HTML5
FONCTIONNALITÉS MULTIMÉDIA

Le tag <canvas> : dessin, jeux, etc.

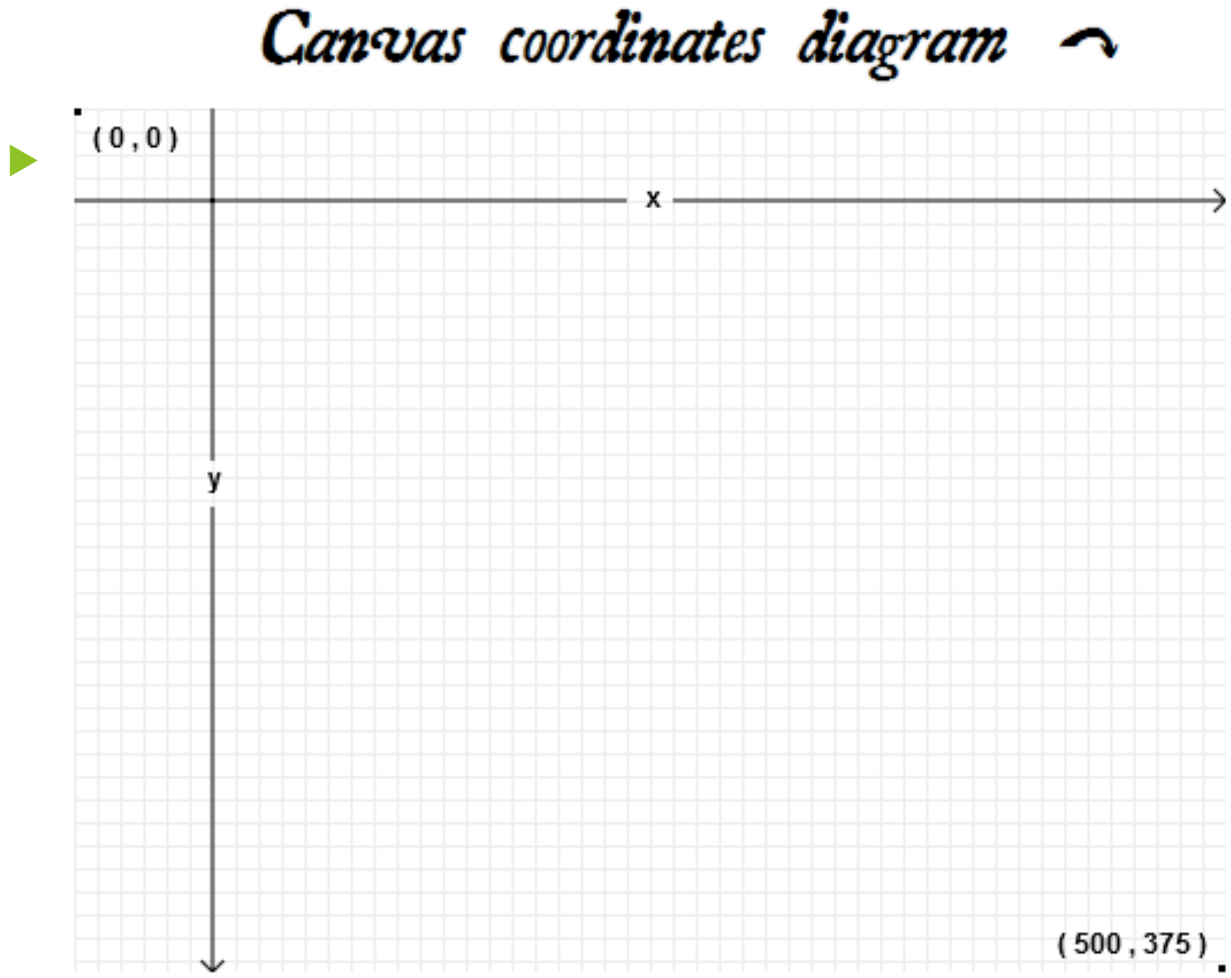
<canvas> permet de dessiner en bitmap

- Dessin bitmap,
- Permet :
  - dessin de formes (droites, courbes, rectangles, texte),
  - formes arrondies, support de la transparence,
  - Nombreux filtres (blur, etc),
  - Nombreuses fonctions de gestion d'images,
- Support de la 3D via WebGL, version web d'OpenGL

WebGL

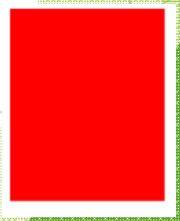
WebGL est un standard pour la programmation en 3D avec le navigateur comme plateforme. La spécification finale du standard a vu le jour en 2010 et est définie par le Khronos Group. Il permet de réaliser des animations, des interfaces ou des jeux en 3D fonctionnant à la fois en ligne et hors connexion.

