Programmation WEB

HTML5

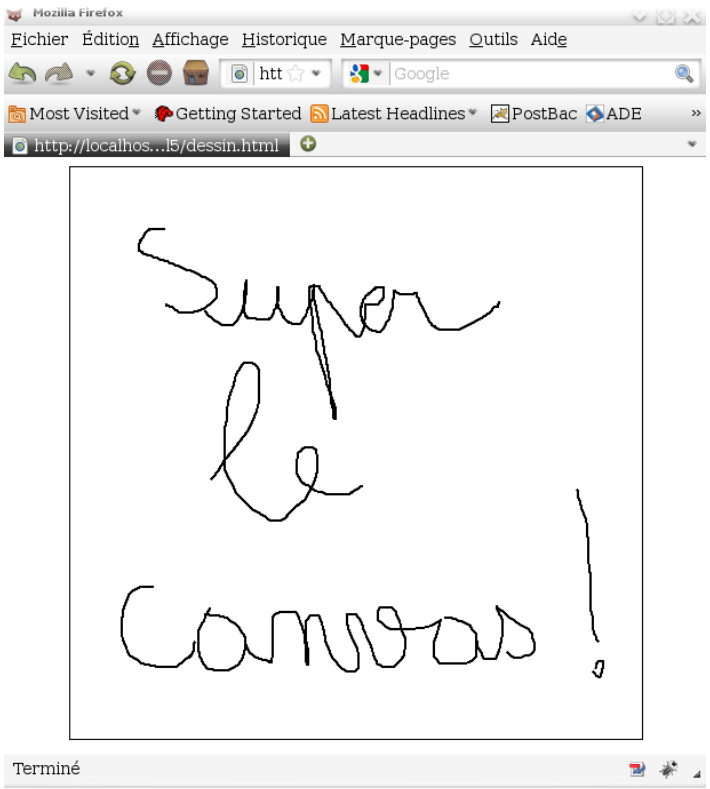
Programmation licence

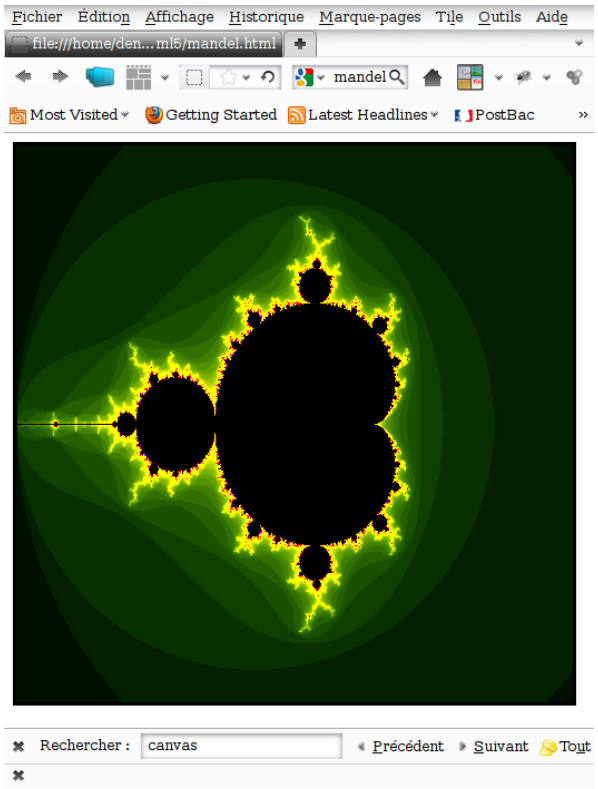
IUT de Fontainebleau

31 janvier 2014

Introduction

- Nouvel élément qui permet de déssiner à l'aide de javascript.
- Fait partie d'HTML 5.
- Implanté par la plupart des navigateurs.
- Alternative à Flash, et au SVG.

