immagini dei componenti ui, i grafici, i diagrammi, il testo. Quando li si utilizza per foto e foto come immagini, le dimensioni del file possono essere molto più grandi di quelle di JPEG. .

Il formato PNG fornisce anche il supporto dell'animazione anche se il supporto del browser è limitato e l'accesso all'animazione per l'utilizzo sulla tela può essere effettuato solo tramite API e librerie Javascript

Il canvas può essere utilizzato per codificare le immagini PNG tramite `canvas.toDataURL` e `canvas.toBlob` anche se il formato di output è limitato a 32Bit RGBA compressi. Il PNG fornirà una copia pixel perfetta della tela.

PNG su wikipedia

GIF

Le GIF vengono utilizzate per brevi animazioni, ma possono anche essere utilizzate per fornire grafici, diagrammi e immagini di testo di alta qualità. Le GIF hanno un supporto dei colori molto limitato con un massimo di 256 colori per fotogramma. Con le immagini gif di elaborazione delle immagini della mannaia si ottengono risultati sorprendentemente buoni, specialmente quando sono animati. Le GIF forniscono trasparenza anche se questo è limitato a on o off

AS con PNG, le animazioni GIF non sono direttamente accessibili per l'uso sul canvas e per accedere è necessaria un'API o una libreria Javascript. GIF non può essere salvato tramite la tela e richiederà e API o libreria per farlo.

GIF su wikipedia

Examples

Caricamento e visualizzazione di un'immagine

Per caricare un'immagine e posizionarla sulla tela

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

Creare un'immagine

Esistono diversi modi per creare un'immagine

- `new Image()`
- `document.createElement("img")`
- `<img src = 'imageUrl' id='myImage'>` Come parte del corpo HTML e recuperato con `document.getElementById('myImage')`

L'immagine è un `HTMLImageElement`

Proprietà Image.src

L'immagine `src` può essere qualsiasi URL di immagine valido o dataURL codificato. Vedi le osservazioni di questo argomento per ulteriori informazioni su formati e supporto di immagine.

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

* Il dataURL è un'immagine GIF 1 x 1 pixel contenente il nero

Note sul caricamento e errori

L'immagine inizierà il caricamento quando viene impostata la sua proprietà src. Il caricamento è sincronizzato ma l'evento `onload` non verrà chiamato fino a quando la funzione o il codice non saranno usciti / restituiti.

Se si ottiene un'immagine dalla pagina (ad esempio, `document.getElementById("myImage")`) e il relativo `src` è impostato potrebbe essere caricato o meno. Puoi controllare lo stato dell'immagine con `HTMLImageElement.complete` che sarà `true` se completo. Questo non significa che l'immagine sia caricata, significa che lo ha anche

- caricato
- c'era un errore
- la proprietà src non è stata impostata ed è uguale alla stringa vuota `""`

Se l'immagine proviene da una fonte inaffidabile e potrebbe non essere accessibile per una serie di motivi, genererà un evento di errore. Quando ciò accade l'immagine sarà in uno stato rotto. Se tenti di disegnarlo sulla tela, verrà generato il seguente errore

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

Fornendo l'evento `image.onerror = myImgErrorHandler` puoi prendere le misure appropriate per evitare errori.

Disegnare un'immagine svg

Per disegnare un'immagine SVG vettoriale, l'operazione non è diversa da un'immagine raster: Per prima cosa è necessario caricare l'immagine SVG in un elemento HTMLImage, quindi utilizzare il metodo `drawImage()` .

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

Le immagini SVG hanno alcuni vantaggi rispetto a quelle raster, dal momento che non perderai la qualità, a prescindere dalla scala su cui la disegnerai sulla tela. Ma attenzione, potrebbe anche essere un po 'più lento del disegno di un'immagine raster.

Tuttavia, le immagini SVG hanno più restrizioni rispetto alle immagini raster.

- **Per motivi di sicurezza, nessun contenuto esterno può essere caricato da un'immagine SVG a cui si fa riferimento in un HTMLImageElement (`<img>`)**
  Nessun foglio di stile esterno, nessuna immagine esterna referenziata in elementi SVGImage (`<image/>`), nessun filtro esterno o elemento collegato dall'attributo `xlink:href` (`<use xlink:href="anImage.SVG#anElement"/>`) o il funcIRI (`url()`) metodo di attributo ecc.
  Inoltre, i fogli di stile aggiunti nel documento principale non avranno alcun effetto sul documento SVG una volta fatto riferimento in un elemento HTMLImmagine.
  Infine, nessuno script verrà eseguito all'interno dell'immagine SVG.
  Soluzione: è necessario aggiungere tutti gli elementi esterni all'interno dello SVG stesso prima di fare riferimento all'elemento HTMLImage. (per immagini o caratteri, è necessario aggiungere una versione dataURI delle risorse esterne).

- **L'elemento radice (`<svg>`) deve avere gli attributi width e height impostati su un valore assoluto.**
  Se si dovesse utilizzare la lunghezza relativa (ad esempio `%`), il browser non sarà in grado di sapere a cosa è relativo. Alcuni browser (Blink) tenteranno di indovinare, ma la maggior parte semplicemente ignorerà la tua immagine e non disegnerà nulla, senza un avvertimento.

- **Alcuni browser macchieranno la tela quando un'immagine SVG è stata disegnata su di essa.**
  In particolare, Internet-Explorer <Edge in ogni caso e Safari 9 quando `<foreignObject>` è presente nell'immagine SVG.

Caricamento di base e riproduzione di un video sulla tela.

La tela può essere utilizzata per visualizzare video da una varietà di fonti. Questo esempio mostra come caricare un video come risorsa file, visualizzarlo e aggiungere un semplice clic sullo schermo play / pause toggle.

