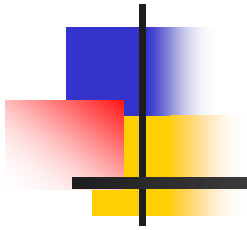
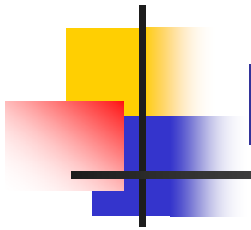


Le multimédia dans les SID

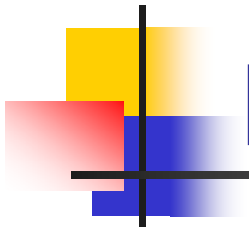
SVG

Scalable Vector Graphics



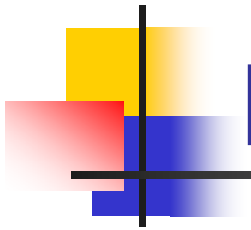
Introduction

- SVG ?
  - SVG est une application de XML dont l'objet est la description d'objets graphiques en 2 dimensions
- Le monde du multimédia bouge
  - Il est à la recherche de formats pour la création de RIA pour des applications Web, applications embarquées, sur des petits devices



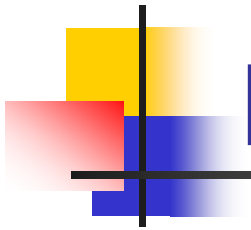
Historique

- Début
  - Le W3C : s'est orienté vers PNG : (Portable Network Graphics créé pour remplacer le GIF) et CGM : Computer Graphics Metafile) pour créer
    - VML : Vector Markup Language
  - Rejeté comme standard Web car Adobe Systems, Sun Microsystems et d'autres ont proposé un concurrent connu sous le nom de PGML (Precision Graphics Markup Language), un langage pour des artistes graphistes
  - Les deux standards ont été fusionnés et améliorés pour créer le format SVG
- Un groupe d'expert est mis en place :
  - Adobe, Corel, Macromedia, Microsoft, Sun, HP, Kodak, Canon, ILOG
    - Le SVG 1.0 est né en Septembre 2001
  - Pendant trois ans, deux implémentations majeures s'affirment :
    - Adobe et Batik (Java-based toolkit pour SVG)



Historique

- SVG 1.0
  - Propose tout ce que propose une plate-forme graphique 2D
    - Formes géométriques simples, Béziers
    - Texte avancé
    - Dégradés, motifs, clip, masques, opacité
  - Les premiers outils
    - Un plug-in browser Adobe
    - Une plate-forme Java open-source Batik
    - Des outils de dessins vectoriels exportant SVG (Illustrator, Corel DRAW)
    - Des outils conçus autour de SVG (Jasc WebDraw)



Historique

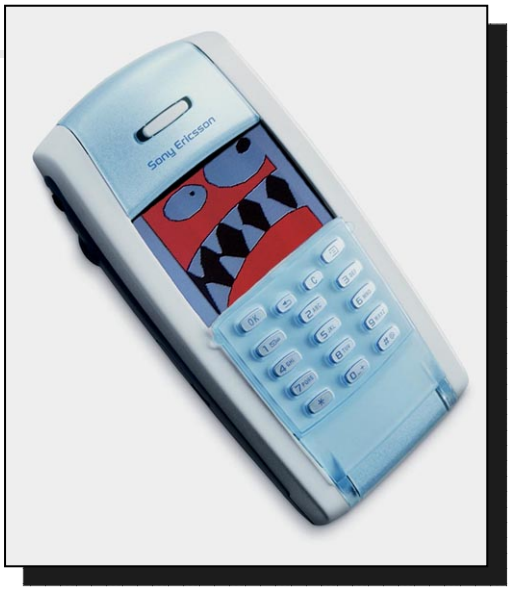
- Continuation
  - Comme SVG 1.0 a eu un certain succès
    - De nouvelles problématiques se sont apparus
    - De nouveaux acteurs du monde mobile se joignent au groupe de travail : Nokia, Ericsson, Motorola, KDDI, Sharp, ZOOMON
- Évolution vers SVG 1.1
  - Idée : simplifier le code et proposer des modules plus faciles à intégrer sur des petits devices
    - SVG Tiny
      - une adaptation pour les téléphones portables multimédias
    - SVG Basic
      - son équivalent pour les ordinateurs de poche
        - SVG permet d'envoyer, de manière plus souple, des messages animés

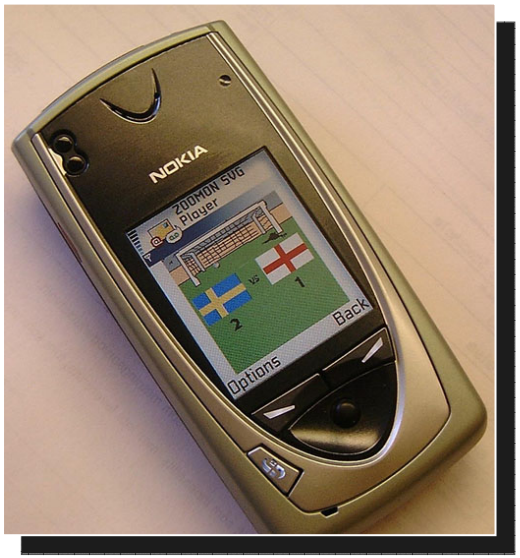
Historique

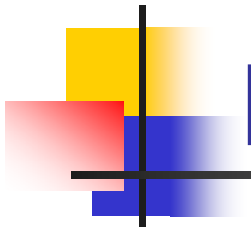
- SVG Basic
  - Assez proche des capacités du SVG initial (Full)
  - Contient un sous-ensemble de filtres de SVG
  - Supporte la CSS Mobile Profile
  - Supporte le DOM scripting (i.e. programmation avec le DOM)
- SVG Tiny
  - Sous-ensemble bien plus léger que Basic
  - Pas de scripting, plutôt utilisation de styles CSS, filtres, symboles, texte avancé, dégradés, motifs, opacité, référence externes
  - Supporte l'animation, polices SVG, images, texte, formes et courbes

Historique

- Implémentations Mobile
  - De très bon rapports d'implémentation
    - SVG Basic - BitFlash, eSVG ...
    - SVG Tiny - ZOOMON, TinyLine, Access ...
  - D'autres acteurs majeurs arrivent avec comme but : Tiny dans MMS :
    - *Multimedia messaging service* : version étendue aux données multimédia (audio, photo, vidéo) des célèbres SMS

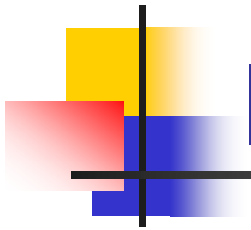






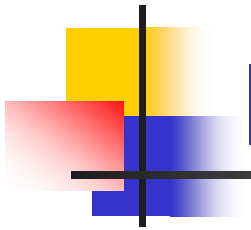
Historique

- Présent et avenir
  - SVG 1.1 et SVG Mobile sont devenus des recommandations en Janvier 2003
  - Dès lors le groupe de travail W3C se met au travail
    - propose SVG 1.2
- Technologies Clés
  - SVG est plus qu'une syntaxe XML
  - CSS est utilisé pour le stylage
  - SMIL est utilisé pour le multimédia
  - Le DOM est utilisé pour l'intégration applicative
  - XLink et XPointer pour les références



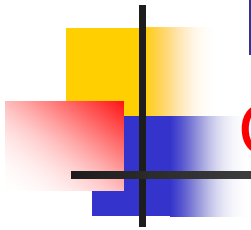
Introduction

- Philosophie de SVG
  - Offrir un langage de description de graphiques 2D en XML
  - En utilisant des objets courants
    - des *formes* vectorielles (traits, courbes, …)
    - des images
    - du texte
  - Les objets graphiques peuvent être
    - groupés, transformés, composés dans d'autres objets et recevoir des attributs de style
  - Les graphiques SVG sont interactifs et dynamiques
    - Leur animation peut être définie soit à l'intérieur des fichiers SVG soit dans un langage de script externe
    - On peut ainsi insérer SVG dans du HTML, XHTML, le générer à partir de XSLT ou de PHP, utiliser CSS et le scripter avec JavaScript (via DOM)



Introduction

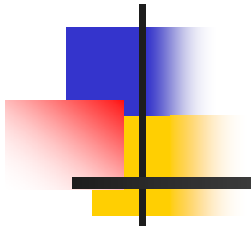
- Philosophie de SVG (suite)
  - Conformité à XML
    - Possibilité d'utiliser tous les outils XML : parsers, outils de transformation, bases de données
  - Conformité aux espaces de nommage
    - Possibilité de mélanger des grammaires XML entre elles
    - Par exemple :
      - un document HTML peut contenir des graphiques SVG, des expressions mathématiques en MathML, des présentations en SMIL, ...