<canvas> système de coordonnées



<canvas> exemple simple

```
<canvas id="myCanvas" width="200" height="200">
Votre navigateur ne supporte pas le tag
canvas.</canvas>
<script type="text/javascript">
  var canvas=document.getElementById('myCanvas');
  var ctx=canvas.getContext('2d');
  ctx.fillStyle='#FF0000';
  ctx.fillRect(0,0,80,100);
</script>
```



Principe d'utilisation

1. Déclarer le canvas, ne pas oublier l'attribut id et sa taille :
```
<canvas id="a" width="300" height="225"></canvas>
```
2. Récupérer dans une variable cet élément, dans du code JavaScript
```
var a_canvas = document.getElementById("a");
```
3. Récupérer le contexte graphique du canvas pour dessiner
```
var a_context = a_canvas.getContext("2d");
```
4. Dessiner à l'aide du contexte graphique
```
b_context.fillRect(50, 25, 150, 100);
```

- Le contexte est comme un "crayon" qui dessine dans l'objet qui a permis de l'obtenir

Principes du dessin

- Prenons l'exemple du rectangle des exemples précédents
  - **fillStyle** est une propriété du contexte, similaire à du CSS. Peut prendre comme valeur une couleur, une pattern (texture) ou un gradient (dégradé). Par défaut couleur = noir.

    Toute le dessin en mode « plein » se fera avec cette propriété activée : les rectangles pleins seront noirs, les cercles pleins seront noirs, etc. Tant qu'on ne modifie pas cette propriété, tous les ordres de dessin la prendront en compte, c'est comme une variable globale du contexte

Principe du dessin

- **fillRect(x, y, width, height)** : dessine un rectangle plein. On indique le point en haut à gauche, la largeur et la hauteur. Utilise le fillStyle courant.
- **strokeStyle** est comme fillStyle mais pour les formes « en fil de fer », non pleines. Ex : un cercle dont on ne veut que le contours. Mêmes valeurs possibles que pour fillStyle.
- **strokeRect(x, y, width, height)** : idem fillRect mais rectangle en fil de fer, non plein. Utilise le strokeStyle courant.
- **clearRect(x, y, width, height)** : efface le rectangle courant (couleur = noir transparent, en fait le remet dans l'état initial)

Faire « reset » d'un <canvas>

- On peut remettre le canvas dans l'état initial aussi en resettant la largeur ou sa hauteur

```
var b_canvas =
document.getElementById("b");
b_canvas.width = b_canvas.width;
```

- Cela efface le canvas mais remet aussi son contexte dans l'état initial

Principe du dessin de lignes par chemin

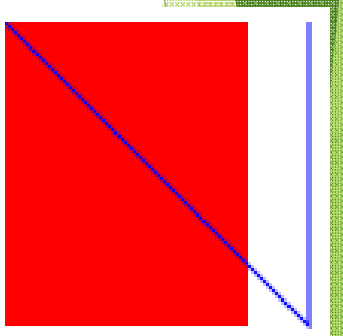
- Contrairement à de nombreuses approches, pour dessiner des formes dans un canvas on utilise la notion de chemin (path).
  - On met le crayon à un endroit donné (moveTo)
  - On choisit la couleur et ce que l'on veut dessiner (strokeStyle ou fillStyle)
  - On dit jusqu'où on veut dessiner (lineTo par exemple)
  - Encore un coup (lineTo, on trace deux lignes jointives…)
  - On dessine (stroke ou fill)
  - On aurait pu choisir la couleur juste avant le stroke ou le fill

Exemple de tracés de lignes

```
// Lignes verticales

for (var x = 0.5; x < 500; x += 10) {
  context.moveTo(x, 0);
  context.lineTo(x, 375);
}

// Lignes horizontales

for (var y = 0.5; y < 375; y += 10) {
  context.moveTo(0, y);
  context.lineTo(500, y);
}

// Rien n'a été encore dessiné !!!
context.strokeStyle = "#0000FF"; // en bleu
context.stroke(); // on dessine !
```

Exemple complet

```
<!DOCTYPE html>
<html>
<body>
<canvas id="myCanvas">Your browser does not support
the canvas tag.</canvas>
<script type="text/javascript">
  var canvas=document.getElementById('myCanvas');
  var ctx=canvas.getContext('2d');
  ctx.fillStyle='#FF0000';
  ctx.fillRect(0,0,80,100);

  ctx.moveTo(0,0);
  ctx.lineTo(100, 100);
  ctx.lineTo(100,0);
  ctx.strokeStyle = "#0000FF";
  ctx.stroke();
</script>
</body>
</html>
```



