eBook Gratuit

APPRENEZ html5-canvas

eBook gratuit non affilié créé à partir des **contributeurs de Stack Overflow.**

#html5-canvas

Table des matières

À propos

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.

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

Chapitre 1: Démarrer avec html5-canvas

Examples

Comment ajouter l'élément Canvas Html5 à une page Web

Html5-Canvas ...

- Est un élément Html5.
- Est pris en charge dans la plupart des navigateurs modernes (Internet Explorer 9+).
- Est-ce qu'un élément visible est transparent par défaut
- A une largeur par défaut de 300px et une hauteur par défaut de 150px.
- Nécessite JavaScript car tout le contenu doit être ajouté par programme au canevas.

Exemple: Créez un élément Html5-Canvas en utilisant le balisage Html5 et 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>
```

Taille et résolution du canevas

La taille d'un canevas est la zone qu'il occupe sur la page et est définie par les propriétés CSS width et height.

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

La résolution du canevas définit le nombre de pixels qu'il contient. La résolution est définie en définissant les propriétés de largeur et de hauteur de l'élément de canevas. S'il n'est pas spécifié, le canevas est défini par défaut sur 300 par 150 pixels.

Le canevas suivant utilisera la taille CSS ci-dessus, mais comme la `width` et la `height` ne sont pas spécifiées, la résolution sera de 300 par 150.

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

Cela aura pour résultat que chaque pixel sera étiré de manière inégale. L'aspect des pixels est 1: 2. Lorsque le canevas est étiré, le navigateur utilise le filtrage bilinéaire. Cela a pour effet de rendre flous les pixels étirés.

Pour de meilleurs résultats lors de l'utilisation de la zone de dessin, assurez-vous que la résolution de la zone de dessin correspond à la taille de l'écran.

Suite au style CSS ci-dessus pour correspondre à la taille d'affichage, ajoutez le canevas avec la `width` et la `height` définies sur le même nombre de pixels que le style définit.

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

Toile hors écran

Souvent, lorsque vous travaillez avec le canevas, vous aurez besoin d'un canevas pour contenir des données de pixels d'intrum. Il est facile de créer un canevas hors écran, d'obtenir un contexte 2D. Un canevas hors écran utilisera également le matériel graphique disponible pour le rendu.

Le code suivant crée simplement un canevas et le remplit de pixels bleus.

```
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);
```

Plusieurs fois, le canevas hors écran sera utilisé pour de nombreuses tâches et vous pouvez avoir de nombreuses toiles. Pour simplifier l'utilisation du canevas, vous pouvez associer le contexte de canevas au canevas.

```
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);
```

Détection de la position de la souris sur la toile

Cet exemple montre comment obtenir la position de la souris par rapport au canevas, tel que `(0,0)` soit le coin supérieur gauche du canevas HTML5. `e.clientX` et `e.clientY` obtiendront les positions de la souris par rapport au haut du document. Pour que cela change en fonction du haut du canevas, nous soustrayons les positions `left` et `right` du canevas du client X et 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);
});
```

Exemple Runnable

L'utilisation de `Math.round` doit garantir que les positions `x,y` sont des nombres entiers, car le rectangle englobant du canevas peut ne pas avoir de positions entières.

Bonjour le monde

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);
```

Résultat

Un index pour Html5 Canvas Capabilities & Uses

Capacités de la toile

Canvas vous permet de dessiner par programmation sur votre page Web:

- Des images ,
- Des textes ,
- Lignes et courbes .

Les dessins sur toile peuvent être intensément stylisés:

- largeur de course ,
- couleur de course ,
- couleur de remplissage de forme ,
- opacité ,
- l'observation ,
- gradients linéaires et gradients radiaux ,
- visage de la police ,
- taille de la police ,
- l'alignement du texte ,
- le texte peut être caressé, rempli ou à la fois caressé et rempli ,
- redimensionnement de l'image ,
- recadrage d'images ,
- composition

Usages de la toile

Les dessins peuvent être combinés et positionnés n'importe où sur la toile pour pouvoir être utilisés pour créer:

- Applications de peinture / dessin,
- Jeux interactifs rapides,
- Analyses de données comme des graphiques, des graphiques,
- Imagerie de type Photoshop,
- Publicité Flash et contenu Web Flashy.

Canvas vous permet de manipuler les couleurs des composants rouge, vert, bleu et alpha des images. Cela permet au canevas de manipuler des images avec des résultats similaires à ceux de

Photoshop.

- Recolorer une partie quelconque d'une image au niveau du pixel (si vous utilisez HSL, vous pouvez même recolorer une image tout en conservant l'important Lighting & Saturation pour que le résultat ne ressemble pas à de la peinture giflée sur l'image),
- "Knockout" l'arrière-plan autour d'une personne / d'un objet dans une image,
- Détecter et remplir une partie d'une image (par exemple, changer la couleur d'un pétale de fleur sur lequel l'utilisateur a cliqué de vert à jaune - juste le pétale cliqué!),
- Faire du gauchissement en perspective (par exemple, envelopper une image autour de la courbe d'une tasse),
- Examiner une image pour le contenu (par exemple, reconnaissance faciale),
- Répondez aux questions concernant une image: y a-t-il une voiture garée sur cette image de mon parking ?,
- Appliquer des filtres d'image standard (niveaux de gris, sépia, etc.)
- Appliquez n'importe quel filtre d'image exotique que vous pouvez imaginer (détection de bord Sobel),
- Combinez des images. Si la chère grand-mère Sue ne pouvait pas se rendre à la réunion de famille, il suffit de la "photoshop" dans l'image de la réunion. Je n'aime pas le cousin Phil - juste "photoshop it out,
- Lire une vidéo / Saisissez une image dans une vidéo,
- Exportez le contenu de la toile en tant que .jpg | .png image (vous pouvez même éventuellement recadrer ou annoter l'image et exporter le résultat sous la forme d'une nouvelle image),

A propos du déplacement et de l'édition de dessins de canevas (par exemple pour créer un jeu d'action):

- Une fois que quelque chose a été dessiné sur le canevas, ce dessin existant ne peut pas être déplacé ou modifié. Cette idée fausse commune selon laquelle les dessins sur toile sont mobiles mérite d'être clarifiée: *les dessins sur toile existants ne peuvent pas être modifiés ou déplacés!*
- La toile attire très, très rapidement. Canvas peut dessiner des centaines d'images, de textes, de lignes et de courbes en une fraction de seconde. Il utilise le GPU lorsqu'il est disponible pour accélérer le dessin.
- Canvas crée l'illusion du mouvement en dessinant rapidement et à plusieurs reprises quelque chose puis en le redessinant dans une nouvelle position. Comme la télévision, cette redéfinition constante donne à l'œil l'illusion du mouvement.

Tourner

La méthode `rotate(r)` du contexte 2D fait pivoter le canevas par la quantité `r` spécifiée de *radians* autour de l'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);
};
```

Démo en direct sur JSfiddle

Enregistrer le canevas dans le fichier image

Vous pouvez enregistrer un canevas dans un fichier image à l'aide de la méthode `canvas.toDataURL()` , qui renvoie l' *URI de données* pour les données d'image du canevas.

La méthode peut prendre deux paramètres facultatifs `canvas.toDataURL(type, encoderOptions)` : `type` est le format de l'image (si omis, la valeur par défaut est `image/png`); `encoderOptions` est un nombre compris entre 0 et 1 indiquant la qualité de l'image (la valeur par défaut est 0,92).

Ici, nous dessinons un canevas et attachons l'URI de données du canevas au lien "Télécharger sur 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;
};
```

Démo en direct sur JSfiddle.

Lire Démarrer avec html5-canvas en ligne: https://riptutorial.com/fr/html5-canvas/topic/1892/demarrer-avec-html5-canvas

Chapitre 2: Animation

Examples

Animation simple avec contexte 2D et requestAnimationFrame

Cet exemple vous montre comment créer une animation simple en utilisant le canevas et le contexte 2D. On suppose que vous savez créer et ajouter un canevas au DOM et obtenir le contexte

```
// 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
```

Une démo de cet exemple à jsfiddle

Animer à un intervalle spécifié (ajouter un nouveau rectangle toutes les 1 secondes)

Cet exemple ajoute un nouveau rectangle au canevas toutes les 1 secondes (== un intervalle de 1 seconde)

Code annoté:

```
<!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>
```

Animer à une heure spécifiée (une horloge animée)

Cet exemple anime une horloge indiquant les secondes comme un coin rempli

Code annoté:

```
<!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>
```

Utilisez requestAnimationFrame () NOT setInterval () pour les boucles d'animation

`requestAnimationFrame` est similaire à setInterval, mais a ces améliorations importantes:

- Le code d'animation est synchronisé avec les rafraîchissements de l'affichage pour des raisons d'efficacité Le code d'effacement + redessiner est planifié, mais pas exécuté immédiatement. Le navigateur exécute le code clair + redessiner uniquement lorsque l'affichage est prêt à être actualisé. Cette synchronisation avec le cycle de rafraîchissement augmente les performances de votre animation en donnant à votre code le temps le plus disponible.
- Chaque boucle est toujours terminée avant qu'une autre boucle ne soit autorisée à démarrer. Cela évite la "déchirure", où l'utilisateur voit une version incomplète du dessin. L'œil remarque particulièrement la déchirure et est distrait lorsque la déchirure se produit. Ainsi, empêcher les déchirures rend votre animation plus fluide et plus cohérente.
- L'animation s'arrête automatiquement lorsque l'utilisateur bascule sur un autre onglet du navigateur. Cela permet d'économiser de l'énergie sur les appareils mobiles, car l'appareil ne gaspille pas de puissance en calculant une animation que l'utilisateur ne peut actuellement pas voir.

Les affichages de périphérique seront actualisés environ 60 fois par seconde, donc requestAnimationFrame peut être redessiné en continu à environ 60 "images" par seconde. L'œil voit le mouvement à 20-30 images par seconde, donc requestAnimationFrame peut facilement créer l'illusion du mouvement.

Notez que requestAnimationFrame est rappelé à la fin de chaque animateCircle. C'est parce que chaque requestAnimatonFrameonly demande une seule exécution de la fonction d'animation.

Exemple: simple `requestAnimationFrame

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

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

```
    // start the animation
    requestAnimationFrame(animate);

    function animate(currentTime){

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

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

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

Pour illustrer les avantages de requestAnimationFrame, cette question stackoverflow a une démonstration en direct

Animer une image sur le canevas

Cet exemple charge et anime l'image sur le canevas

Conseil important! Assurez-vous de donner à votre image le temps `image.onload` pour charger complètement en utilisant `image.onload` .

Code annoté

```
<!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>
```

Ne dessinez pas d'animations dans vos gestionnaires d'événements (une simple application de croquis)

Au cours de `mousemove` vous recevez 30 événements de souris par seconde. Vous ne pourrez peut-être pas redessiner vos dessins 30 fois par seconde. Même si vous le pouvez, vous gaspillez probablement votre puissance de calcul en dessinant lorsque le navigateur n'est pas prêt à dessiner (gaspillage == entre les cycles de rafraîchissement de l'affichage).

Par conséquent, il est judicieux de séparer les événements de saisie de vos utilisateurs (comme mousemove) du dessin de vos animations.

- Dans les gestionnaires d'événements, enregistrez toutes les variables d'événement qui contrôlent l'emplacement des dessins sur le canevas. Mais ne dessine rien.
- Dans une boucle `requestAnimationFrame` , `requestAnimationFrame` tous les dessins dans le

canevas à l'aide des informations enregistrées.

En ne dessinant pas dans les gestionnaires d'événements, vous n'obligez pas Canvas à essayer d'actualiser des dessins complexes à la vitesse des événements de la souris.

En effectuant tous les dessins dans `requestAnimationFrame` vous `requestAnimationFrame` tous les avantages décrits ici. Utilisez 'requestanimationFrame' et non 'setInterval' pour les boucles d'animation .

Code annoté:

```
<!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>
```

Faciliter l'utilisation des équations de Robert Penners

Un assouplissement fait que certaines **variables** changent de **manière inégale** sur une **durée** .

"variable" doit pouvoir être exprimé sous forme de nombre et peut représenter une variété remarquable de choses:

- une coordonnée X,
- la largeur d'un rectangle,
- un angle de rotation,
- la composante rouge d'une couleur R, G, B.
- tout ce qui peut être exprimé sous forme de nombre.

La "durée" doit pouvoir être exprimée sous forme de nombre et peut aussi être une variété de choses:

- une période de temps,
- une distance à parcourir,

- une quantité de boucles d'animation à exécuter,
- tout ce qui peut être exprimé comme

"inégalement" signifie que la variable progresse de façon inégale vers les valeurs finales:

- plus vite au début et plus lent à la fin - ou vice-versa,
- dépasse la fin mais revient à la fin à la fin de la durée,
- avance / recule élastiquement à plusieurs reprises pendant la durée,
- "rebondit" sur la fin tout en se reposant à la fin de la durée.

Attribution: Robert Penner a créé le "standard de référence" des fonctions d'assouplissement.

Citer: 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;
    },
};
```

Exemple d'utilisation:

```
// 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);
    }
}
```

Définir la fréquence d'images à l'aide de requestAnimationFrame

Utiliser requestAnimationFrame peut sur certains systèmes mettre à jour plus d'images par seconde que les 60fps. 60fps est le taux par défaut si le rendu peut suivre. Certains systèmes fonctionneront à 120 images par seconde, voire plus.

Si vous utilisez la méthode suivante, vous ne devez utiliser que des cadences qui sont des divisions entières de 60, de sorte que `(60 / FRAMES_PER_SECOND) % 1 === 0` soit `true` ou que vous obtiendrez des fréquences d'images incohérentes.

```
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
```

Animer de [x0, y0] à [x1, y1]

Utiliser des vecteurs pour calculer les incréments [x, y] de [startX, startY] à [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;
```

Exemple de code:

```
// 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);
}
```

Lire Animation en ligne: https://riptutorial.com/fr/html5-canvas/topic/4822/animation

Chapitre 3: Chemin (syntaxe uniquement)

Syntaxe

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

Examples

Vue d'ensemble des commandes de dessin du chemin de base: lignes et courbes

==========

TODO: Liez chacune des commandes de dessin ci-dessous à leurs exemples individuels. Je ne sais pas comment faire car les liens vers les exemples individuels pointent vers le dossier "draft".

TODO: Ajouter des exemples pour ces commandes "action" du chemin: stroke (), fill (), clip ()

==========

Chemin

Un chemin définit un ensemble de lignes et de courbes pouvant être visiblement dessinées sur le canevas.

Un chemin n'est pas automatiquement dessiné sur le canevas. Mais les lignes et les courbes du chemin peuvent être dessinées sur le canevas en utilisant un trait stylé. Et la forme créée par les lignes et les courbes peut également être remplie avec un remplissage stylé.

Les chemins ont des utilisations qui vont au-delà du dessin sur le canevas:

- Hit test si une coordonnée x, y est à l'intérieur de la forme du chemin.
- Définition d'une zone de découpage dans laquelle seuls les dessins à l'intérieur de la zone de découpage seront visibles. Tous les dessins en dehors de la zone de découpage ne seront pas dessinés (== transparent), ce qui est similaire au dépassement CSS.

Les commandes de dessin de chemin de base sont les suivantes:

- beginpath
- déménager à
- lineTo
- arc
- quadraticCurveTo
- bezierCurveTo
- arcTo
- rect
- prèsPath

Description des commandes de dessin de base:

beginPath

```
context.beginPath()
```

Commence à assembler un nouvel ensemble de commandes de chemin et supprime également tout chemin précédemment assemblé.

Le rejet est un point important et souvent négligé. Si vous ne commencez pas un nouveau chemin, toutes les commandes de chemin émises précédemment seront automatiquement redessinées.

Il déplace également le dessin "stylo" vers l'origine supérieure gauche du canevas (coordonnée == [0,0]).

déménager à

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

Déplace l'emplacement actuel du stylet sur la coordonnée [startX, startY].

Par défaut, tous les dessins de chemin sont connectés ensemble. Ainsi, le point d'arrivée d'une ligne ou d'une courbe est le point de départ de la ligne ou de la courbe suivante. Cela peut entraîner une ligne inattendue entre deux dessins adjacents. La `context.moveTo` commande essentiellement « prend le stylo dessin » et place à une nouvelle coordonnée de sorte que la ligne automatique de connexion ne soit pas établi.

lineTo

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

Dessine un segment de ligne à partir de l'emplacement actuel du stylet pour coordonner [endX, endY]

Vous pouvez assembler plusieurs commandes `.lineTo` pour dessiner une polyligne. Par exemple, vous pouvez assembler 3 segments de ligne pour former un triangle.

arc

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

Dessine un arc de cercle selon un point central, un rayon et des angles de départ et de fin. Les angles sont exprimés en radians. Pour convertir des degrés en radians, vous pouvez utiliser cette formule: `radians = degrees * Math.PI / 180; .`

L'angle 0 fait face directement au centre de l'arc. Pour dessiner un cercle complet, vous pouvez faire terminerAngle = startAngle + 360 degrés (360 degrés == Math.PI *2): `context.arc (10,10,20,0, Math.PI* 2);

Par défaut, l'arc est dessiné dans le sens des aiguilles d'une montre. Un paramètre optionnel [true | false] indique à l'arc d'être dessiné dans le sens inverse des aiguilles d'une montre: `context.arc(10,10,20,0,Math.PI*2,true)`

quadraticCurveTo

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

Dessine une courbe quadratique commençant à l'emplacement actuel du stylet jusqu'à une coordonnée de fin donnée. Une autre coordonnée de contrôle donnée détermine la forme (courbure) de la courbe.

bezierCurveTo

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

Dessine une courbe de Bézier cubique commençant à l'emplacement actuel du stylet jusqu'à une coordonnée de fin donnée. Deux autres coordonnées de contrôle données déterminent la forme (courbure) de la courbe.

arcTo

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

Dessine un arc de cercle avec un rayon donné. L'arc est dessiné dans le sens des aiguilles d'une montre à l'intérieur du coin formé par l'emplacement actuel du stylo et reçoit deux points: Point1 & Point2.

Une ligne reliant l'emplacement actuel du stylet et le début de l'arc est automatiquement dessinée avant l'arc.

rect

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

Dessine un rectangle à partir d'un coin supérieur gauche et d'une largeur et hauteur.

`context.rect` est une commande de dessin unique car elle ajoute des rectangles déconnectés. Ces rectangles déconnectés ne sont pas automatiquement connectés par des lignes.

prèsPath

```
context.closePath()
```

Dessine une ligne entre l'emplacement actuel du stylet et la coordonnée du chemin de début.

Par exemple, si vous tracez 2 lignes formant 2 branches d'un triangle, `closePath` "fermera" le triangle en dessinant la troisième branche du triangle du point d'extrémité de la 2ème jambe au point de départ de la première jambe.

Le nom de cette commande est souvent mal compris. `context.closePath` N'EST PAS un délimiteur de fin pour `context.beginPath` . Encore une fois, la commande `closePath` dessine une ligne - elle ne "ferme" pas un `beginPath` .

lineTo (une commande de chemin)

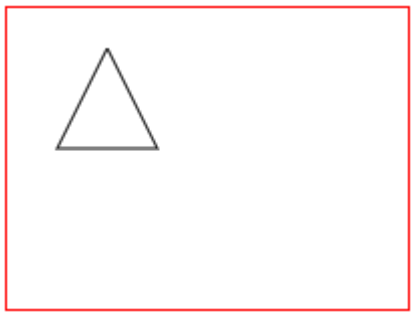
```
context.lineTo(endX, endY)
```

Dessine un segment de ligne à partir de l'emplacement actuel du stylet pour coordonner [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>
```

Vous pouvez assembler plusieurs commandes .lineTo pour dessiner une polyligne. Par exemple, vous pouvez assembler 3 segments de ligne pour former un triangle.



```
<!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>
```

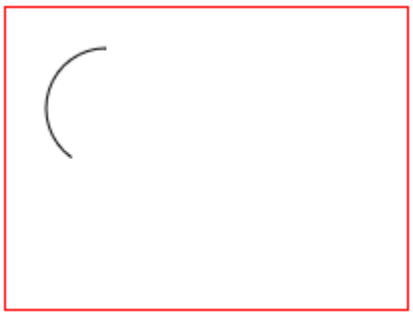
arc (une commande de chemin)

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

Dessine un arc de cercle selon un point central, un rayon et des angles de départ et de fin. Les angles sont exprimés en radians. Pour convertir des degrés en radians, vous pouvez utiliser cette formule: `radians = degrees * Math.PI / 180; .`

L'angle 0 fait face directement au centre de l'arc.

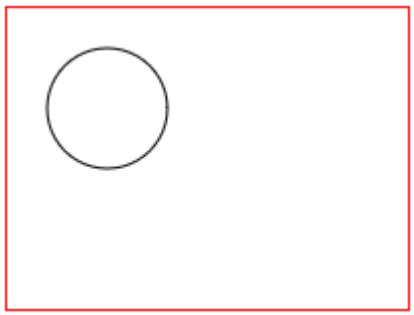
Par défaut, l'arc est dessiné dans le sens des aiguilles d'une montre. Un paramètre optionnel [true | false] indique à l'arc d'être dessiné dans le sens inverse des aiguilles d'une montre:
`context.arc(10,10,20,0,Math.PI*2,true)`



```
<!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>
```

Pour dessiner un cercle complet, vous pouvez utiliser la méthode terminerAngle = startingAngle + 360 degrés (360 degrés == Math.PI2).



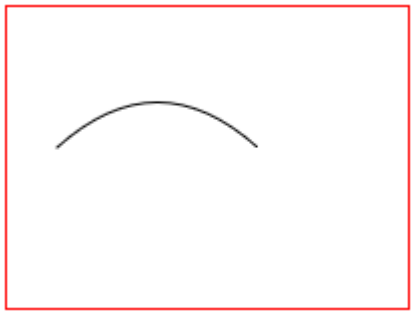
```
<!doctype html>
<html>
<head>
<style>
    body{ background-color:white; }
    #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

    // 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 (une commande de chemin)

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

Dessine une courbe quadratique commençant à l'emplacement actuel du stylet jusqu'à une coordonnée de fin donnée. Une autre coordonnée de contrôle donnée détermine la forme (courbure) de la courbe.



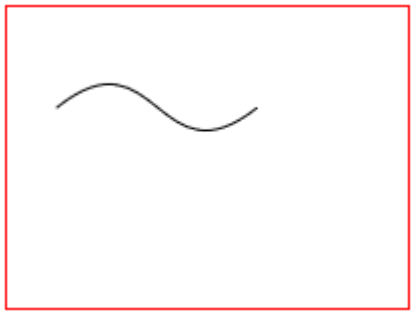
```
<!doctype html>
<html>
<head>
<style>
    body{ background-color:white; }
    #canvas{border:1px solid red; }
</style>
<script>
window.onload=(function(){

    // 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 (une commande de chemin)

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

Dessine une courbe de Bézier cubique commençant à l'emplacement actuel du stylet jusqu'à une coordonnée de fin donnée. Deux autres coordonnées de contrôle données déterminent la forme (courbure) de la courbe.