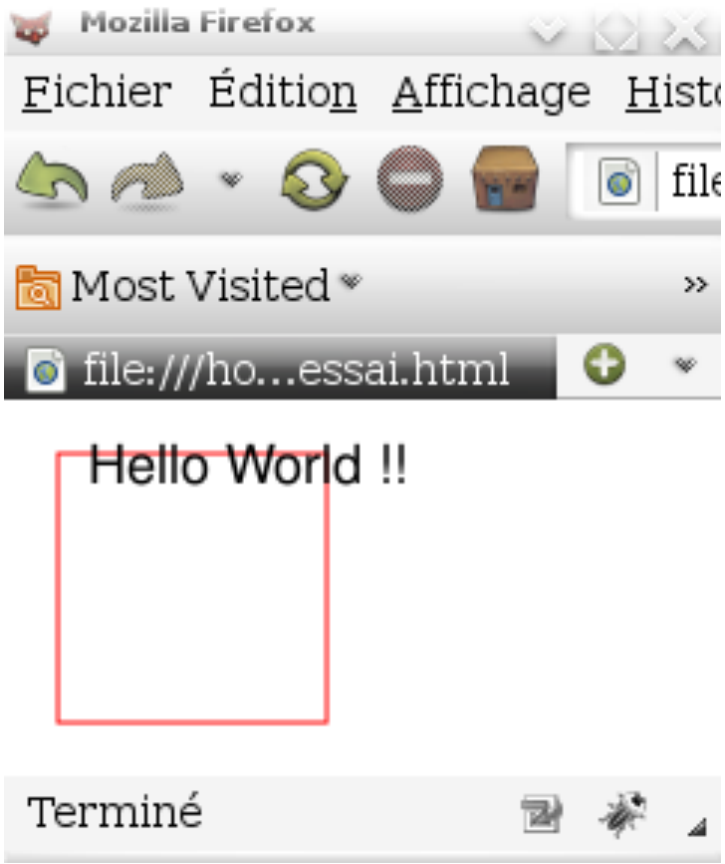




Un petit exemple ...

```
<html>
  <head>
    <script>
      function init()
      {
        ctx=document.getElementById("cs").getContext("2d");
        ctx. beginPath();
        ctx.moveTo(10,10);
        ctx.lineTo(100,10);
        ctx.lineTo(100,100);
        ctx.lineTo(10,100);
        ctx.lineTo(10,10);
        ctx.closePath();
        ctx.strokeStyle="#ff0000";
        ctx.stroke();
        ctx.font="0.9em FreeSans"
        ctx.fillText("Hello word !!!",20,20);
      }
    </script>
  </head>
  <body onload="init()">
    <canvas id="cs" height="110" width="200"></canvas>
  </body>
</html>
```

... qui donne


Mozilla Firefox
Fichier Édition Affichage
Most Visited
file:///ho...essai.html
Hello World !!
Terminé

Inclusion

- On l'inclut dans la page avec la balise <canvas>

```
<canvas id="monCanvas" width="300" height="150"></canvas>
```

- L'identifiant permet ensuite de le manipuler :

```
var cs=document.getElementById('monCanvas');
```

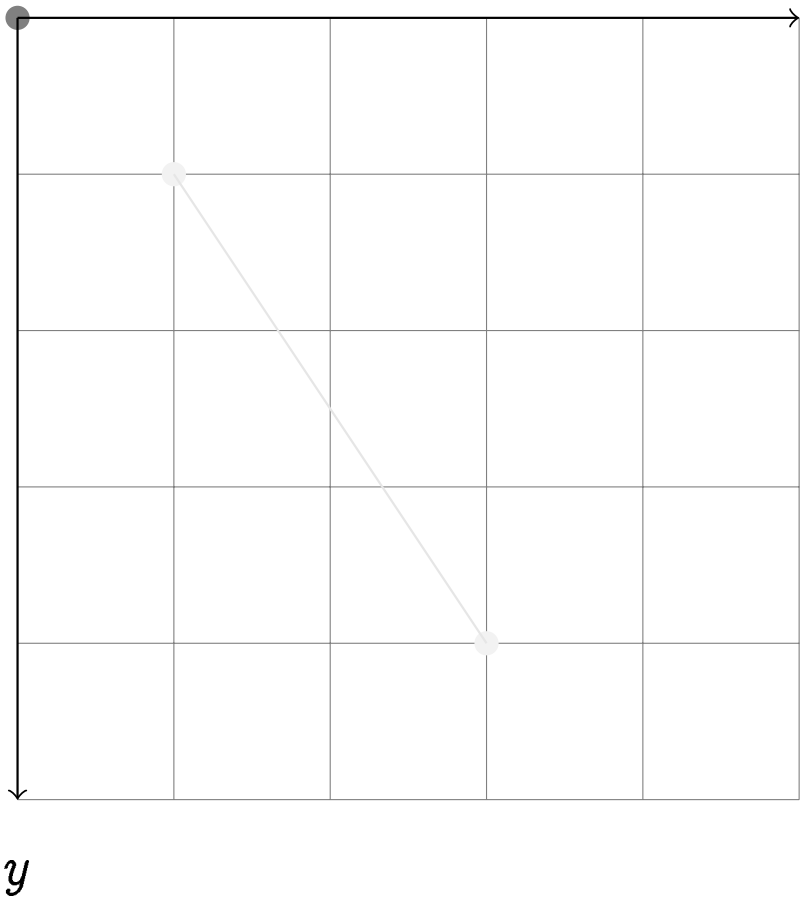
- La classe correspondante HTMLCanvasElement :

```
attribute unsigned long width;
attribute unsigned long height;
DOMString toDataURL(in optional DOMString type,
    in any... args);
object getContext(in DOMString contextId);
```