Introduction

Outils de visualisation

- Viewers SVG
  - Télécharger SVG viewer qui se pluggue directement dans vos navigateurs
- Éditeur graphique :
  - SVGGenerator
    - http://www.svg.free.fr/
    - Code source en Java disponible
    - On peut passer du mode graphique au mode SVG
    - Documentation : http://www.svg.free.fr/rapport/
  - http://lti.epfl.ch/EditorSVG/



SVG
Scalable Vector Graphics

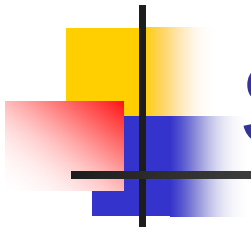
Eléments du langage

Structure d'un document SVG

- Déclaration d'un document SVG

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
   "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
      //utile pour un document standalone
<svg width="5cm" height="4cm" version="1.1"
   xmlns="http://www.w3.org/2000/svg">
            Contenu ici
            …
</svg>
```

  - Attention : mettre le bon namespace car les versions ont bien changé



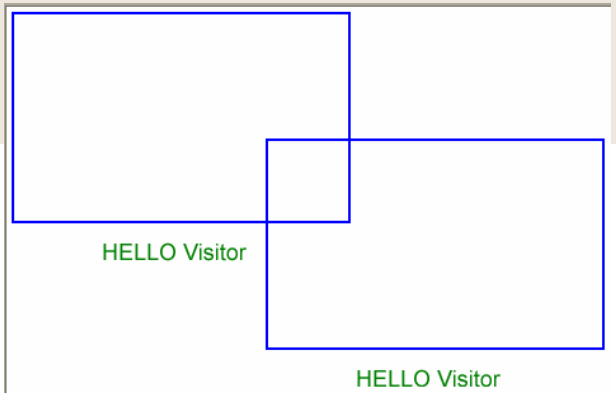
Structure d'un document SVG

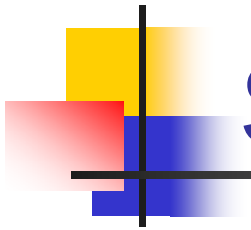
- Un document SVG se compose :
  - d'un ou plusieurs fragments délimités par la balise <svg>
- <svg> est la racine d'un graphisme SVG
  - On peut imbriquer des éléments svg parmi d'autres et les positionner
  - Chaque <svg> crée un nouveau système de coordonnées
    - ainsi on peut facilement réutiliser des fragments graphiques sans devoir modifier les coordonnées
- Paramètres de <svg> :
  - x, y : coordonnées du coin sup gauche de l'espace
  - width, height : dimensions

Structure d'un document SVG

- Exemple : exemple.svg

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg SYSTEM "../svg10.dtd" >
<svg xmlns="http://www.w3.org/2000/svg">
   <rect x="5" y="5" width="265" height="165" style="fill:none;stroke:blue; stroke-width:2"
/>
   <text x="75" y="200" style="font-size:18;font-family:Helvetica; fill:green">
          HELLO Visitor </text>
<svg with="200" height="200" x="200" y="100">
   <rect x="5" y="5" width="265" height="165" style="fill:none;stroke:blue;stroke-width:2"
/>
   <text x="75" y="200" style="font-size:18;font-family:Helvetica; fill:green">
          HELLO Visitor </text>
</svg>
</svg>
```





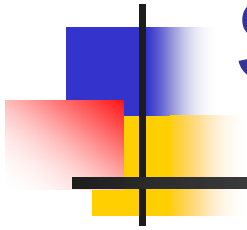
Style

- SVG possède 2 syntaxes différentes pour définir la mise en forme d'un élément :
  - L'attribut *style* reprend la syntaxe et les styles de CSS2

```
<rect x="200" y="200" width="60"
height="30"   style="fill:red;stroke:blue;stroke-width:3" />
```

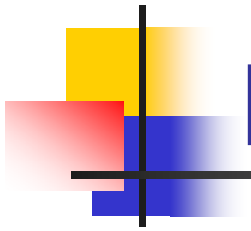
  - Pour chaque style, il existe aussi un attribut de présentation SVG

```
<rect x="200" y="100" width="60" height="30" fill="red"
stroke="blue" stroke-width="3" />
```



SVG

Les éléments de base



Éléments de base

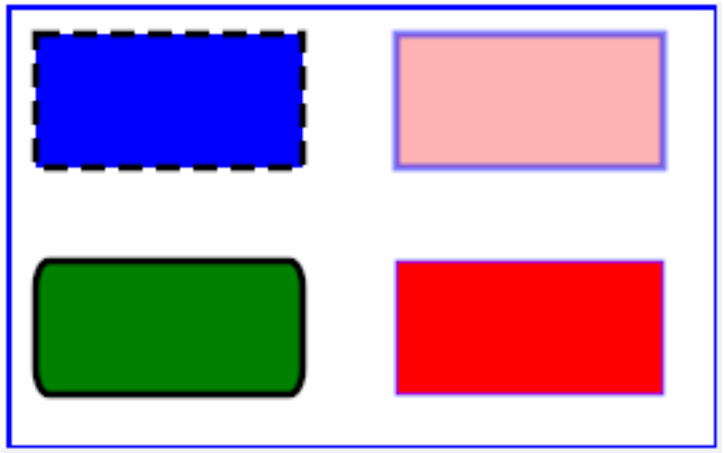
- Rectangles <rect>
- Cercles <circle>
- Ellipse <ellipse>
- Ligne <line>
- Poly-ligne <polyline>
- Polygone <polygon>
- Forme arbitraire <path>
- Texte <text>

Rectangle <rect> : rectangle.svg

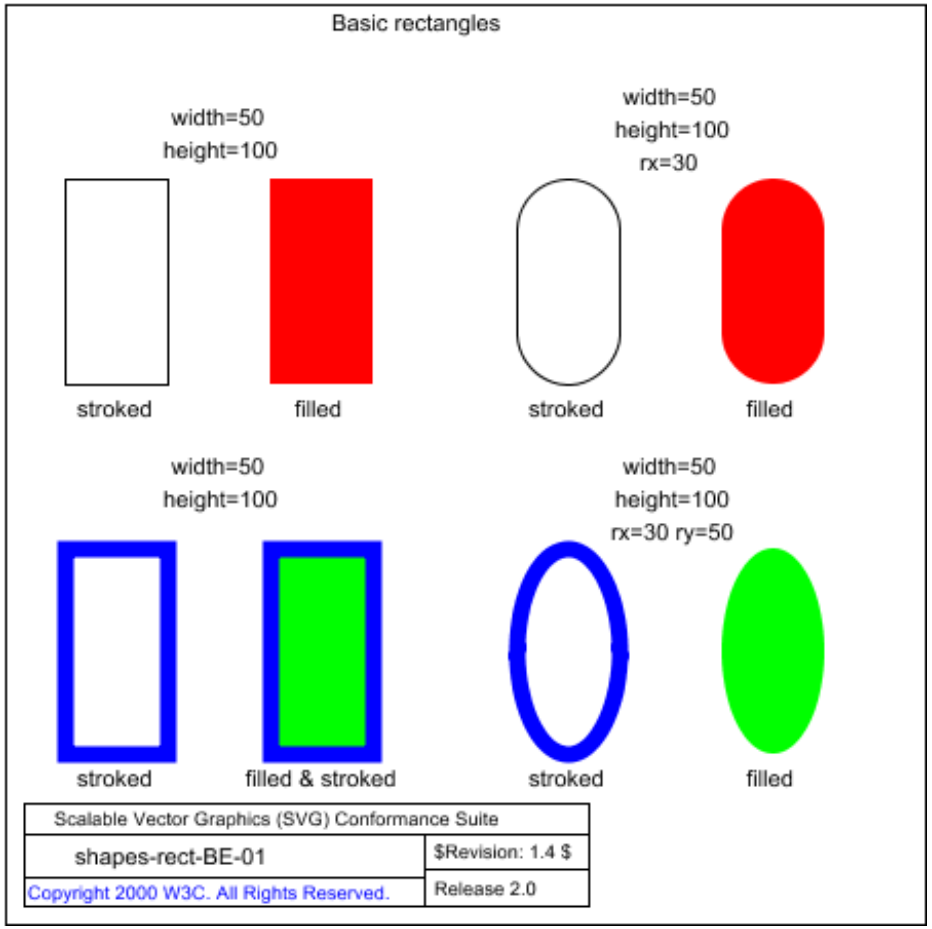
- Attributs de base
  - x, y = <0,0> par défaut
    - position du coin supérieur gauche
  - width, height =
    - taille du rectangle
  - fill
    - couleur remplissage
  - stroke
    - couleur du trait
  - stroke-width
    - épaisseur du trait
  - stroke-opacity
  - fill-opacity