```
<!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>
```

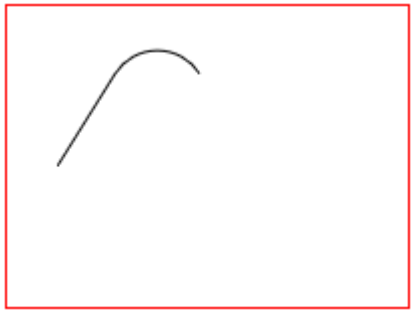
```
</body>
</html>
```

arcTo (une commande de chemin)

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

Dessine un arc de cercle avec un rayon donné. L'arc est dessiné dans le sens des aiguilles d'une montre à l'intérieur du coin formé par l'emplacement actuel du stylo et reçoit deux points: Point1 & Point2.

Une ligne reliant l'emplacement actuel du stylet et le début de l'arc est automatiquement dessinée avant l'arc.



```
<!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 (une commande de chemin)

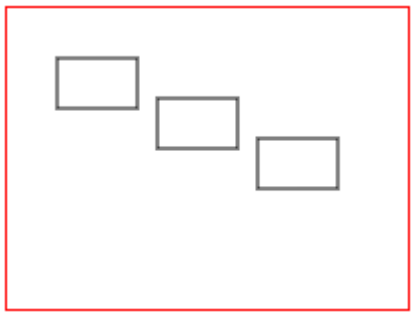
```
context.rect(leftX, topY, width, height)
```

Dessine un rectangle à partir d'un coin supérieur gauche et d'une largeur et hauteur.

```
<!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` est une commande de dessin unique car elle ajoute des rectangles déconnectés.

Ces rectangles déconnectés ne sont pas automatiquement connectés par des lignes.



```
<!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 (une commande de chemin)

```
context.closePath()
```

Dessine une ligne entre l'emplacement actuel du stylet et la coordonnée du chemin de début.

Par exemple, si vous tracez 2 lignes formant 2 branches d'un triangle, closePath "fermera" le triangle en dessinant la troisième branche du triangle du point d'extrémité de la 2ème jambe au point de départ de la première jambe.

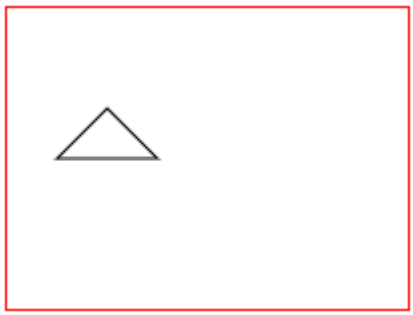
Une idée fausse expliquée!

Le nom de cette commande est souvent mal compris.

`context.closePath` N'EST PAS un délimiteur de fin pour `context.beginPath` .

Encore une fois, la commande closePath dessine une ligne - elle ne "ferme" pas un beginPath.

Cet exemple dessine 2 jambes d'un triangle et utilise `closePath` pour compléter (fermer ?!) le triangle en dessinant la troisième jambe. `closePath` fait, `closePath` trace une ligne entre le point d'extrémité de la deuxième étape et le point de départ de la première étape.



```
<!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 (une commande de chemin)

```
context.beginPath()
```

Commence à assembler un nouvel ensemble de commandes de chemin et supprime également tout chemin précédemment assemblé.

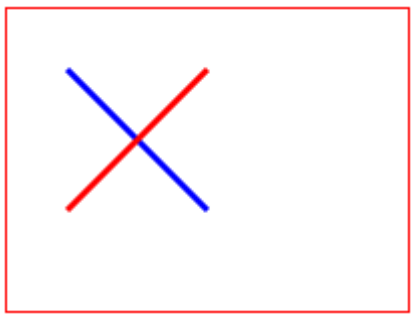
Il déplace également le dessin "stylo" vers l'origine supérieure gauche du canevas (coordonnée == [0,0]).

Bien que facultatif, vous devriez TOUJOURS démarrer un chemin avec `beginPath`

Le rejet est un point important et souvent négligé. Si vous ne commencez pas un nouveau chemin avec `beginPath` , toutes les commandes de chemin émises précédemment seront automatiquement redessinées.

Ces deux démos tentent toutes deux de dessiner un "X" avec un trait rouge et un trait bleu.

Cette première démo utilise correctement `beginPath` pour lancer son deuxième trait rouge. Le résultat est que le "X" a correctement un trait rouge et un trait bleu.



```
<!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);
```

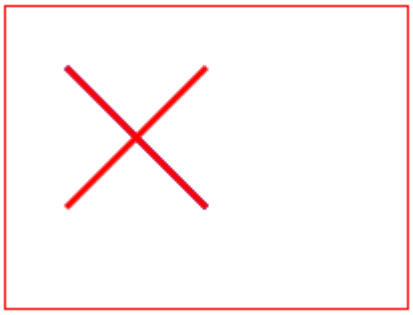
```
    ctx.lineTo(100,100);
    ctx.strokeStyle='blue';
    ctx.lineWidth=3;
    ctx.stroke();

    // 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>
```

Cette deuxième démo laisse incorrectement `beginPath` sur le deuxième coup. Le résultat est que le "X" a incorrectement les deux traits rouges.

Le deuxième `stroke()` dessine le deuxième trait rouge.

Mais sans un second `beginPath` , ce même deuxième `stroke()` **redessine** aussi incorrectement le premier trait.

Comme le second `stroke()` est maintenant rouge, le premier trait bleu est **remplacé** par un trait rouge incorrectement coloré.



```
<!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
```

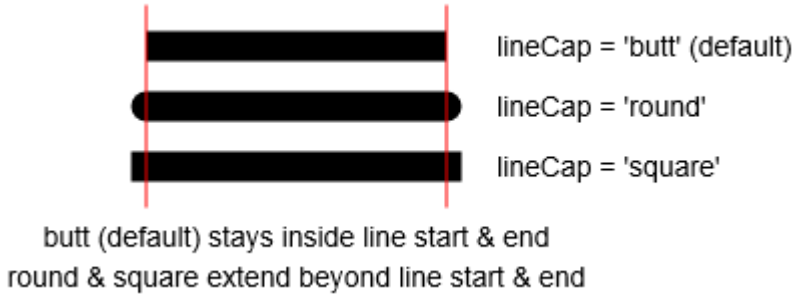
```
    ctx.beginPath();
    ctx.moveTo(30,30);
    ctx.lineTo(100,100);
    ctx.strokeStyle='blue';
    ctx.lineWidth=3;
    ctx.stroke();

    // 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 (attribut de style de chemin)

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

Définit le style de majuscule des points de départ de ligne et des points de fin.

- **butt** , le style lineCap par défaut, affiche les majuscules au carré qui ne dépassent pas les points de début et de fin de la ligne.
- **rond** , montre des calottes arrondies qui dépassent les points de départ et d'arrivée de la ligne.
- **square** , montre des calques carrés qui dépassent les points de départ et d'arrivée de la ligne.




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

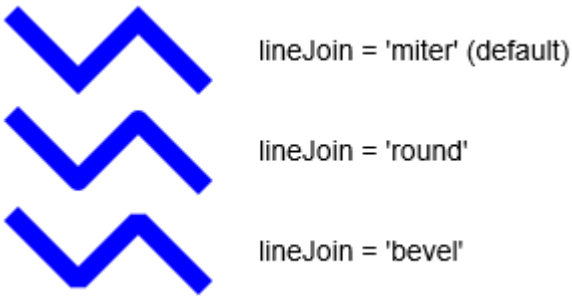
```
<script>
window.onload=(function(){

    // 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 (attribut de style de chemin)

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

Définit le style utilisé pour connecter les segments de ligne adjacents.

- **mitre** , la valeur par défaut, joint les segments de ligne avec une jointure pointue.
- **rond** , joint des segments de ligne avec un joint arrondi.
- **biseau** , rejoint les segments de ligne avec un joint émoussé.




```
<!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 (attribut de style de chemin)

```
context.strokeStyle=color
```

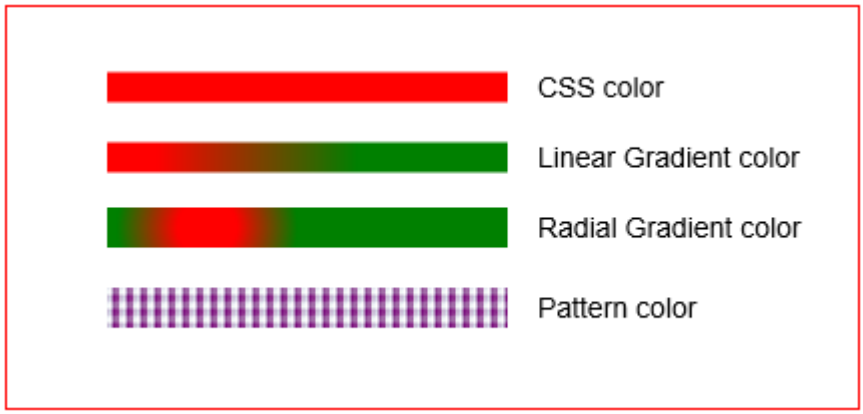
Définit la couleur qui sera utilisée pour tracer le contour du chemin actuel.

Ce sont `color` options de `color` (celles-ci doivent être citées):

- **Une couleur nommée CSS** , par exemple `context.strokeStyle='red'`
- **Une couleur hexadécimale** , par exemple `context.strokeStyle='#FF0000'`
- **Une couleur RVB** , par exemple `context.strokeStyle='rgb(red,green,blue)'` où rouge, vert et bleu sont des nombres entiers compris entre 0 et 255 indiquant la force de chaque couleur de composant.
- **Une couleur HSL** , par exemple `context.strokeStyle='hsl(hue,saturation,lightness)'` où la teinte est un entier compris entre 0 et 360 sur la roue chromatique et où saturation et luminosité sont des pourcentages (0-100%) indiquant la force de chaque composant .
- **Une couleur HSLA** , par exemple `context.strokeStyle='hsl(hue,saturation,lightness,alpha)'` où la teinte est un entier compris entre 0 et 360 sur la roue chromatique et où la saturation et la luminosité sont des pourcentages (0-100%) indiquant la force de chaque composant et alpha est une valeur décimale de 0,00-1,00 indiquant l'opacité.

Vous pouvez également spécifier ces options de couleur (ces options sont des objets créés par le contexte):

- **Un dégradé linéaire** qui est un objet de dégradé linéaire créé avec `context.createLinearGradient`
- **Un dégradé radial** qui est un objet de dégradé radial créé avec `context.createRadialGradient`
- **Un modèle** qui est un objet de modèle créé avec `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 (attribut de style de chemin)

```
context.fillStyle=color
```

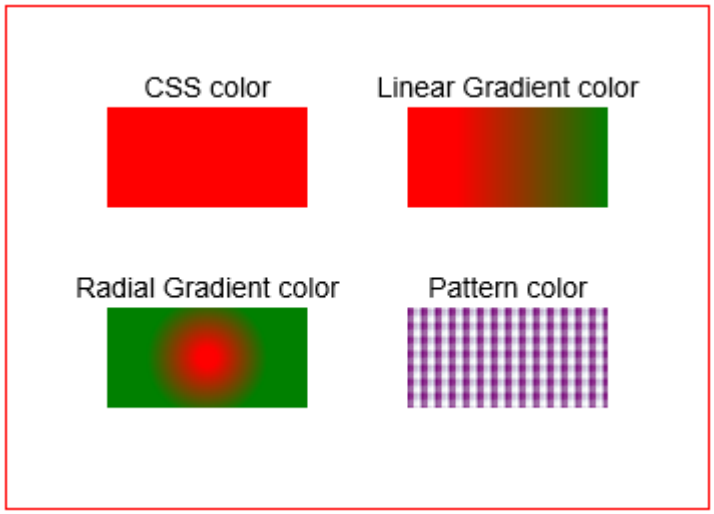
Définit la couleur qui sera utilisée pour remplir l'intérieur du chemin actuel.

Ce sont des options de couleur (celles-ci doivent être citées):

- **Une couleur nommée CSS** , par exemple `context.fillStyle='red'`
- **Une couleur hexadécimale** , par exemple `context.fillStyle='#FF0000'`
- **Une couleur RVB** , par exemple `context.fillStyle='rgb(red,green,blue)'` où rouge, vert et bleu sont des nombres entiers compris entre 0 et 255 indiquant la force de la couleur de chaque composant.
- **Une couleur HSL** , par exemple `context.fillStyle='hsl(hue,saturation,lightness)'` où la teinte est un entier compris entre 0 et 360 sur la roue chromatique et où la saturation et la luminosité sont des pourcentages (0-100%) indiquant la force de chaque composant .
- **Une couleur HSLA** , par exemple `context.fillStyle='hsl(hue,saturation,lightness,alpha)'` où la teinte est un entier compris entre 0 et 360 sur la roue chromatique et où la saturation et la luminosité sont des pourcentages (0-100%) indiquant la force de chaque composant et alpha est une valeur décimale de 0,00-1,00 indiquant l'opacité.

Vous pouvez également spécifier ces options de couleur (ces options sont des objets créés par le contexte):

- **Un dégradé linéaire** qui est un objet de dégradé linéaire créé avec `context.createLinearGradient`
- **Un dégradé radial** qui est un objet de dégradé radial créé avec `context.createRadialGradient`
- **Un modèle** qui est un objet de modèle créé avec `context.createPattern`



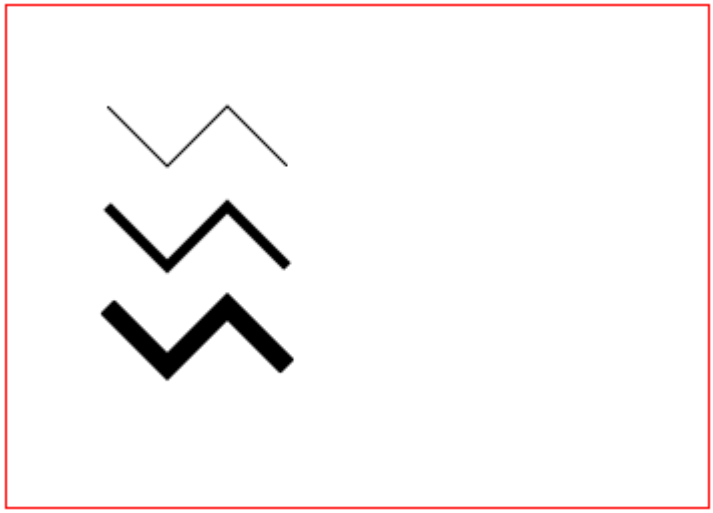

```
<!doctype html>
<html>
<head>
<style>
```

```
    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 (attribut de style de chemin)

```
context.lineWidth=lineWidth
```

Définit la largeur de la ligne qui va marquer le contour du chemin



```
<!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 (attributs de style de chemin)

```
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
```

Cet ensemble d'attributs ajoutera une ombre autour d'un chemin.

Les chemins remplis et les chemins carpés peuvent avoir une ombre.

L'ombre est la plus sombre (opaque) sur le périmètre de la trajectoire et devient de plus en plus claire à mesure qu'elle s'éloigne du périmètre de la trajectoire.

- **shadowColor** indique quelle couleur CSS sera utilisée pour créer l'ombre.
- **shadowBlur** est la distance sur laquelle l'ombre s'étend vers l'extérieur du chemin.
- **shadowOffsetX** est une distance par laquelle l'ombre est décalée horizontalement du chemin. Une distance positive déplace l'ombre vers la droite, une distance négative déplace l'ombre vers la gauche.
- **shadowOffsetY** est une distance par laquelle l'ombre est décalée verticalement du chemin. Une distance positive déplace l'ombre vers le bas, une distance négative déplace l'ombre vers le haut.

À propos de shadowOffsetX & shadowOffsetY

Il est important de noter que *l'ombre entière est déplacée dans son intégralité* . Cela provoquera le déplacement d'une partie de l'ombre sous les chemins remplis et donc une partie de l'ombre ne sera pas visible.

À propos des traits ombrés

Lors de l'observation d'un trait, l'intérieur et l'extérieur du trait sont ombrés. L'ombre est la plus sombre au coup et s'éclaircit lorsque l'ombre s'étend vers l'extérieur dans les deux directions à partir du coup.

Désactiver l'ombrage une fois terminé

Après avoir dessiné vos ombres, vous voudrez peut-être désactiver l'ombre pour dessiner plus de chemins. Pour désactiver l'observation, définissez `shadowColor` sur transparent.

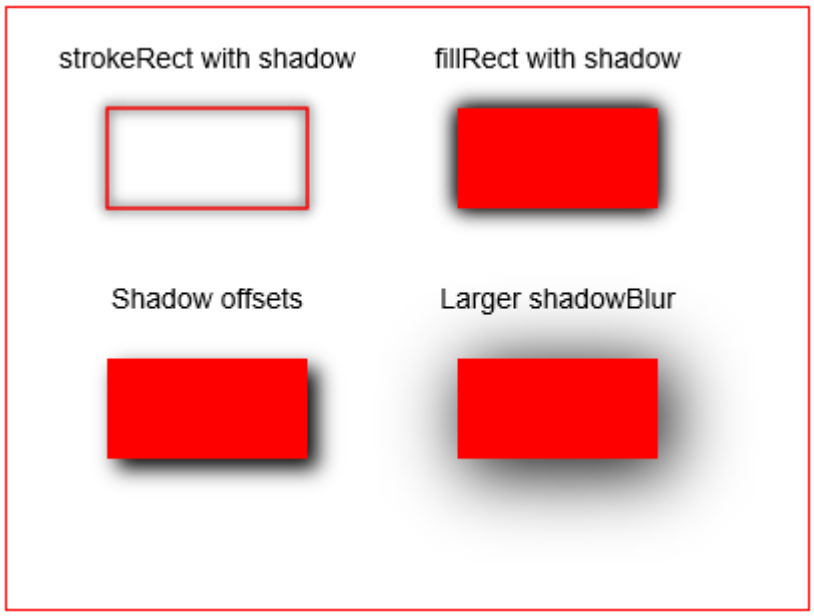
```
context.shadowColor = 'rgba(0,0,0,0)';
```

Considérations de performance

Les ombres (comme les dégradés) nécessitent des calculs approfondis et vous devez donc utiliser les ombres avec parcimonie.

Soyez particulièrement prudent lorsque vous animez, car dessiner des ombres plusieurs fois par

seconde aura un impact considérable sur les performances. Une solution de contournement si vous devez animer des chemins ombrés consiste à pré-créer le chemin ombré sur un second "shadow-canvas". Le shadow-canvas est un canevas normal créé en mémoire avec `document.createElement` - il n'est pas ajouté au DOM (c'est juste un canevas intermédiaire). Ensuite, dessinez le shadow-canvas sur le canevas principal. C'est beaucoup plus rapide car les calculs d'ombre n'ont pas besoin d'être faits plusieurs fois par seconde. Tout ce que vous faites est de copier un canevas pré-construit sur votre canevas visible.




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

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

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

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

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

    // a wider blur (==extends further from the path)
    ctx.shadowColor='black';
    ctx.shadowBlur=35;
    ctx.fillStyle='red';
    ctx.fillRect(225,175,100,50);

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

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

createLinearGradient (crée un objet de style de chemin)

```
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]
```

Crée un dégradé linéaire réutilisable (objet).

L'objet peut être affecté à n'importe quel `strokeStyle` et / ou `fillStyle` .

Puis stroke () ou fill () coloreront le chemin avec les couleurs de dégradé de l'objet.

La création d'un objet dégradé est un processus en deux étapes:

1. Créez l'objet dégradé lui-même. Lors de la création, vous définissez une ligne sur le canevas où le dégradé va commencer et se terminer. L'objet dégradé est créé avec `var gradient = context.createLinearGradient` .
2. Ajoutez ensuite 2 (ou plus) couleurs qui composent le dégradé. Cela se fait en ajoutant plusieurs arrêts de couleur à l'objet `gradient.addColorStop` avec `gradient.addColorStop` .

Arguments:

- **startX, startY** est la coordonnée de toile où le dégradé commence. Au point de départ (et avant), le canevas est solidement la couleur du `gradientPercentPosition` le plus bas.
- **endX, endY** est la coordonnée de la toile où le dégradé se termine. Au point final (et après), le canevas est solidement la couleur de la position de `gradientPercentPosition` la plus élevée.

- **gradientPercentPosition** est un nombre flottant compris entre 0,00 et 1,00 affecté à un arrêt de couleur. Il s'agit essentiellement d'un point de passage en pourcentage le long de la ligne où cet arrêt de couleur particulier s'applique.
    - Le dégradé commence au pourcentage 0,00, qui est [startX, startY] sur le canevas.
    - Le dégradé se termine au pourcentage 1,00 qui est [endX, endY] sur le canevas.
    - *Note technique:* Le terme "pourcentage" n'est pas techniquement correct puisque les valeurs vont de 0,00 à 1,00 plutôt que de 0% à 100%.
- **CssColor** est une couleur CSS assignée à cet arrêt de couleur particulier.

L'objet dégradé est un objet que vous pouvez utiliser (et réutiliser!) Pour que les traits et les remplissages de votre tracé deviennent dégradés.

Note latérale: L'objet dégradé n'est pas interne à l'élément Canvas ni à son contexte. C'est un objet JavaScript distinct et réutilisable que vous pouvez attribuer à n'importe quel chemin souhaité. Vous pouvez même utiliser cet objet pour colorer un chemin sur un autre élément Canvas (!)

Les arrêts de couleur sont des points de passage (en pourcentage) le long de la ligne de dégradé. A chaque point de passage de couleur, le dégradé est entièrement (== opaque) coloré avec la couleur qui lui est attribuée. Les points intermédiaires sur la ligne de dégradé entre les arrêts de couleur sont colorés en tant que dégradés de la couleur this et de la couleur précédente.

Conseil important sur les dégradés de la toile!

Lorsque vous créez un objet dégradé, l'intégralité du canevas est remplie de manière invisible par ce dégradé.

Lorsque vous `stroke()` ou `fill()` un chemin, le dégradé invisible est révélé, mais seulement révélé sur ce chemin en cours de trait ou de remplissage.

1. Si vous créez un dégradé linéaire rouge-magenta comme ceci:

```
// 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. Alors Canvas verra "invisible" votre création de dégradé comme ceci:

3. Mais tant que vous n'aurez pas `stroke()` ou `fill()` avec le dégradé, vous ne verrez aucun dégradé sur le canevas.

4. Enfin, si vous tracez ou remplissez un tracé à l'aide du dégradé, le dégradé "invisible" devient visible sur le canevas ... mais uniquement là où le tracé est dessiné.

```
<!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 (crée un objet de style de chemin)

```
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]
```

Crée un dégradé radial réutilisable (objet). L'objet dégradé est un objet que vous pouvez utiliser (et réutiliser!) Pour que les traits et les remplissages de votre tracé deviennent dégradés.

Sur...

Le dégradé radial Canvas est *extrêmement différent* des dégradés radiaux traditionnels.

La définition "officielle" (presque indéchiffrable!) Du dégradé radial de Canvas se trouve au bas de cet article. Ne le regardez pas si vous avez une mauvaise disposition !!

En termes (presque compréhensibles):

- Le dégradé radial comporte 2 cercles: un cercle de "coulée" et un cercle "d'affichage".
- Le cercle de lancement projette la lumière dans le cercle d'affichage.
- Cette lumière est le dégradé.
- La forme de cette lumière dégradée est déterminée par la taille et la position relatives des deux cercles.