CanvasRenderingContext2D

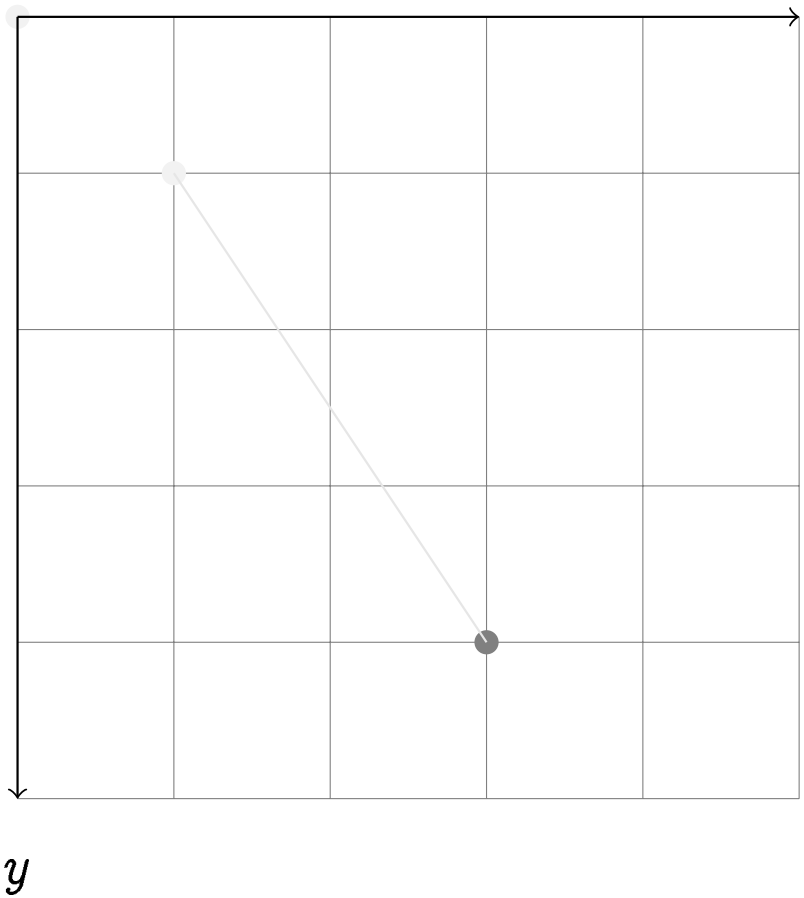
- Chaque canvas utilise à la fois un et un seul contexte graphique, retourné par la méthode `getContext()` avec l'argument 2d.
- Un contexte graphique mémorise un certain nombre de paramètres nécessaires aux différentes primitives de dessin (même idée que dans la Xlib, openGL, etc ...)
  1. La matrice de transformation (le repère dans lequel on dessine)
  2. La region de clipping.
  3. style de dessin, de remplissage, épaisseur du trait, police, etc ...
- La méthode `save()` empile le contexte courant
- La méthode `restore()` dépile le contexte au sommet, en écrasant la valeur courante.