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20001102//EN"
"http://www.w3.org/TR/2000/CR-SVG-20001102/DTD/svg-20001102.dtd">
<svg>
    <rect x="5" y="5" width="265" height="165«
      style="fill:none;stroke:blue; stroke-width:2" />
   <rect x="15" y="15" width="100" height="50«
     fill="blue" stroke="black" stroke-width="3" stroke-
     dasharray="9 5"/>
   <rect x="15" y="100" width="100" height="50"
     fill="green" stroke="black" stroke-width="3" rx="5"
     ry="10"/>
   <rect x="150" y="15" width="100" height="50"
     fill="red" stroke="blue" stroke-opacity="0.5" fill-
     opacity="0.3" stroke-width="3"/>
   <rect x="150" y="100" width="100" height="50"
     style="fill:red;stroke:blue;stroke-width:1"/>
</svg>
```

Exemples



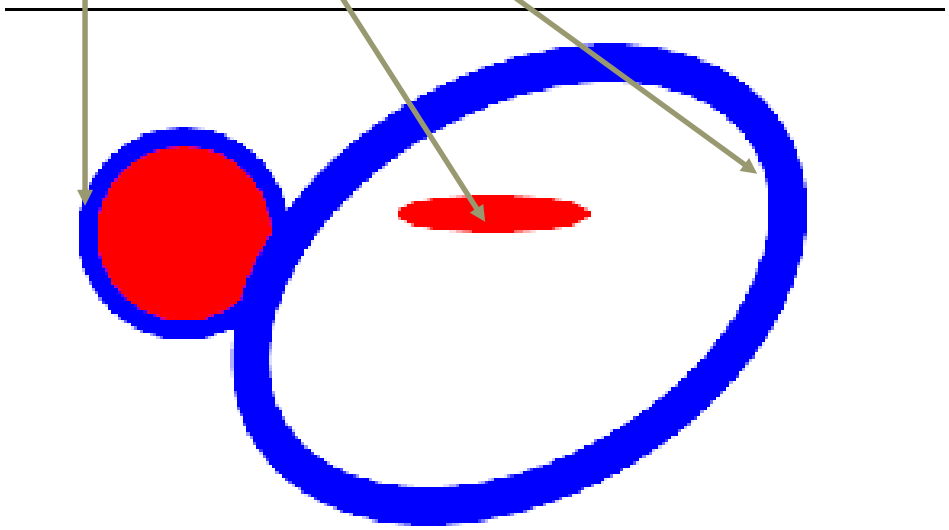
rectangle.svg



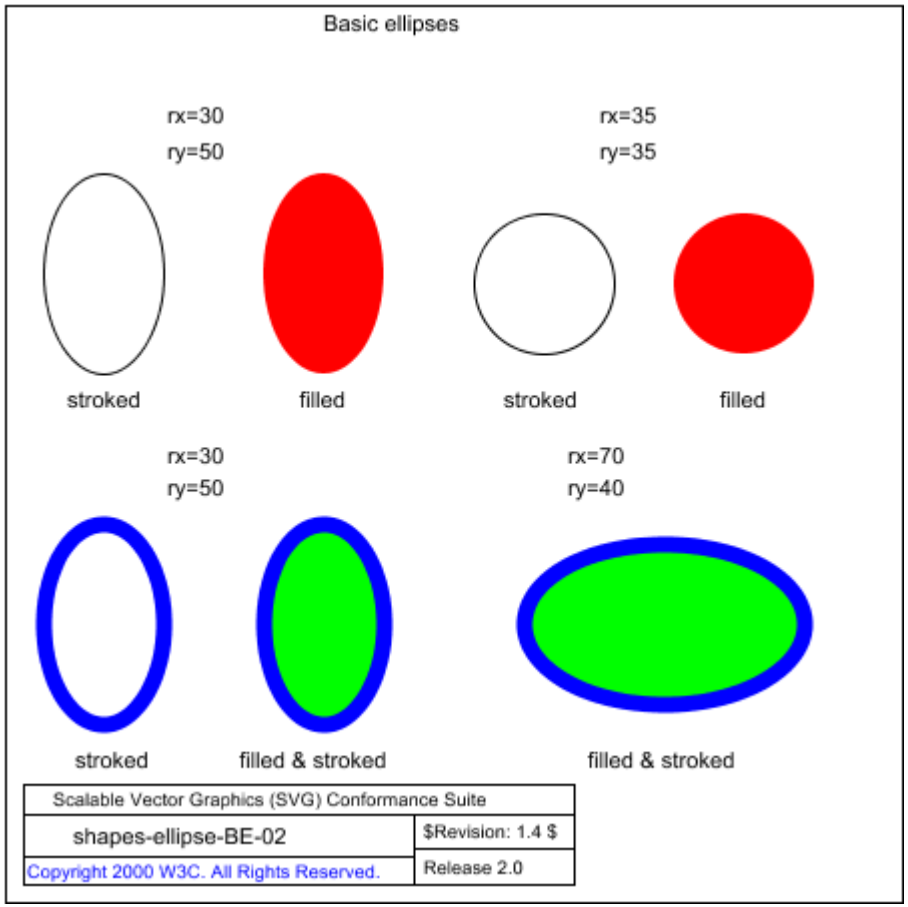
Cercle <circle>, ellipse <ellipse>

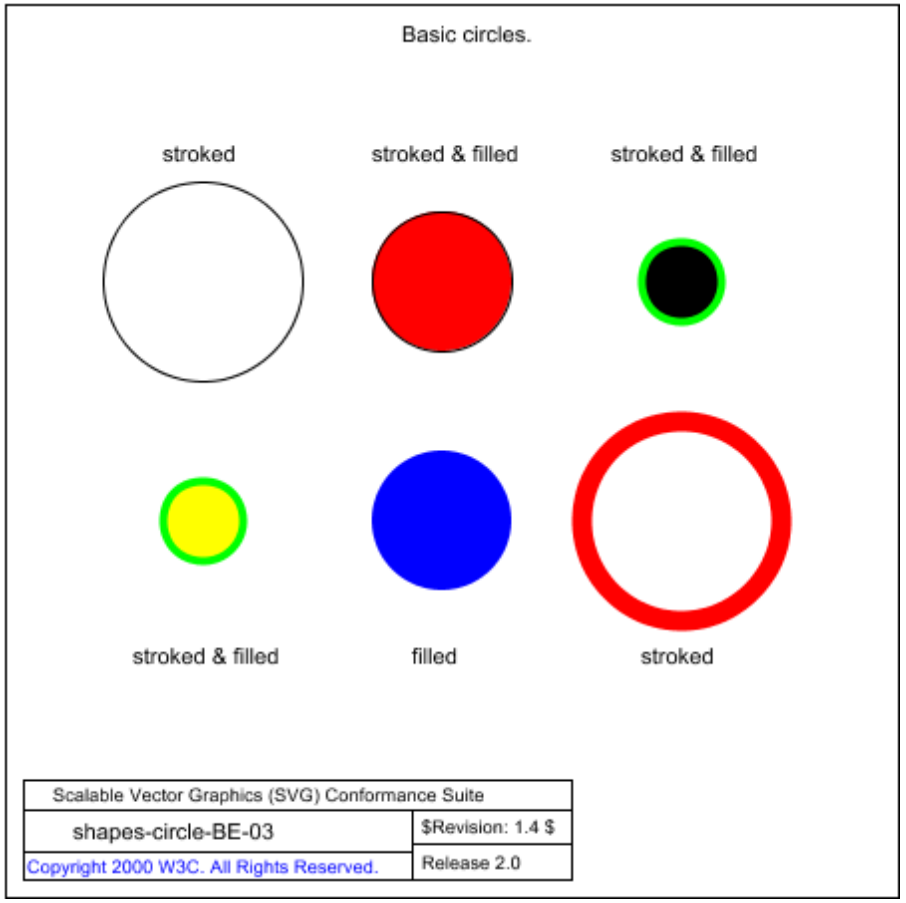
- Attributs de base
  - cx,cy :
    - position du centre ("c" = circle)
  - r =
    - rayon du cercle
  - rx,ry =
    - rayons de l'ellipse