La création d'un objet dégradé est un processus en deux étapes:

1. Créez l'objet dégradé lui-même. Lors de la création, vous définissez une ligne sur le canevas où le dégradé va commencer et se terminer. L'objet dégradé est créé avec `var gradient = context.radialLinearGradient` .
2. Ajoutez ensuite 2 (ou plus) couleurs qui composent le dégradé. Cela se fait en ajoutant plusieurs arrêts de couleur à l'objet `gradient.addColorStop` avec `gradient.addColorStop` .

Arguments:

- **centerX1, centerY1, radius1** définit un premier cercle où le dégradé sera affiché.
- **centerX2, centerY2, rayon2** définit un deuxième cercle qui **projette la lumière dégradée** dans le premier cercle.
- **gradientPercentPosition** est un nombre flottant compris entre 0,00 et 1,00 affecté à un arrêt de couleur. Il s'agit essentiellement d'un point de cheminement en pourcentage définissant où cet arrêt de couleur particulier s'applique le long du dégradé.
    - Le dégradé commence au pourcentage 0,00.
    - Le gradient se termine en pourcentage 1,00.
    - *Note technique:* Le terme "pourcentage" n'est pas techniquement correct puisque les valeurs vont de 0,00 à 1,00 plutôt que de 0% à 100%.
- **CssColor** est une couleur CSS assignée à cet arrêt de couleur particulier.

Note latérale: L'objet dégradé n'est pas interne à l'élément Canvas ni à son contexte. C'est un objet JavaScript distinct et réutilisable que vous pouvez attribuer à n'importe quel chemin souhaité. Vous pouvez même utiliser cet objet pour colorer un chemin sur un autre élément Canvas (!)

Les arrêts de couleur sont des points de passage (en pourcentage) le long de la ligne de dégradé. A chaque point de passage de couleur, le dégradé est entièrement (== opaque) coloré avec la couleur qui lui est attribuée. Les points intermédiaires sur la ligne de dégradé entre les arrêts de couleur sont colorés en tant que dégradés de la couleur this et de la couleur précédente.

Conseil important sur les dégradés de la toile!

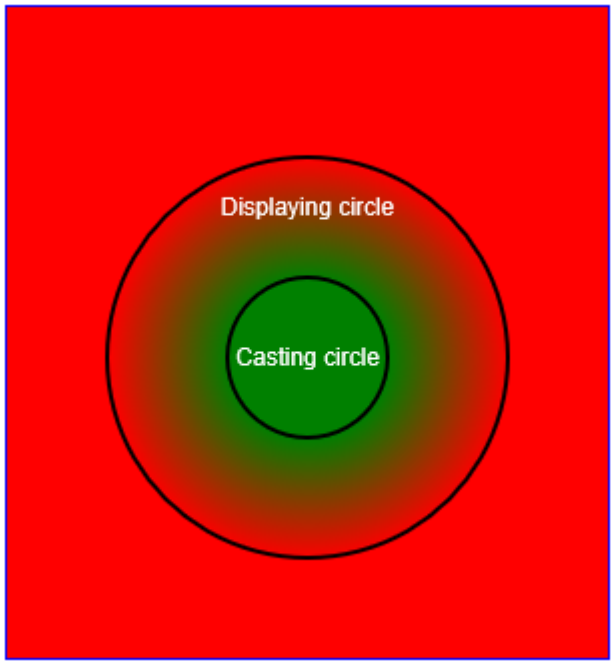
Lorsque vous créez un objet dégradé, la totalité du dégradé radial est "invisible" sur la toile.

Lorsque vous `stroke()` ou `fill()` un chemin, le dégradé invisible est révélé, mais seulement révélé sur ce chemin en cours de trait ou de remplissage.

1. Si vous créez un dégradé radial vert sur rouge comme ceci:

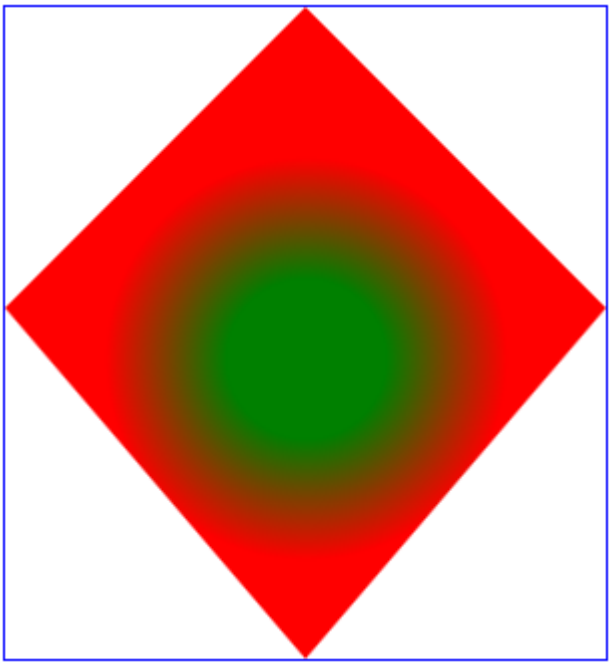
```
// create a radialGradient
var x1=150;
var y1=150;
var x2=280;
var y2=150;
var r1=100;
var r2=120;
var gradient=ctx.createRadialGradient(x1,y1,r1,x2,y2,r2);
gradient.addColorStop(0,'red');
gradient.addColorStop(1,'green');
ctx.fillStyle=gradient;
```

2. Alors Canvas verra "invisible" votre création de dégradé comme ceci:



3. Mais tant que vous n'aurez pas `stroke()` ou `fill()` avec le dégradé, vous ne verrez aucun dégradé sur le canevas.

4. Enfin, si vous tracez ou remplissez un tracé à l'aide du dégradé, le dégradé "invisible" devient visible sur le canevas ... mais uniquement là où le tracé est dessiné.



```
<!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>
```

Les détails officiels effrayants

Qui décide de ce que `createRadialGradient fait?

Le W3C émet les spécifications officielles recommandées que les navigateurs utilisent pour créer l'élément Canvas Html5.

La spécification W3C pour `createRadialGradient` lit comme suit:

Que crée `createRadialGradient`

> `createRadialGradient` … crée efficacement un cône, touché par les deux cercles définis lors de la création du dégradé, avec la partie du cône avant le cercle de départ (0.0) en utilisant la couleur du premier décalage, la partie du cône après le cercle de fin (1.0) en utilisant la couleur du dernier décalage et les zones extérieures au cône intactes par le dégradé (noir transparent).

Comment ça marche en interne

> La `createRadialGradient(x0, y0, r0, x1, y1, r1)` prend six arguments, les trois

premiers représentant le cercle de départ d'origine (x0, y0) et le rayon r0, les trois derniers représentant le cercle de fin d'origine (x1 , y1) et le rayon r1. Les valeurs sont en unités d'espace de coordonnées. Si l'un des r0 ou r1 est négatif, une exception IndexSizeError doit être levée. Sinon, la méthode doit renvoyer un radial CanvasGradient initialisé avec les deux cercles spécifiés.

Les dégradés radiaux doivent être rendus en procédant comme suit:

1. Si x0 = x1 et y0 = y1 et r0 = r1, le dégradé radial ne doit rien peindre. Abandonnez ces étapes.
2. Soit x (ω) = (x1-x0) ω + x0; Soit y (ω) = (y1-y0) ω + y0; Soit r (ω) = (r1-r0) ω + r0 Soit la couleur à ω la couleur à cette position sur le dégradé (avec les couleurs provenant de l'interpolation et de l'extrapolation décrites ci-dessus).
3. Pour toutes les valeurs de ω où r (ω)> 0, en commençant par la valeur de ω la plus proche de l'infini positif et se terminant par la valeur de ω la plus proche de l'infini négatif, tracer la circonférence du cercle de rayon r (ω) x (ω), y (ω)), avec la couleur à ω, mais ne peignant que sur les parties de la toile qui n'ont pas encore été peintes par les cercles précédents dans cette étape pour ce rendu du dégradé.

createPattern (crée un objet de style de chemin)

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

Crée un motif réutilisable (objet).

L'objet peut être affecté à n'importe quel `strokeStyle` et / ou `fillStyle` .

Alors stroke () ou fill () peindront le chemin avec le motif de l'objet.

Arguments:

- **imageObject** est une image qui sera utilisée comme motif. La source de l'image peut être:
    - HTMLImageElement --- un élément img ou une nouvelle image (),
    - HTMLCanvasElement --- un élément canvas,
    - HTMLVideoElement --- un élément vidéo (va saisir l'image vidéo actuelle)
    - ImageBitmap,
    - Goutte.
- **repeat** détermine comment l'imageObject sera répété à travers le canevas (un peu comme un arrière-plan CSS). Cet argument doit être délimité par des guillemets et les valeurs valides sont:
    - "répéter" --- le motif remplira la toile horizontalement et verticalement
    - "repeat-x" --- le motif ne se répète qu'horizontalement (1 ligne horizontale)
    - "repeat-y" --- le motif ne se répète que verticalement (1 ligne verticale)
    - "ne répéter aucun" --- le motif apparaît une seule fois (en haut à gauche)

L'objet pattern est un objet que vous pouvez utiliser (et réutiliser!) Pour créer des traits et des remplissages sur votre chemin.

Note latérale: L'objet pattern n'est pas interne à l'élément Canvas ni à son contexte. C'est un objet JavaScript distinct et réutilisable que vous pouvez attribuer à n'importe quel chemin souhaité. Vous pouvez même utiliser cet objet pour appliquer un motif à un chemin sur un autre élément Canvas (!)

Conseil important sur les motifs de la toile!

Lorsque vous créez un objet de modèle, l'intégralité du canevas est remplie de manière invisible (sous réserve de l'argument de `repeat`).

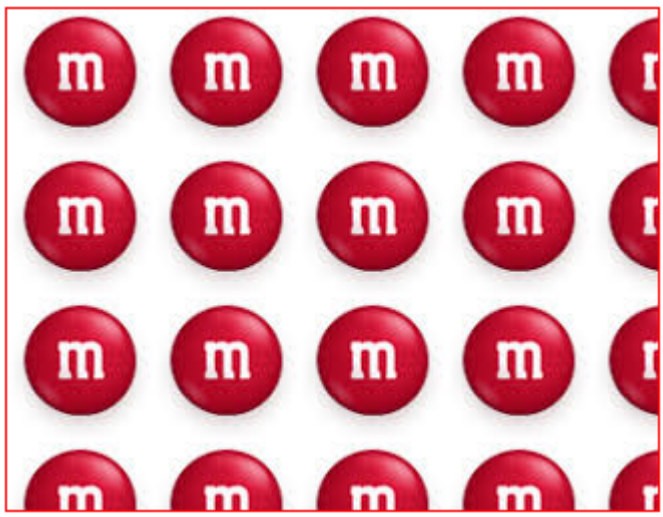
Lorsque vous `stroke()` ou `fill()` un chemin, le motif invisible est révélé, mais ne se révèle que sur ce chemin caressé ou rempli.

1. Commencez avec une image que vous souhaitez utiliser comme motif. Important (!): Assurez-vous que votre image est complètement chargée (en utilisant `patternimage.onload`) avant de tenter de l'utiliser pour créer votre motif.

2. Vous créez un motif comme celui-ci:

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

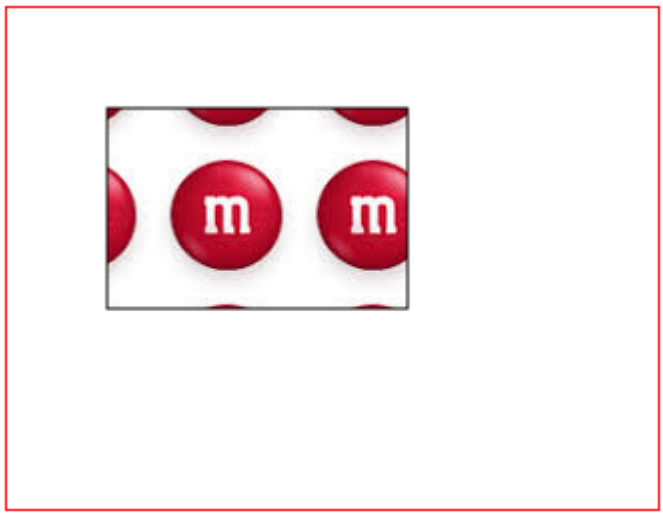
3. Alors Canvas verra "invisiblement" votre création de motif comme ceci:



4. Mais jusqu'à ce que vous `stroke()` ou `fill()` avec le motif, vous ne verrez aucun motif sur le canevas.

5. Enfin, si vous traitez ou remplissez un tracé à l'aide du motif, le motif "invisible" devient

visible sur le canevas ... mais uniquement là où le tracé est dessiné.



```
<!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>
```

stroke (une commande de chemin)

```
context.stroke()
```

Fait en sorte que le périmètre du chemin soit tracé en fonction du `context.strokeStyle` actuel et que le tracé tracé soit dessiné visuellement sur le canevas.

Avant d'exécuter `context.stroke` (ou `context.fill`), le chemin existe en mémoire et n'est pas encore dessiné visuellement sur le canevas.

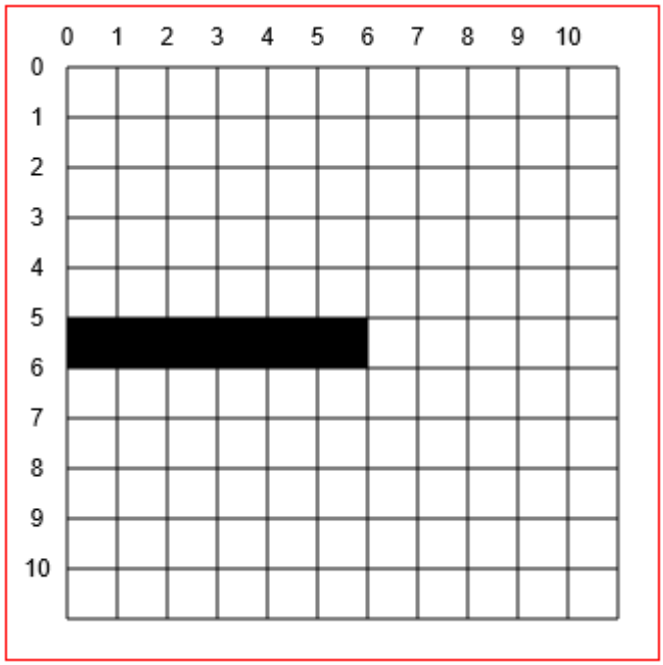
Les traits inhabituels sont dessinés

Considérons cet exemple Path qui dessine une ligne noire de 1 pixel de `[0,5]` à `[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();
```

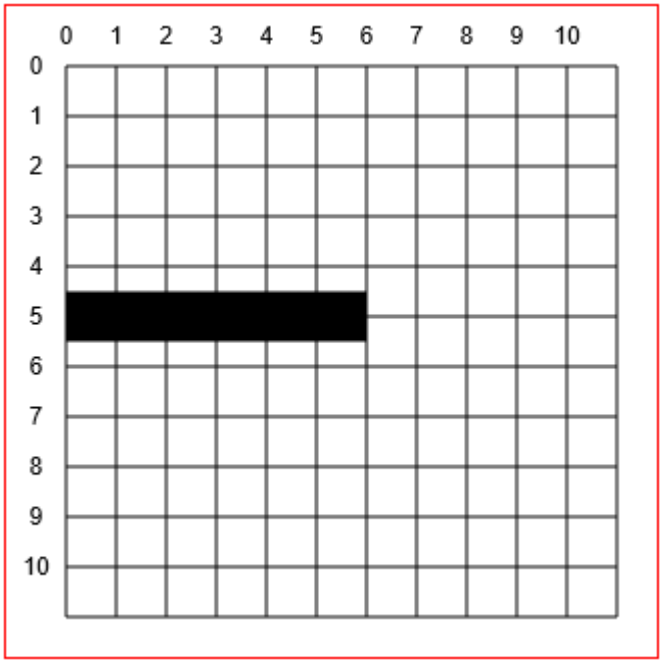
Question: Qu'est-ce que le navigateur dessine sur la toile?

Vous vous attendez probablement à obtenir 6 pixels noirs sur y = 5



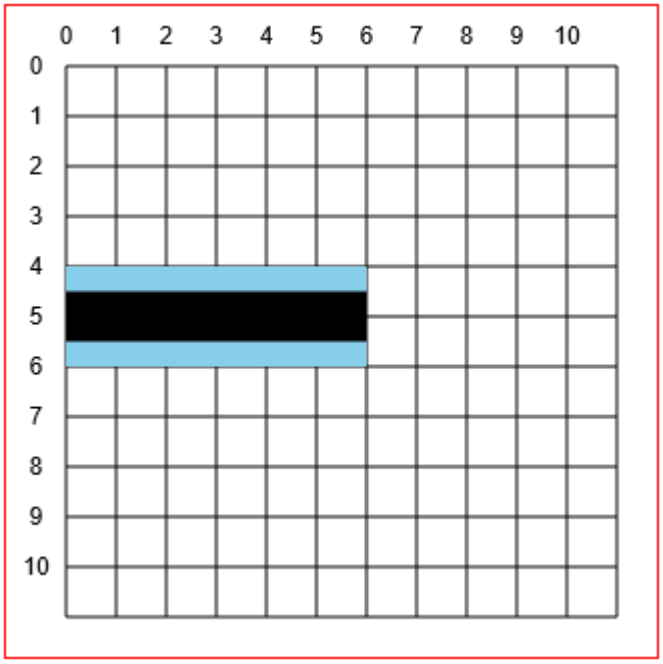
Mais (!) ... Canvas trace toujours des traits à mi-chemin de chaque côté du chemin défini!

Donc puisque la ligne est définie à `y==5.0` Canvas veut tracer la ligne entre `y==4.5` et `y==5.5`

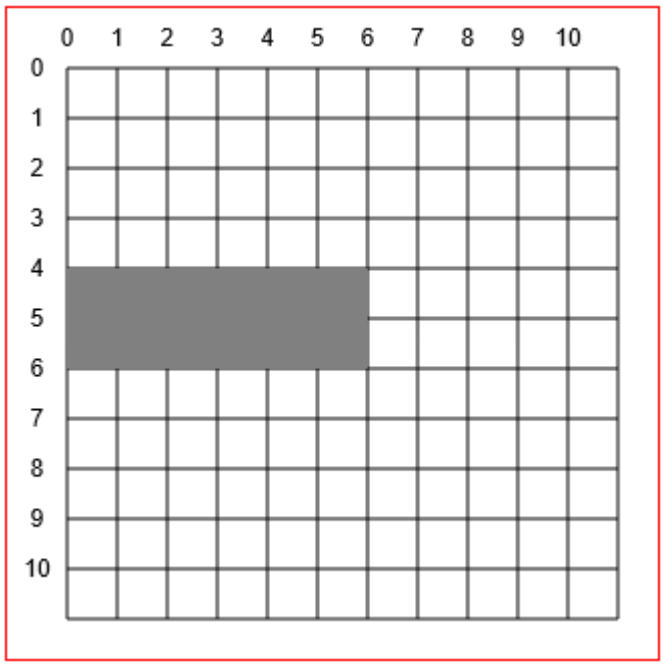


Mais, encore une fois (!) ... L'écran de l'ordinateur ne peut pas dessiner des demi-pixels!

Alors, que faire avec les demi-pixels indésirables (en bleu ci-dessous)?



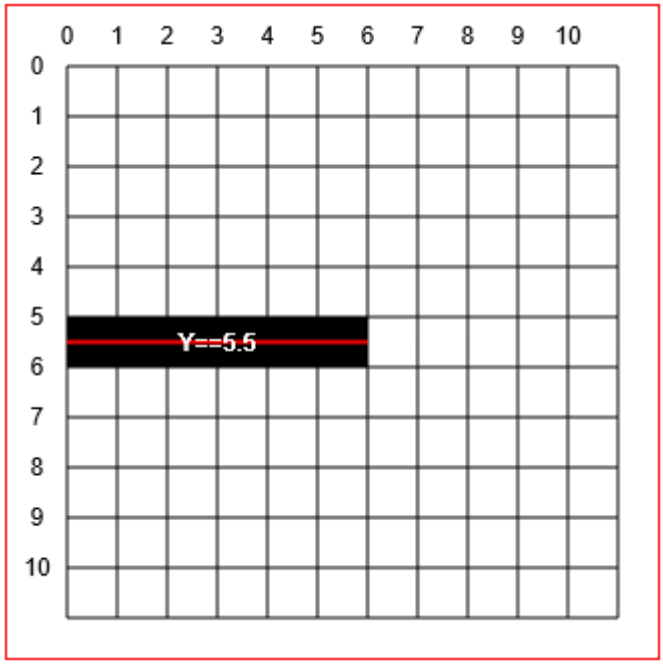
La réponse est que Canvas ordonne en fait à l'affichage de dessiner une ligne de 2 pixels de largeur entre `4.0` et `6.0` . Il colore également la ligne plus claire que le `black` défini. Ce comportement de dessin étrange est "anti-aliasing" et aide Canvas à éviter de dessiner des traits qui semblent irréguliers.



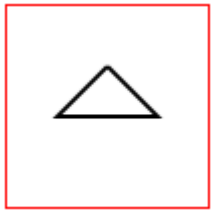
Une astuce de réglage qui fonctionne UNIQUEMENT pour les traits exactement horizontaux et verticaux

Vous pouvez obtenir une ligne noire pleine de 1 pixel en spécifiant que la ligne doit être dessinée sur le demi-pixel:

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



Exemple de code utilisant `context.stroke()` pour dessiner un tracé tracé sur le canevas:



```
<!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>
```

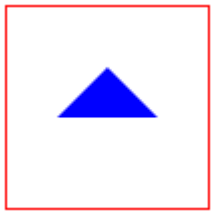
remplir (une commande de chemin)

```
context.fill()
```

Fait en sorte que l'intérieur du chemin soit rempli en fonction du `context.fillStyle` actuel et que le chemin rempli soit dessiné visuellement sur le canevas.

Avant d'exécuter `context.fill` (ou `context.stroke`), le chemin existe en mémoire et n'est pas encore dessiné visuellement sur le canevas.

Exemple de code utilisant `context.fill()` pour dessiner un chemin rempli sur le canevas:



```
<!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 (une commande de chemin)

```
context.clip
```

Limite les futurs dessins à afficher uniquement dans le chemin actuel.

Exemple: Découpez cette image dans un chemin triangulaire

```
<!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>
```

Lire Chemin (syntaxe uniquement) en ligne: https://riptutorial.com/fr/html5-canvas/topic/3241/chemin--syntaxe-uniquement-

Chapitre 4: Collisions et intersections

Examples

Est-ce que 2 cercles entrent en collision?

```
// 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);
}
```

2 rectangles sont-ils en collision?

```
// 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 cercle et un rectangle entrent-ils en collision?