Repère



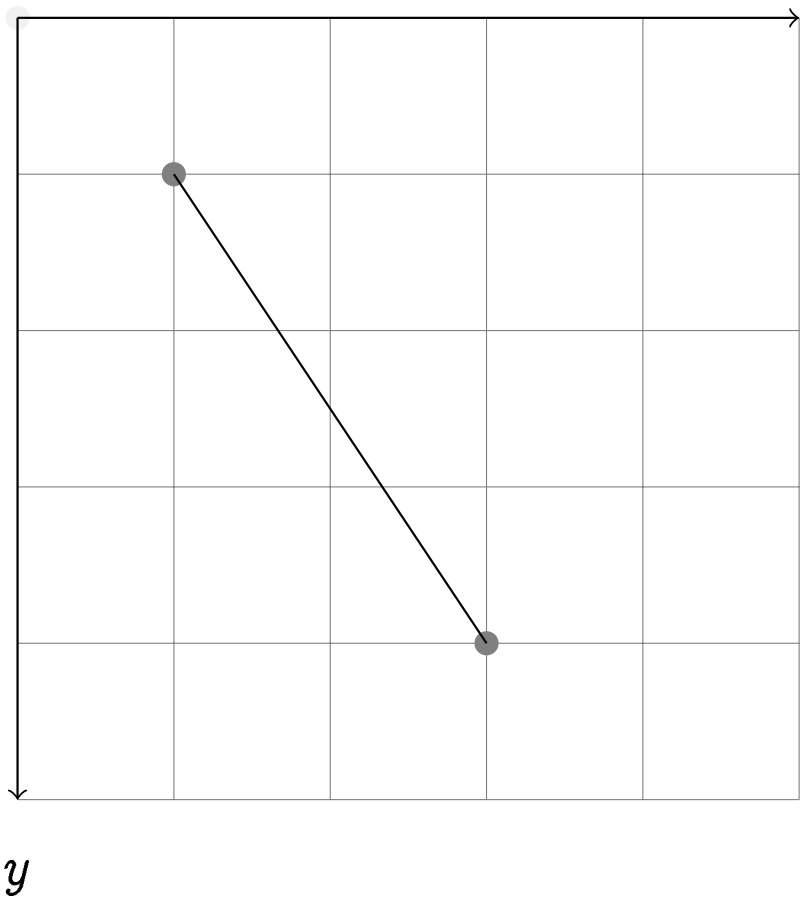
- moveTo(3,4)
- lineTo(1,1)

Repère



- moveTo(3,4)
- lineTo(1,1)

Repère



- moveTo(3,4)
- lineTo(1,1)

Couleurs et styles

```
// colors and styles
attribute any strokeStyle; // (default black)
attribute any fillStyle; // (default black)
CanvasGradient createLinearGradient(in float x0, in float y0,
        in float x1, in float y1);
CanvasGradient createRadialGradient(in float x0, in float y0,
        in float r0, in float x1, in float y1, in float r1);
CanvasPattern createPattern(in HTMLImageElement image,
        in DOMString repetition);
CanvasPattern createPattern(in HTMLCanvasElement image,
        in DOMString repetition);
CanvasPattern createPattern(in HTMLVideoElement image,
        in DOMString repetition);
```

Attributs de tracé

```
// line caps/joins
attribute float lineWidth; // (default 1)
attribute DOMString lineCap; // "butt",
"round", "square" (default "butt")
    attribute DOMString lineJoin; // "round",
"bevel", "miter" (default "miter")
    attribute float miterLimit; // (default 10)
// shadows
attribute float shadowOffsetX; // (default 0)
attribute float shadowOffsetY; // (default 0)
attribute float shadowBlur; // (default 0)
attribute DOMString shadowColor; // (default transparent black)
```

Formes simples

- Elles n'affectent pas le path courant.

```
// rects
  void clearRect(in float x, in float y, in float w, in float h);
  void fillRect(in float x, in float y, in float w, in float h);
  void strokeRect(in float x, in float y, in float w, in float h);
```