Questa domanda con risposta automatica StackOverflow Come faccio a visualizzare un video utilizzando il tag canvas HTML5 mostra il seguente codice di esempio in azione.

Solo un'immagine

Un video è solo un'immagine per quanto riguarda la tela. Puoi disegnarlo come qualsiasi immagine. La differenza è che il video può suonare e ha un suono.

Ottieni canvas e impostazioni di base

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

Creazione e caricamento del video

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

A differenza degli elementi delle immagini, i video non devono essere completamente caricati per essere visualizzati sulla tela. I video forniscono anche una serie di eventi extra che possono essere utilizzati per monitorare lo stato del video.

In questo caso desideriamo sapere quando il video è pronto per la riproduzione. `oncanplay` significa che è stato caricato un numero sufficiente di video per riprodurne alcuni, ma potrebbe non essere sufficiente per suonare fino alla fine.

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

In alternativa puoi usare `oncanplaythrough` che si `oncanplaythrough` quando una parte del video è stata caricata in modo tale da poter essere riprodotta fino alla fine.

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

Utilizzare solo uno degli eventi canPlay non entrambi.

L'evento can play (equivalente al caricamento di immagini)

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

Visualizzazione

Il video non verrà riprodotto sulla tela. Devi disegnarlo per ogni nuovo fotogramma. Poiché è difficile conoscere il frame rate esatto e quando si verificano, il miglior approc è quello di visualizzare il video come se fosse a 60fps. Se la frequenza dei fotogrammi è inferiore, w solo renderizza due volte lo stesso fotogramma. Se la frequenza dei fotogrammi è superiore, non c'è nulla che possa essere visto per vedere i fotogrammi aggiuntivi, quindi li ignoriamo.

L'elemento video è solo un elemento dell'immagine e può essere disegnato come qualsiasi immagine, è possibile ridimensionare, ruotare, spostare il video, specularlo, sfumarlo, ritagliarlo e visualizzare solo le parti, disegnarlo due volte la seconda volta con una modalità composita globale aggiungere FX come schiarire, schermo, ecc.

```
function updateCanvas(){
    ctx.clearRect(0,0,canvas.width,canvas.height); // Though not always needed
                                                   // you may get bad pixels from
                                                   // previous videos so clear to be
                                                   // safe
    // only draw if loaded and ready
    if(videoContainer !== undefined && videoContainer.ready){
        // find the top left of the video on the canvas
        var scale = videoContainer.scale;
        var vidH = videoContainer.video.videoHeight;
        var vidW = videoContainer.video.videoWidth;
        var top = canvas.height / 2 - (vidH /2 ) * scale;
        var left = canvas.width / 2 - (vidW /2 ) * scale;
        // now just draw the video the correct size
        ctx.drawImage(videoContainer.video, left, top, vidW * scale, vidH * scale);
        if(videoContainer.video.paused){ // if not playing show the paused screen
            drawPayIcon();
        }
    }
    // all done for display
    // request the next frame in 1/60th of a second
    requestAnimationFrame(updateCanvas);
}
```

Controllo di pausa riproduzione di base

Ora abbiamo il video caricato e visualizzato tutto ciò di cui abbiamo bisogno è il controllo di gioco. Lo faremo come un clic per attivare la riproduzione sullo schermo. Quando il video è in riproduzione e l'utente fa clic sul video viene messo in pausa. Quando viene messo in pausa, il clic riprende la riproduzione. Aggiungeremo una funzione per scurire il video e disegnare un'icona di riproduzione (triangolo)

```
function drawPayIcon(){
     ctx.fillStyle = "black";  // darken display
     ctx.globalAlpha = 0.5;
     ctx.fillRect(0,0,canvas.width,canvas.height);
     ctx.fillStyle = "#DDD"; // colour of play icon
     ctx.globalAlpha = 0.75; // partly transparent
     ctx.beginPath(); // create the path for the icon
     var size = (canvas.height / 2) * 0.5;  // the size of the icon
```

```
     ctx.moveTo(canvas.width/2 + size/2, canvas.height / 2); // start at the pointy end
     ctx.lineTo(canvas.width/2 - size/2, canvas.height / 2 + size);
     ctx.lineTo(canvas.width/2 - size/2, canvas.height / 2 - size);
     ctx.closePath();
     ctx.fill();
     ctx.globalAlpha = 1; // restore alpha
}
```

Ora l'evento di pausa di riproduzione

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

Sommario

Riproduzione di un video è molto semplice con la tela, anche l'aggiunta di effetti in tempo reale è facile. Ci sono tuttavia alcune limitazioni sui formati, come puoi giocare e cercare. MDN HTMLMediaElement è il posto dove ottenere il riferimento completo all'oggetto video.

Una volta che l'immagine è stata disegnata sulla tela, puoi usare `ctx.getImageData` per accedere ai pixel che contiene. Oppure puoi usare `canvas.toDataURL` per scattare una foto e scaricarla. (Solo se il video proviene da una fonte attendibile e non macchia la tela).

Nota se il video ha un suono, quindi riprodurrà anche il suono.

Buon video.

Cattura tela e Salva come video WebM

Creazione di un video WebM da cornici su tela e riproduzione in tela, caricamento o download.

Esempio di acquisizione e riproduzione della tela

```
name = "CanvasCapture"; // Placed into the Mux and Write Application Name fields of the WebM
header
quality = 0.7; // good quality 1 Best < 0.7 ok to poor
fps = 30; // I have tried all sorts of frame rates and all seem to work
          // Do some test to workout what your machine can handle as there
          // is a lot of variation between machines.
var video = new Groover.Video(fps,quality,name)
function capture(){
    if(video.timecode < 5000){ // 5 seconds
```

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

Piuttosto che fare uno sforzo enorme solo per essere respinto, questo è un inserto veloce per vedere se accettabile. Darà tutti i dettagli se accettato. Includere anche opzioni di acquisizione aggiuntive per una migliore velocità di acquisizione HD (rimosso da questa versione, può catturare HD 1080 a 50 fps su macchine buone).