```
// 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);
}
```

2 segments de ligne sont-ils interceptés?

La fonction dans cet exemple renvoie `true` si deux segments de ligne se croisent et `false` si ce n'est pas le cas.

L'exemple est conçu pour les performances et utilise la fermeture pour contenir les variables de travail

```
    // 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.
    })();
```

Exemple d'utilisation

```
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'exemple est facilement modifié pour renvoyer le point d'interception. Remplacez le code entre le `code point A` et la `A end` par

```
if(u1 >= 0 && u1 <= 1){
    return {
        x : p0.x + v1.x * u1,
        y : p0.y + v1.y * u1,
    };
}
```

Ou si vous voulez obtenir le point d'interception sur les lignes, en ignorant le début et la fin des segments de ligne, remplacez le code entre les `code point B` et `B end` par

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

Les deux modifications `{x : xCoord, y : yCoord}` false s'il n'y a pas d'interception ou renvoient le point d'interception en tant que `{x : xCoord, y : yCoord}`

Un segment de ligne et un cercle entrent-ils en collision?

```
// [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 );
}
```

Le segment de droite et le rectangle sont-ils en conflit?

```
// 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);
}
```

2 polygones convexes sont-ils en collision?

Utilisez le théorème de l'axe de séparation pour déterminer si 2 polygones convexes se croisent

LES POLYGONS DOIVENT ÊTRE CONVEXÉS

Attribution: Markus Jarderot @ Comment vérifier l'intersection entre 2 rectangles pivotés?

```
// 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;
```

```
};
```

2 polygones entrent-ils en collision? (les polys concaves et convexes sont autorisés)

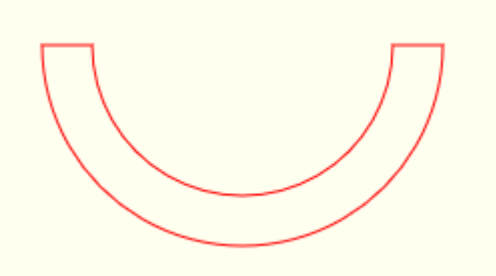
Teste tous les côtés des polygones pour les intersections afin de déterminer si 2 polygones entrent en collision.

```
// 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);
}
```

Est-ce qu'un point X, Y à l'intérieur d'un arc?

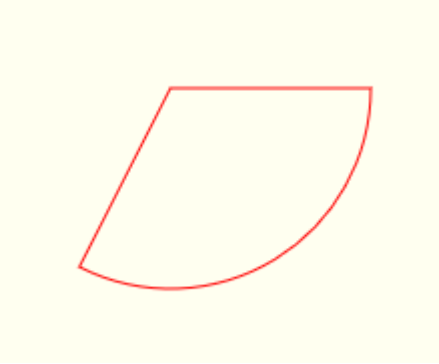
Teste si le point [x, y] se trouve dans un arc fermé.



```
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);
}
```

Est-ce qu'un point X, Y à l'intérieur d'un coin?

Teste si le point [x, y] se trouve dans un coin.



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

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

Un point X, Y est-il dans un cercle?

Teste si un point [x, y] se trouve dans un cercle.

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

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

Un point X, Y est-il dans un rectangle?

Teste si un point [x, y] se trouve dans un rectangle.

```
// 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);
}
```

Lire Collisions et intersections en ligne: https://riptutorial.com/fr/html5-canvas/topic/5017/collisions-et-intersections

Chapitre 5: Compositing

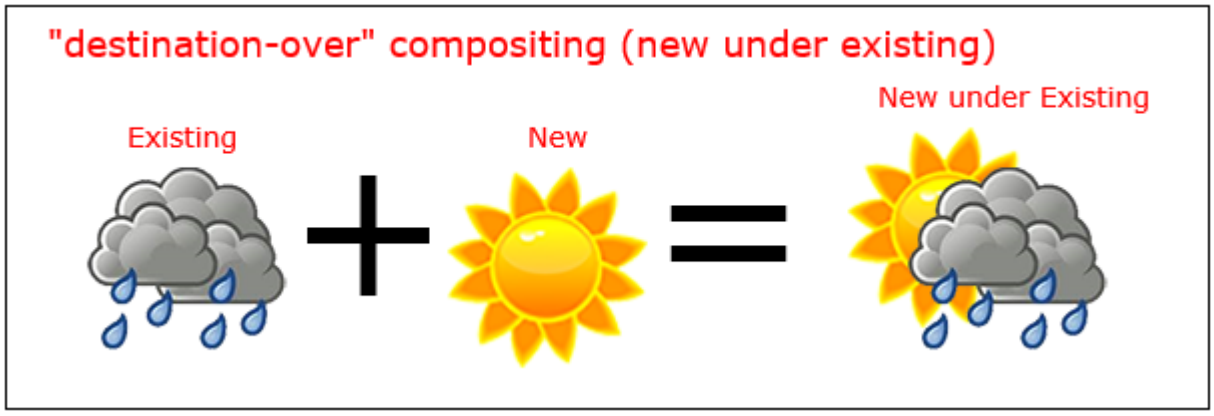
Examples

Dessiner derrière des formes existantes avec "destination-over"

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

"destination-over" compositing place le nouveau dessin *sous les dessins existants* .

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



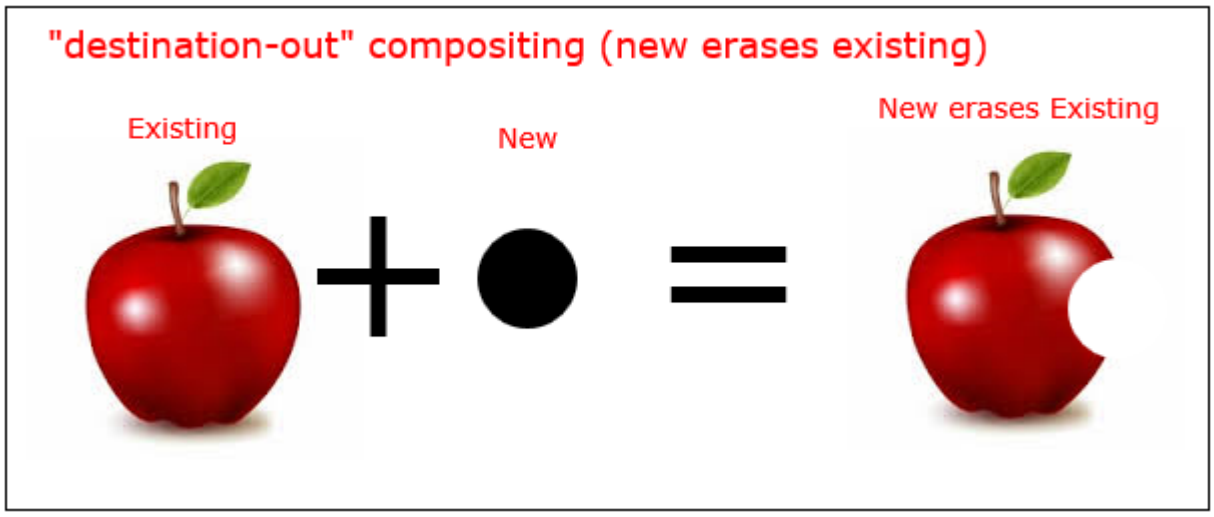
Effacer les formes existantes avec "destination-out"

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

Le compositing "destination-out" utilise de nouvelles formes pour effacer les dessins existants.

La nouvelle forme n'est pas réellement dessinée - elle est simplement utilisée comme un "cookie-cutter" pour effacer les pixels existants.

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

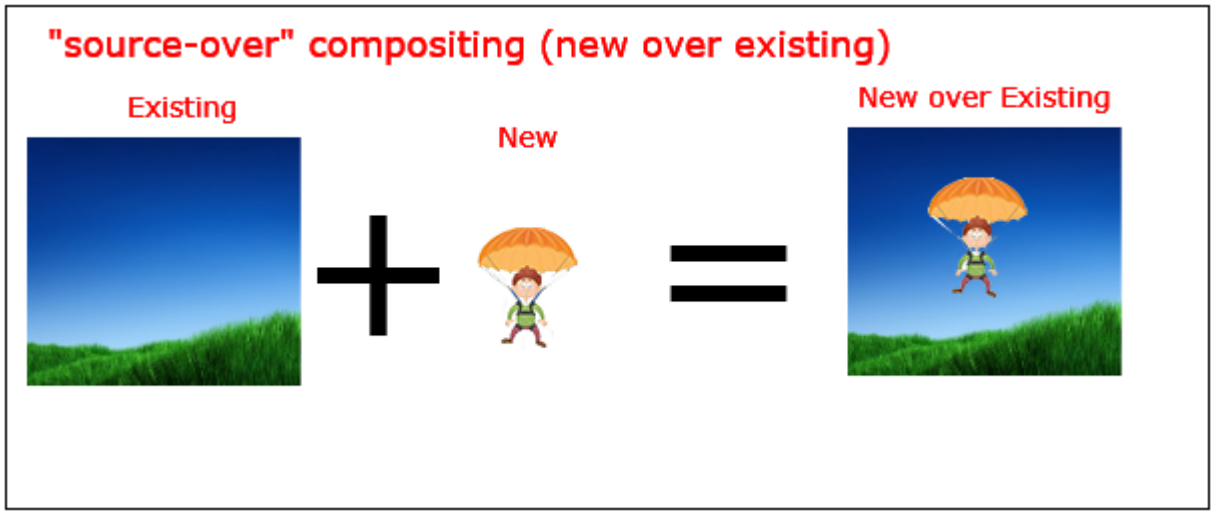


Composition par défaut: les nouvelles formes sont dessinées sur des formes existantes

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

"source-over" compositing **[default]** , place tous les nouveaux dessins sur les dessins existants.

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



Clip images à l'intérieur des formes avec "destination-in"

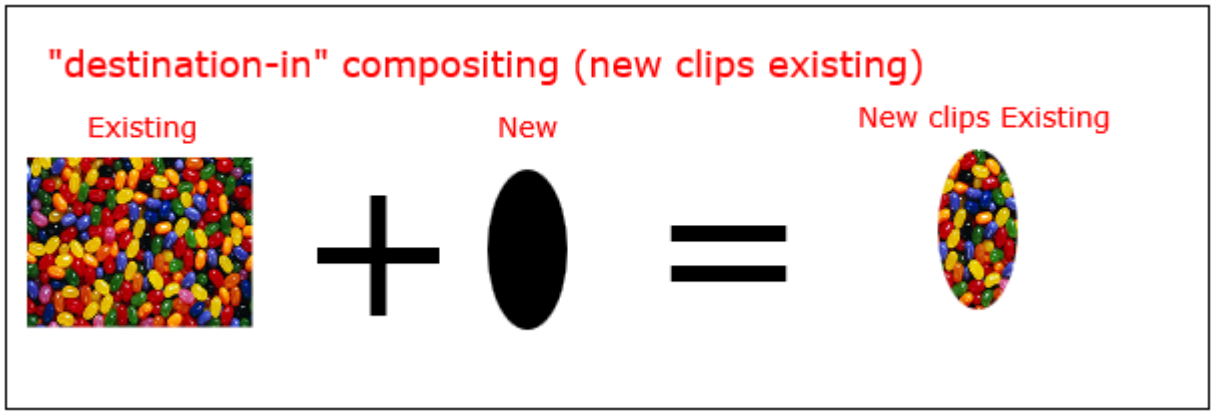
```
context.globalCompositeOperation = "destination-in"
```

"destination-in" composant des clips des dessins existants dans une nouvelle forme.

Remarque: toute partie du dessin existant qui ne fait pas partie du nouveau dessin est effacée.

```
context.drawImage(picture,0,0);
```

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



Clip images à l'intérieur des formes avec "source-in"

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

`source-in` compositing clips de nouveaux dessins à l'intérieur d'une forme existante.

Remarque: toute partie du nouveau dessin qui ne fait pas partie du dessin existant est effacée.

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

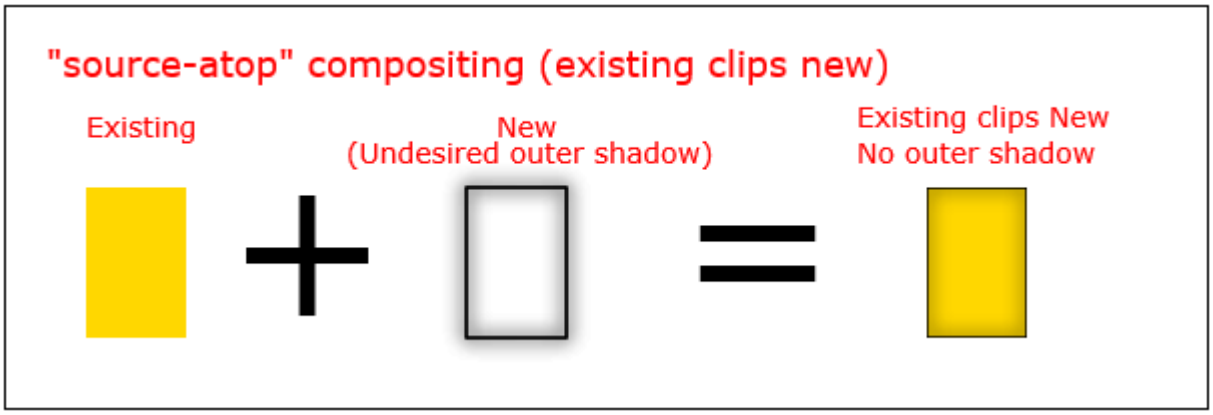
Ombres intérieures avec "source-atop"

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

`source-atop` compositing clips nouvelle image dans une forme existante.

```
// 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);
```



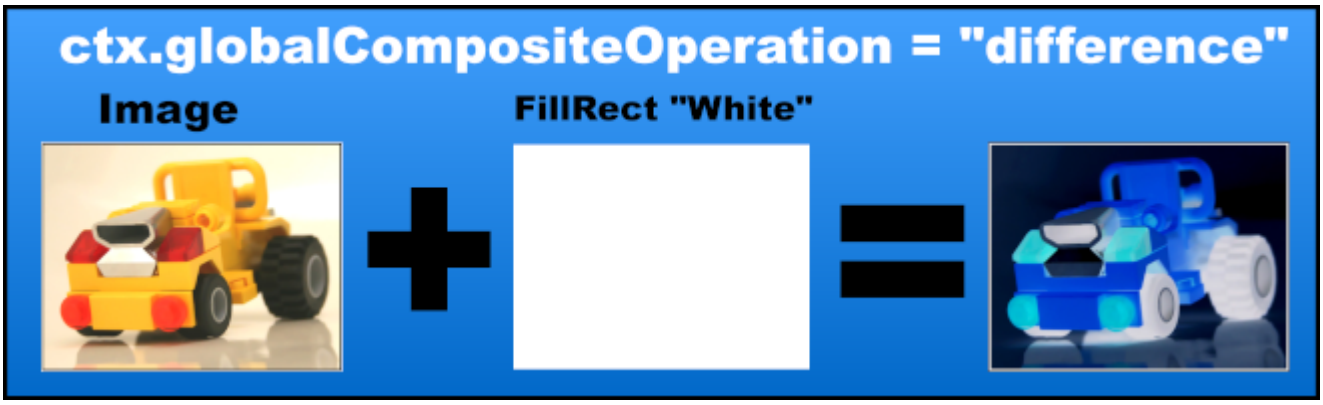
Inverser ou annuler l'image avec "différence"

Rendre un rectangle blanc sur une image avec l'opération composite

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

Le montant de l'effet peut être contrôlé avec le paramètre alpha

```
// 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);
```



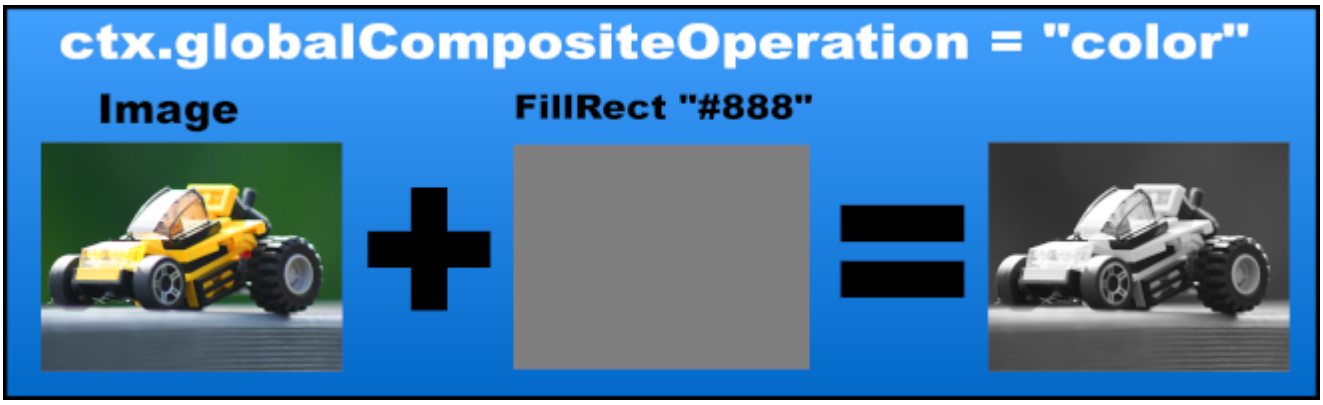
Noir et blanc avec "couleur"

Supprimer la couleur d'une image via

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

Le montant de l'effet peut être contrôlé avec le paramètre alpha

```
// 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);
```



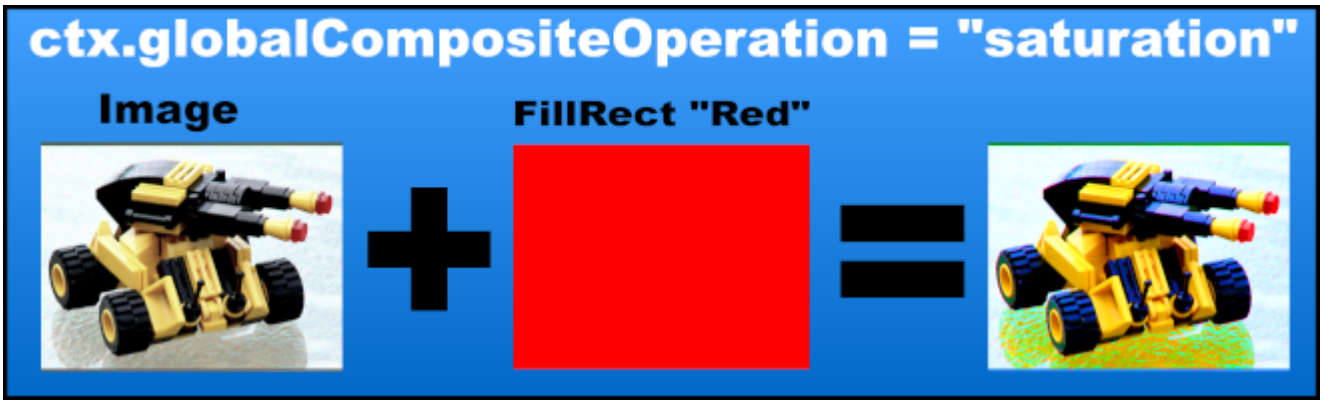
Augmente le contraste des couleurs avec "saturation"

Augmente le niveau de saturation d'une image avec

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

La quantité d'effet peut être contrôlée avec le paramètre alpha ou la quantité de saturation dans la couche de remplissage

```
// 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);
```



Sépia FX avec "luminosité"

Créez un effet sépia coloré avec

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

Dans ce cas, la couleur sépia est rendue d'abord l'image.

La quantité d'effet peut être contrôlée avec le paramètre alpha ou la quantité de saturation dans la couche de remplissage

```
// 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);
```

Changer l'opacité avec "globalAlpha"

```
context.globalAlpha=0.50
```

Vous pouvez modifier l'opacité des nouveaux dessins en définissant le paramètre `globalAlpha` sur une valeur comprise entre 0,00 (entièrement transparent) et 1,00 (totalement opaque).

Le `globalAlpha` par défaut est 1.00 (complètement opaque).

Les dessins existants ne sont pas affectés par `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);
```

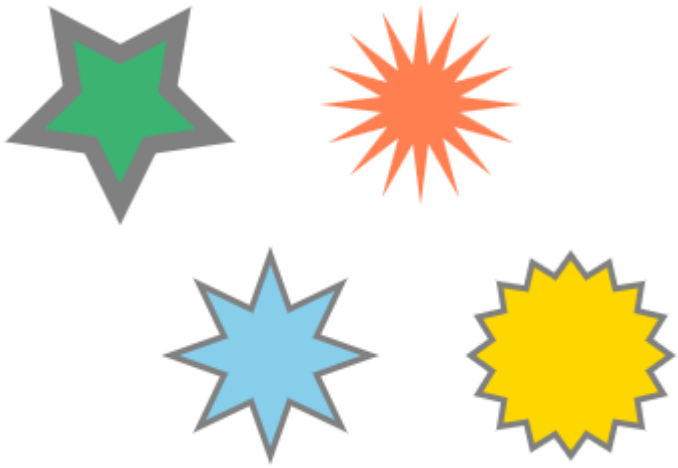
Lire Compositing en ligne: https://riptutorial.com/fr/html5-canvas/topic/5547/compositing

Chapitre 6: Des polygones

Examples

Étoiles

Dessinez des étoiles avec un style flexible (taille, couleurs, nombre de points).



```
// 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()
}
```

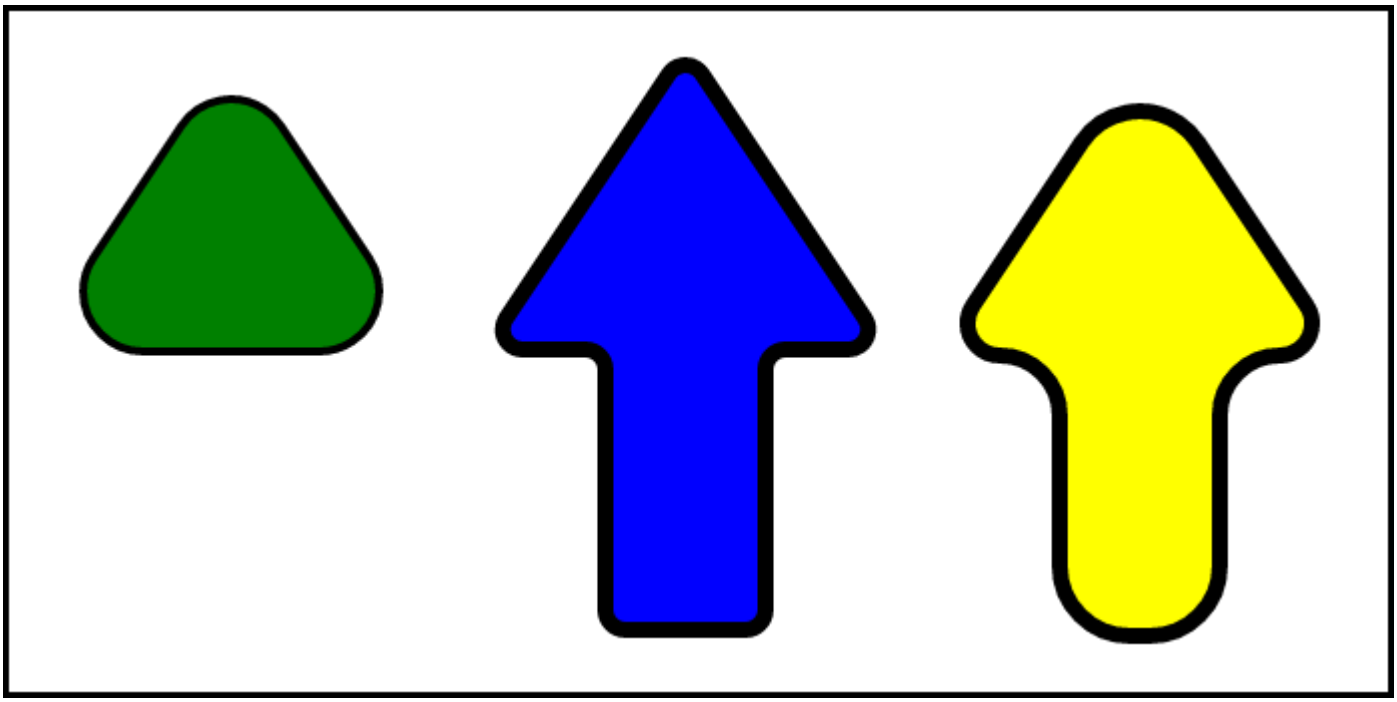
Polygone régulier

Un polygone régulier a tous les côtés égaux en longueur.