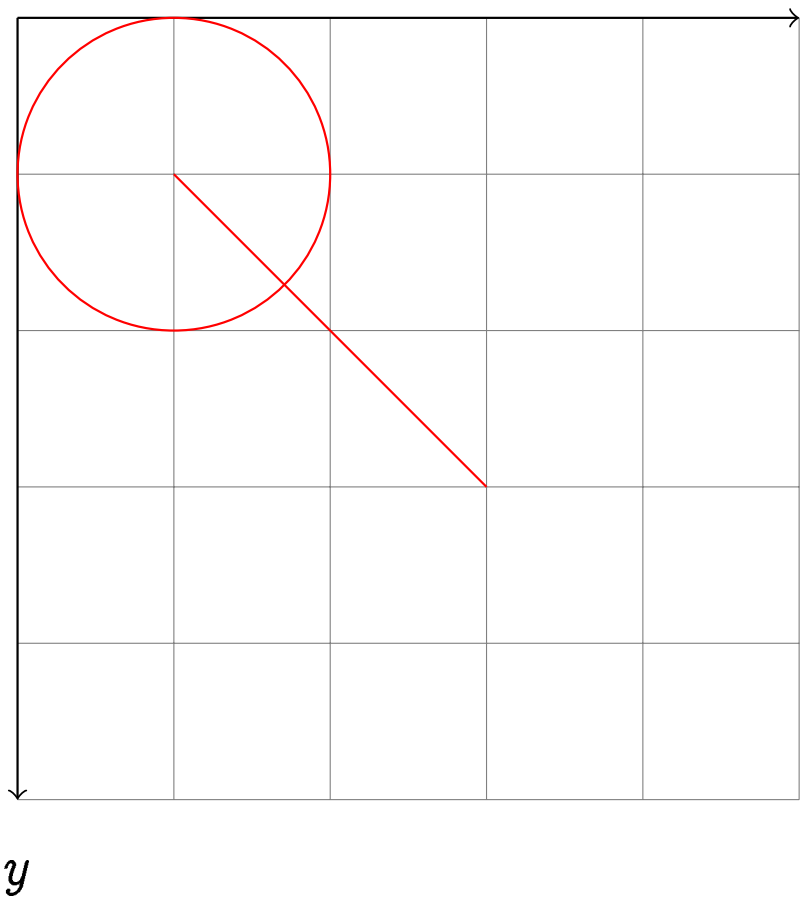
- Pour les autres formes, on a recours à la notion de chemin.

Chemins

- Les chemins sont une liste de sous-chemins.
- Les sous-chemins sont un ou plusieurs points connectés par des segments ou des courbes.
- Le contexte graphique a toujours un path courant.

`beginPath()`	initialisation du chemin courant
`closePath()`	ferme le chemin courant et en commence un nouveau
`moveTo(x,y)`	démarre un nouveau sous-chemin au point
`fill()`	remplit le chemin
`stroke()`	dessine le chemin
`lineTo(x,y)`	ajoute un segment à partir du point précédent
`rect(x,y,w,h)`	ajoute un sous-chemin (fermé) rectangulaire
`arc(x,y,...)`	ajoute un sous-chemin circulaire
`arcTo(...)`	ajoute un sous-chemin connectant deux points par un arc de cercle
`bezierCurveTo`	courbe de bézier (cubique)
`quadraticCurveTo`	courbe de bézier (quadratique)

Exemple




```
ctx.strokeStyle='rgb(255,0,0)';
ctx.beginPath();
ctx.arc(10,10,10,0,
    Math.PI*2,false);
ctx.moveTo(10,10);
ctx.lineTo(30,30);
ctx.stroke();
```

Chemins

```
// path API
  void beginPath();
  void closePath();
  void moveTo(in float x, in float y);
  void lineTo(in float x, in float y);
  void quadraticCurveTo(in float cpx, in float cpy,
          in float x, in float y);
  void bezierCurveTo(in float cp1x, in float cp1y,
          in float cp2x, in float cp2y, in float x, in float y);
  void arcTo(in float x1, in float y1, in float x2,
          in float y2, in float radius);
  void rect(in float x, in float y, in float w,
          in float h);
  void arc(in float x, in float y, in float radius,
          in float startAngle, in float endAngle,
          in boolean anticlockwise);
  void fill();
  void stroke();
  void clip();
  boolean isPointInPath(in float x, in float y);
```