Questo è stato ispirato da Wammy, ma è una completa riscrittura con codifica mentre si va metodologia, riducendo notevolmente la memoria necessaria durante l'acquisizione. Può acquisire più di 30 secondi di dati migliori, gestendo algoritmi.

> **I** frame di **nota** sono codificati in immagini WebP. Solo Chrome supporta la codifica del canvas webP. Per gli altri browser (Firefox e Edge) sarà necessario utilizzare un codificatore WebP di terze parti come Libwebp Javascript Codifica delle immagini WebP tramite Javascript è lento. (includerà l'aggiunta del supporto di immagini Webp raw se accettato).

Il codificatore webM ispirato a Whammy: WebM Javascript in tempo reale

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

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

    const stream = {
        num : function(num){ // writes int
            var parts = [];
            while(num > 0){ parts.push(num & 0xff); num = num >> 8; }
            return new Uint8Array(parts.reverse());
        },
        str : function(str){ // writes string
            var i, len, arr;
            len = str.length;
            arr = new Uint8Array(len);
            for(i = 0; i < len; i++){arr[i] = str.charCodeAt(i);}
            return arr;
        },
        compInt : function(num){ // could not find full details so bit of a guess
            if(num < 128){       // number is prefixed with a bit (1000 is on byte 0100 two,
0010 three and so on)
                num += 0x80;
                return new Uint8Array([num]);
            }else
            if(num < 0x4000){
                num += 0x4000;
                return new Uint8Array([num>>8, num])
            }else
            if(num < 0x200000){
                num += 0x200000;
                return new Uint8Array([num>>16, num>>8, num])
            }else
            if(num < 0x10000000){
                num += 0x10000000;
                return new Uint8Array([num>>24, num>>16, num>>8, num])
            }
        }
    }
    const ids = { // header names and values
        videoData          : 0x1a45dfa3,
        Version            : 0x4286,
        ReadVersion        : 0x42f7,
        MaxIDLength        : 0x42f2,
        MaxSizeLength      : 0x42f3,
        DocType            : 0x4282,
        DocTypeVersion     : 0x4287,
        DocTypeReadVersion : 0x4285,
        Segment            : 0x18538067,
        Info               : 0x1549a966,
        TimecodeScale      : 0x2ad7b1,
        MuxingApp          : 0x4d80,
        WritingApp         : 0x5741,
        Duration           : 0x4489,
        Tracks             : 0x1654ae6b,
        TrackEntry         : 0xae,
        TrackNumber        : 0xd7,
```

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

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

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

Leggi Tipi di media e tela online: https://riptutorial.com/it/html5-canvas/topic/3689/tipi-di-media-e-tela

Capitolo 17: Trascinando le forme del percorso e le immagini su tela

Examples

Come forme e immagini DAVVERO (!) "Muovere" sulla tela

Un problema: la tela ricorda solo i pixel, non le forme o le immagini

Questa è l'immagine di un pallone da spiaggia circolare e, naturalmente, non è possibile trascinare la palla intorno all'immagine.

Potrebbe sorprendervi che, proprio come un'immagine, se disegnate un cerchio su una tela non potete trascinare quel cerchio attorno alla tela. Questo perché la tela non ricorderà dove ha disegnato il cerchio.

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

Cosa la tela NON sa ...

- ... dove hai disegnato il cerchio (non sa x, y = [20,30]).
- ... la dimensione del cerchio (non conosce raggio = 15).
- ... il colore del cerchio. (non sa che il cerchio è blu).

Cosa sa la tela ...

Canvas conosce il colore di ogni pixel sulla sua superficie di disegno.

La tela può dirti che in x, y == [20,30] c'è un pixel blu, ma non sa se questo pixel blu fa parte di un cerchio.

Cosa significa...

Ciò significa che tutto ciò che è disegnato sulla tela è permanente: immobile e immutabile.

- La tela non può spostare il cerchio o ridimensionare il cerchio.
- La tela non può ricolorare il cerchio o cancellare il cerchio.
- La tela non può dire se il mouse si trova sopra il cerchio.
- La tela non può dire se il cerchio è in collisione con un altro cerchio.
- La tela non può lasciare che un utente trascini il cerchio attorno alla tela.

Ma Canvas può dare l'ILLUSIONE del movimento

La tela può dare l' **illusione del movimento** cancellando continuamente il cerchio e ridisegnandolo in una posizione diversa. Ridisegnando la tela molte volte al secondo, l'occhio è ingannato nel vedere il cerchio muoversi attraverso la tela.

- Cancellare la tela
- Aggiorna la posizione del cerchio
- Ridisegna il cerchio nella sua nuova posizione
- Ripeti, ripeti, ripeti ...

Questo codice dà l' **illusione del movimento** ridisegnando continuamente un cerchio in nuove posizioni.

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

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

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

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

Trascinando cerchi e rettangoli attorno alla tela

Cos'è una "forma"?

In genere si salvano le forme creando un oggetto "forma" JavaScript che rappresenta ciascuna forma.

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

Certo, non stai davvero risparmiando forme. Invece, stai salvando la definizione di come disegnare le forme.

Quindi metti tutti gli oggetti forma in una matrice per un facile riferimento.

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

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

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

Uso degli eventi del mouse per eseguire il trascinamento

Trascinando una forma o un'immagine è necessario rispondere a questi eventi del mouse:

In fuga:

Verifica se qualsiasi forma è sotto il mouse. Se una forma è sotto il mouse, l'utente intende trascinare quella forma. Quindi mantieni un riferimento a quella forma e imposta un flag `isDragging` vero / falso che indica che è in corso un trascinamento.

Su mousemove:

Calcola la distanza trascinata dal mouse dall'ultimo evento `mousemove` e modifica la posizione della forma trascinata in base a tale distanza. Per cambiare la posizione della forma, si modificano le proprietà di posizione `x,y` nell'oggetto di quella forma.

Su mouseup o mouseout:

L'utente intende interrompere l'operazione di trascinamento, quindi deselezionare il flag "isDragging". Il trascinamento è completato.

Demo: trascinamento di cerchi e rettangoli

sulla tela

Questa demo trascina cerchi e rettangoli sulla tela rispondendo agli eventi del mouse e dando l'illusione di movimento cancellando e ridisegnando.

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

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

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

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

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

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

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

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

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

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

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

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

function handleMouseMove(e){
```