```
// 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);
}
```

Rendre un polygone arrondi.

Crée un chemin à partir d'un ensemble de points `[{x:?,y:?},{x:?,y:?},...,{x:?,y:?}]` avec des coins arrondis de rayon. Si l'angle du coin est trop petit pour s'adapter au rayon ou si la distance entre les coins ne permet pas de faire de la place, le rayon des coins est réduit au mieux.



Exemple d'utilisation

```
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();
```

Fonction de rendu

```
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();
}
```

Lire Des polygones en ligne: https://riptutorial.com/fr/html5-canvas/topic/5493/des-polygones

Chapitre 7: Design réactif

Examples

Créer un canevas pleine page réactif

Code de démarrage pour créer et supprimer un canevas pleine page qui répond au redimensionnement d'événements via JavaScript.

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

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

Si vous n'avez plus besoin de la toile, vous pouvez la supprimer en appelant `removeCanvas()`

Une démo de cet exemple à jsfiddle

Coordonnées de la souris après le redimensionnement (ou le défilement)

Les applications Canvas reposent souvent sur l'interaction de l'utilisateur avec la souris, mais

lorsque la fenêtre est redimensionnée, les coordonnées d'événement de la souris sur lesquelles repose le canevas sont probablement modifiées car le redimensionnement entraîne un décalage du canevas par rapport à la fenêtre. Ainsi, la conception réactive nécessite que la position de décalage de la zone de dessin soit recalculée lorsque la fenêtre est redimensionnée - et également recalculée lorsque la fenêtre défile.

Ce code écoute les événements de redimensionnement de fenêtre et recalcule les décalages utilisés dans les gestionnaires d'événement de souris:

```
// 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);
    // ...
}
```

Animations sur toile réactives sans événements de redimensionnement.

Les événements de redimensionnement de la fenêtre peuvent se déclencher en réponse au mouvement du périphérique de saisie de l'utilisateur. Lorsque vous redimensionnez un canevas, il est automatiquement effacé et vous êtes obligé de restituer le contenu. Pour les animations, vous faites ceci chaque image via la fonction de boucle principale appelée par `requestAnimationFrame` qui fait de son mieux pour maintenir le rendu synchronisé avec le matériel d'affichage.

Le problème avec l'événement de redimensionnement est que lorsque la souris est utilisée pour redimensionner la fenêtre, les événements peuvent être déclenchés beaucoup plus rapidement que le taux standard de 60 images par seconde du navigateur. Lorsque l'événement de redimensionnement quitte le tampon de retour, celui-ci n'est plus synchronisé avec le périphérique d'affichage, ce qui peut entraîner des effets de cisaillement et d'autres effets négatifs. Il y a également beaucoup d'allocation et de libération de mémoire inutiles qui peuvent influer davantage sur l'animation lorsque le CPG se nettoie quelque temps après.

Evénement de redimensionnement annoncé

Une manière courante de gérer les taux de déclenchement élevés de l'événement de redimensionnement consiste à éliminer l'événement de redimensionnement.

```
// 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'exemple ci-dessus retarde le redimensionnement du canevas jusqu'à 100 ms après l'événement de redimensionnement. Si, au cours de cette période, d'autres événements de redimensionnement sont déclenchés, le délai d'expiration du redimensionnement existant est annulé et un nouveau planifié. Cela consomme efficacement la plupart des événements de redimensionnement.

Il y a toujours des problèmes, le plus notable est le délai entre le redimensionnement et la visualisation du canevas redimensionné. La réduction du temps de réponse améliore cela, mais le redimensionnement est toujours en décalage avec le périphérique d'affichage. Vous avez également le rendu de la boucle d'animation principale sur un canevas mal ajusté.

Plus de code peut réduire les problèmes! Plus de code crée également ses propres problèmes.

Simple et le meilleur redimensionnement

Après avoir essayé de nombreuses manières différentes de lisser le redimensionnement de la toile, du complexe absurde au simple fait d'ignorer le problème (qui se soucie de toute façon?), Je suis tombé sur un ami de confiance.

Kiss est quelque chose la plupart des programmeurs doivent être conscients de **((K** EEP **I** t **S S** tupid œuvre) *La bête se réfère à moi de ne pas avoir pensé il y a quelques années.)* Et il se trouve la meilleure solution est la plus simple de tous.

Il suffit de redimensionner le canevas à partir de la boucle d'animation principale. Il reste synchronisé avec le périphérique d'affichage, il n'y a pas de rendu inutile et la gestion des ressources est au minimum possible tout en maintenant une fréquence d'images maximale. Vous n'avez pas non plus besoin d'ajouter un événement de redimensionnement à la fenêtre ou des fonctions de redimensionnement supplémentaires.

Vous ajoutez le redimensionnement où vous effaceriez normalement le canevas en vérifiant si la taille du canevas correspond à la taille de la fenêtre. Si ce n'est pas le redimensionner.

```
// 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);
}
```

Lire Design réactif en ligne: https://riptutorial.com/fr/html5-canvas/topic/5495/design-reactif

Chapitre 8: Effacer l'écran

Syntaxe

- void clearRect (x, y, largeur, hauteur)
- ImageData createImageData (width, height)

Remarques

Aucune de ces méthodes ne produira des pixels transparents si le contexte a été créé avec le paramètre `alpha: false` .

Examples

Rectangles

Vous pouvez utiliser la méthode `clearRect` pour effacer toute section rectangulaire du canevas.

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

> **Remarque:** `clearRect` dépend de la matrice de transformation.

Pour faire face à cela, il est possible de réinitialiser la matrice de transformation avant d'effacer le canevas.

```
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
```

> **Remarque:** `ctx.save` et `ctx.restore` ne sont requis que si vous souhaitez conserver l'état du contexte 2D du canevas. Dans certaines situations, la sauvegarde et la restauration peuvent être lentes et généralement évitées si elles ne sont pas requises.

Données d'image brutes

Il est possible d'écrire directement sur les données d'image rendues en utilisant `putImageData` . En créant de nouvelles données d'image puis en les affectant au canevas, vous effacez l'écran entier.

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

Remarque: `putImageData` n'est pas affecté par les transformations appliquées au contexte. Il écrira des données directement dans la région de pixels rendue.

Formes complexes

Il est possible d'effacer des régions de formes complexes en modifiant la propriété `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';
```

Effacer la toile avec dégradé.

Plutôt que d'utiliser `clearRect` qui rend tous les pixels transparents, vous pouvez souhaiter un arrière-plan.

Effacer avec un dégradé

```
// 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);
```

Ceci est environ deux fois moins rapide que 0,008 `0.008ms` comme `0.008ms` à 0,004 `0.004ms` mais les 4 millions de secondes ne devraient pas avoir d'impact négatif sur une animation en temps réel. (Les temps varient considérablement en fonction de l'appareil, de la résolution, du navigateur et de la configuration du navigateur. Les temps sont uniquement pour comparaison)

Effacer la toile en utilisant l'opération composite

Effacer le canevas en utilisant l'opération de compositing. Cela `clearRect()` le canevas indépendamment des transformations mais n'est pas aussi rapide que `clearRect()` .

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

tout ce qui sera dessiné ensuite effacera le contenu précédent.

Lire Effacer l'écran en ligne: https://riptutorial.com/fr/html5-canvas/topic/5245/effacer-l-ecran

Chapitre 9: Glisser le chemin Formes et images sur la toile

Examples

Comment les formes et les images VRAIMENT (!) "Bougent" sur la toile

Un problème: Canvas ne mémorise que les pixels, pas les formes ou les images

Ceci est une image d'un ballon de plage circulaire, et bien sûr, vous ne pouvez pas faire glisser la balle autour de l'image.

Cela peut vous surprendre, tout comme une image, si vous dessinez un cercle sur une toile, vous ne pouvez pas faire glisser ce cercle autour de la toile. C'est parce que la toile ne se souviendra plus où elle a dessiné le cercle.

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

Ce que la toile ne sait pas ...

- ... où vous avez dessiné le cercle (il ne sait pas x, y = [20,30]).
- ... la taille du cercle (il ne connaît pas le rayon = 15).
- ... la couleur du cercle. (il ne sait pas que le cercle est bleu).

Ce que la toile sait ...

Canvas connaît la couleur de chaque pixel sur sa surface de dessin.

Le canevas peut vous dire qu'à x, y == [20,30] il y a un pixel bleu, mais il ne sait pas si ce pixel bleu fait partie d'un cercle.

Qu'est-ce que cela signifie...

Cela signifie que tout ce qui est dessiné sur la toile est permanent: immuable et immuable.

- Le canevas ne peut pas déplacer le cercle ou redimensionner le cercle.
- La toile ne peut pas recolorer le cercle ou effacer le cercle.
- La toile ne peut pas dire si la souris survole le cercle.
- La toile ne peut pas dire si le cercle est en collision avec un autre cercle.
- Le canevas ne peut pas laisser un utilisateur faire glisser le cercle autour du canevas.

Mais Canvas peut donner l'ILLUSION du mouvement

La toile peut donner l' **illusion d'un mouvement** en effaçant continuellement le cercle et en le redessinant dans une position différente. En redessinant le canevas plusieurs fois par seconde, l'œil est trompé en voyant le cercle se déplacer sur la toile.

- Effacer la toile
- Mettre à jour la position du cercle
- Redessine le cercle dans sa nouvelle position
- Répétez, répétez, répétez ...

Ce code donne l' **illusion du mouvement** en redessinant continuellement un cercle dans de nouvelles positions.

```
// 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);
}
```

Faire glisser des cercles et des rectangles autour de la toile

Qu'est ce qu'une "forme"?

Vous enregistrez généralement vos formes en créant un objet JavaScript "shape" représentant chaque forme.

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

Bien sûr, vous ne sauvegardez pas vraiment les formes. Au lieu de cela, vous enregistrez la définition de la manière de dessiner les formes.

Ensuite, placez chaque objet de forme dans un tableau pour pouvoir vous y référer facilement.

```
// 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'} );
```

Utilisation des événements de la souris pour faire glisser

Faire glisser une forme ou une image nécessite de répondre à ces événements de la souris:

Sur mousedown:

Testez si une forme est sous la souris. Si une forme est sous la souris, l'utilisateur a l'intention de faire glisser cette forme. Conservez donc une référence à cette forme et définissez un `isDragging` true / false `isDragging` indiquant qu'un glisser est en cours.

Sur mousemove:

Calculez la distance de déplacement de la souris depuis le dernier événement `mousemove` et modifiez la position de la forme déplacée sur cette distance. Pour modifier la position de la forme, vous modifiez les propriétés de position `x,y` dans l'objet de cette forme.

Sur mouseup ou mouseout:

L'utilisateur a l'intention d'arrêter l'opération de glisser, donc effacez le drapeau "isDragging". Le glisser est terminé.

Démo: Faire glisser des cercles et des

rectangles sur la toile

Cette démo entraîne des cercles et des rectangles sur la toile en répondant aux événements de la souris et en donnant l'illusion d'un mouvement en nettoyant et en redessinant.

```
// 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);
        }
    }
}
```

Faire glisser des formes irrégulières autour de la toile

La plupart des dessins sur toile sont rectangulaires (rectangles, images, blocs de texte) ou circulaires (cercles).

Les cercles et les rectangles ont des tests mathématiques pour vérifier si la souris est à l'intérieur. Cela rend les tests de cercles et de rectangles faciles, rapides et efficaces. Vous pouvez "tester" des centaines de cercles ou de rectangles en une fraction de seconde.

Vous pouvez également faire glisser des formes irrégulières. Mais les formes irrégulières ne comportent pas de test mathématique rapide. Heureusement, les formes irrégulières ont un test de réussite intégré pour déterminer si un point (souris) se trouve dans la forme: `context.isPointInPath` . Bien `isPointInPath` fonctionne bien, il n'est pas aussi efficace que les tests de mathématiques pur - il est souvent jusqu'à 10 fois plus lent que les tests de mathématiques

purs.

Lorsque vous utilisez `isPointInPath` , vous devez `isPointInPath` "redéfinir" le chemin testé immédiatement avant d'appeler `isPointInPath` . "Redéfinir" signifie que vous devez émettre les commandes de dessin du chemin (comme ci-dessus), mais vous n'avez pas besoin de marquer () ou de remplir () le chemin avant de le tester avec `isPointInPath` . De cette façon, vous pouvez tester les chemins précédemment dessinés sans avoir à écraser (contour / remplissage) les chemins précédents sur le canevas lui-même.

La forme irrégulière n'a pas besoin d'être aussi commune que le triangle quotidien. Vous pouvez également tester tous les chemins irréguliers.

Cet exemple annoté montre comment faire glisser des formes de chemins irréguliers ainsi que des cercles et des rectangles:

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

Faire glisser des images autour de la toile

Voir cet exemple pour une explication générale du glissement des formes autour du canevas.

Cet exemple annoté montre comment faire glisser des images autour du canevas

```
// 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);
        }
    }
}
```

Lire Glisser le chemin Formes et images sur la toile en ligne: https://riptutorial.com/fr/html5-canvas/topic/5318/glisser-le-chemin-formes-et-images-sur-la-toile

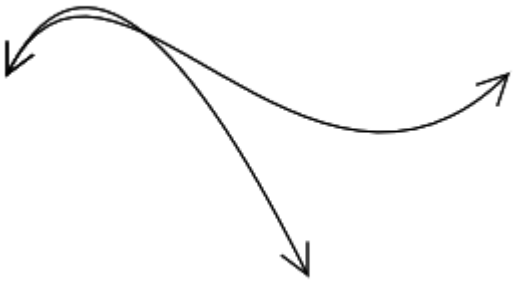
Chapitre 10: Graphiques et diagrammes

Examples

Ligne avec des pointes de flèche

```
// 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);
}
```

Courbe de Bézier cubique et quadratique avec pointes de flèches

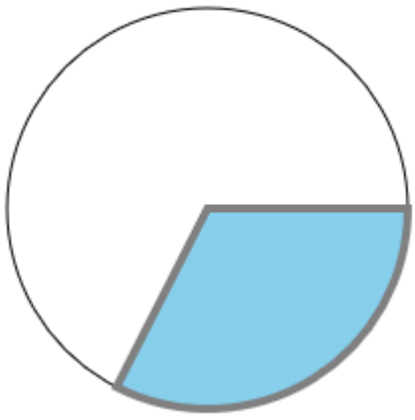


```
// Usage:
var p0={x:50,y:100};
var p1={x:100,y:0};
var p2={x:200,y:200};
var p3={x:300,y:100};

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);
}
```

Coin

Le code dessine uniquement le coin ... cercle dessiné ici pour la perspective seulement.



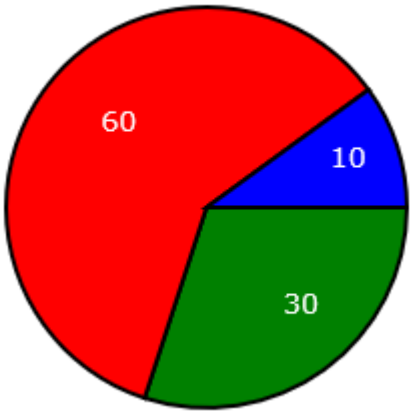
```
// Usage
var wedge={
    cx:150, cy:150,
    radius:100,
    startAngle:0,
    endAngle:Math.PI*.65
}
```

```
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();
}
```

Arc à la fois de remplissage et de course

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

Graphique à secteurs avec démo




```
<!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>
```

Lire Graphiques et diagrammes en ligne: https://riptutorial.com/fr/html5-canvas/topic/5492/graphiques-et-diagrammes

Chapitre 11: Images

Examples

Recadrage d'image avec toile

Cet exemple montre une fonction de recadrage d'image simple qui prend une image et des coordonnées de recadrage et renvoie l'image recadrée.

```
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;
}
```

Utiliser

```
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 toile Tained

Lorsque vous ajoutez du contenu provenant de sources extérieures à votre domaine ou du système de fichiers local, le canevas est marqué comme corrompu. Tenter d'accéder aux données de pixel ou de convertir en dataURL génère une erreur de sécurité.

```
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
}
```

Cet exemple est juste un bout pour attirer quelqu'un avec une compréhension détaillée élaborée.

"Context.drawImage" n'affiche-t-il pas l'image sur le canevas?

Assurez-vous que votre objet image est entièrement chargé avant d'essayer de le dessiner sur le canevas avec `context.drawImage` . Sinon, l'image ne sera pas affichée.

En JavaScript, les images ne sont pas chargées immédiatement. Au lieu de cela, les images sont chargées de manière asynchrone et pendant le temps qu'elles prennent pour charger JavaScript continue à exécuter tout code qui suit `image.src` . Cela signifie que `context.drawImage` peut être exécuté avec une image vide et n'affiche donc rien.

Exemple: assurez-vous que l'image est complètement chargée avant d'essayer de la dessiner avec .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
}
```

Exemple de chargement de plusieurs images avant d'essayer de dessiner avec l'une d'elles

Il y a plus de chargeurs d'images fonctionnels, mais cet exemple illustre comment le faire

```
// 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);
}
```

Mise à l'échelle de l'image pour l'ajuster ou la remplir.

Mise à l'échelle pour s'adapter

Signifie que l'image entière sera visible mais qu'il peut y avoir un espace vide sur les côtés ou en haut et en bas si l'image n'est pas le même aspect que le canevas. L'exemple montre l'image

mise à l'échelle pour s'adapter. Le bleu sur les côtés est dû au fait que l'image n'a pas le même aspect que la toile.

Mise à l'échelle pour remplir

Signifie que l'image est mise à l'échelle de sorte que tous les pixels du canevas soient couverts par l'image. Si l'aspect de l'image n'est pas le même que celui du canevas, certaines parties de l'image seront tronquées. L'exemple montre l'image mise à l'échelle pour remplir. Notez que le haut et le bas de l'image ne sont plus visibles.

Exemple d'échelle pour s'adapter

```
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);
}
```

Exemple d'échelle à remplir

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

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

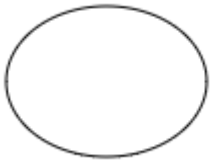
La seule différence entre les deux fonctions est d'obtenir la balance. L'ajustement utilise l'échelle d'ajustement minimum pour que le remplissage utilise l'échelle d'ajustement maximale.

Lire Images en ligne: https://riptutorial.com/fr/html5-canvas/topic/3210/images

Chapitre 12: Les chemins

Examples

Ellipse



Remarque: Les navigateurs sont en train d'ajouter une commande de dessin `context.ellipse` intégrée, mais cette commande n'est pas adoptée universellement (notamment pas dans IE). Les méthodes ci-dessous fonctionnent dans tous les navigateurs.

Dessinez une ellipse en fonction de la coordonnée supérieure gauche souhaitée:

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

Dessinez une ellipse en fonction de la coordonnée souhaitée du point central:

```
// 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();
```

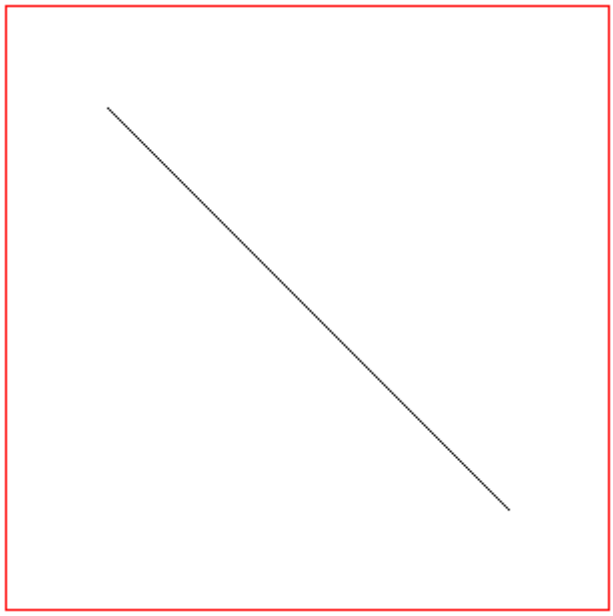
```
    var x = cx + radius * Math.cos(0);
    var y = cy - ratio * radius * Math.sin(0);
    ctx.lineTo(x,y);

    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();
}
```

Ligne sans flou

Lorsque Canvas dessine une ligne, il ajoute automatiquement un anti-aliasing pour soigner visuellement les "irrégularités". Le résultat est une ligne moins déchiquetée mais plus floue.

Cette fonction trace une ligne entre 2 points sans anti-aliasing en utilisant l' algorithme Bresenham's_line . Le résultat est une ligne nette sans enchevêtrement.

Remarque importante: *Cette méthode pixel par pixel est une méthode de dessin beaucoup plus lente que* `context.lineTo` *.*



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

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

```
    y = Math.floor(y);
    // 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
}
```

Lire Les chemins en ligne: https://riptutorial.com/fr/html5-canvas/topic/5133/les-chemins

Chapitre 13: Manipulation de pixels avec "getImageData" et "putImageData"

Examples

Introduction à "context.getImageData"

Html5 Canvas vous permet d'extraire et de modifier la couleur de tout pixel de la toile.

Vous pouvez utiliser la manipulation de pixels de Canvas pour:

- Créez un sélecteur de couleur pour une image ou sélectionnez une couleur sur une roue chromatique.
- Créez des filtres d'image complexes comme le flou et la détection des contours.
- Recolorez n'importe quelle partie d'une image au niveau du pixel (si vous utilisez HSL, vous pouvez même recolorer une image tout en conservant l'important Lighting & Saturation afin que le résultat ne ressemble pas à quelqu'un qui a giflé de la peinture sur l'image). Remarque: Canvas a désormais Blend Compositing qui peut également recolorer une image dans certains cas.
- "Knockout" l'arrière-plan autour d'une personne / d'un objet dans une image,
- Créez un outil de peinture pour détecter et remplir une partie d'une image (par exemple, changez la couleur d'un pétale de fleur sur lequel l'utilisateur a cliqué de vert à jaune).
- Examiner une image pour le contenu (par exemple, la reconnaissance faciale).

Problèmes courants:

- Pour des raisons de sécurité, `getImageData` est désactivé si vous avez dessiné une image provenant d'un autre domaine que la page Web elle-même.
- `getImageData` est une méthode relativement coûteuse car elle crée un grand tableau de données en pixels et n'utilise pas le GPU pour aider ses efforts. Remarque: Canvas a maintenant une composition de fusion qui peut faire la même manipulation de pixels que `getImageData` .
- Pour les images .png, `getImageData` peut ne pas rapporter exactement les mêmes couleurs que dans le fichier .png d'origine car le navigateur est autorisé à effectuer une correction gamma et une alpha-prémultiplication lors du dessin d'images sur le canevas.

Obtenir des couleurs de pixel

Utilisez `getImageData` pour récupérer les couleurs des pixels pour tout ou partie de votre contenu de canevas.

La méthode `getImageData` renvoie un objet `imageData`

L'objet `imageData` a une propriété `.data` contenant les informations de couleur des pixels.

La propriété `data` est un `Uint8ClampedArray` contenant les données de couleur Rouge, Vert, Bleu et

Alpha (opacité) pour tous les pixels demandés.