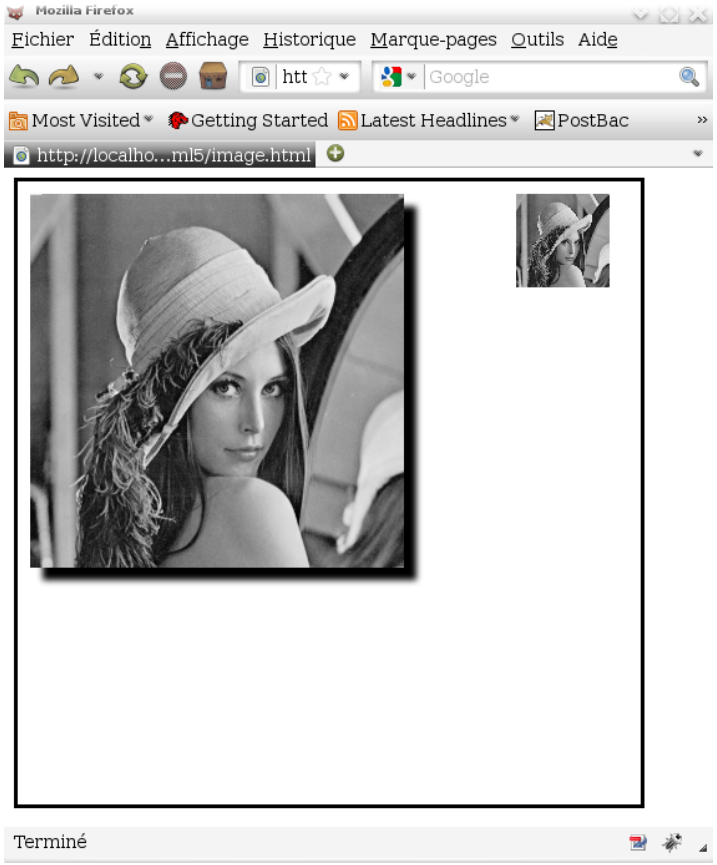
Dessiner du texte

```
// text
attribute DOMString font; // (default 10px sans-serif)
attribute DOMString textAlign; // "start", "end", "left",
        "right", "center" (default: "start")
attribute DOMString textBaseline; // "top", "hanging",
        "middle", "alphabetic", "ideographic",
            "bottom" (default: "alphabetic")
void fillText(in DOMString text, in float x, in float y,
        in optional float maxWidth);
void strokeText(in DOMString text, in float x, in float y,
        in optional float maxWidth);
TextMetrics measureText(in DOMString text);
```

Dessin d'images

```
// drawing images
void drawImage(in HTMLImageElement image, in float dx, in float dy,
        in optional float dw, in float dh);
void drawImage(in HTMLImageElement image, in float sx, in float sy,
        in float sw, in float sh, in float dx, in float dy,
        in float dw, in float dh);
void drawImage(in HTMLCanvasElement image, in float dx, in float dy,
        in optional float dw, in float dh);
void drawImage(in HTMLCanvasElement image, in float sx, in float sy,
        in float sw, in float sh, in float dx, in float dy,
        in float dw, in float dh);
void drawImage(in HTMLVideoElement image, in float dx, in float dy,
        in optional float dw, in float dh);
void drawImage(in HTMLVideoElement image, in float sx, in float sy,
        in float sw, in float sh, in float dx, in float dy,
        in float dw, in float dh);
```