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

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

Trascinando forme irregolari attorno alla tela

La maggior parte dei disegni su tela è rettangolare (rettangoli, immagini, blocchi di testo) o circolare (cerchi).

Cerchi e rettangoli hanno test matematici per verificare se il mouse è all'interno di essi. Ciò rende il test di cerchi e rettangoli facile, veloce ed efficiente. Puoi "colpire" centinaia di cerchi o rettangoli in una frazione di secondo.

Puoi anche trascinare forme irregolari. Ma le forme irregolari non hanno un rapido hit matematico. Fortunatamente, le forme irregolari hanno un hit-test integrato per determinare se un punto (mouse) si trova all'interno della forma: `context.isPointInPath` . Mentre `isPointInPath` funziona bene, non è altrettanto efficiente quanto i test di successo puramente matematici: spesso è fino a 10 volte più lento rispetto ai purissimi test di matematica.

Un requisito quando si utilizza `isPointInPath` è che è necessario "ridefinire" il percorso da testare immediatamente prima di chiamare `isPointInPath` . "Ridefinisci" significa che è necessario emettere i comandi di `isPointInPath` del percorso (come sopra), ma non è necessario tracciare () o riempire () il percorso prima di `isPointInPath` con `isPointInPath` . In questo modo puoi testare i percorsi disegnati in precedenza senza dover sovrascrivere (traccia / riempi) quei percorsi precedenti sulla tela stessa.

La forma irregolare non ha bisogno di essere comune come il triangolo di tutti i giorni. Puoi anche eseguire il test dei percorsi Wildly irregolari.

Questo esempio annotato mostra come trascinare forme del percorso irregolari e cerchi e rettangoli:

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

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

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

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

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

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

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

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

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

function handleMouseUp(e){
    // return if we're not dragging
    if(!isDragging){return;}
```

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

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

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

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

```
        }
    }
}

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

Trascinando le immagini attorno alla tela

Vedi questo esempio per una spiegazione generale del trascinamento di forme attorno alla tela.

Questo esempio annotato mostra come trascinare le immagini attorno alla tela

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

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

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

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

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

// load the image
var card=new Image();
card.onload=function(){
    // define one image and save it in the shapes[] array
    shapes.push( {x:30, y:10, width:127, height:150, image:card} );
    // draw the shapes on the canvas
    drawAll();
```

```
    // listen for mouse events
    canvas.onmousedown=handleMouseDown;
    canvas.onmousemove=handleMouseMove;
    canvas.onmouseup=handleMouseUp;
    canvas.onmouseout=handleMouseOut;
};
// put your image src here!
card.src='https://dl.dropboxusercontent.com/u/139992952/stackoverflow/card.png';


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

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

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

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

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

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

Leggi Trascinando le forme del percorso e le immagini su tela online: https://riptutorial.com/it/html5-canvas/topic/5318/trascinando-le-forme-del-percorso-e-le-immagini-su-tela

Capitolo 18: trasformazioni

Examples

Disegnare rapidamente molte immagini tradotte, ridimensionate e ruotate

Ci sono molte situazioni in cui si desidera disegnare un'immagine che viene ruotata, ridimensionata e tradotta. La rotazione dovrebbe avvenire attorno al centro dell'immagine. Questo è il modo più rapido per farlo sulla tela 2D. Queste funzioni sono molto adatte ai giochi 2D in cui l'aspettativa è di rendere alcune centinaia persino fino a 1000+ immagini ogni 60esimo di secondo. (dipende dall'hardware)

```
// assumes that the canvas context is in ctx and in scope
function drawImageRST(image, x, y, scale, rotation){
    ctx.setTransform(scale, 0, 0, scale, x, y); // set the scale and translation
    ctx.rotate(rotation);                       // add the rotation
    ctx.drawImage(image, -image.width / 2, -image.height / 2); // draw the image offset by
half its width and height
}
```

Una variante può anche includere il valore alfa che è utile per i sistemi di particelle.

```
function drawImageRST_Alpha(image, x, y, scale, rotation, alpha){
    ctx.setTransform(scale, 0, 0, scale, x, y); // set the scale and translation
    ctx.rotate(rotation);                       // add the rotation
    ctx.globalAlpha = alpha;
    ctx.drawImage(image, -image.width / 2, -image.height / 2); // draw the image offset by
half its width and height
}
```

È importante notare che entrambe le funzioni lasciano il contesto della tela in uno stato casuale. Anche se le funzioni non saranno influenzate, altri renderanno il mio essere. Una volta terminato il rendering delle immagini, potrebbe essere necessario ripristinare la trasformazione predefinita

```
ctx.setTransform(1, 0, 0, 1, 0, 0); // set the context transform back to the default
```