Attributs du dessin « fil de fer »

- Largeur du trait : **context.lineWidth=10;**
- Couleur du trait : **context.strokeStyle=couleur**, gradient ou texture (exe plus loin)
  - context.strokeStyle = "#ff0000";
  - context.strokeStyle = "red";
- Arrondis aux extrêmités : **context.lineCap=[value];**
  - context.lineCap = "butt";
  - context.lineCap = "round";
  - context.lineCap = "square";

Gestion des ombres

- Quatre attributs pour cela :
  - `context.shadowColor`
  - `context.shadowBlur`
  - `context.shadowOffsetX`
  - `context.shadowOffsetY`
- Exemple :

```
context.rect(188, 40, 200, 100);
context.fillStyle = "#8ED6FF";
context.shadowColor = "#bbbbbb";
context.shadowBlur = 20; // floutage de l'ombre
context.shadowOffsetX = 15; // décalage en X
context.shadowOffsetY = 15; // décalage en Y
context.fill();
```

Exercices

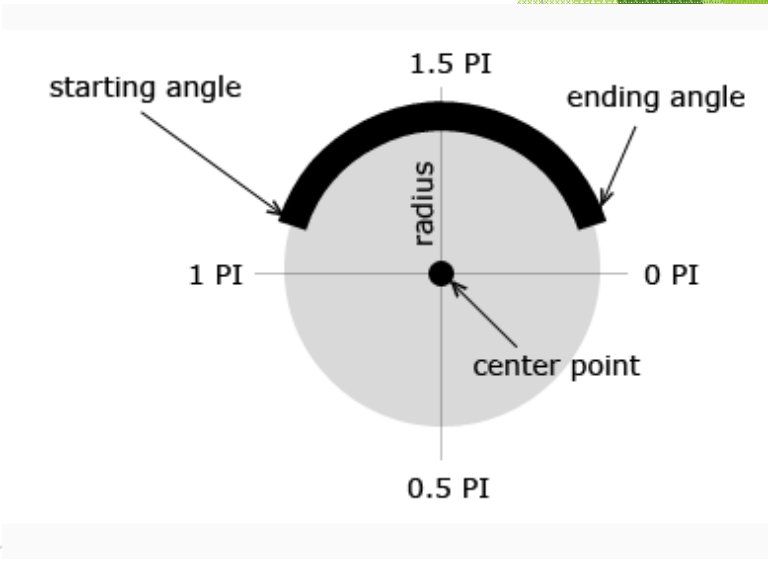
1. Créer un canvas contenant une ligne similaire à celle de l’image avec une ombre

2. Dessiner ce bateau :

Dessiner un Arc (de cercle)

```
context.arc(centreX, centreY, rayon,
            angleDepart, angleArrivee,
            sensInverseAiguillesMontre);
```

- Les angles en radians, le dernier paramètre est booléen. True indique qu'on travaille dans le sens inverse des aiguilles d'une montre quand on trace.
- Le sens n'est pas le sens trigonométrique classique.



Piège du sens de progression des angles

```
context.arc(centerX, centerY, radius, 0, Math.PI/4, false);
context.lineWidth = 15;
context.strokeStyle = "black"; // line color
context.stroke();
```

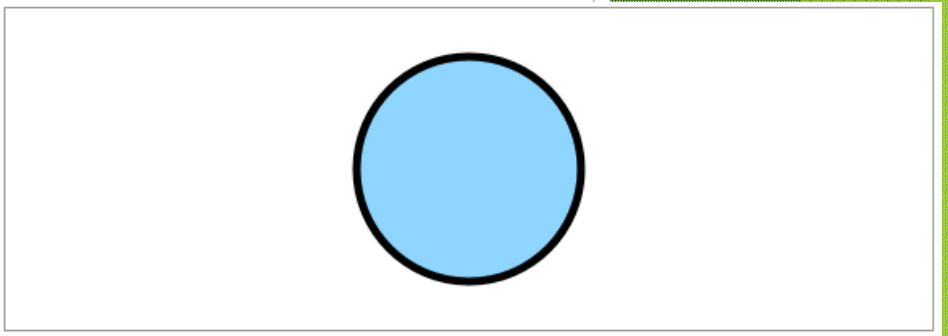
- Donne :

- Mais :

```
context.arc(centerX, centerY, radius, 0, Math.PI/4, true);
context.lineWidth = 15;
context.strokeStyle = "black"; // line color
context.stroke();
```