```
<circle cx="90" cy="110" r="50" fill="red"
        stroke="blue" stroke-width="10" />
<ellipse cx="250" cy="100" rx="50" ry="10"
        fill="red" />
<ellipse cx="160" cy="250" transform="rotate(-
        30)" rx="150" ry="100" fill="none"
        stroke="blue" stroke-width="20" />
```



Exemples


Basic ellipses
rx=30
ry=50
rx=35
ry=35
stroked
filled
stroked
filled
rx=30
ry=50
rx=70
ry=40
stroked
filled & stroked
filled & stroked
Scalable Vector Graphics (SVG) Conformance Suite
shapes-ellipse-BE-02
$Revision: 1.4 $
Copyright 2000 W3C. All Rights Reserved.
Release 2.0


Basic circles.
stroked
stroked & filled
stroked & filled
stroked & filled
filled
stroked
Scalable Vector Graphics (SVG) Conformance Suite
shapes-circle-BE-03
$Revision: 1.4 $
Copyright 2000 W3C. All Rights Reserved.
Release 2.0

Ligne <line>, Poly-ligne <polyline>

- Attributs de base pour <line>
  - x1,y1 =
    - Point de départ
  - x2,y2 =
    - Point d'arrivée
- Attributs de base pour <polyline>
  - points =
    - série de points x,y qui seront liés

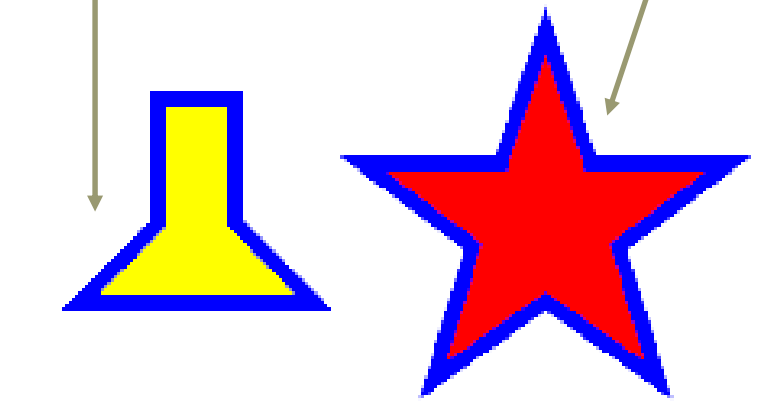
```
<line x1="300" y1="200" x2="400" y2="100"
        stroke = "red" stroke-width="5" />
<line x1="300" y1="100" x2="400" y2="200"
        stroke = "red" stroke-width="5" />
```

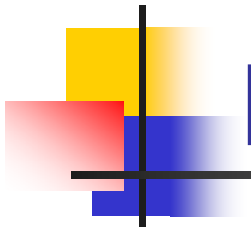
```
<polyline fill="none" stroke="blue"
stroke-width="10" points="50,200,100,200,100,
120,150,120,150,200,200,200" />
```

Polygone <polygon>

- Définition
  - forme fermée
- Attributs de base
  - points = "<chemin de points>"
    - Séries de points x,y qui seront reliés (également le dernier au premier point)

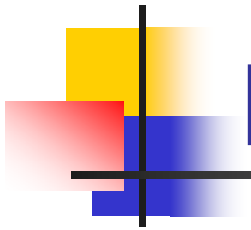
```
<polygon fill="yellow" stroke="blue" stroke-
        width="10"  points="50,250,100,2
        00,100,120,150,120,150,200,200,
        250" />
<polygon fill="red" stroke="blue" stroke-
        width="10" points="350,75
        379,161 469,161 397,215
        423,301 350,250 277, 301
        303,215 231,161 321,161" />
```





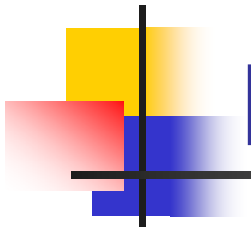
Forme arbitraire <path>

- Définition
  - L'élément <path> permet de définir des formes arbitraires (shapes) par connexion de lignes, arcs et courbes
- Commandes "path data" de base :
  - **Principe** :
    - Le dessin est réalisé par une liste de commandes
    - Chaque commande est désignée par une lettre (M : moveto, L : lineto, etc.) suivie d'une coordonnée, soit par rapport :
      - au repère absolu si la commande est en MAJUSCULE (ex: M: moveto)
      - au repère relatif si la commande est en MINUSCULE (ex: m: moveto)



Forme arbitraire <path>

- Commandes (suite)
  - **M et m** : commande « moveto »:
    - Déplacement du *crayon* au point spécifié
    - M permet de créer un nouveau PATH (ou SUB-PATH) qui sert d'origine pour un repère relatif
    - Syntaxe : M|m (x y)+
      - Exemple : M100 100 200 200
  - **L et l** : commande « lineto » :
    - Dessine des lignes du point courant vers le(s) point(s) indiqué(s)
    - Syntaxe : L|l (x y)+
      - Exemple : L 200,300 100,200



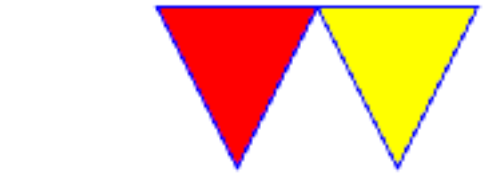
Forme arbitraire <path>

- Commandes "path data" de base :
  - **Z et z** :
    - ferme le sous chemin courant (trace une ligne depuis le point courant vers le point défini avec un M ou m)
  - **H et h (V et v) :**
    - dessine des lignes horizontales (verticales) à une coordonnée spécifiée
      - Syntaxe :
        - H |h (x)+
        - V |v (y)+
      - Exemple :
        - h 100

Forme arbitraire

- Exemple :
  - on pose le crayon (*M 50 100*)
  - ensuite on tire un trait vers le coin du bas (*L 100 100*) et vers le coin en haut à droite (*150 100)*
  - finalement on ferme (z)
  - Notez que le triangle jaune (pareil) a été fait avec *<polygon>*

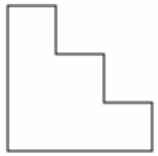
```
<path d="M 50 50 L 100 150 150 50 z"
        fill="red" stroke="blue" stroke-
        width="2" />

<polygon points="150 50 200 150 250 50"
        fill="yellow" stroke="blue" stroke-
        width="2" />
```



Forme arbitraire

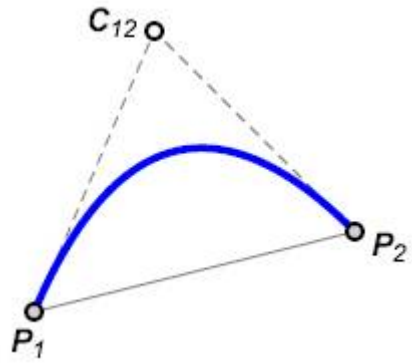
- Exemple :
  - on se positionne à 240, 96 (*M 240,96*)
  - on dessine une ligne horizontale jusqu'à 270
  - on dessine une ligne verticale jusqu'à 126
  - …
  - on ferme le tout (*z*)