Se si utilizza la versione alpha (secondo esempio) e quindi la versione standard, è necessario assicurarsi che lo stato alfa globale venga ripristinato

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

Un esempio di utilizzo delle funzioni di cui sopra per rendere alcune particelle e alcune immagini

```
// assume particles to contain an array of particles
for(var i = 0; i < particles.length; i++){
    var p = particles[i];
    drawImageRST_Alpha(p.image, p.x, p.y, p.scale, p.rot, p.alpha);
    // no need to rest the alpha in the loop
}
```

```
// you need to reset the alpha as it can be any value
ctx.globalAlpha = 1;

drawImageRST(myImage, 100, 100, 1, 0.5);  // draw an image at 100,100
// no need to reset the transform
drawImageRST(myImage, 200, 200, 1, -0.5); // draw an image at 200,200
ctx.setTransform(1,0,0,1,0,0);            // reset the transform
```

Ruota un'immagine o un percorso attorno al suo punto centrale

I passaggi da 1 a 5 qui sotto consentono a qualsiasi immagine o forma del percorso di essere spostati ovunque nell'area di disegno e ruotati su qualsiasi angolo senza modificare le coordinate del punto originale dell'immagine / percorso.

1. Sposta l'origine della tela [0,0] al punto centrale della forma

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

2. Ruota la tela dell'angolo desiderato (in radianti)

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

3. Sposta l'origine della tela nell'angolo in alto a sinistra

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

4. Disegna l'immagine o la forma del percorso usando le sue coordinate originali.

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

5. Pulisci sempre! Impostare lo stato di trasformazione su dove era prima del # 1

- *Passaggio n. 5, opzione n. 1:* annullare ogni trasformazione nell'ordine inverso

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

- *Passaggio n. 5, opzione n. 2:* se la tela era in uno stato non trasformato (impostazione predefinita) prima di iniziare il passaggio n. 1, è possibile annullare gli effetti dei passaggi n. 1-3 ripristinando tutte le trasformazioni al loro stato predefinito

```
// set transformation to the default state (==no transformation applied)
context.setTransform(1,0,0,1,0,0)
```

Esempio di demo del codice:

```
// canvas references & canvas styling
var canvas=document.createElement("canvas");
canvas.style.border='1px solid red';
document.body.appendChild(canvas);
canvas.width=378;
canvas.height=256;
var ctx=canvas.getContext("2d");
ctx.fillStyle='green';
ctx.globalAlpha=0.35;

// define a rectangle to rotate
var rect={ x:100, y:100, width:175, height:50 };

// draw the rectangle unrotated
ctx.fillRect( rect.x, rect.y, rect.width, rect.height );

// draw the rectangle rotated by 45 degrees (==PI/4 radians)
ctx.translate( rect.x+rect.width/2, rect.y+rect.height/2 );
ctx.rotate( Math.PI/4 );
ctx.translate( -rect.x-rect.width/2, -rect.y-rect.height/2 );
ctx.fillRect( rect.x, rect.y, rect.width, rect.height );
```

Introduzione alle trasformazioni

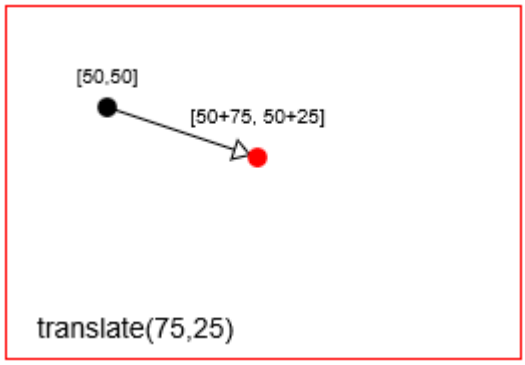
Le trasformazioni alterano la posizione di partenza di un dato punto spostando, ruotando e ridimensionando quel punto.

- **Traslazione:** sposta un punto di una `distanceX` e una `distanceX` `distanceY`
- **Rotazione:** ruota un punto di un `radian angle` attorno al suo punto di rotazione. Il punto di rotazione predefinito in Html Canvas è l'origine in alto a sinistra [x = 0, y = 0] della tela. Ma puoi riposizionare il punto di rotazione usando le traduzioni.
- **Ridimensionamento:** consente di ridimensionare la posizione di un punto mediante un `scalingFactorX` di scala e un `scalingFactorY` di scala dal punto di ridimensionamento. Il punto di ridimensionamento predefinito in Html Canvas è l'origine in alto a sinistra [x = 0, y = 0] della tela. Ma puoi riposizionare il punto di ridimensionamento usando le traduzioni.

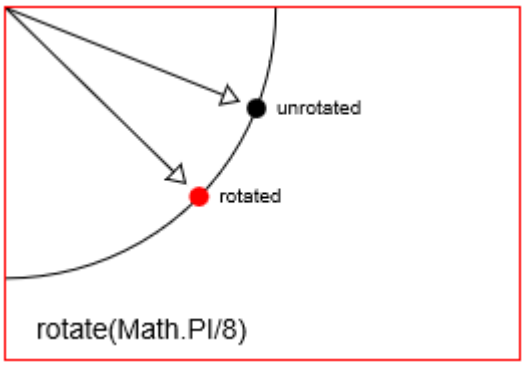
È anche possibile eseguire trasformazioni meno comuni, come la cesoiatura (inclinazione),

impostando direttamente la matrice di trasformazione della tela utilizzando `context.transform` .