- Donne :

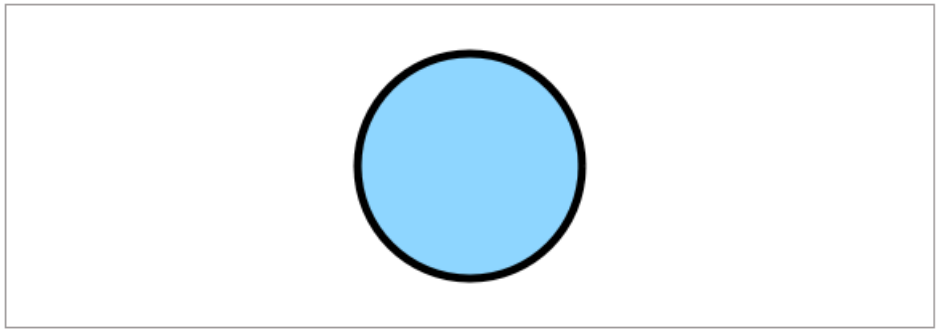
Dessiner un cercle



```
<!DOCTYPE HTML>
<html>
<head>
<style>
  #myCanvas {
    border: 1px solid #9C9898;
  }
</style>
<script>
  window.onload = function(){
    var canvas = document.getElementById("myCanvas");
    var context = canvas.getContext("2d");
    var centerX = canvas.width / 2;
    var centerY = canvas.height / 2;
    var radius = 70;

    context.beginPath();
    context.arc(centerX, centerY, radius, 0, 2 * Math.PI, false);
    context.fillStyle = "#8ED6FF"; // Cercle bleu plein
    context.fill(); // on dessine tout ce qui est en attente en plein
    context.lineWidth = 5;         // Cercle fil de fer noir, largeur trait 5
    context.strokeStyle = "black";

    context.closePath();
    context.stroke(); // on dessine tout ce qui est en attente en fil de fer

  };
</script>
</head>
```

Dessiner un cercle (fin)

```
<body>
  <canvas id="myCanvas" width="578"
height="200"></canvas>
</body>
</html>
```



- Démo

Dessiner un demi-cercle

- Il suffit de remplacer dans l'exemple précédent

```
context.arc(centerX, centerY, radius, 0, 2*Math.PI, false);
```

- Par :

```
context.arc(centerX, centerY, radius, 0, Math.PI, false);
```

- Le dernier paramètre, dans le cas d'un cercle partiel, indique s'il vaut « true » que l'on désire dessiner non pas en allant dans le sens des aiguilles d'une montre, mais dans le sens inverse. On aurait eu le demi-cercle allant de 0 à PI par le haut (le demi-cercle supérieur)

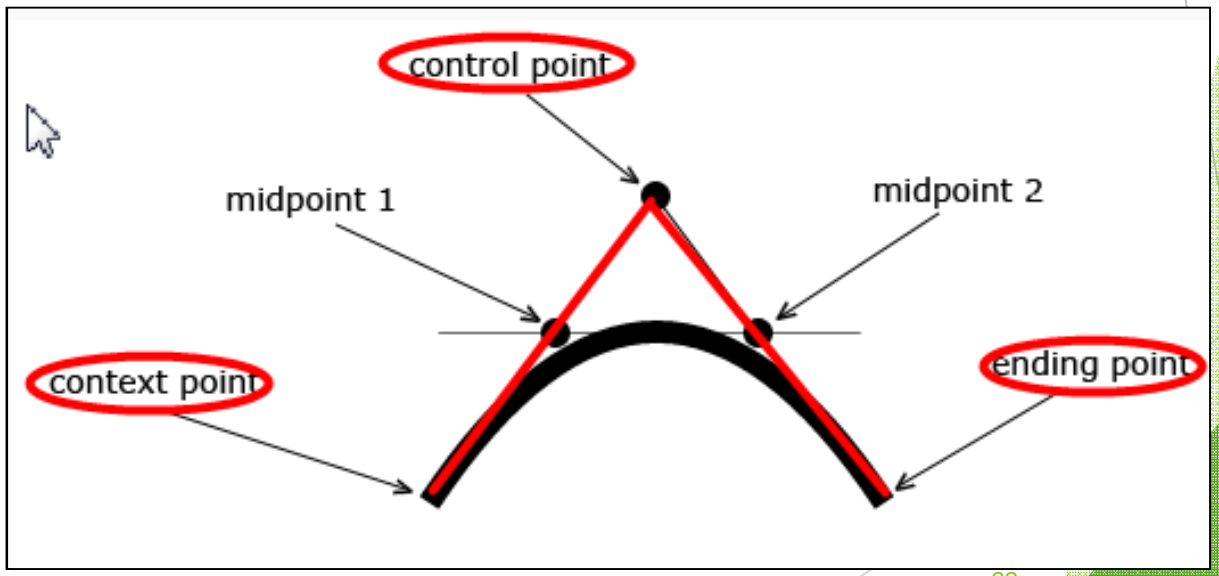
Exercices

- Dessiner un arc en ciel

Courbes quadriques

```
context.moveTo(startX, startY);
context.quadraticCurveTo(controlX, controlY,
                         endX, endY);
```