```
<path id="Stairs_stroke_MHVZ"
style="fill:blue;stroke:#000000;" d="
M 240 96 H 270 V 126 H 300 V 156
H 330 V 186 H 240 V 96 Z "/>
```



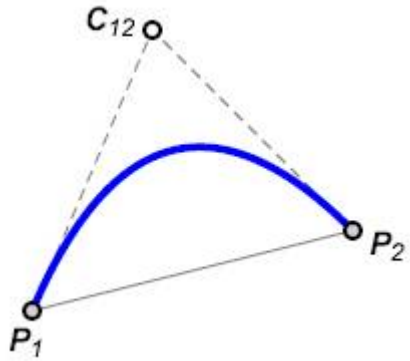
Courbes

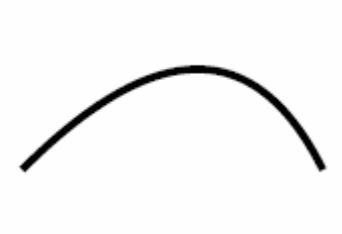
- Courbe de bézier quadratique
  - On part d'un segment P1P2, et on ajoute un troisième point C12 en dehors de ce segment
  - La courbe de Bézier passe par les points de contrôle P1 et P2, et approche seulement le point C12
  - C'est comme si C12 exerce une certaine attraction sur le segment et le déforme en une courbe régulière



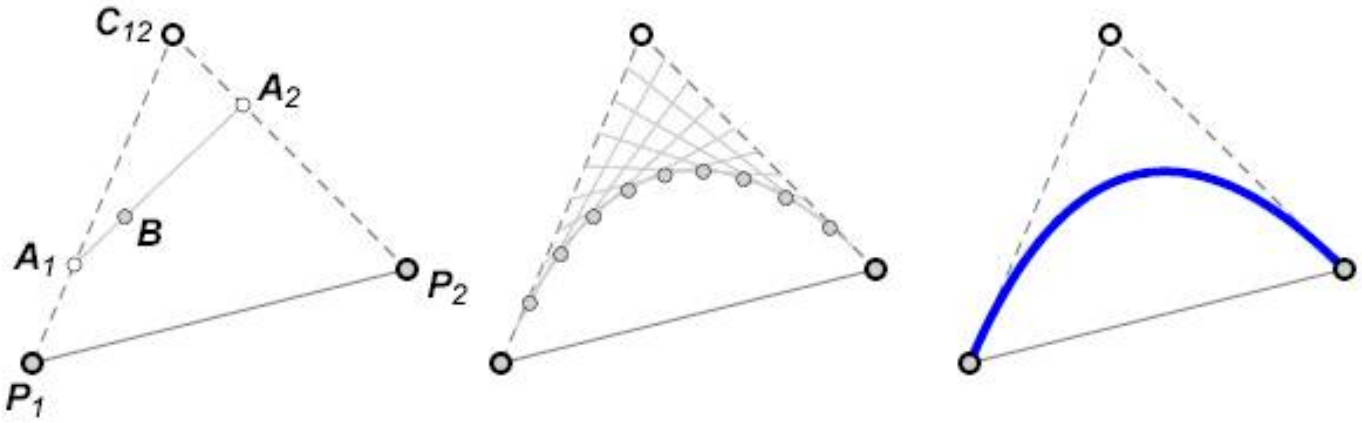
Courbes

- Syntaxe : Q|q x1, y1, x, y
  - Où :
    - x1, y1 : coordonnées du point de contrôle C12
    - X, y : coordonnées de P2, fin de tracé
    - P1 est supposé être le point précédent du tracé à partir duquel on trace la courbe de Bézier
  - Exemple :
    - <path fill="white" stroke="black" stroke-width="2" d="M 50,100 Q100,50,125,100" />



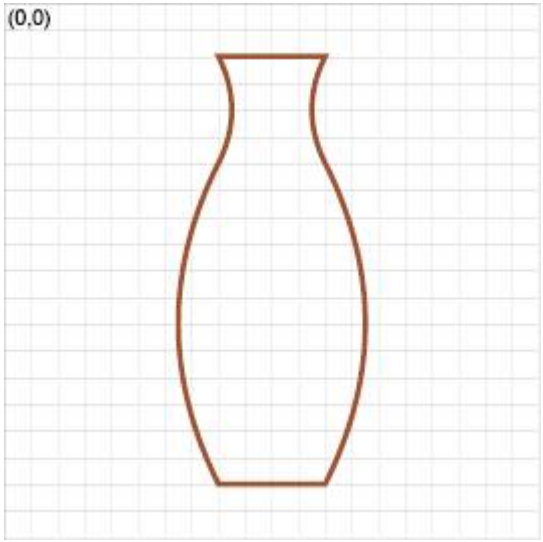


- Algorithme de construction et intérêt de la représentation graphique
  - On place un point A1 à 30% du segment P1 C12 ainsi que A2 à 30% de C12 P2
  - On joint ces points et place B à 30% du segment A1 A2 obtenu
  - Ce point B appartient à la courbe de Bézier
  - En répétant cette construction en changeant le rapport de 0% à 100% avec un pas de 10%, on obtient la figure centrale
  - Tous les segments construits sont des tangentes à la courbe - ils constituent l'enveloppe convexe de la courbe - et tous ces points B appartiennent à la courbe de Bézier qui n'est dans ce cas qu'une simple parabole



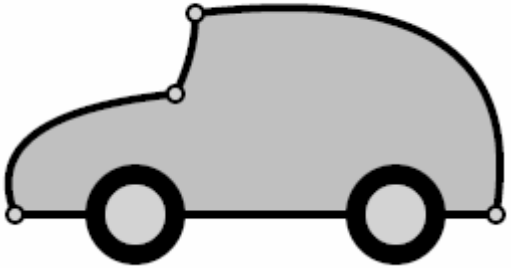
Courbes

- Courbe de Bézier quadratique
  - Exemple : bezier-quadratique.svg
    - <path stroke="sienna" stroke-width="2" fill="none"
      d="M 80,180
      Q 50,120 80,60
      Q 90, 40 80,20
      Q 100, 20 120,20
      Q 110, 40 120,60
      Q 150,120 120,180
      Z" />

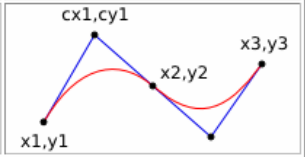
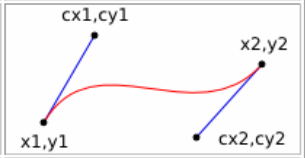
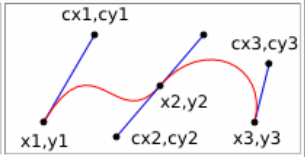
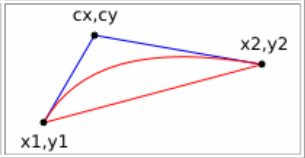


Exemple : bezier-voiture.svg