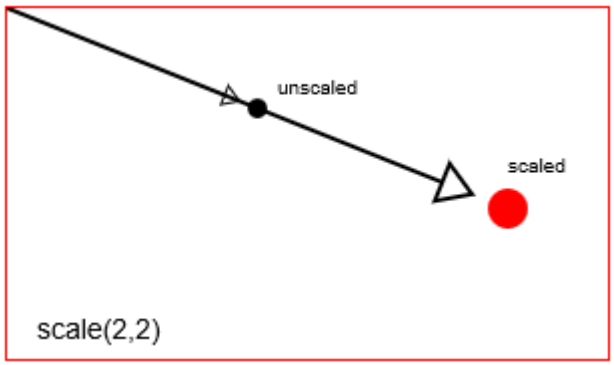
Traduci (== sposta) un punto con `context.translate(75,25)`



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



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



Canvas realizza effettivamente le trasformazioni alterando l'intero sistema di coordinate della tela.

- `context.translate` sposta l'origine del canvas [0,0] dall'angolo in alto a sinistra in una nuova posizione.
- `context.rotate` ruoterà l'intero sistema di coordinate del canvas attorno all'origine.
- `context.scale` ridimensiona l'intero sistema di coordinate del canvas attorno all'origine. Pensa a ciò aumentando la dimensione di ogni x, y sulla tela: `every x*=scaleX` e `every y*=scaleY` .

Le trasformazioni della tela sono persistenti. Tutti i nuovi disegni continueranno a essere trasformati fino a quando non si ripristinerà la trasformazione della tela al suo stato predefinito (== totalmente non trasformato). Puoi ripristinare i valori predefiniti con:

```
// reset context transformations to the default (untransformed) state
context.setTransform(1,0,0,1,0,0);
```

Una matrice di trasformazione per tracciare le forme tradotte, ruotate e ridimensionate

Canvas ti permette di `context.translate` , `context.rotate` e `context.scale` per disegnare la tua forma nella posizione e dimensione richiesta.

La tela stessa usa una matrice di trasformazione per tracciare efficientemente le trasformazioni.

- Puoi cambiare la matrice di Canvas con `context.transform`
- Puoi cambiare la matrice di Canvas con singoli comandi di `translate, rotate & scale` scalatura
- Puoi sovrascrivere completamente la matrice di Canvas con `context.setTransform` ,
- *Ma non puoi leggere la matrice di trasformazione interna di Canvas: è di sola scrittura.*

Perché usare una matrice di trasformazione?

Una matrice di trasformazione consente di aggregare molte singole traduzioni, rotazioni e ridimensionamenti in un'unica matrice facilmente riapplicabile.

Durante animazioni complesse è possibile applicare dozzine (o centinaia) di trasformazioni a una forma. Utilizzando una matrice di trasformazione è possibile (quasi) riapplicare istantaneamente quelle dozzine di trasformazioni con una singola riga di codice.

Alcuni esempi utilizza:

- **Verifica se il mouse si trova all'interno di una forma che hai tradotto, ruotato e ridimensionato**

  Esiste un `context.isPointInPath` integrato che verifica se un punto (ad es. Il mouse) si trova all'interno di un percorso, ma questo test integrato è molto lento rispetto al test che utilizza una matrice.

  Testare in modo efficiente se il mouse si trova all'interno di una forma comporta prendere la posizione del mouse riportata dal browser e trasformarla nello stesso modo in cui la forma è stata trasformata. Quindi puoi applicare hit-test come se la forma non fosse stata trasformata.

- **Ridisegna una forma che è stata ampiamente tradotta, ruotata e ridimensionata.**

  Invece di riapplicare le singole trasformazioni con più `.translate, .rotate, .scale` è possibile applicare tutte le trasformazioni aggregate in una singola riga di codice.

- **Forme di prova di collisione che sono state tradotte, ruotate e ridimensionate**

  È possibile utilizzare la geometria e la trigonometria per calcolare i punti che costituiscono le

forme trasformate, ma è più rapido utilizzare una matrice di trasformazione per calcolare quei punti.

Una matrice di trasformazione "Classe"

Questo codice rispecchia i comandi di trasformazione nativi `context.translate` , `context.rotate` , `context.scale` . A differenza della matrice tela nativa, questa matrice è leggibile e riutilizzabile.

metodi:

- `translate` , `rotate` , `scale` i comandi di trasformazione del contesto e consenti di alimentare le trasformazioni nella matrice. La matrice contiene efficientemente le trasformazioni aggregate.
- `setContextTransform` prende un contesto e imposta la matrice del contesto uguale a questa matrice di trasformazione. Questo riapplica in modo efficiente tutte le trasformazioni memorizzate in questa matrice al contesto.
- `resetContextTransform` reimposta la trasformazione del contesto nel suo stato predefinito (== non trasformato).
- `getTransformedPoint` prende un punto di coordinate non trasformato e lo converte in un punto trasformato.
- `getScreenPoint` prende un punto di coordinate trasformato e lo converte in un punto non trasformato.
- `getMatrix` restituisce le trasformazioni aggregate sotto forma di matrice matrice.