Un exemple

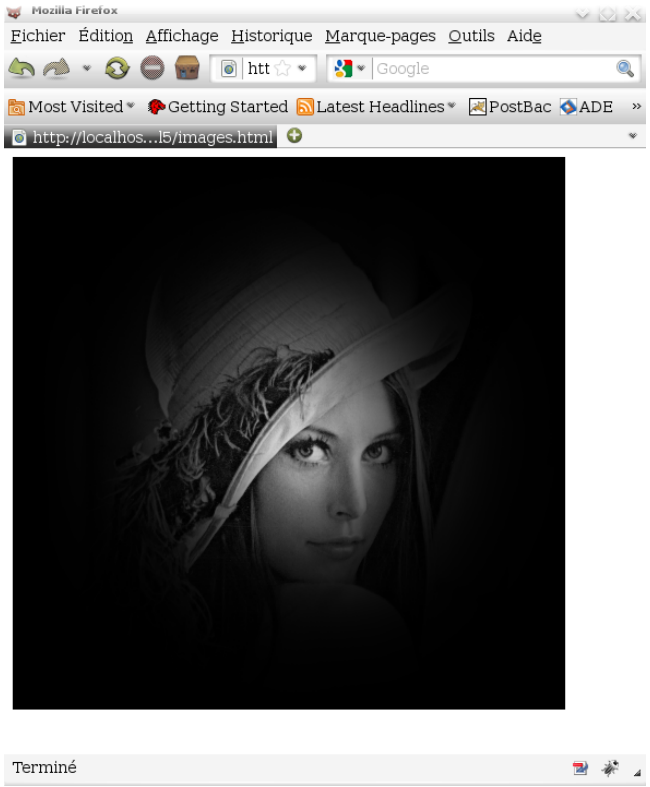



```
var linux=new Image();
linux.src="lena.jpg";
var canvas=document.getElementById(
    'cs');
var ctx=canvas.getContext("2d");

ctx.drawImage(linux,400,10,75,75);
ctx.shadowOffsetX = 10;
ctx.shadowOffsetY = 10;
ctx.shadowBlur = 10;
ctx.shadowColor = "black";
ctx.drawImage(linux,10,10);
```

Manipulation de pixels

```
// pixel manipulation
  ImageData createImageData(in float sw, in float sh);
  ImageData createImageData(in ImageData imagedata);
  ImageData getImageData(in float sx, in float sy, in float sw,
          in float sh);
  void putImageData(in ImageData imagedata, in float dx,
          in float dy, optional in float dirtyX, in float dirtyY,
          in float dirtyWidth, in float dirtyHeight);
};
interface ImageData {
  readonly attribute CanvasPixelArray data;
  readonly attribute unsigned long height;
  readonly attribute unsigned long width;
};

interface CanvasPixelArray {
  readonly attribute unsigned long length;
  getter octet (in unsigned long index);
  setter void (in unsigned long index, in octet value);
};
```

- L'attribut data de type CanvasPixelArray est un tableau de pixel au format rgba, de dimension $4 \times h \times w$, représentant la couleur de chaque pixel de l'image parcouru de gauche à droite et de haut en bas.




```
ctx.drawImage(lena,0,0);
image=ctx.getImageData(0,0,500,500);
var w=image.width;
var h=image.height;
var tab=image.data;
for(i=0;i<h;i++)
{  for(j=0;j<w;j++)
  { e=Math.exp(-((i-250)*(i-250)+
        (j-250)*(j-250))/10000.0);
  tab[(i*h+j)*4]*=e;
  tab[(i*h+j)*4+1]*=e;
  tab[(i*h+j)*4+2]*=e;
  }
}
```