- Ci-dessous, le « context point » est le dernier point du chemin en cours.
- Démonstration

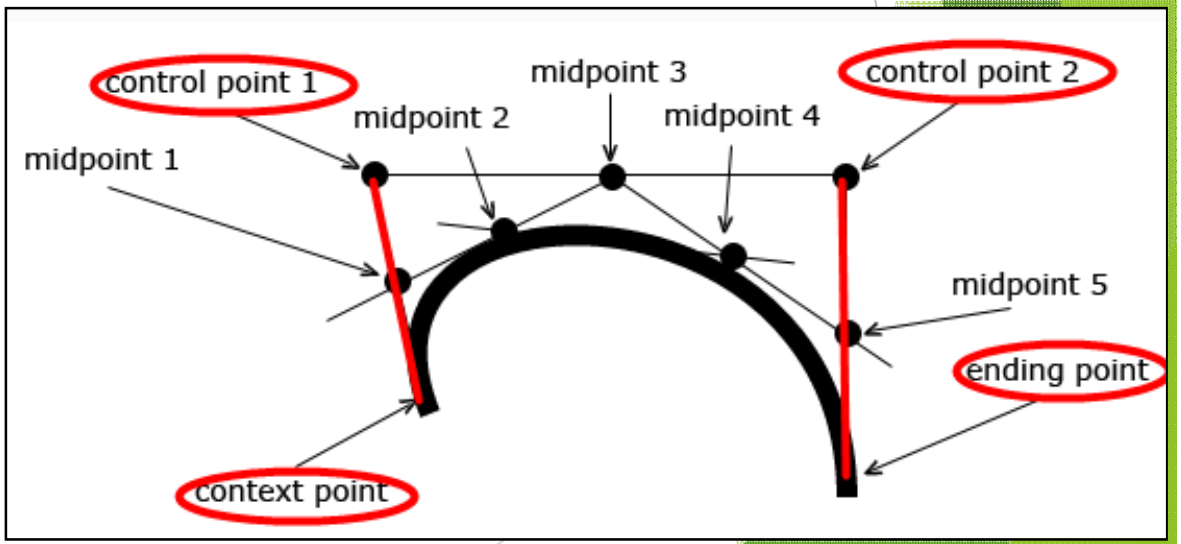


Courbes de Bezier

```
context.moveTo(startX, startY);
context.bezierCurveTo(controlX1, controlY1,
                      controlX2, controlY2,
                      endX, endY);
```

Ci-dessous, le « context point » est le dernier point du chemin en cours.

- Démonstration



Formes personnalisées

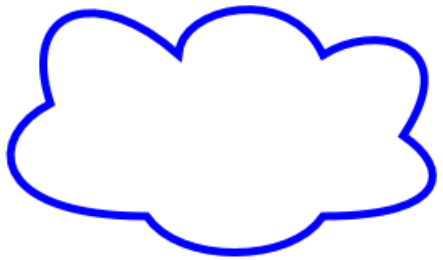
- On peut créer des formes personnalisées à l'aide des méthodes beginPath() et closePath() du contexte.

```
var canvas = document.getElementById("myCanvas");
var context = canvas.getContext("2d");
context.beginPath(); // début de la forme personnalisée

... Ici des ordres de dessin

context.closePath(); // fin de la forme

// ensuite on dessine la forme, par exemple :
context.lineWidth = 5;
context.strokeStyle = "#0000ff";
context.stroke();
```

Formes personnalisées (suite)