Codice:

```
var TransformationMatrix=( function(){
    // private
    var self;
    var m=[1,0,0,1,0,0];
    var reset=function(){ var m=[1,0,0,1,0,0]; }
    var multiply=function(mat){
        var m0=m[0]*mat[0]+m[2]*mat[1];
        var m1=m[1]*mat[0]+m[3]*mat[1];
        var m2=m[0]*mat[2]+m[2]*mat[3];
        var m3=m[1]*mat[2]+m[3]*mat[3];
        var m4=m[0]*mat[4]+m[2]*mat[5]+m[4];
        var m5=m[1]*mat[4]+m[3]*mat[5]+m[5];
        m=[m0,m1,m2,m3,m4,m5];
    }
    var screenPoint=function(transformedX,transformedY){
        // invert
        var d =1/(m[0]*m[3]-m[1]*m[2]);
        im=[ m[3]*d, -m[1]*d, -m[2]*d, m[0]*d, d*(m[2]*m[5]-m[3]*m[4]), d*(m[1]*m[4]-
m[0]*m[5]) ];
        // point
        return({
```

```
            x:transformedX*im[0]+transformedY*im[2]+im[4],
            y:transformedX*im[1]+transformedY*im[3]+im[5]
        });
    }
    var transformedPoint=function(screenX,screenY){
        return({
            x:screenX*m[0] + screenY*m[2] + m[4],
            y:screenX*m[1] + screenY*m[3] + m[5]
        });
    }
    // public
    function TransformationMatrix(){
        self=this;
    }
    // shared methods
    TransformationMatrix.prototype.translate=function(x,y){
        var mat=[ 1, 0, 0, 1, x, y ];
        multiply(mat);
    };
    TransformationMatrix.prototype.rotate=function(rAngle){
        var c = Math.cos(rAngle);
        var s = Math.sin(rAngle);
        var mat=[ c, s, -s, c, 0, 0 ];
        multiply(mat);
    };
    TransformationMatrix.prototype.scale=function(x,y){
        var mat=[ x, 0, 0, y, 0, 0 ];
        multiply(mat);
    };
    TransformationMatrix.prototype.skew=function(radianX,radianY){
        var mat=[ 1, Math.tan(radianY), Math.tan(radianX), 1, 0, 0 ];
        multiply(mat);
    };
    TransformationMatrix.prototype.reset=function(){
        reset();
    }
    TransformationMatrix.prototype.setContextTransform=function(ctx){
        ctx.setTransform(m[0],m[1],m[2],m[3],m[4],m[5]);
    }
    TransformationMatrix.prototype.resetContextTransform=function(ctx){
        ctx.setTransform(1,0,0,1,0,0);
    }
    TransformationMatrix.prototype.getTransformedPoint=function(screenX,screenY){
        return(transformedPoint(screenX,screenY));
    }
    TransformationMatrix.prototype.getScreenPoint=function(transformedX,transformedY){
        return(screenPoint(transformedX,transformedY));
    }
    TransformationMatrix.prototype.getMatrix=function(){
        var clone=[m[0],m[1],m[2],m[3],m[4],m[5]];
        return(clone);
    }
    // return public
    return(TransformationMatrix);
})();
```

demo:

Questa demo utilizza la "Classe" di Transformation Matrix sopra per:

- Traccia (== salva) la matrice di trasformazione di un rettangolo.
- Ridisegna il rettangolo trasformato senza utilizzare i comandi di trasformazione del contesto.
- Verifica se il mouse ha fatto clic all'interno del rettangolo trasformato.

Codice:

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

    var canvas=document.getElementById("canvas");
    var ctx=canvas.getContext("2d");
    var cw=canvas.width;
    var ch=canvas.height;
    function reOffset(){
        var BB=canvas.getBoundingClientRect();
        offsetX=BB.left;
        offsetY=BB.top;
    }
    var offsetX,offsetY;
    reOffset();
    window.onscroll=function(e){ reOffset(); }
    window.onresize=function(e){ reOffset(); }

    // Transformation Matrix "Class"

    var TransformationMatrix=( function(){
        // private
        var self;
        var m=[1,0,0,1,0,0];
        var reset=function(){ var m=[1,0,0,1,0,0]; }
        var multiply=function(mat){
            var m0=m[0]*mat[0]+m[2]*mat[1];
            var m1=m[1]*mat[0]+m[3]*mat[1];
            var m2=m[0]*mat[2]+m[2]*mat[3];
            var m3=m[1]*mat[2]+m[3]*mat[3];
            var m4=m[0]*mat[4]+m[2]*mat[5]+m[4];
            var m5=m[1]*mat[4]+m[3]*mat[5]+m[5];
            m=[m0,m1,m2,m3,m4,m5];
        }
        var screenPoint=function(transformedX,transformedY){
            // invert
            var d =1/(m[0]*m[3]-m[1]*m[2]);
            im=[ m[3]*d, -m[1]*d, -m[2]*d, m[0]*d, d*(m[2]*m[5]-m[3]*m[4]), d*(m[1]*m[4]-
m[0]*m[5]) ];
            // point
            return({
                x:transformedX*im[0]+transformedY*im[2]+im[4],
                y:transformedX*im[1]+transformedY*im[3]+im[5]
            });
        }
        var transformedPoint=function(screenX,screenY){
```

```
            return({
                x:screenX*m[0] + screenY*m[2] + m[4],
                y:screenX*m[1] + screenY*m[3] + m[5]
            });
        }
        // public
        function TransformationMatrix(){
            self=this;
        }
        // shared methods
        TransformationMatrix.prototype.translate=function(x,y){
            var mat=[ 1, 0, 0, 1, x, y ];
            multiply(mat);
        };
        TransformationMatrix.prototype.rotate=function(rAngle){
            var c = Math.cos(rAngle);
            var s = Math.sin(rAngle);
            var mat=[ c, s, -s, c, 0, 0 ];
            multiply(mat);
        };
        TransformationMatrix.prototype.scale=function(x,y){
            var mat=[ x, 0, 0, y, 0, 0 ];
            multiply(mat);
        };
        TransformationMatrix.prototype.skew=function(radianX,radianY){
            var mat=[ 1, Math.tan(radianY), Math.tan(radianX), 1, 0, 0 ];
            multiply(mat);
        };
        TransformationMatrix.prototype.reset=function(){
            reset();
        }
        TransformationMatrix.prototype.setContextTransform=function(ctx){
            ctx.setTransform(m[0],m[1],m[2],m[3],m[4],m[5]);
        }
        TransformationMatrix.prototype.resetContextTransform=function(ctx){
            ctx.setTransform(1,0,0,1,0,0);
        }
        TransformationMatrix.prototype.getTransformedPoint=function(screenX,screenY){
            return(transformedPoint(screenX,screenY));
        }
        TransformationMatrix.prototype.getScreenPoint=function(transformedX,transformedY){
            return(screenPoint(transformedX,transformedY));
        }
        TransformationMatrix.prototype.getMatrix=function(){
            var clone=[m[0],m[1],m[2],m[3],m[4],m[5]];
            return(clone);
        }
        // return public
        return(TransformationMatrix);
    })();