```
<svg width="600" height="400" viewBox="0 0 300
    200">
  <defs>
    <circle id="crvpnt" r="2" stroke="black"
     fill="lightgray" />
    <circle id="wheel" r="10" stroke="black" stroke-
     width="5"  fill="lightgray" />
  </defs>
  <path fill="silver" stroke="black" stroke-width="2"
        d="M 50,100 Q40,75 90,70Q95,60
    95,50Q180,40 170,100Z" />
  <use xlink:href="#crvpnt" x="50" y="100" />
  <use xlink:href="#crvpnt" x="90" y="70" />
  <use xlink:href="#crvpnt" x="95" y="50" />
  <use xlink:href="#crvpnt" x="170" y="100" />
  <use xlink:href="#wheel"  x="80" y="100" />
  <use xlink:href="#wheel"  x="145" y="100" />
  <text x="20" y="140" font-size="7">
d="M 50,100 Q40,75 90,70Q95,60 95,50Q180,40
    170,100Z"
  </text>
</svg>
```

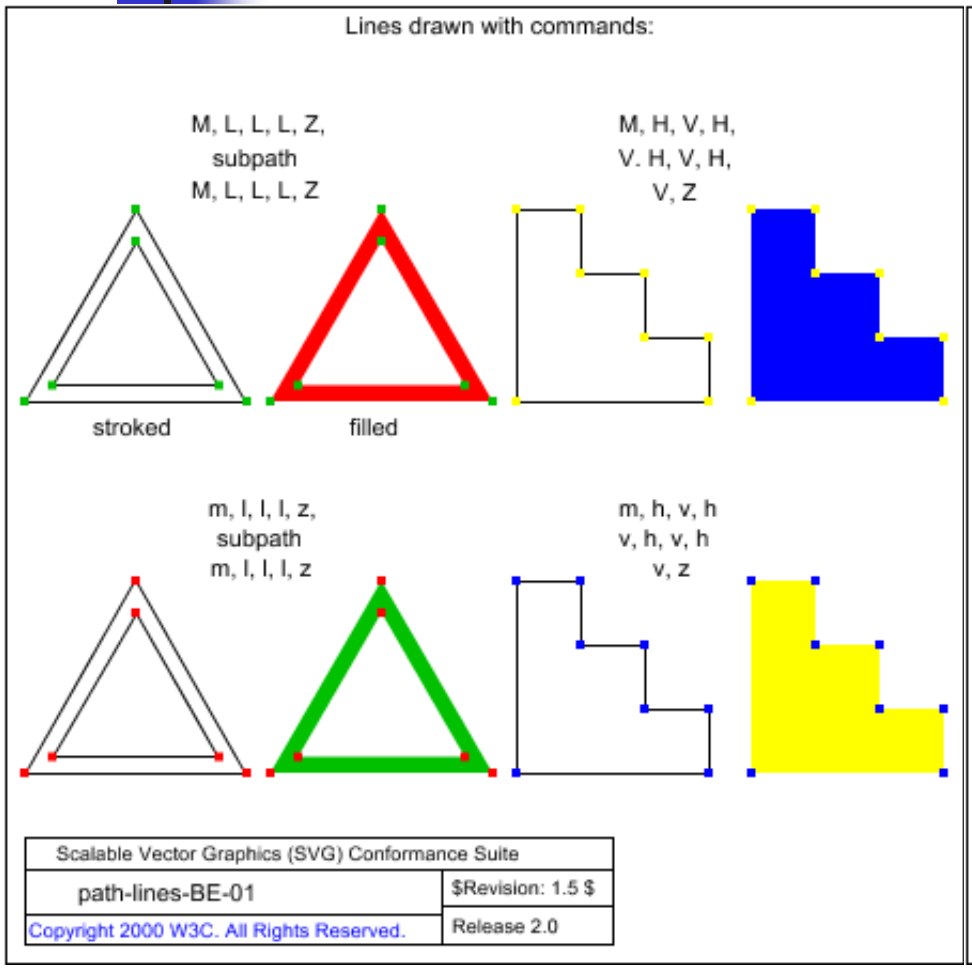


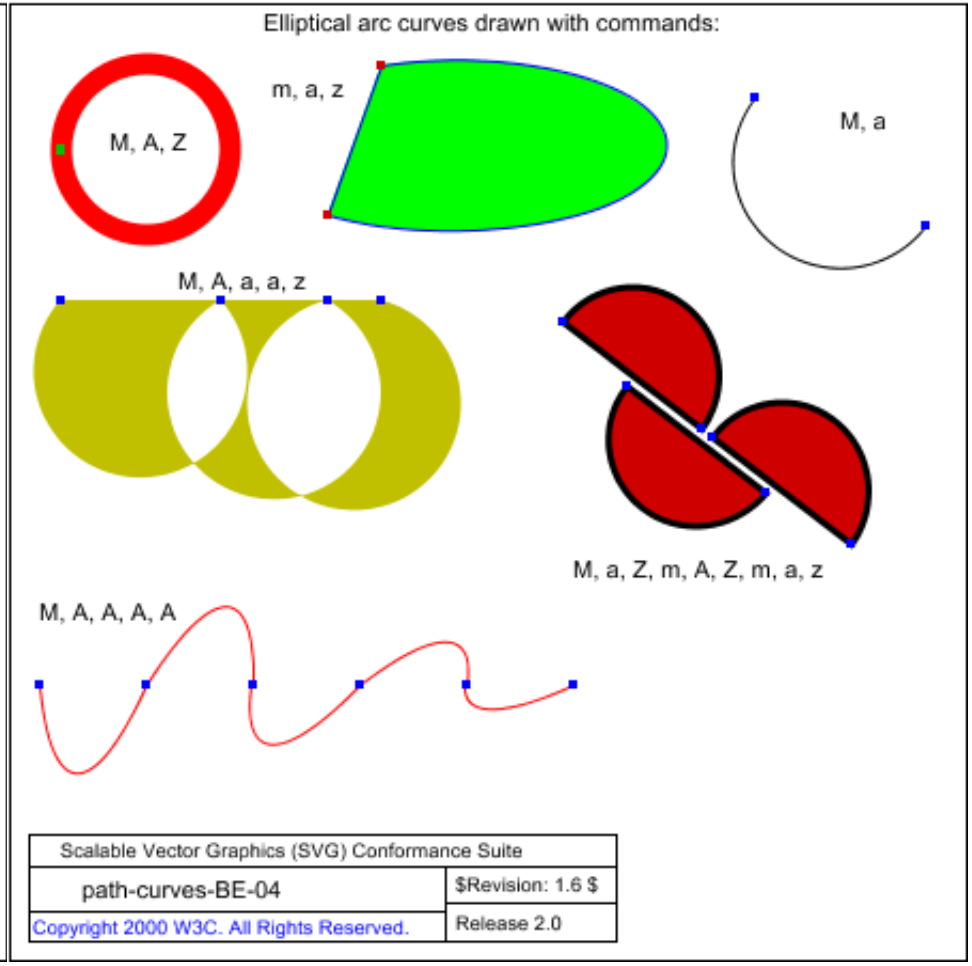
Autres courbes de Bézier

Courbe de Bézier quadratique avec partage de point de contrôle `<path d="M x1 y1 Q cx cy x2 y2 T x3 y3"/>`	
Courbe de Bézier cubique `<path d="M x1 y1 C cx1 cy1 cx2 cy2 x2 y2"/>`	
Courbe de Bézier cubique avec partage de point de contrôle `<path d="M x1 y1 C cx1 cy1 cx2 cy2 x2 y2 S cx3 cy3 x3 y3"/>`	
Fermeture du chemin par une ligne `<path d="M x1 y1 Q cx cy x2 y2 Z"/>`	

Forme arbitraire

path-lines-BE-01.svg, path-curves-BE-04.svg



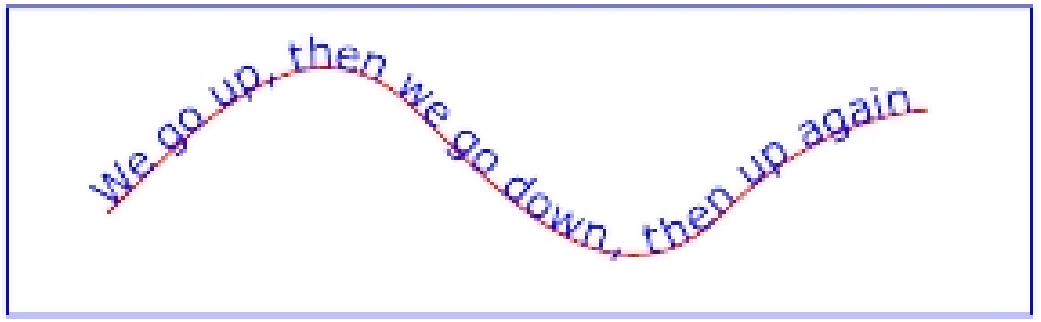


Texte <text>

- Lien avec les chemins
  - On peut imposer au texte de *suivre* un chemin prédéfini par la balise <textPath>
  - On le référence par :

    <textPath xlink:href="uri" />

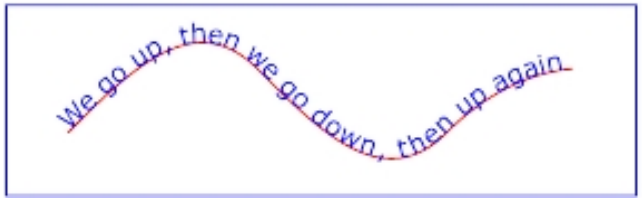
    - Uri : emplacement du path

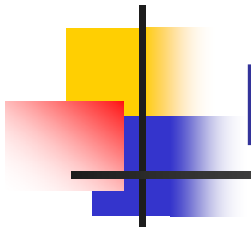


Texte <text>

- Exemple

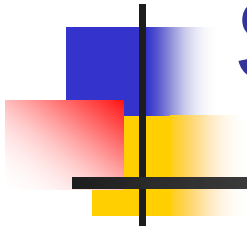
```
<svg width="10cm" height="3cm" viewBox="0 0 1000 300">
    <defs>
     <path id="MyPath"
         d="M 100 200
            C 200 100 300   0 400 100
            C 500 200 600 300 700 200
            C 800 100 900 100 900 100" />
    </defs>
    <use xlink:href="#MyPath" style="stroke:red" />
       <!-- pour mettre en rouge le chemin -->
       <text style="font-family:Verdana; font-size:42.3333; fill:blue">
              <textPath xlink:href="#MyPath">
              We go up, then we go down, then up again
              </textPath>
       </text>
</svg>
```





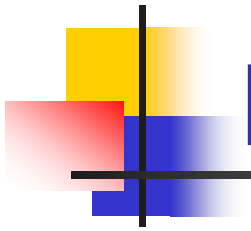
Exercice

- Énoncé
  - Construire un nouveau logo pour Miage
  - Le logo est composé de plusieurs parties
    - Un œil central : utiliser les courbes de Bézier pour le dessin
    - Un cil l'entourant : utiliser le path pour le dessiner
    - Un texte, de type « Miage de Nancy » entourant le cil
  - Servez-vous des exemples :
    - Exemples/Logo_Miage



SVG

Structure d’un document SVG



Les éléments

- SVG possède plusieurs constructions pour regrouper des objets dans des blocs
  - Liste des éléments les plus importants :
    - Fragment d'un document SVG : <svg>
    - Groupement d'éléments : <g>
    - Objets abstraits réutilisables (chablons) : <symbol>
    - Section de définition : <defs>
    - Utilisation d'éléments : <use>
  - Les objets SVG (comme les objets HTML) héritent le style de leurs parents (les styles sont de type : "cascading")

Structure d'un document SVG

- Groupement d'éléments : <g>
  - sert à regrouper des éléments qui vont ensemble, qui se partagent des attributs : couleur, style, …
  - ses fils héritent ses propriétés
- Exemples

```
<g style="fill:red" id="Grands rectangles rouges">
        <rect x="100" y="100" width="200" height="200 />
        <rect x="300" y="400" width="100" height="100 />
</g>
<g style="fill:blue" id="Petits rectangles bleus">
        <rect x="10" y="10" width="20" height="20 />
        <rect x="30" y="40" width="10" height="10 />
</g>
```

Structure d'un document SVG

- Groupement de définitions : <defs>
  - autorise la définition d'objets référencés plus loin dans le même fichier
- Exemple :

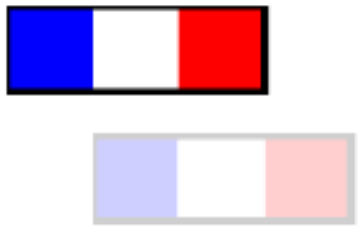
```
<defs>
    <linearGradient id="Gradient01">
        <stop offset="20%" stop-color="#39F" />
        <stop offset="90%" stop-color="#F3F" />
    </linearGradient>
</defs>
<rect x="1cm" y="1cm" width="6cm" height="1cm"  fill="url(#Gradient01)" />
```

  - Attention, un fichier SVG n'autorise qu'un seul <defs>. Il faut donc définir tout ce que l'on voudra réutiliser dans cette structure

Structure d'un document SVG

- Réutilisation d'objets : <symbol>-<use>
  - permet de définir des objets graphiques réutilisables et non des définitions, dans les cas suivants
    - Objet à instancier de multiples fois
    - Objet classique référencé par de nombreux éléments