```
// 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;
```

Vous pouvez obtenir la position de tout pixel [x, y] dans `data` tableau de `data` comme ceci:

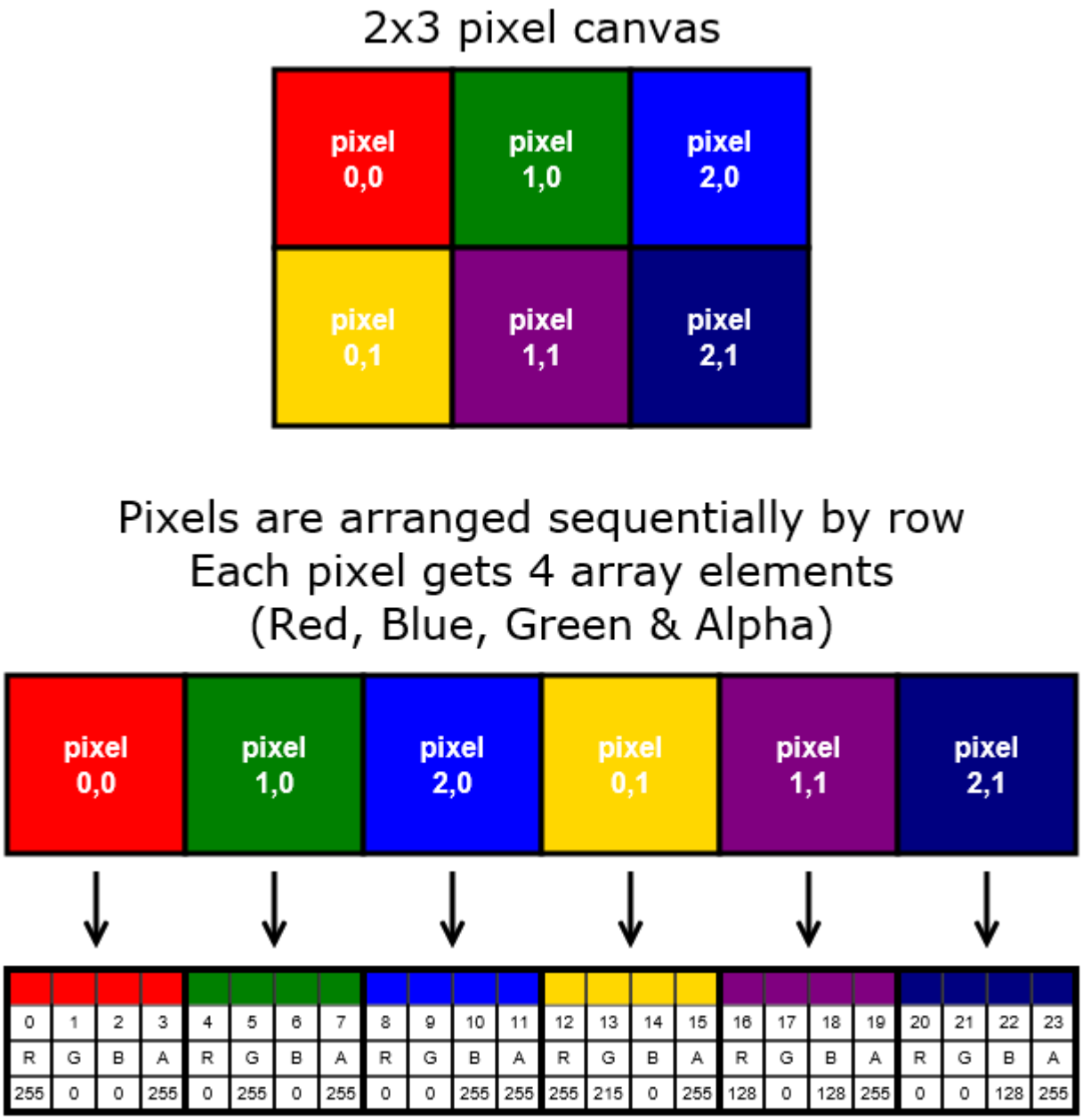
```
// the data[] array position for pixel [x,y]
var n = y * canvas.width + x;
```

Et puis vous pouvez récupérer les valeurs rouges, vertes, bleues et alpha de ce pixel comme ceci:

```
// 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];
```

Une illustration montrant comment le tableau de données de pixels est structuré

`context.getImageData` est illustré ci-dessous pour un petit tableau au format 2x3 pixels:



Lire Manipulation de pixels avec "getImageData" et "putImageData" en ligne: https://riptutorial.com/fr/html5-canvas/topic/5573/manipulation-de-pixels-avec--getimagedata--et--putimagedata-

Chapitre 14: Naviguer le long d'un chemin

Examples

Trouver des points le long d'une courbe de Bézier cubique

Cet exemple trouve un tableau de points espacés approximativement le long d'une courbe de Bézier cubique.

Il décompose les segments de chemin créés avec `context.bezierCurveTo` en points situés le long de cette courbe.

```
// 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);
}
```

Trouver des points le long d'une courbe quadratique

Cet exemple trouve un tableau de points espacés de manière à peu près égale sur une courbe quadratique.

Il décompose les segments de chemin créés avec `context.quadraticCurveTo` en points situés le long de cette courbe.

```
// 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);
}
```

Trouver des points le long d'une ligne

Cet exemple trouve un tableau de points espacés de manière à peu près égale le long d'une ligne.

Il décompose les segments de chemin créés avec `context.lineTo` en points situés le long de cette ligne.

```
// 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);
}
```

Recherche de points sur un chemin entier contenant des courbes et des lignes

Cet exemple trouve un tableau de points espacés de façon à peu près égale sur tout un chemin.

Il décompose tous les segments Path créés avec `context.lineTo` , `context.quadraticCurveTo` et / ou `context.bezierCurveTo` en points situés le long de ce chemin.

Usage

```
// 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 qui calcule automatiquement les points le long du chemin

Ce code modifie les commandes de dessin de Canvas Context afin que les commandes non seulement dessinent la ligne ou la courbe, mais créent également un tableau de points tout au long du chemin:

- beginPath,
- déménager à,
- lineTo,
- quadraticCurveTo,
- bezierCurveTo.

Note importante!

Ce code modifie les fonctions de dessin réelles du contexte. Lorsque vous avez fini de tracer des points le long du chemin, vous devez appeler les commandes `stopPlottingPathCommands` fournies pour `stopPlottingPathCommands` les fonctions de dessin du contexte dans leur état non modifié.

Le but de ce contexte modifié est de vous permettre de «brancher» le calcul du tableau de points dans votre code existant sans avoir à modifier vos commandes de dessin de chemin existantes. Mais, vous n'avez pas besoin d'utiliser ce contexte modifié - vous pouvez appeler séparément les fonctions individuelles qui décomposent une ligne, une courbe quadratique et une courbe de Bézier cubique, puis concaténer manuellement ces tableaux de points individuels en un seul tableau de points pour le chemin entier.

Vous récupérez une copie du tableau de points résultant à l'aide de la fonction `getPathPoints` fournie.

Si vous tracez plusieurs chemins avec le contexte modifié, le tableau de points contiendra un ensemble unique de points concaténés pour tous les chemins multiples dessinés.

Si, au contraire, vous souhaitez obtenir des tableaux de points séparés, vous pouvez récupérer le tableau actuel avec `getPathPoints` , puis effacer ces points du tableau avec la fonction `clearPathPoints` fournie.

```
// 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;
    }
}
```

Une démo complète:

```
// 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);
}
```

Longueur d'une courbe quadratique

Étant donné les 3 points d'une courbe quadratique, la fonction suivante renvoie la longueur.

```
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);
}
```

Dérivé de la fonction de bézier quadratique F (t) = a * (1 - t) ^{2} + 2 * b * (1 - t) * t + c * t ^{2}

Split bezier courbes à la position

Cet exemple divise les courbes cubiques et bezier en deux.

La fonction `splitCurveAt` divise la courbe à la `position` où `0.0` = début, `0.5` = milieu et `1` = fin. Il peut diviser les courbes quadratiques et cubiques. Le type de courbe est déterminé par le dernier argument `x4` . Si non `undefined` ou `null` alors il suppose que la courbe est cubique sinon la courbe est un quadratique

Exemple d'utilisation

Fractionnement de la courbe de Bézier quadratique en deux

```
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();
```

Fractionnement de la courbe du bezier cubique en deux

```
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 fonction split

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

> **Remarque: les** arguments situés à l'intérieur de [x4, y4] sont facultatifs.
>
> **Remarque:** La fonction possède un code en commentaire `/* */` facultatif qui traite des cas d'arête où les courbes résultantes peuvent avoir une longueur nulle ou se situer en dehors du début ou des extrémités de la courbe d'origine. Comme tente de fractionner une courbe en dehors de la plage valide pour la `position >= 0` ou la `position >= 1` , une erreur de plage est générée. Cela peut être supprimé et fonctionnera très bien, même si les courbes résultantes ont une longueur nulle.

```
// 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;
}
```

Trim bezier courbe.

Cet exemple vous montre comment couper un bezier.

La fonction trimBezier coupe les extrémités de la courbe en renvoyant la courbe `fromPos` `toPos` . `fromPos` et `toPos` sont compris entre 0 et 1 inclus. Il peut couper les courbes quadratiques et cubiques. Le type de courbe est déterminé par le dernier argument `x4` . Si non `undefined` ou `null` alors il suppose que la courbe est cubique sinon la courbe est un quadratique

La courbe découpée est renvoyée sous la forme d'un tableau de points. 6 points pour les courbes quadratiques et 8 pour les courbes cubiques.

Exemple d'utilisation

Découper une courbe quadratique.

```
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();
```

Découper une courbe cubique.

```
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();
```

Exemple de fonction

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

> **Remarque: les** arguments situés à l'intérieur de [x4, y4] sont facultatifs.
>
> **Remarque:** cette fonction requiert la fonction dans l'exemple Courbes de fracturation de Bézier At dans cette section

```
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;
}
```

Longueur d'une courbe de Bézier cubique (une approximation rapprochée)

Étant donné les 4 points d'une courbe de Bézier cubique, la fonction suivante renvoie sa longueur.

Méthode: La longueur d'une courbe de Bézier cubique n'a pas de calcul mathématique direct. Cette méthode de "force brute" recherche un échantillon de points le long de la courbe et calcule la distance totale parcourue par ces points.

Précision: La longueur approximative est exacte à 99 +% en utilisant la taille d'échantillonnage par défaut de 40.

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


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

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

Trouver un point sur la courbe

Cet exemple trouve un point sur une courbe bezier ou cubique à la `position` où la `position` est la distance unitaire sur la courbe. 0 <= `position` <= 1. La position est bloquée sur la plage. définir 0,1 respectivement.

Passez les coordonnées de la fonction 6 pour le bezier quadratique ou 8 pour le cube.

Le dernier argument facultatif est le vecteur renvoyé (point). S'il n'est pas donné, il sera créé.

Exemple d'utilisation

```
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 fonction

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

> **Remarque: les** arguments situés à l'intérieur de [x4, y4] sont facultatifs.
>
> **Remarque:** `x4` , `y4` si `null` ou `undefined` signifie que la courbe est un bezier quadratique.
> `vec`

est facultatif et contiendra le point retourné s'il est fourni. Sinon, il sera créé.

```
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;
}
```

Étendue de la recherche de la courbe quadratique

Lorsque vous avez besoin de trouver le rectangle englobant d'une courbe de Bézier quadratique, vous pouvez utiliser la méthode performante suivante.

```
// 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;
}
```

Pour un examen plus détaillé pour la résolution de mesure voir la réponse Pour obtenir une mesure de Bézier quadratique qui comprend des démos exécutables.

Lire Naviguer le long d'un chemin en ligne: https://riptutorial.com/fr/html5-canvas/topic/5281/naviguer-le-long-d-un-chemin

Chapitre 15: Ombres

Examples

Effet d'autocollant en utilisant des ombres

Ce code ajoute des ombres croissantes à une image pour créer une version "autocollant" de l'image.

Remarques:

- En plus d'être un objet ImageObject, l'argument "img" peut également être un élément Canvas. Cela vous permet de personnaliser vos propres dessins. Si vous dessinez du texte sur l'argument Canvas, vous pouvez également coller ce texte.
- Les images entièrement opaques n'auront aucun effet d'autocollant car l'effet est dessiné autour de groupes de pixels opaques bordés de pixels transparents.

```
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);
}
```

Comment arrêter d'autres ombres

Une fois que l'observation est activée, chaque nouveau dessin sur la toile sera ombré.

Désactivez d'autres masques en définissant `context.shadowColor` sur une couleur transparente.

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

... render some shadowed drawings ...

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

L'observation est coûteuse en calculs - Cachez cette ombre!

Attention! Appliquez les ombres avec parcimonie!

Appliquer l'observation est coûteux et multiplicativement coûteux si vous appliquez l'observation dans une boucle d'animation.

Au lieu de cela, mettez en cache une version ombrée de votre image (ou un autre dessin):

- Au début de votre application, créez une version ombrée de votre image dans un deuxième canevas en mémoire uniquement: `var memoryCanvas = document.createElement('canvas') ...`
- Chaque fois que vous avez besoin de la version ombrée, dessinez cette image pré-ombrée du canevas en mémoire dans le canevas visible: `context.drawImage(memoryCanvas,x,y)`

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

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

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

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

Ajouter de la profondeur visuelle avec des ombres

L'utilisation traditionnelle de l'ombrage consiste à donner des dessins bidimensionnels à l'illusion de la profondeur 3D.

Cet exemple montre le même "bouton" avec et sans ombrage

```
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)';
```

Ombres intérieures

Canvas n'a pas d' `inner-shadow` CSS.

- La toile va masquer l'extérieur d'une forme remplie.
- La toile ira à la fois à l'intérieur et à l'extérieur d'une forme caressée.

Mais il est facile de créer des ombres intérieures en utilisant le compositing.

Coups avec une ombre intérieure

Pour créer des traits avec une ombre intérieure, utilisez `destination-in` compositing qui provoque le contenu existant reste que si le contenu existant est chevauché par un nouveau contenu. Le contenu existant qui n'est pas recouvert par un nouveau contenu est effacé.

1. **Traitez une forme avec une ombre.** L'ombre s'étendra à la fois vers l'extérieur et l'intérieur du trait. Nous devons nous débarrasser de l'ombre extérieure en ne laissant que l'ombre intérieure désirée.
2. **Définissez la composition à l' `destination-in` de la `destination-in`** conserver l'ombre que si elle est recouverte par de nouveaux dessins.
3. **Remplissez la forme.** Cela fait que le trait et l'ombre intérieure restent pendant que l'ombre extérieure est effacée. *Eh bien, pas exactement! Comme un trait est à mi-distance et à moitié en dehors de la forme remplie, la moitié extérieure de la course sera également effacée. La solution consiste à doubler le `context.lineWidth` afin que la moitié du trait double soit toujours dans la forme remplie.*

```
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();
}
```

Filled Filled avec une ombre intérieure

Pour créer des remplissages avec une ombre interne, suivez les étapes 1 à 3 ci-dessus, mais utilisez davantage `destination-over` composition par `destination-over` ce qui entraîne la création de nouveaux contenus **sous du contenu existant** .

4. **Définissez la composition sur la `destination-over`** ce qui entraîne le remplissage **sous** le masque interne existant.
5. **Désactivez l'observation** en définissant `context.shadowColor` sur une couleur transparente.
6. **Remplissez la forme** avec la couleur souhaitée. La forme sera remplie sous l'ombre interne existante.

```
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();
}
```

Remplissage sans traits avec une ombre intérieure

Pour dessiner une forme remplie avec une ombre interne, mais sans le moindre trait, vous pouvez dessiner le trait hors-toile et utiliser `shadowOffsetX` pour repousser l'ombre sur le canevas.

```
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();
}
```

Lire Ombres en ligne: https://riptutorial.com/fr/html5-canvas/topic/5322/ombres

Chapitre 16: Texte

Examples

Texte de dessin

Le dessin sur toile ne se limite pas aux formes et aux images. Vous pouvez également dessiner du texte sur le canevas.

Pour dessiner du texte sur le canevas, obtenez une référence au canevas, puis appelez la méthode `fillText` sur le contexte.

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

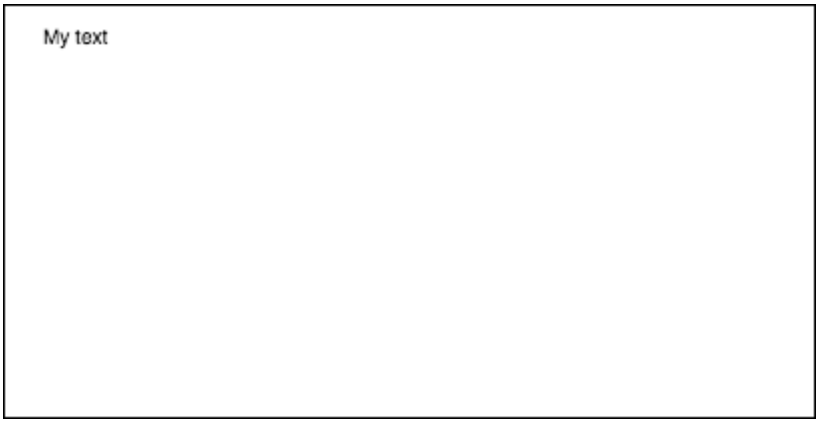
Les trois arguments **requis** qui sont passés dans `fillText` sont les suivants:

1. Le texte que vous souhaitez afficher
2. La position horizontale (axe des x)
3. La position verticale (axe des y)

En outre, il existe un quatrième argument **facultatif** , que vous pouvez utiliser pour spécifier la largeur maximale de votre texte en pixels. Dans l'exemple ci-dessous, la valeur de `200` limite la largeur maximale du texte à 200px:

```
ctx.fillText("My text", 0, 0, 200);
```

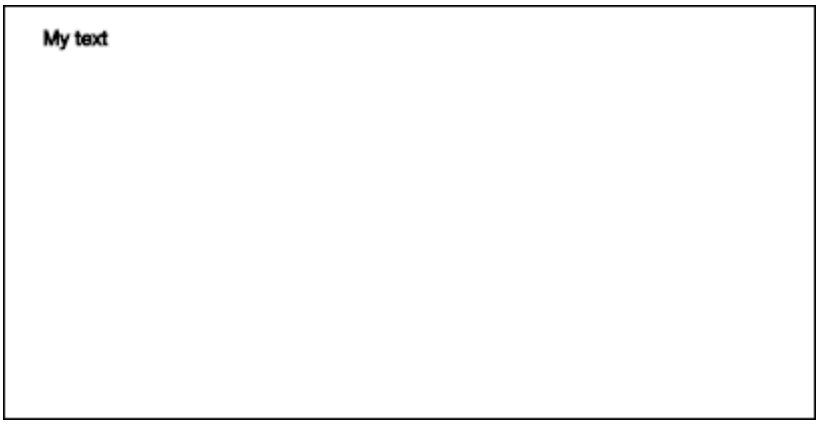
Résultat:



Vous pouvez également dessiner du texte sans remplissage et simplement un contour à l'aide de la méthode `strokeText` :

```
ctx.strokeText("My text", 0, 0);
```

Résultat:



Sans les propriétés de mise en forme de la police, le canevas affiche par défaut le texte à 10 pixels dans sans-serif, ce qui rend difficile la différence entre le résultat des méthodes `fillText` et `strokeText` . Voir l' exemple de formatage de texte pour plus de détails sur la manière d'augmenter la taille du texte et d'appliquer d'autres modifications esthétiques au texte.

Texte de formatage

Le formatage de police par défaut fourni par les méthodes `fillText` et `strokeText` n'est pas très esthétique. Heureusement, l'API canvas fournit des propriétés pour le formatage du texte.

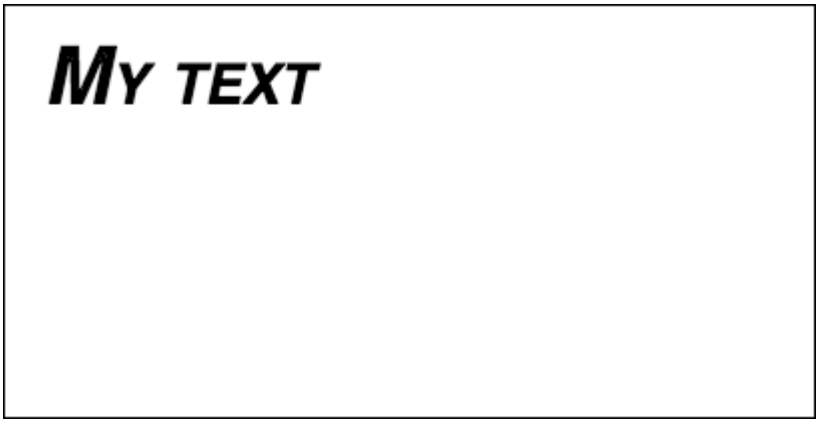
En utilisant la propriété de `font` , vous pouvez spécifier:

- le style de police
- variante de police
- poids de la police
- taille de police / hauteur de ligne
- famille de polices

Par exemple:

```
ctx.font = "italic small-caps bold 40px Helvetica, Arial, sans-serif";
ctx.fillText("My text", 20, 50);
```

Résultat:



En utilisant la propriété `textAlign` , vous pouvez également modifier l'alignement du texte sur:

- la gauche
- centre
- droite
- fin (même chose à droite)
- commencer (comme à gauche)

Par exemple:

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

Envelopper le texte en paragraphes

Native Canvas API ne dispose pas d'une méthode pour envelopper le texte sur la ligne suivante lorsqu'une largeur maximale souhaitée est atteinte. Cet exemple encapsule le texte dans des paragraphes.