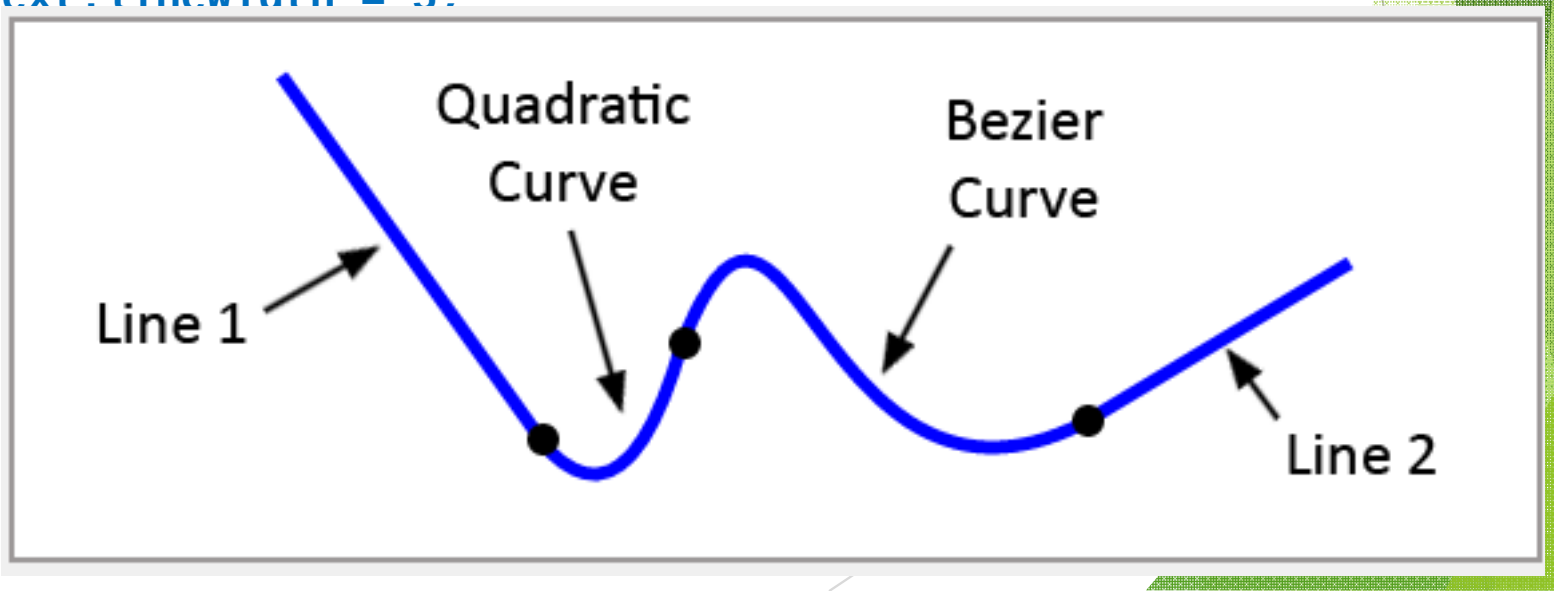
- Dans la partie entre le beginPath() et le closePath() on peut utiliser
  - lineTo(), arcTo(), quadraticCurveTo(), ou bezierCurveTo()

```
context.beginPath(); // début d'une forme
personnalisée
context.moveTo(170, 80);
context.bezierCurveTo(130, 100, 130, 150, 230, 150);
context.bezierCurveTo(250, 180, 320, 180, 340, 150);
context.bezierCurveTo(420, 150, 420, 120, 390, 100);
context.bezierCurveTo(430, 40, 370, 30, 340, 50);
context.bezierCurveTo(320, 5, 250, 20, 250, 50);
context.bezierCurveTo(200, 5, 150, 20, 170, 80);
context.closePath(); // fin de la forme
context.lineWidth = 5;
context.strokeStyle = "#0000ff";
context.stroke();
```



Formes personnalisées (suite)

```
context.beginPath();
context.moveTo(100, 20);
context.lineTo(200, 160);
context.quadraticCurveTo(230, 200, 250, 120);
context.bezierCurveTo(290, -40, 300, 200, 400,
150); context.lineTo(500, 90);
context.endPath();
context.lineWidth = 5;
cont
cont
```



Dessin de dégradés

- On appelle un dégradé « gradient » en anglais.
- On peut créer des objets de ce type pour des dégradés « linéaires » ou « radiaux »
- Exemple de forme avec dégradé radial et linéaire comme couleur de remplissage

- On peut cliquer ces exemples pour voir les démos

Création d'un gradient linéaire

- Gradient linéaire

```
var my_gradient =
context.createLinearGradient(0, 0, 300, 0);
```

- On donne le point de départ (0,0) et le point d'arrivée (300, 0), situé 300 pixels plus à droite que le point de départ, dans cet exemple
- Puis on donne un ensemble de couleurs que le dégradé devra interpoler :

```
my_gradient.addColorStop(0, "black");
my_gradient.addColorStop(1, "white");
```

Utilisation d’un gradient

- Il reste à mettre ce gradient comme valeur d’un fillStyle ou d’un strokeStyle

```
context.fillStyle = my_gradient;
context.fillRect(0, 0, 300, 225);
```

- En changeant les points de départ et d’arrivée du gradient on peut faire un dégradé vertical, en changeant les couleurs un dégradé arc en ciel etc.