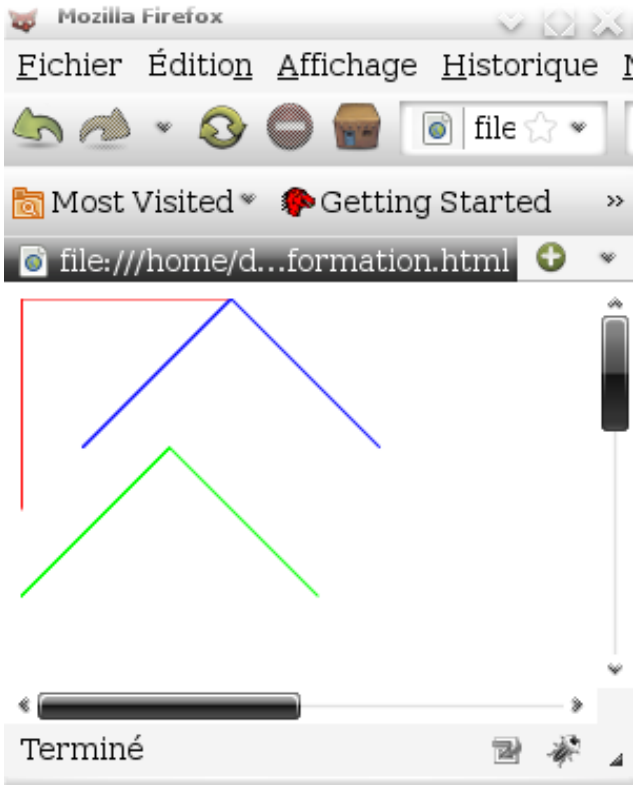
Transformations

- Le contexte graphique possède une matrice de transformation (le repère actuel), initialisée avec l'identité (le repère initial).
- Chaque transformation modifie la matrice courante :

```
// transformations (default transform is the identity matrix)
void rotate(in float angle);
void scale(in float x, in float y);
void setTransform(in float m11, in float m12, in float m21,
        in float m22, in float dx, in float dy);
void transform(in float m11, in float m12, in float m21,
        in float m22, in float dx, in float dy);
void translate(in float x, in float y);
```

-  Les transformations s'appliquent dans l'odre inverse d'appel, et ne commutent pas !
- Multiplication à droite.

Exemple




```
cs=document.getElementById('cs');
ctx=cs.getContext("2d");
repere("#ff0000");

ctx.rotate(Math.PI/4);
ctx.translate(100,0);
repere("#00ff00");

ctx.setTransform(1,0,0,1,0,0);

ctx.translate(100,0);
ctx.rotate(Math.PI/4);
repere("#0000ff");
```

`setTransform` réinialise la matrice de transformation, `transform` la multiplie.

- Chaque pixel, $\begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$, avant dessin, est multiplé par la matrice de transformation.
- Matrice de rotation :

$$\begin{pmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

- Matrice de translation :

$$\begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix}$$

- Matrice de mise à l'échelle :

$$\begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

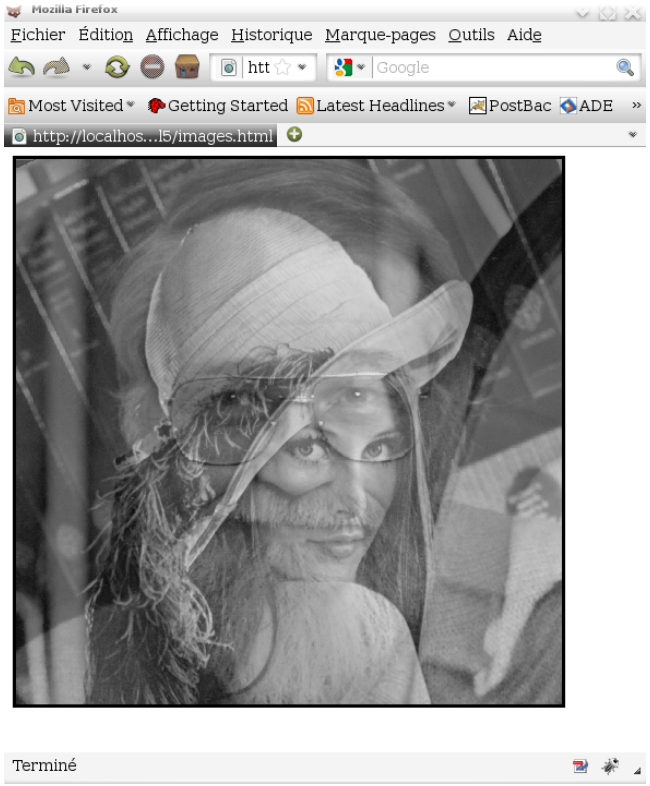
Composition

```
// compositing
attribute float globalAlpha; // (default 1.0)
attribute DOMString globalCompositeOperation; // (default source-over)
```

- `globalAlpha` est un coefficient entre 0 et 1 qui est pris en compte dans l'affichage du pixel.
- `globalCompositeOperation` indique comment est combiné le pixel à afficher avec le pixel existant.
- Par exemple, pour l'opérateur par défaut `source-over`, A représentant l'image à afficher, et B l'image existante, on a

$$C_r = C_a \alpha_a + C_b \alpha_b (1 - \alpha_a)$$

Fondu enchainé




```
ctx.drawImage(lena,0,0);
for(i=0;i<=10;i++)
{
  ctx.globalAlpha=i/10.0;
  alert("ok");
  ctx.drawImage(brian,0,0);
  ctx.stroke();
}
```