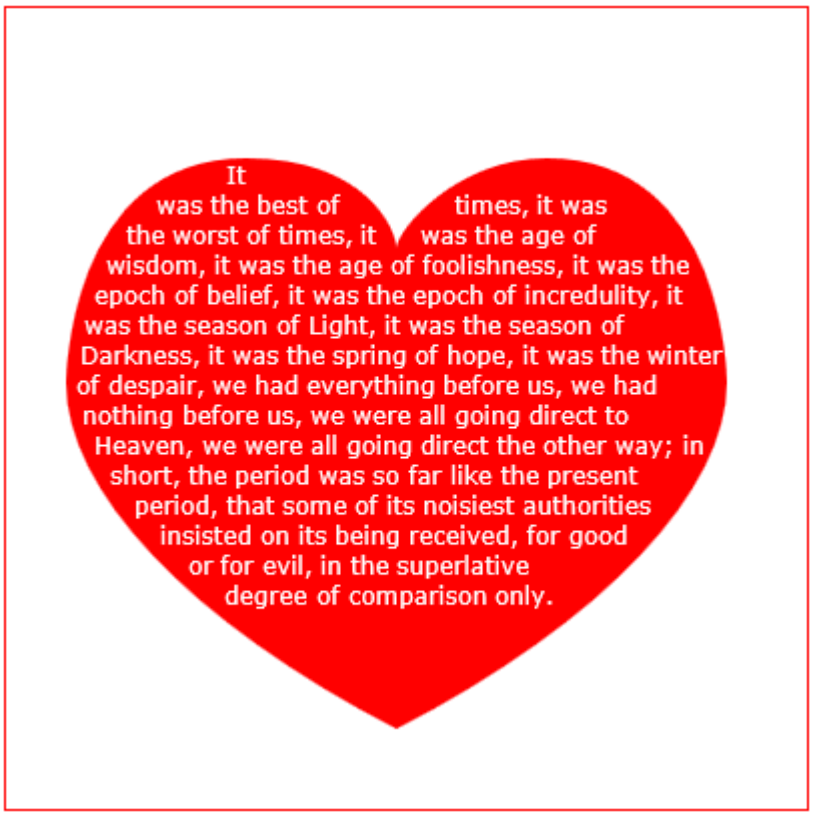
```
function wrapText(text, x, y, maxWidth, fontSize, fontFace){
  var firstY=y;
  var words = text.split(' ');
  var line = '';
  var lineHeight=fontSize*1.286; // a good approx for 10-18px sizes

  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);
}
```

Dessinez des paragraphes de texte dans des formes irrégulières

Cet exemple dessine des paragraphes de texte dans toutes les parties du canevas qui ont des pixels opaques.

Cela fonctionne en trouvant le prochain bloc de pixels opaques qui est assez grand pour contenir le mot spécifié suivant et en remplissant ce bloc avec le mot spécifié.

Les pixels opaques peuvent provenir de n'importe quelle source: commandes de dessin de chemin et / ou images.




```
<!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>
```

Remplir le texte avec une image

Cet exemple remplit le texte avec une image spécifiée.

Important! L'image spécifiée doit être entièrement chargée avant d'appeler cette fonction ou le dessin échouera. Utilisez `image.onload` pour vous assurer que l'image est complètement chargée.

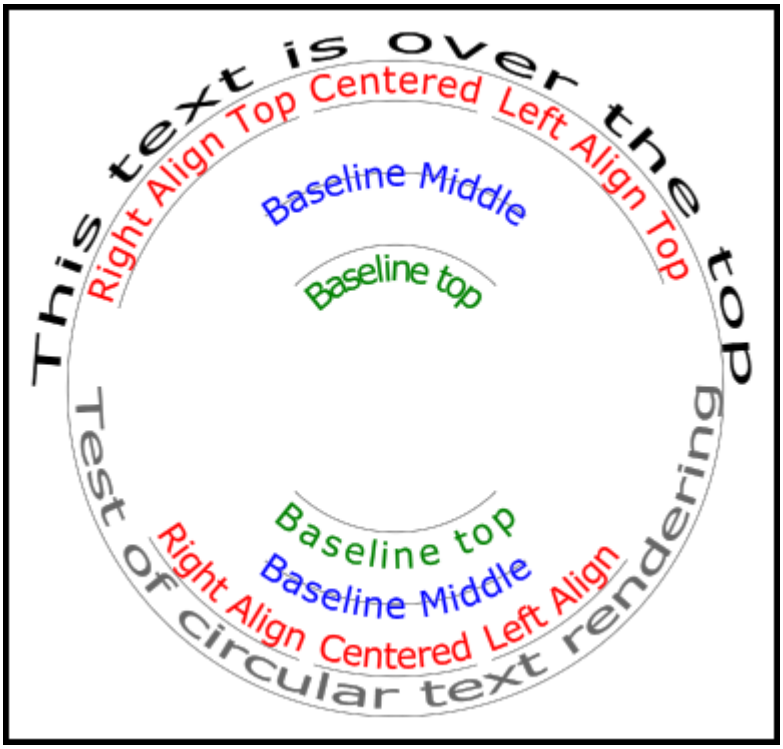

```
function drawImageInsideText(canvas,x,y,img,text,font){
    var c=canvas.cloneNode();
    var ctx=c.getContext('2d');
    ctx.font=font;
    ctx.fillText(text,x,y);
    ctx.globalCompositeOperation='source-atop';
    ctx.drawImage(img,0,0);
    canvas.getContext('2d').drawImage(c,0,0);
}
```

Rendu du texte le long d'un arc.

Cet exemple montre comment rendre du texte le long d'un arc. Il comprend comment vous pouvez ajouter des fonctionnalités à `CanvasRenderingContext2D` en étendant son prototype.

Cet exemple est dérivé du texte circulaire de la réponse stackoverflow.

Exemple de rendu



Exemple de code

L'exemple ajoute 3 nouvelles fonctions de rendu de texte au prototype de contexte 2D.

- **ctx.fillCircleText (texte, x, y, rayon, début, fin, avant);**
- **ctx.strokeCircleText (texte, x, y, rayon, début, fin, avant);**
- **ctx.measureCircleText (texte, rayon);**

```
(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;
})();
```

Description des fonctions

Cet exemple ajoute 3 fonctions au `CanvasRenderingContext2D prototype` . `fillCircleText` , `strokeCircleText` et `measureCircleText`

CanvasRenderingContext2D.fillCircleText (texte, x, y, rayon, début, [fin, [avant]]);

CanvasRenderingContext2D.strokeCircleText (texte, x, y, rayon, début, [fin, [avant]]);

- **text:** Texte à afficher sous forme de chaîne.
- **x** , **y** : position du centre du cercle sous forme de nombres.
- **rayon:** rayon du cercle en pixels
- **start:** angle en radians pour commencer.
- **[fin]:** facultatif. Si inclus, `ctx.textAlign` est ignoré et le texte est mis à l'échelle entre le début et la fin.
- **[forward]:** option par défaut 'true'. si la direction du texte est vraie, si la «fausse» direction est en arrière.

Les deux fonctions utilisent textBaseline pour positionner le texte verticalement autour du rayon. Pour les meilleurs résultats, utilisez `ctx.TextBaseline` .

Les fonctions lanceront un `TypeError` est l'un des arguments numériques en tant que NaN.

Si l'argument de `text` réduit à une chaîne vide ou `ctx.globalAlpha = 0` la fonction ne fait que passer et ne fait rien.

CanvasRenderingContext2D.measureCircleText (text, radius);

```
- **text:** String of text to measure.
- **radius:** radius of circle in pixels.
```

Renvoie un objet contenant diverses mesures de taille pour le rendu du texte circulaire

```
   - **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.
```

Exemples d'utilisation

```
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();
```

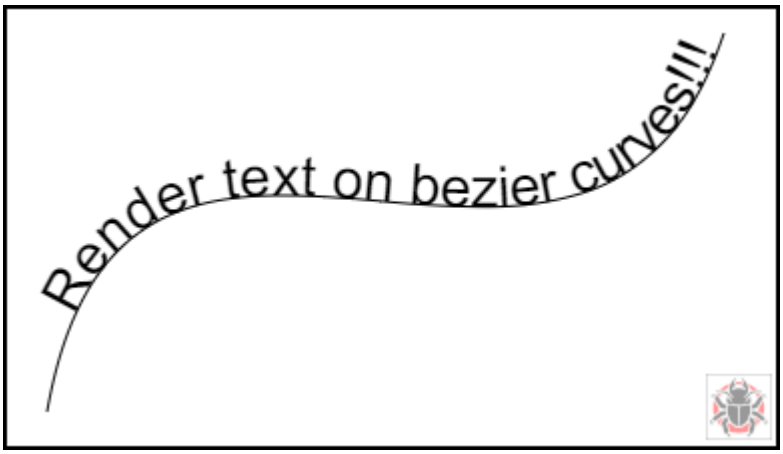
```
ctx.stroke();
```

REMARQUE: Le texte rendu n'est qu'une approximation du texte circulaire. Par exemple, si deux ls sont rendues, les deux lignes ne seront pas parallèles, mais si vous rendez un "H", les deux arêtes seront parallèles. C'est parce que chaque caractère est rendu aussi proche que possible de la direction requise, plutôt que chaque pixel soit correctement transformé pour créer un texte circulaire.

NOTE: `const multiplyCurrentTransform = true;` défini dans cet exemple est utilisé pour définir la méthode de transformation utilisée. Si `false` la transformation pour le rendu du texte circulaire est absolue et ne dépend pas de l'état de la transformation en cours. Le texte ne sera affecté par aucune échelle, rotation ou traduction antérieure. Cela augmentera les performances de la fonction de rendu, après que la fonction soit appelée, la transformation sera définie sur `setTransform(1,0,0,1,0,0)` par défaut `setTransform(1,0,0,1,0,0)`

Si `multiplyCurrentTransform = true` (définie par défaut dans cet exemple) le texte utilisera la transformée en cours afin que le texte peut être mis à l' échelle traduit, en biais, rotation, etc. , mais la modification de la transformation en cours befor appelant les `fillCircleText` et `strokeCircleText` fonctions. Selon l'état actuel du contexte 2D, cela peut être un peu plus lent que `multiplyCurrentTransform = false`

Texte sur courbe, beziers cubiques et quadratiques



textOnCurve (texte, décalage, x1, y1, x2, y2, x3, y3, x4, y4)

Rend le texte sur les courbes quadratiques et cubiques.

- `text` est le texte à rendre
- distance de `offset` du début de la courbe au texte> = 0
- `x1,y1` - `x3,y3` points de courbe quadratique ou
- `x1,y1` - `x4,y4` points de courbe cubique ou

Exemple d'utilisation:

```
textOnCurve("Hello world!",50,100,100,200,200,300,100); // draws text on quadratic curve
                                                        // 50 pixels from start of curve


textOnCurve("Hello world!",50,100,100,200,200,300,100,400,200);
                                                        // draws text on cubic curve
                                                        // 50 pixels from start of curve
```

La fonction d'assistance et la fonction de recourbement

```
// 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 fonction d'assistance de courbe est conçue pour améliorer les performances de recherche de points sur le 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){
```

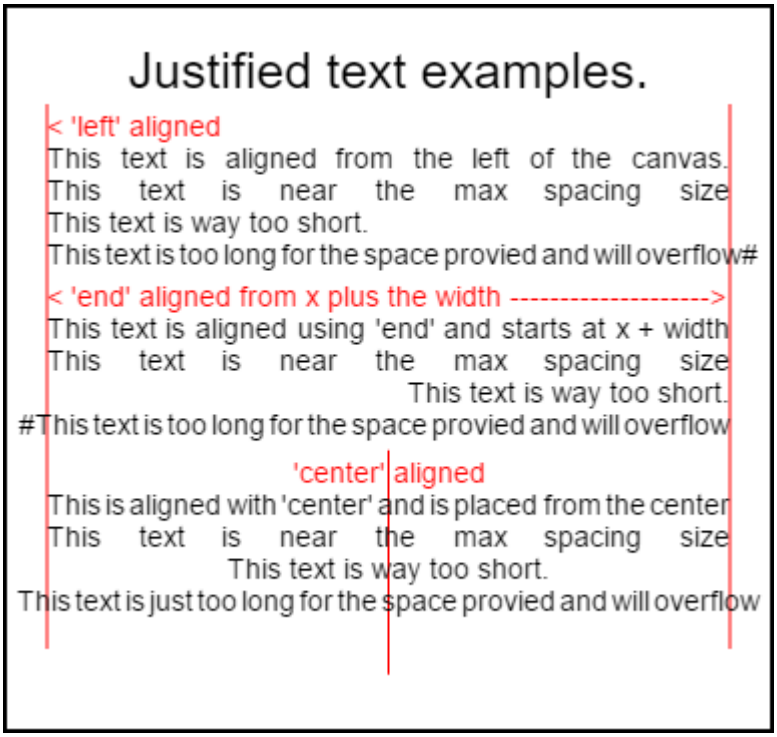
```
        tx1 = x1; ty1 = y1;
        tx2 = x2; ty2 = y2;
        tx3 = x3; ty3 = y3;
        tx1 += (tx2 - tx1) * c;
        ty1 += (ty2 - ty1) * c;
        tx2 += (tx3 - tx2) * c;
        ty2 += (ty3 - ty2) * c;
        tx3 += (x4 - tx3) * c;
        ty3 += (y4 - ty3) * c;
        tx1 += (tx2 - tx1) * 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
}
```

Texte justifié

Cet exemple affiche un texte justifié. Il ajoute des fonctionnalités supplémentaires à la `CanvasRenderingContext2D` en étendant son prototype ou un objet global `justifiedText` (voir la note en option A).

Exemple de rendu.



Le code pour rendre cette image se trouve dans les exemples d'utilisation en bas .

L'exemple

La fonction en tant que fonction anonyme immédiatement invoquée.

```
(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*/
})();
```

> **Remarque A:** Si vous ne souhaitez pas étendre le prototype `CanvasRenderingContext2D` Supprimez de l'exemple tout le code entre `// code point A` et `// code point B` et décommentez le code marqué `/* Uncomment from here to the closing comment`

Comment utiliser

Trois fonctions sont ajoutées à `CanvasRenderingContext2D` et sont disponibles pour tous les objets de contexte 2D créés.

- ctx.fillJustifyText (texte, x, y, largeur, [paramètres]);
- ctx.strokeJustifyText (texte, x, y, largeur, [paramètres]);
- ctx.measureJustifiedText (texte, largeur, [paramètres]);

Remplissez et contournez la fonction de texte, remplissez ou contournez le texte et utilisez les mêmes arguments. `measureJustifiedText` renvoie la largeur réelle à laquelle le texte sera rendu. Cela peut être égal, inférieur ou supérieur à la `width` de l'argument en fonction des paramètres actuels.

> **Remarque: les** arguments situés à l'intérieur de `[` et `]` sont facultatifs.

Arguments de fonction

- **text:** Chaîne contenant le texte à afficher.
- **x, y:** coordonnées pour rendre le texte à.
- **width:** Largeur du texte justifié. Le texte augmentera / diminuera les espaces entre les mots pour s'adapter à la largeur. Si l'espace entre les mots est supérieur à `maxSpaceSize` (valeur par défaut = 6), l'espacement normal sera utilisé et le texte ne remplira pas la largeur requise. Si l'espacement est inférieur à `minSpaceSize` (valeur par défaut = 0,5), l'espacement normal est utilisé, la taille de l'espace minimum est utilisée et le texte dépasse la largeur demandée.
- **paramètres:** facultatif. Objet contenant des tailles d'espace min et max.

L'argument des `settings` est facultatif et, s'il n'est pas inclus, le rendu du texte utilisera le dernier paramètre défini ou le paramètre par défaut (illustré ci-dessous).

Les valeurs min et max sont les tailles min et max pour les mots séparant le caractère [espace]. La valeur par défaut `maxSpaceSize = 6` signifie que lorsque l'espace entre les caractères est> 63 * ctx.measureText (""), le texte .width ne sera pas justifié. Si le texte à justifier comporte des espaces inférieurs à `minSpaceSize = 0.5` (valeur par défaut 0,5) * `ctx.measureText(" ").width` l'espacement sera défini sur `minSpaceSize * ctx.measureText(" ").width` la largeur de justification.

Les règles suivantes sont appliquées, min et max doivent être des nombres. Sinon, les valeurs associées ne seront pas modifiées. Si `minSpaceSize` est plus grand que `maxSpaceSize` deux paramètres d'entrée sont invalides et min max ne sera pas modifié.

Exemple d'objet avec des valeurs par défaut

```
settings = {
    maxSpaceSize : 6;   // Multiplier for max space size.
    minSpaceSize : 0.5; // Multiplier for minimum space size
};
```

> **REMARQUE:** Ces fonctions de texte introduisent un changement de comportement subtil pour la propriété `textAlign` du contexte 2D. "Left", "right", "center" et "start" se comportent comme prévu, mais "end" ne s'alignera pas de la droite de l'argument de la fonction `x` mais plutôt de la droite de `x + width`
>
> **Remarque: les** paramètres (taille de l'espace min et max) sont globaux pour tous les objets de contexte 2D.

USAGE Exemples

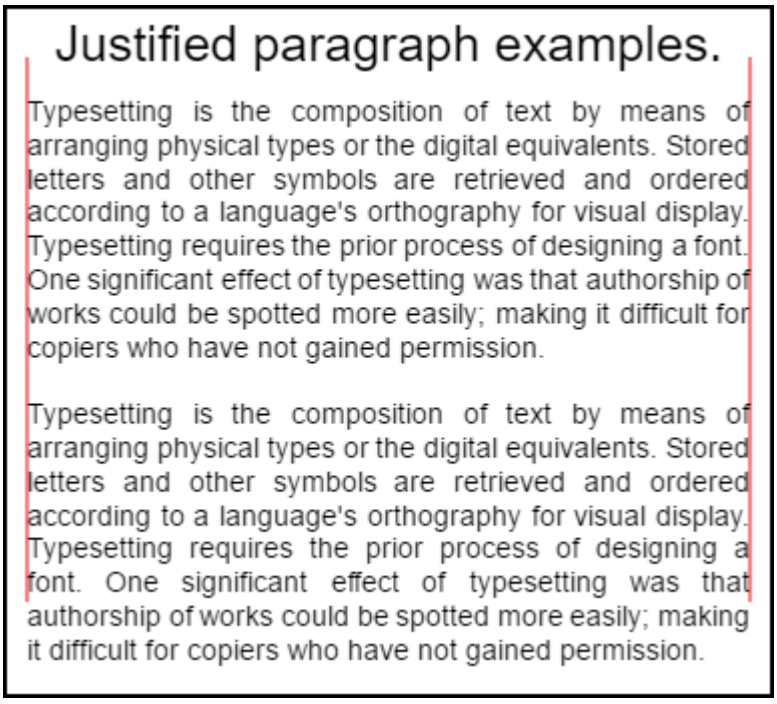
```
var i = 0;
text[i++] = "This text is aligned from the left of the canvas.";
text[i++] = "This text is near the max spacing size";
text[i++] = "This text is way too short.";
text[i++] = "This text is too long for the space provied and will overflow#";
text[i++] = "This text is aligned using 'end' 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);
```

Paragraphes justifiés.

Rend le texte en tant que paragraphes justifiés. **Exige** l'exemple **Texte justifié**

Exemple de rendu



Le paragraphe supérieur a **setting.compact = true** et bottom **false** et l'interligne est **1.2** plutôt que la valeur par défaut **1.5** . Rendu par exemple d'utilisation du code en bas de cet exemple.

Exemple de code

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

> **REMARQUE** cela étend le prototype `CanvasRenderingContext2D` . Si vous ne souhaitez pas que cela se produise, utilisez l'exemple de **texte justifié** pour déterminer comment modifier cet exemple pour qu'il fasse partie de l'espace de noms global.
>
> **NOTE** `CanvasRenderingContext2D.prototype.fillJustifyText` un ReferenceError si cet exemple ne trouve pas la fonction `CanvasRenderingContext2D.prototype.fillJustifyText`

Comment utiliser

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

Voir **Texte justifié** pour plus de détails sur les arguments. Les arguments entre `[` et `]` sont facultatifs.

L'argument `settings` a deux propriétés supplémentaires.

- **compact:** Par défaut `true` . Si true essaie de mettre autant de mots que possible par ligne. Si la valeur est false, le système tente d'obtenir l'espacement des mots aussi proche que possible de l'espacement normal.
- **lineSpacing** Par défaut `1.5` . Espace par défaut de la ligne `1.5` la distance entre la ligne et la suivante en termes de taille de police

Les propriétés manquantes dans l'objet de paramètres seront par défaut à leurs valeurs par défaut ou aux dernières valeurs valides. Les propriétés ne seront modifiées que si les nouvelles valeurs sont valides. Pour `compact` valeurs valides `compact` ne sont que des booléens `true` ou `false` valeurs de `true` ne sont pas considérées comme valides.

Objet retour

Les deux fonctions renvoient un objet contenant des informations pour vous aider à placer le paragraphe suivant. L'objet contient les propriétés suivantes.

- **nextLine** Position de la ligne suivante après les pixels du paragraphe.
- **fontSize** Taille de la police. (veuillez noter que n'utilisez que des polices définies en pixels, par exemple `14px arial`)
- **lineHeight** Distance en pixels d'une ligne à la suivante

Cet exemple utilise un algorithme simple qui fonctionne une ligne à la fois pour trouver le meilleur ajustement pour un paragraphe. Cela ne signifie pas que c'est le meilleur ajustement (plutôt que l'algorithme le mieux adapté). Vous pouvez souhaiter améliorer l'algorithme en créant un algorithme à plusieurs lignes sur les lignes générées. Déplacement des mots de la fin d'une ligne au début de la suivante ou du début à la fin. Le meilleur aspect est obtenu lorsque l’espacement

sur l’ensemble du paragraphe a la plus petite variation et est le plus proche de l’espacement normal du texte.

Comme cet exemple dépend de l'exemple de **texte justifié,** le code est très similaire. Vous souhaiterez peut-être déplacer les deux dans une fonction. Remplacez la fonction `justifiedTextSettings` dans l'autre exemple par celle utilisée dans cet exemple. Copiez ensuite tout le reste du code de cet exemple dans le corps de la fonction anonyme de l'exemple de **texte justifié** . Vous n'aurez plus besoin de tester les dépendances trouvées à `// Code point A` Il peut être supprimé.

Exemple d'utilisation

```
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);
```

> **Remarque:** pour le texte aligné à `left` ou au `start` la dernière ligne du paragraphe aura toujours un espacement normal. Pour tous les autres alignements, la dernière ligne est traitée comme toutes les autres.

Remarque: Vous pouvez insérer le début du paragraphe avec des espaces. Bien que cela puisse ne pas être cohérent d'un paragraphe à l'autre. C'est toujours une bonne chose d'apprendre ce que fait une fonction et de la modifier. Un exercice serait d'ajouter un paramètre aux paramètres qui indente la première ligne d'un montant fixe. La boucle while devra faire apparaître temporairement le premier mot (+ indent) `words[0].width += ?` et puis, lors du rendu des lignes, indenter la première ligne.

Lire Texte en ligne: https://riptutorial.com/fr/html5-canvas/topic/5235/texte

Chapitre 17: Transformations

Examples

Dessiner rapidement de nombreuses images traduites, mises à l'échelle et pivotées

Il existe de nombreuses situations où vous souhaitez dessiner une image qui est pivotée, mise à l'échelle et traduite. La rotation devrait avoir lieu autour du centre de l'image. C'est le moyen le plus rapide de le faire sur la toile 2D. Ces fonctions sont bien adaptées aux jeux en 2D où l'on s'attend à rendre quelques centaines voire plus de 1000 images toutes les 60 secondes. (dépend du matériel)

```
// 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
}
```

Une variante peut également inclure la valeur alpha qui est utile pour les systèmes de particules.

```
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
}
```

Il est important de noter que les deux fonctions laissent le contexte du canevas dans un état aléatoire. Bien que les fonctions ne seront pas affectées les autres rendant mon être. Lorsque vous avez fini de rendre les images, vous devrez peut-être restaurer la transformation par défaut

```
ctx.setTransform(1, 0, 0, 1, 0, 0); // set the context transform back to the default
```

Si vous utilisez la version alpha (deuxième exemple) et la version standard, vous devrez vous assurer que l'état alpha global est restauré

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

Un exemple d'utilisation des fonctions ci-dessus pour rendre certaines particules et quelques images

```
// 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
```

Faire pivoter une image ou un chemin autour de son centre

Les étapes 1 à 5 ci-dessous permettent de déplacer n'importe quelle image ou forme de chemin n'importe où sur la toile et de la faire pivoter à n'importe quel angle sans modifier les coordonnées du point d'origine de l'image / du chemin.