```
<symbol id="bleublancrouge">
        <rect x="0" fill="blue" width="10" height="10"/>
        <rect x="10" fill="white" width="10" height="10"/>
        <rect x="20" fill="red" width="10" height="10"/>
        <rect x="0" fill="none" width="30" height="10" stroke
        ="black"/>
</symbol>
<use x="10" y="5" xlink:href="#bleublancrouge" />
<use x="20" y="20" xlink:href="#bleublancrouge"
opacity="0.3" />
```



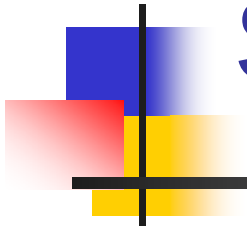
Autre exemple

```
<svg width="10cm" height="3.5cm" viewBox="0 0 100 30"
xmlns="http://www.w3.org/2000/svg"
xmlns:xlink="http://www.w3.org/1999/xlink">
<rect id="MyRect" width="60" height="10"/>
<use x="20" y="10" fill="yellow" xlink:href="#MyRect" />
<use x="20" y="20" fill="red" xlink:href="#MyRect" />
</svg>
```

Image

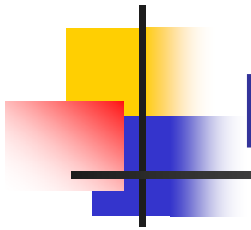
- <image>
  - Se définit également par référencement
  - Formats bitmap supportés :
    - png et jpeg
  - Le référencement sert à :
    - Préciser les dimensions
    - Changer le point de vue (voir plus loin)

```
<image x="10" y="50" width="200"
height="100" xlink:href="cathedrale_ge.jpg">
<title>Eglise large</title> </image>

<image x="250" y="50" width="50"
height="100" xlink:href="cathedrale_ge.jpg">
<title>Eglise longue</title> </image>

 <image x="310" y="50" width="100"
height="100" xlink:href="cathedrale_ge.jpg
preserveAspectRatio="xMinYMin meet">
<title>Eglise juste</title> </image>
```



SVG

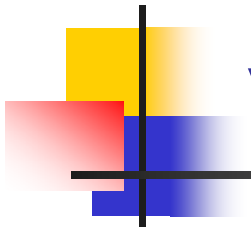
Le système de coordonnées



Le canevas ou espace utilisateur

- Définition
  - Au départ, l'élément <svg> le plus externe établit un espace utilisateur par les attributs "width" et "height"
  - A chaque espace utilisateur est associé un point de vue ou viewport :
    - il possède un système de coordonnées qui commence en haut à gauche du rectangle
  - Ensuite, chaque élément <svg> interne redéfinit un nouvel espace utilisateur et un nouveau point de vue associé

Comment changer le point de vue ou Viewport ?

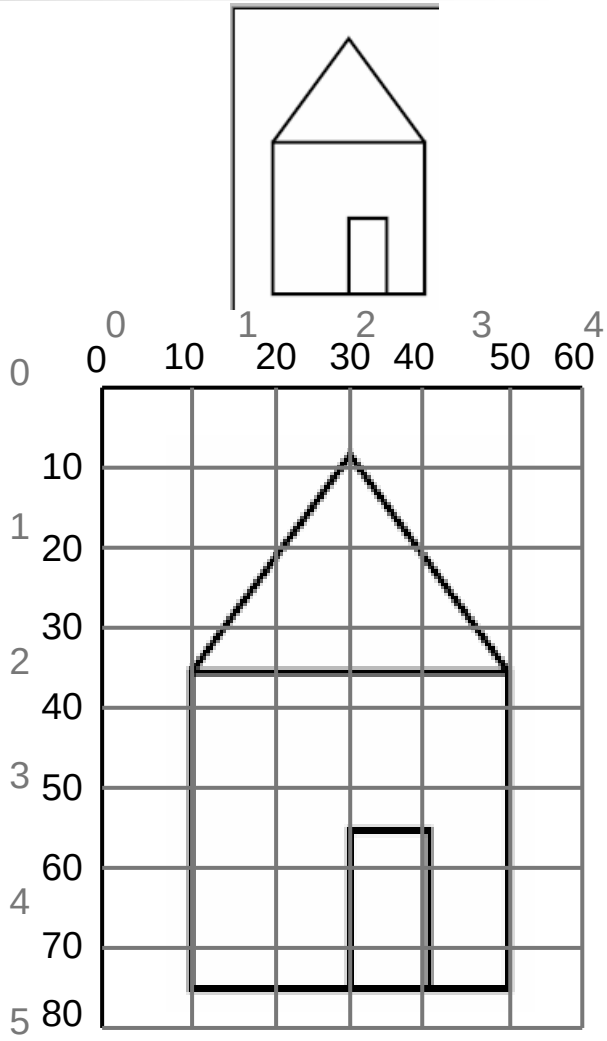


- On agit sur l'attribut viewBox
  - Syntaxe :
    - viewBox = "<min-x> <min-y> <width> <height>"
  - Ceci permet de préciser un nouvel espace donné par :
    - une nouvelle origine
    - de nouvelles dimensions
- Autres intérêts :
  - Changer l'expression des mesures de l'espace svg :
    - Par ex. des pixels en cm
  - Changer le ratio :
    - Pouvoir mettre en place un système dans lequel chaque coordonnée représente, par ex., 1/16 de centimètres
    - Dans ce système, un carré qui fait 40×40 unités serait affiché avec une taille de 2,5 cm par côté

Comment changer le Viewport ?

- Exemple : changer le type de mesures : cm ➔ px pour dessiner

```
<?xml version="1.0" standalone="no"?>
<svg width="4cm" height="5cm"
   viewBox="0 0 64 80">
    <rect x="10" y="35" width="40"
    height="40" style="stroke:black;
    fill:none;"/>
          <!--toit -->
   <polyline points="10 35, 30 7.68, 50 35"
   style="stroke:black; fill:none;"/>
          <!-- porte -->
   <polyline points="30 75, 30 55, 40 55, 40
   75" style="stroke:black; fill:none;"/>
</svg>
```



Comment changer le Viewport ?

- Attention !
  - Si le ratio est différent, on peut avoir un problème d'échelle
  - Exemple :

```
<svg width="45px" height="135px" viewBox="0 0 90 90">
```

    - viewBox a un ratio de 1/1 (90, 90) et la zone de dessin, 1/3 (45,135), d'où déformation ou rétrécissement du point de vue

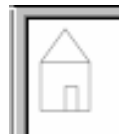
Comment changer tout en conservant l'aspect ?

- Utiliser l'attribut : preserveAspectRatio
  - avec les valeurs suivantes :
    - rien (par défaut) : la mise à l'échelle s'arrange pour que les valeurs extrêmes en x et en y touchent les bords du rectangle de point de vue
    - xMinYmin : mise à l'échelle uniforme ;
      - min des x = x_{min} du point de vue
      - min des y = y_{min} du point de vue

        <rect x="10" y="35" width="40" height="40" style="stroke:black; fill:none;" preserveAspectRatio = "xMinYmin"
    - XMidYMin : mise à l'échelle uniforme ;
      - val moyenne des x = x_{moyen} du point de vue
      - min des y = y_{min} du point de vue
    - XMaxYMin : idem
    - XMinYMid : idem
    - et toutes les variantes Min, Mid, Max pour X et Y

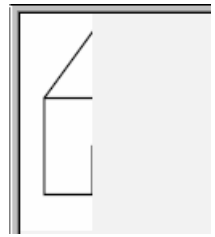
Conserver le ratio largeur/hauteur

- "meet" ou "Slice " :
  - meet (défaut) :
    - garder l'"aspect ratio", toute la viewBox est visible et adaptée au dessin, le graphisme sera toujours visible mais n'utilise peut-être pas tout le viewPort