    // DEMO starts here

    // create a rect and add a transformation matrix
    // to track it's translations, rotations & scalings
    var rect={x:30,y:30,w:50,h:35,matrix:new TransformationMatrix()};

    // draw the untransformed rect in black
    ctx.strokeRect(rect.x, rect.y, rect.w, rect.h);
    // Demo: label
    ctx.font='11px arial';
```

```
    ctx.fillText('Untransformed Rect',rect.x,rect.y-10);

    // transform the canvas & draw the transformed rect in red
    ctx.translate(100,0);
    ctx.scale(2,2);
    ctx.rotate(Math.PI/12);
    // draw the transformed rect
    ctx.strokeStyle='red';
    ctx.strokeRect(rect.x, rect.y, rect.w, rect.h);
    ctx.font='6px arial';
    // Demo: label
    ctx.fillText('Same Rect: Translated, rotated & scaled',rect.x,rect.y-6);
    // reset the context to untransformed state
    ctx.setTransform(1,0,0,1,0,0);

    // record the transformations in the matrix
    var m=rect.matrix;
    m.translate(100,0);
    m.scale(2,2);
    m.rotate(Math.PI/12);

    // use the rect's saved transformation matrix to reposition,
    //     resize & redraw the rect
    ctx.strokeStyle='blue';
    drawTransformedRect(rect);

    // Demo: instructions
    ctx.font='14px arial';
    ctx.fillText('Demo: click inside the blue rect',30,200);

    // redraw a rect based on it's saved transformation matrix
    function drawTransformedRect(r){
        // set the context transformation matrix using the rect's saved matrix
        m.setContextTransform(ctx);
        // draw the rect (no position or size changes needed!)
        ctx.strokeRect( r.x, r.y, r.w, r.h );
        // reset the context transformation to default (==untransformed);
        m.resetContextTransform(ctx);
    }

    // is the point in the transformed rectangle?
    function isPointInTransformedRect(r,transformedX,transformedY){
        var p=r.matrix.getScreenPoint(transformedX,transformedY);
        var x=p.x;
        var y=p.y;
        return(x>r.x && x<r.x+r.w && y>r.y && y<r.y+r.h);
    }

    // listen for mousedown events
    canvas.onmousedown=handleMouseDown;
    function handleMouseDown(e){
        // tell the browser we're handling this event
        e.preventDefault();
        e.stopPropagation();
        // get mouse position
        mouseX=parseInt(e.clientX-offsetX);
        mouseY=parseInt(e.clientY-offsetY);
        // is the mouse inside the transformed rect?
        if(isPointInTransformedRect(rect,mouseX,mouseY)){
            alert('You clicked in the transformed Rect');
        }
```

```
    }

    // Demo: redraw transformed rect without using
    //       context transformation commands
    function drawTransformedRect(r,color){
        var m=r.matrix;
        var tl=m.getTransformedPoint(r.x,r.y);
        var tr=m.getTransformedPoint(r.x+r.w,r.y);
        var br=m.getTransformedPoint(r.x+r.w,r.y+r.h);
        var bl=m.getTransformedPoint(r.x,r.y+r.h);
        ctx.beginPath();
        ctx.moveTo(tl.x,tl.y);
        ctx.lineTo(tr.x,tr.y);
        ctx.lineTo(br.x,br.y);
        ctx.lineTo(bl.x,bl.y);
        ctx.closePath();
        ctx.strokeStyle=color;
        ctx.stroke();
    }

}); // end window.onload
</script>
</head>
<body>
    <canvas id="canvas" width=512 height=250></canvas>
</body>
</html>
```

Leggi trasformazioni online: https://riptutorial.com/it/html5-canvas/topic/5494/trasformazioni

Titoli di coda

S. No	Capitoli	Contributors
1	Iniziare con html5-canvas	almcd, Blindman67, Community, Daniel Dees, Kaiido, markE, ndugger, Spencer Wieczorek, Stephen Leppik, user2314737
2	Animazione	Blindman67, markE
3	Cancellare lo schermo	bjanes, Blindman67, Kaiido, markE, Mike C, Ronen Ness
4	Collisioni e intersezioni	Blindman67, markE
5	compositing	Blindman67, markE
6	Design reattivo	Blindman67, markE, mnoronha
7	Grafici e diagrammi	Blindman67, markE
8	immagini	Blindman67, Kaiido, markE
9	Manipolazione pixel con "getImageData" e "putImageData"	markE
10	Navigazione lungo un percorso	Blindman67, markE
11	percorsi	Blindman67, markE
12	Percorso (solo sintassi)	AgataB, markE
13	poligoni	Blindman67, markE
14	Shadows	markE
15	Testo	almcd, Blindman67, markE, RamenChef
16	Tipi di media e tela	Blindman67, Bobby, Kaiido
17	Trascinando le forme del percorso e le immagini su tela	markE

18	trasformazioni	Blindman67, markE