1. Déplacer l'origine du canevas [0,0] vers le centre de la forme

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

2. Faire pivoter la toile de l'angle souhaité (en radians)

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

3. Déplacer l'origine du canevas vers le coin supérieur gauche

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

4. Dessinez l'image ou la forme du chemin en utilisant ses coordonnées d'origine.

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

5. Toujours nettoyer! Rétablit l'état de transformation à l'endroit où il était avant # 1

- *Étape n ° 5, option n ° 1:* Annulez chaque transformation dans l'ordre inverse

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

- *Étape 5, Option 2:* Si le canevas était dans un état non transformé (valeur par défaut) avant de commencer l'étape 1, vous pouvez annuler les effets des étapes 1 à 3 en réinitialisant toutes les transformations à leur état par défaut.

```
// set transformation to the default state (==no transformation applied)
context.setTransform(1,0,0,1,0,0)
```

Exemple de démonstration de code:

```
// 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 );
```

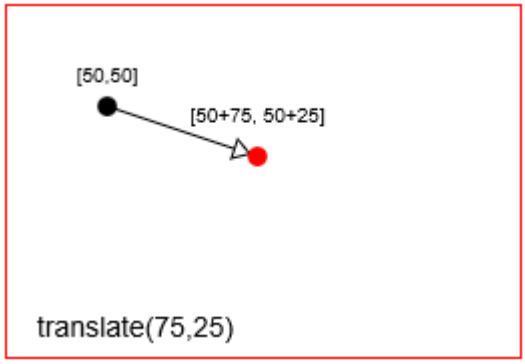
Introduction aux transformations

Les transformations modifient la position de départ d'un point donné en déplaçant, en tournant et en mettant à l'échelle ce point.

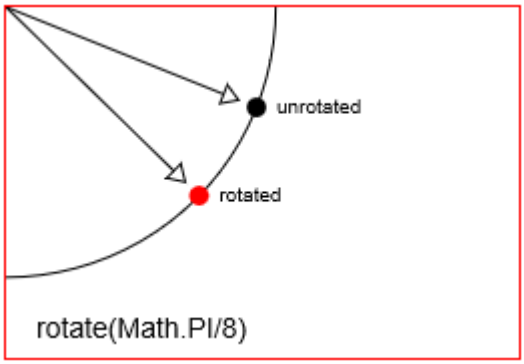
- **Traduction:** Déplace un point de `distanceX` et de `distanceY` .
- **Rotation: fait** pivoter un point par un `radian angle` autour de son point de rotation. Le point de rotation par défaut dans Canevas HTML est l'origine supérieure gauche [x = 0, y = 0] du canevas. Mais vous pouvez repositionner le point de rotation en utilisant des traductions.
- **Mise à l'échelle:** `scalingFactorY` à l'échelle la position d'un point par un `scalingFactorX` et un `scalingFactorY` depuis son point de mise à l'échelle. Le point de mise à l'échelle par défaut dans Canvas HTML est l'origine supérieure gauche [x = 0, y = 0] du canevas. Mais vous pouvez repositionner le point de mise à l'échelle à l'aide de traductions.

Vous pouvez également effectuer des transformations moins courantes, telles que le cisaillement (skewing), en définissant directement la matrice de transformation du canevas à l'aide de `context.transform`.

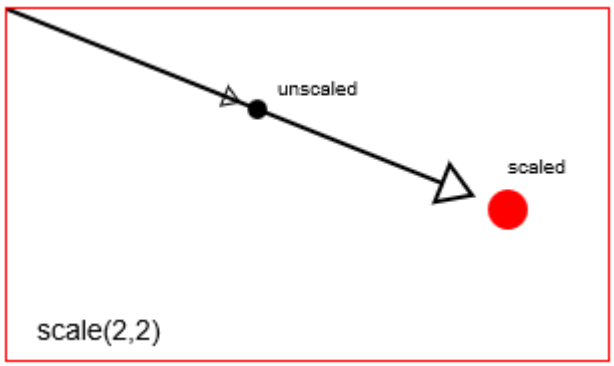
Traduire (== déplacer) un point avec `context.translate(75,25)`



Faire pivoter un point avec `context.rotate(Math.PI/8)`



Mettre à l'échelle un point avec `context.scale(2,2)`



Canvas réalise en réalité des transformations en modifiant le système de coordonnées complet de la toile.

- `context.translate` déplace l'origine du canevas [0,0] du coin supérieur gauche vers un nouvel emplacement.
- `context.rotate` fera pivoter tout le système de coordonnées du canevas autour de l'origine.
- `context.scale` mettra à l'échelle le système de coordonnées du canevas entier autour de l'origine. Considérez ceci comme augmentant la taille de chaque x, y sur le canevas: `every x*=scaleX` et `every y*=scaleY`.

Les transformations de la toile sont persistantes. Tous les nouveaux dessins continueront à être transformés jusqu'à ce que vous réinitialisiez la transformation du canevas à son état par défaut (== totalement non transformé). Vous pouvez revenir à la valeur par défaut avec:

```
// reset context transformations to the default (untransformed) state
context.setTransform(1,0,0,1,0,0);
```

Une matrice de transformation pour suivre les formes traduites, pivotées et mises à l'échelle

Canvas vous permet de `context.translate` , `context.rotate` et `context.scale` afin de dessiner votre forme dans la position et la taille souhaitées.

Canvas lui-même utilise une matrice de transformation pour suivre efficacement les transformations.

- Vous pouvez changer la matrice de Canvas avec `context.transform`
- Vous pouvez changer la matrice de Canvas avec des commandes individuelles de `translate, rotate & scale`
- Vous pouvez complètement écraser la matrice de Canvas avec `context.setTransform` ,
- *Mais vous ne pouvez pas lire la matrice de transformation interne de Canvas - elle est en écriture seule.*

Pourquoi utiliser une matrice de transformation?

Une matrice de transformation vous permet d'agréger de nombreuses traductions, rotations et échelles individuelles en une seule matrice facilement réappliquée.

Lors d'animations complexes, vous pouvez appliquer des dizaines (ou des centaines) de transformations à une forme. En utilisant une matrice de transformation, vous pouvez (presque) réappliquer instantanément ces dizaines de transformations avec une seule ligne de code.

Quelques exemples d'utilisation:

- **Tester si la souris est dans une forme que vous avez traduite, tournée et mise à l'échelle**

  Il y a un `context.isPointInPath` intégré qui teste si un point (par exemple la souris) se trouve dans une forme de chemin, mais ce test intégré est très lent comparé au test utilisant une matrice.

  Tester efficacement si la souris est dans une forme implique de prendre la position de la souris signalée par le navigateur et de la transformer de la même manière que la forme a été transformée. Vous pouvez ensuite appliquer le test de frappe comme si la forme n'était pas transformée.

- **Redessinez une forme qui a été largement traduite, pivotée et mise à l'échelle.**

  Au lieu de réappliquer des transformations individuelles avec plusieurs `.translate, .rotate, .scale` vous pouvez appliquer toutes les transformations agrégées sur une seule ligne de code.

- **Formes de test de collision traduites, pivotées et mises à l'échelle**

  Vous pouvez utiliser la géométrie et la trigonométrie pour calculer les points constituant les formes transformées, mais il est plus rapide d'utiliser une matrice de transformation pour calculer ces points.

Une matrice de transformation "classe"

Ce code reflète les commandes de transformation `context.translate` , `context.rotate` et `context.scale` natives. Contrairement à la matrice de canevas native, cette matrice est lisible et réutilisable.

Méthodes:

- `translate` , `rotate` , `scale` les commandes de transformation du contexte et vous permettre d'alimenter les transformations dans la matrice. La matrice contient efficacement les transformations agrégées.
- `setContextTransform` prend un contexte et définit la matrice de ce contexte sur cette matrice de transformation. Cela réapplique efficacement toutes les transformations stockées dans cette matrice au contexte.
- `resetContextTransform` réinitialise la transformation du contexte à son état par défaut (== non transformé).
- `getTransformedPoint` prend un point de coordonnées non transformé et le convertit en un point transformé.
- `getScreenPoint` prend un point de coordonnées transformé et le convertit en un point non transformé.
- `getMatrix` renvoie les transformations agrégées sous la forme d'un tableau matriciel.

Code:

```
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);
})();
```

Démo:

Cette démo utilise la matrice de transformation "Class" ci-dessus pour:

- Suivre (== enregistrer) la matrice de transformation d'un rectangle.
- Redessinez le rectangle transformé sans utiliser les commandes de transformation de contexte.
- Testez si la souris a cliqué à l'intérieur du rectangle transformé.

Code:

```
<!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>
```

Lire Transformations en ligne: https://riptutorial.com/fr/html5-canvas/topic/5494/transformations

Chapitre 18: Types de support et la toile

Remarques

Cette rubrique couvre les différents types de support et leur utilisation avec le canevas dans une interface 2D.

Les types de média ont des catégories génériques et spécifiques au format

Types de médias

- Des animations
- Vidéos
- Images
- Images HD
- Image vectorielle
- Images animées

Formats de média

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

Images

Il existe une grande variété de formats d'images supportés par les navigateurs, mais aucun navigateur ne les supporte tous. Si vous souhaitez utiliser des navigateurs Wiki et des formats d'images pris en charge avec des formats d' images particuliers, vous avez une bonne vue d'ensemble.

Le meilleur support est pour les 3 principaux formats, "jpeg", "png" et "gif" avec tous les principaux navigateurs fournissant un support.

JPEG

Les images JPEG conviennent mieux aux photos et aux images de type photo. Ils ne se prêtent pas bien aux graphiques, aux diagrammes et au texte. Les images JPEG ne prennent pas en charge la transparence.

Canvas peut générer des images JPEG via `canvas.toDataURL` et `canvas.toBlob` et fournit un paramètre de qualité. Le format JPEG ne prenant pas en charge la transparence, tous les pixels

transparents seront mélangés au noir pour la sortie finale JPG. L'image résultante ne sera pas une copie parfaite de la toile.

JPEG sur wikipedia

PNG

PNG Image est la plus haute qualité d'image et peut également inclure un canal alpha pour les pixels transparents. Les données d'image sont compressées mais ne produisent pas d'artefacts tels que des images JPG.

Grâce à la compression sans perte et à la prise en charge du canal alpha, les fichiers PNG sont utilisés pour les jeux, les images de composants d'interface utilisateur, les graphiques, les diagrammes et le texte. Lorsque vous les utilisez pour des photos et des photos comme des images, la taille de leur fichier peut être beaucoup plus grande que celle du format JPEG. .

Le format PNG prend également en charge l'animation, même si la prise en charge du navigateur est limitée, et l'accès à l'animation pour une utilisation sur le canevas ne peut être effectué que via des API et des bibliothèques Javascript.

Le canevas peut être utilisé pour encoder des images PNG via `canvas.toDataURL` et `canvas.toBlob` même si le format de sortie est limité au format RGBA 32 bits compressé. Le PNG fournira une copie parfaite de la toile.

PNG chez wikipedia

GIF

Les GIF sont utilisés pour de courtes animations, mais peuvent également être utilisés pour fournir des graphiques, des diagrammes et des textes de haute qualité, tels que des images. Les GIF ont un support couleur très limité avec un maximum de 256 couleurs par image. Avec le traitement des images cleaver, les images gif peuvent produire des résultats étonnamment bons, surtout lorsqu'elles sont animées. Les gifs offrent également de la transparence, bien que cela soit limité à l'activation ou à la désactivation

Comme avec PNG, les animations GIF ne sont pas directement accessibles pour une utilisation sur le canevas et vous aurez besoin d'une API ou d'une bibliothèque Javascript pour y accéder. GIF ne peut pas être enregistré via le canevas et nécessitera une API ou une bibliothèque pour le faire.

GIF chez wikipedia

Examples

Chargement et affichage d'une image

Charger une image et la placer sur la toile

```
var image = new Image();  // see note on creating an image
```

```
image.src = "imageURL";
image.onload = function(){
    ctx.drawImage(this,0,0);
}
```

Créer une image

Il y a plusieurs façons de créer une image

- `new Image()`
- `document.createElement("img")`
- `<img src = 'imageUrl' id='myImage'>` Dans le cadre du corps HTML et récupéré avec `document.getElementById('myImage')`

L'image est un `HTMLImageElement`

Propriété Image.src

L'image `src` peut être une URL d'image valide ou une URL de données encodée. Consultez les remarques de cette rubrique pour plus d'informations sur les formats d'image et la prise en charge.

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

* Le dataURL est une image gif de 1 par 1 pixel contenant du noir

Remarques sur le chargement et les erreurs

L'image commence à se charger lorsque sa propriété src est définie. Le chargement est syncriouse mais l'événement `onload` ne sera pas appelé tant que la fonction ou le code n'est pas sorti / retourné.

Si vous obtenez une image à partir de la page (par exemple `document.getElementById("myImage")`) et que son `src` est défini, il peut ou non être chargé. Vous pouvez vérifier le statut de l'image avec `HTMLImageElement.complete` qui sera `true` si complet. Cela ne signifie pas que l'image a été chargée, cela signifie qu'elle a soit

- chargé
- Il y avait une erreur
- La propriété src n'a pas été définie et est égale à la chaîne vide `""`

Si l'image provient d'une source non fiable et peut ne pas être accessible pour diverses raisons, cela générera un événement d'erreur. Lorsque cela se produit, l'image sera dans un état cassé. Si vous essayez ensuite de le dessiner sur le canevas, l'erreur suivante apparaîtra.

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

En fournissant l' `image.onerror = myImgErrorHandler` , vous pouvez prendre les mesures appropriées pour éviter les erreurs.

Dessiner une image svg

Pour dessiner une image SVG vectorielle, l'opération n'est pas différente d'une image raster: Vous devez d'abord charger votre image SVG dans un élément HTMLImage, puis utiliser la méthode `drawImage()` .

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

Les images SVG présentent certains avantages par rapport aux images matricielles, car vous ne perdrez pas de qualité, quelle que soit l'échelle que vous allez dessiner sur votre toile. Mais attention, cela peut aussi être un peu plus lent que dessiner une image raster.

Cependant, les images SVG comportent plus de restrictions que les images matricielles.

- **Pour des raisons de sécurité, aucun contenu externe ne peut être chargé à partir d'une image SVG référencée dans un objet HTMLImageElement (`<img>`)**
  Aucune feuille de style externe, aucune image externe référencée dans les éléments SVGImage (`<image/>`), aucun filtre ou élément externe lié par l' `xlink:href` (`<use xlink:href="anImage.SVG#anElement"/>`) ou le funcIRI (`url()`) méthode d'attribut etc.
  De même, les feuilles de style ajoutées au document principal n'auront aucun effet sur le document SVG, une fois référencé dans un élément HTMLImage.
  Enfin, aucun script ne sera exécuté dans l'image SVG.
  Solution de contournement: vous devez ajouter tous les éléments externes à l'intérieur du fichier SVG avant de référencer l'élément HTMLImage. (pour les images ou les polices, vous devez ajouter une version de dataURI de vos ressources externes).

- **L'élément root (`<svg>`) doit avoir ses attributs width et height définis sur une valeur absolue.**
  Si vous utilisiez une longueur relative (par exemple `%`), le navigateur ne pourra pas savoir à quoi il est relatif. Certains navigateurs (Blink) essaieront de deviner, mais la plupart ignoreront simplement votre image et ne dessineront rien, sans avertissement.

- **Certains navigateurs vont corrompre le canevas lorsqu'une image SVG y est dessinée.**
  Plus précisément, Internet-Explorer <Edge dans tous les cas et Safari 9 lorsqu'un `<foreignObject>` est présent dans l'image SVG.

Chargement de base et lecture d'une vidéo sur la toile.

Le canevas peut être utilisé pour afficher des vidéos provenant de diverses sources. Cet exemple montre comment charger une vidéo en tant que ressource de fichier, l'afficher et ajouter un simple clic sur la lecture d'écran / pause.

Cette question auto-répondue stackoverflow Comment afficher une vidéo à l'aide de la balise

canvas HTML5 montre l'exemple de code suivant en action.

Juste une image

Une vidéo n'est qu'une image en ce qui concerne la toile. Vous pouvez le dessiner comme n'importe quelle image. La différence étant la vidéo peut jouer et a du son.

Obtenir la toile et la configuration de 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
```

Créer et charger la vidéo

```
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,
};
```

Contrairement aux images, les vidéos ne doivent pas être entièrement chargées pour être affichées sur le canevas. Les vidéos fournissent également une foule d'événements supplémentaires pouvant être utilisés pour surveiller l'état de la vidéo.

Dans ce cas, nous souhaitons savoir quand la vidéo est prête à jouer. `oncanplay` signifie que suffisamment de vidéo a été chargée pour en lire une partie, mais il n'y en a peut-être pas assez pour jouer jusqu'à la fin.

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

Sinon, vous pouvez utiliser une `oncanplaythrough` qui se déclenche lorsque la vidéo est suffisamment chargée pour pouvoir être jouée jusqu'à la fin.

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

N'utilisez qu'un des événements canPlay pas les deux.

L'événement peut jouer (équivalent à une image en charge)

```
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);
}
```

Afficher

La vidéo ne se jouera pas sur la toile. Vous devez le dessiner pour chaque nouvelle image. Comme il est difficile de connaître la fréquence d'images exacte et quand elle se produit, la meilleure approche consiste à afficher la vidéo comme si elle fonctionnait à 60 images par seconde. Si le taux de trame est inférieur, alors il suffit de rendre le même cadre deux fois. Si le taux de trame est plus élevé, il n'y a rien à voir pour voir les trames supplémentaires, alors nous les ignorons.

L'élément vidéo n'est qu'un élément d'image et peut être dessiné comme n'importe quelle image, vous pouvez le mettre à l'échelle, le faire pivoter, le faire pivoter, le découper et afficher uniquement les parties, le dessiner deux fois avec un mode composite global pour ajouter des effets comme l'éclaircir, l'écran, etc.

```
function updateCanvas(){
    ctx.clearRect(0,0,canvas.width,canvas.height); // Though not always needed
                                                   // you may get bad pixels from
                                                   // previous videos so clear to be
                                                   // safe
    // only draw if loaded and ready
    if(videoContainer !== undefined && videoContainer.ready){
        // find the top left of the video on the canvas
        var scale = videoContainer.scale;
        var vidH = videoContainer.video.videoHeight;
        var vidW = videoContainer.video.videoWidth;
        var top = canvas.height / 2 - (vidH /2 ) * scale;
        var left = canvas.width / 2 - (vidW /2 ) * scale;
        // now just draw the video the correct size
        ctx.drawImage(videoContainer.video, left, top, vidW * scale, vidH * scale);
        if(videoContainer.video.paused){ // if not playing show the paused screen
            drawPayIcon();
        }
    }
    // all done for display
    // request the next frame in 1/60th of a second
    requestAnimationFrame(updateCanvas);
}
```

Contrôle de la pause de lecture de base

Maintenant, nous avons la vidéo chargée et affichée, tout ce dont nous avons besoin est le contrôle de lecture. Nous allons le faire comme un clic pour jouer à l'écran. Lorsque la vidéo est

en cours de lecture et que l'utilisateur clique, la vidéo est suspendue. En pause, le clic reprend. Nous allons ajouter une fonction pour assombrir la vidéo et dessiner une icône de lecture (triangle)

```
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
}
```

Maintenant l'événement de pause de lecture

```
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);
```

Résumé

Jouer une vidéo est très facile en utilisant le canevas, l'ajout d'effet en temps réel est également facile. Il existe toutefois des limitations sur les formats, la façon dont vous pouvez jouer et chercher. MDN HTMLMediaElement est l'endroit où obtenir la référence complète à l'objet vidéo.

Une fois l'image dessinée sur le canevas, vous pouvez utiliser `ctx.getImageData` pour accéder aux pixels qu'elle contient. Ou vous pouvez utiliser `canvas.toDataURL` pour `canvas.toDataURL` un alambic et le télécharger. (Uniquement si la vidéo provient d'une source fiable et n'altère pas le canevas).

Notez que si la vidéo a du son, la lecture jouera également le son.

Joyeuses vidéos.

Capture de toile et enregistrer en tant que vidéo WebM

Création d'une vidéo WebM à partir d'images de canevas et de lecture dans un canevas, de téléchargement ou de téléchargement.

Exemple de capture et de lecture de toile

```
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
```

Plutôt que de mettre un effort énorme pour être rejeté, il s'agit d'une insertion rapide pour voir si elle est acceptable. Donnera des détails complets si accepté. Également inclure des options de capture supplémentaires pour de meilleurs taux de capture HD (supprimés de cette version, peut capturer HD 1080 à 50 images par seconde sur de bonnes machines.)

Ceci a été inspiré par Wammy mais est une réécriture complète avec encodage au fur et à mesure de votre méthodologie, réduisant considérablement la mémoire nécessaire lors de la capture. Peut capturer plus de 30 secondes de données de meilleure qualité, en manipulant des algorithmes.

> **Les** cadres de notes sont encodés en images WebP. Seul Chrome prend en charge l'encodage webp. Pour les autres navigateurs (Firefox et Edge), vous devrez utiliser un encodeur WebP tiers tel que Libwebp Javascript Encoding WebP via Javascript est lent. (inclura l'ajout du support d'images brutes webp si accepté).

L'encodeur webM inspiré de Whammy: un WebM Javascript en temps réel

```
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,
    }
})()
```

Lire Types de support et la toile en ligne: https://riptutorial.com/fr/html5-canvas/topic/3689/types-de-support-et-la-toile

Crédits

S. No	Chapitres	Contributeurs
1	Démarrer avec html5-canvas	almcd, Blindman67, Community, Daniel Dees, Kaiido, markE, ndugger, Spencer Wieczorek, Stephen Leppik, user2314737
2	Animation	Blindman67, markE
3	Chemin (syntaxe uniquement)	AgataB, markE
4	Collisions et intersections	Blindman67, markE
5	Compositing	Blindman67, markE
6	Des polygones	Blindman67, markE
7	Design réactif	Blindman67, markE, mnoronha
8	Effacer l'écran	bjanes, Blindman67, Kaiido, markE, Mike C, Ronen Ness
9	Glisser le chemin Formes et images sur la toile	markE
10	Graphiques et diagrammes	Blindman67, markE
11	Images	Blindman67, Kaiido, markE
12	Les chemins	Blindman67, markE
13	Manipulation de pixels avec "getImageData" et "putImageData"	markE
14	Naviguer le long d'un chemin	Blindman67, markE
15	Ombres	markE
16	Texte	almcd, Blindman67, markE, RamenChef
17	Transformations	Blindman67, markE

18	Types de support et la toile	Blindman67, Bobby, Kaiido