Création d'un gradient radial

- On utilise :

```
var grd=
 context.createRadialGradient(startX, startY, startRadius,
                              endX,   endY,   endRadius);
```

- On définit deux cercles imaginaires, un pour le départ et un pour l'arrivée
- On ajoute des « stop colors » comme pour le gradient linéaire (1er paramètre entre 0 et 1)

```
grd.addColorStop(0, "red");
grd.addColorStop(0.17, "orange");
grd.addColorStop(0.33, "yellow");
grd.addColorStop(0.5, "green");
grd.addColorStop(0.666, "blue");
grd.addColorStop(1, "violet");
```

Textures

- On peut utiliser également des textures avec strokeStyle et fillStyle

```
<head>
<script>
  window.onload = function(){
      var canvas = document.getElementById("myCanvas");
      var context = canvas.getContext("2d");
      var imageObj = new Image();


      // Fonction de callback asynchrone appelée par le chargement de l'image
      imageObj.onload = function(){
         // On entre ici lorsque l'image est chargée
         var pattern = context.createPattern(imageObj, "repeat");
         context.rect(10, 10, canvas.width - 20, canvas.height - 20);
         context.fillStyle = pattern;
         context.fill();
      };

      imageObj.src = "wood-pattern.png";
  };
</script>
</head>

<body>
  <canvas id="myCanvas" width="578" height="200"></canvas>
</body>
```

Textures

- La création de la texture :
  `context.createPattern(imageObj, "repeat");`
- Valeurs possibles pour le dernier paramètre : repeat (par défaut), repeat-x, repeat-y, ou no-repeat
- Démonstration

Images

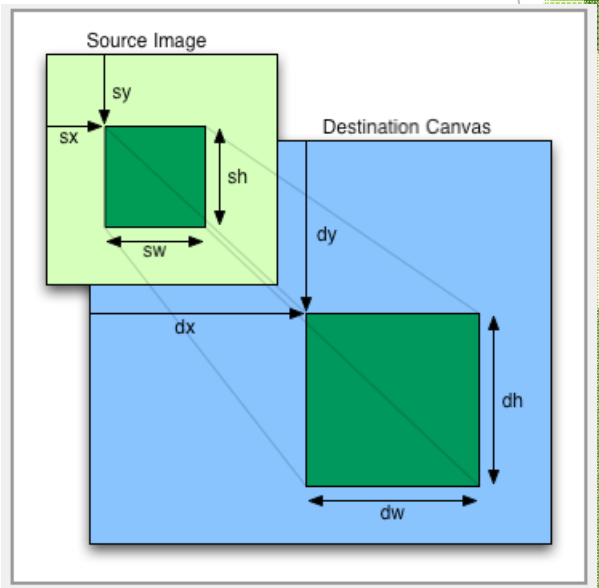
- En fait, on vient de voir un exemple de chargement d'image asynchrone

```
var imageObj = new Image();


// Fonction de callback asynchrone appelée par le chargement
de l'image
imageObj.onload = function(){
   // On entre ici lorsque l'image es
en 0,0 ici
   context.drawImage(imageObj, 0, 0);
};

imageObj.src = "darth-vader.jpg";
```

- Démonstration

Nombreuses options pour drawImage

- Il existe plusieurs variantes pour drawImage
  - drawImage(img, x, y) : dessine l'image à la position x,y, l'image conserve sa taille
  - drawImage(img, x, y, tailleX, tailleY) : l'image est retaillée au moment où elle est dessinée. Démonstration.
  - drawImage(img, sx, sy, sw, sh, dx, dy, dw, dh); permet de ne dessiner qu'une partie de l'image source dans le canvas, tout en spécifiant la taille. Démonstration.



Reference: www.whatwg.org

Dessiner l'image d'une vidéo dans un canvas avec drawImage()

```
<video id="video1" height="115px" width="140px
autoplay="true" controls="true"  src="snoop.mp4">…</video>


<canvas id="canvasDest"
             height="115px"  width="140px"></canvas>

<script type="text/javascript">
  var video1 = document.getElementById("video1");
  var videoWidth = video1.width;
  var videoHeight = video1.height;
  var canvasDest = document.getElementById("canvasDest");
  var ctxDest = canvasDest.getContext("2d");
  ctxDest.drawImage(video1, 0, 0, videoWidth, videoHeight);

</script>
```

- On peut faire cela à la fréquence vidéo !
- Démonstration

Manipulation des pixels d'une image

- A partir du contexte d'un canvas on peut obtenir l'ensemble des pixels qui le composent.

```
bufferImage= ctx.getImageData(0, 0, width, height);
```

- La variable résultat est un tableau à 2 dimensions dont chaque élément est un pixel (R,G,B,A). Exemple pour accèder au ième pixel

```
var r = bufferImage[i + 0];
var g = bufferImage[i + 1];
var b = bPixels[i + 2];
var a = bufferImage[i + 3];
// on peut tester si le pixel est vert
if (r == 0 && g == 255 && b == 0 && a == 255) {…}
```

Manipulation des pixels d'une image (suite)

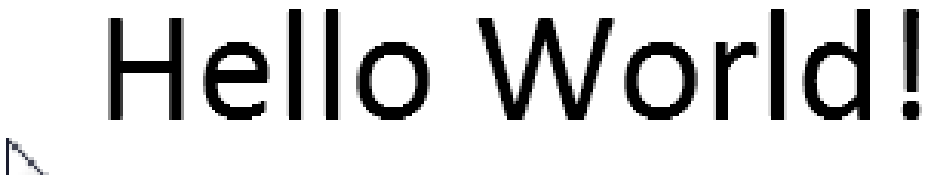
- Application à l'incrustation vidéo… on extrait en temps réel le buffer image d'une vidéo,
- On extrait celui d'une autre vidéo,
- On regarde chaque pixel du buffer destination, s'il est vert, on le remplace par le même pixel du buffer source.
  - Résultat : on incruste la source dans la destination !
- Démonstration (chrome, IE, car vidéos en H.264)
- Explications

Dessin de texte

- Cas simple :

```
context.font = "40pt Calibri";
context.fillText("Hello World!", x, y);
```

- On peut utiliser les



Dessin de texte, exemples


```
context.font = "60pt Calibri";
context.lineWidth = 3;
context.strokeStyle = "blue";
context.strokeText("Hello World!", x,
y);
```

- Mais on peut faire du fill et du stroke !


```
context.fillStyle = "red";
context.fillText("Hello World!", x,
y);
context.strokeStyle = "blue";
context.strokeText("Hello World!", x,
y);
```

Alignement de texte

- On peut « aligner » le texte dessiné par rapport à une ligne verticale imaginaire qui passe par la coordonnée x du (x,y) utilisé pour indiquer l'endroit où on dessine avec strokeText() ou fillText()
- context.textAlign=[value];
- Valeurs possibles : start, end, left, center, ou right
- Démonstration

Texte, concepts avancés

- Non étudiés :
  - Baseline, pour alignement des polices sur une ligne horizontale (pour les lettres montantes et descendantes)
  - Calcul des boîtes englobantes (pour centrer un texte dans une zone par exemple) pour la justification (text wrap) d'un long texte à dessiner dans une zone rectangulaire.

Détection d'événements dans un canvas

- Le canvas peut générer de nombreux événements, notamment pour capturer les clics souris et les déplacements
  - Etudiés dans le TP sur le programme de paint

```
<script>
  function writeMessage(canvas, message){
    var context = canvas.getContext('2d');
    context.clearRect(0, 0, canvas.width, canvas.height);
    context.font = '18pt Calibri';
    context.fillStyle = 'black';
    context.fillText(message, 10, 25);
  }
  window.onload = function(){
    var canvas = document.getElementById('myCanvas');
    var context = canvas.getContext('2d');
    canvas.addEventListener('mousemove', function(evt){
        var message = "Mouse position: " + evt.clientX + ","
                    + evt.clientY;
         writeMessage(canvas, message);
       }, false);
   };
</script>
```

Détection d'événements dans un canvas

- Valeurs possibles très nombreuses, notamment mousedown, mouseup, mousemove mais aussi tous les événements de type Keyboard ou Mouse
  - Voir http://www.w3schools.com/html5/html5_ref_eventattributes.asp

Intéressant pour aller plus loin

- Le site http://www.html5canvastutorials.com/ possède de très nombreux tutoriaux
  - Traitement d'image (filtres, etc)
  - Animation via window.requestAnimFrame (équivalent du repaint() de Java)
  - Transformations géométriques
- Il existe de nombreuses librairies et framework pour utiliser des canvas avec une approche « de haut niveau »
  - processingJS.org
  - Outils commerciaux : bibliothèques JS, outils Adobe, Unity3D, etc.