```
<svg width="45px" height="135px" viewBox="0 0 90 90"
    preserveAspectRatio ="xMinYMin meet">
```

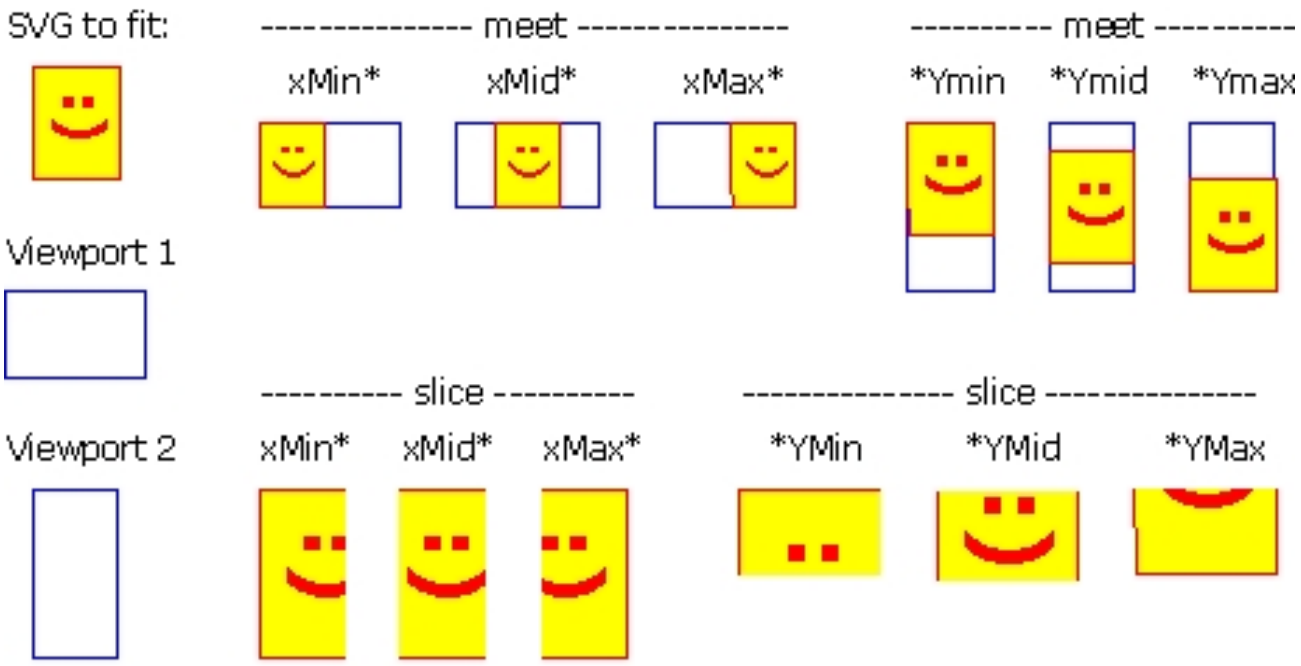
  - slice :
    - garder l'"aspect ratio", la viewBox utilise tout le viewPort et risque de dépasser dans un sens (selon "align"), autrement dit il y aura des graphismes coupés



```
<svg width="45px" height="135px" viewBox="0 0 90 90"
    preserveAspectRatio ="xMinYMin slice">
```

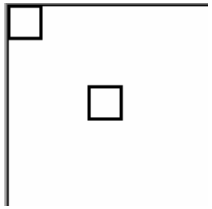
Conserver le ratio largeur/hauteur

- Autre exemple de "meet" ou "Slice" :
  meet-slice.svg



Transformation du système de coordonnées

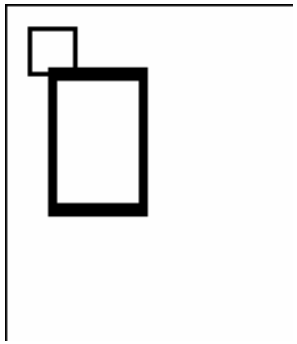
- Définition
  - Jusqu'à présent, les objets sont dessinés "tels quels"
  - On peut vouloir déplacer, effectuer une rotation, ou changer l'échelle d'un objet graphique
  - Pour obtenir ces effets, utiliser l'attribut transform
- Exemple : translation : transform-carre-trans.svg
  - Déplacer un objet avec <use>



```
<svg width="200px" height="200px" >
    <g id="carre">
        <rect x="0" y="0" width="20" height="20"
        style="fill:none; stroke:black; stroke-width: 2;"/>
    </g>
    <use xlink:href="#carre" transform="translate(50,50)"/>
</svg>
```

Transformation du système de coordonnées

- Transformation d'échelle : scale
  - Transform="scale(valeur-x,valeur-y)"
    - multiplie les coordonnées en x par valeur-x et les coordonnées en y par valeur-y



```
<svg width="200px" height="200px" viewBox="0 0 200 200" >
    <g id="carre">
        <rect x="10" y="10" width="20" height="20"
        style="fill:none; stroke:black; stroke-width: 2;"/>
    </g>
    <use xlink:href="#carre" transform="scale(2,3)"/>
</svg>
```

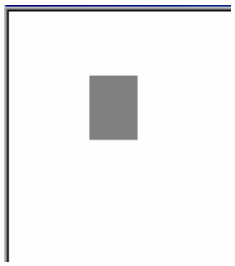
Transformation du système de coordonnées

- Transformation par rotation : rotate
  - Il est possible de faire tourner le système de coordonnées d'un angle quelconque :

```
<svg width="200px" height="200px" viewBox="0 0 200 200" >
        <rect x="70" y="30" width="15" height="20"
        transform="rotate(45)" style="fill:gray;"/>
</svg>
```

Transformation du système de coordonnées

- Enchaînement des transformations
  - Il est possible d'appliquer plusieurs transformations sur un objet graphique
  - Il suffit d'énumérer les transformations séparées par des blancs comme valeur de l'attribut transform

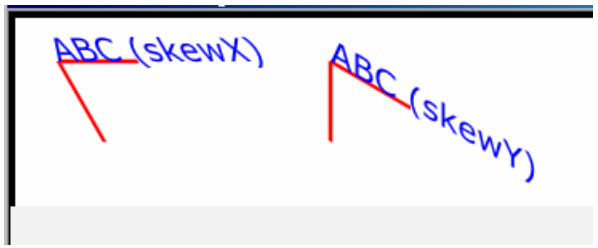


```
<svg width="200px" height="200px" viewBox="0 0 200 200" >
    <g transform="translate(30,20)">
        <g transform="scale(2)">
            <rect x="10" y="10" width="15" height="20"
                style="fill:gray;"/>
        </g>
    </g>
</svg>
```

Transformation du système de coordonnées

- Transformations skewX et skewY
  - skewX(angle) et skewY(angle) permettant de gauchir un des axes
- Exemple : transform-incl.svg

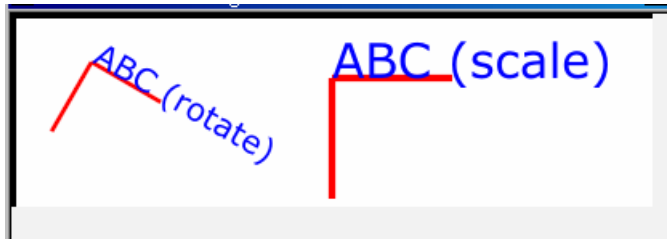
```
<g transform="translate(30,30)">
  <g transform="skewX(30)">
    <g style="fill:none; stroke:red; stroke-width:3">
      <line x1="0" y1="0" x2="50" y2="0" />
      <line x1="0" y1="0" x2="0" y2="50" />
    </g>
    <text x="0" y="0" style="font-size:20; font-family:Verdana; fill:blue">
      ABC (skewX)
    </text>
  </g>
</g>
```



Transformation du système de coordonnées

Exemple : transform-rotate.svg

```
<g transform="translate(50,30)">
  <g transform="rotate(30)">
    <g style="fill:none; stroke:red; stroke-width:3">
      <line x1="0" y1="0" x2="50" y2="0" />
      <line x1="0" y1="0" x2="0" y2="50" />
    </g>
    <text x="0" y="0" style="font-size:20; font-family:Verdana;
  fill:blue">
      ABC (rotate)
    </text>
  </g>
</g>
```



Transformation du système de coordonnées

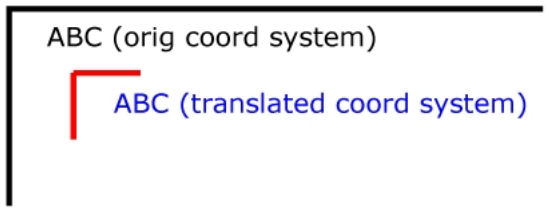
- On peut spécifier un nouvel espace utilisateur par transformation de l'espace original : sys-coords2.svg
  - Par exemple, à partir de l'espace

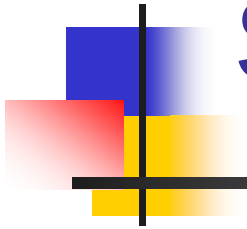
```
<svg width="400px" height="150px">
  <g style="fill:none; stroke:black; stroke-width:3">
    <!-- Draw the axes of the original coordinate system -->
    <line x1="0" y1="1.5" x2="400" y2="1.5" />
    <line x1="1.5" y1="0" x2="1.5" y2="150" />
  </g>
  <g>
    <text x="30" y="30" style="font-size:20 font-
  family:Verdana">
      ABC (orig coord system)
    </text>
  </g>
</svg>
```

Transformation du système de coordonnées

- On pourra obtenir par translation : sys-coords3.svg

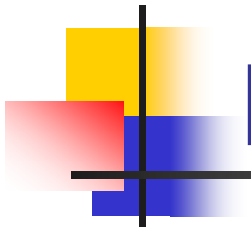
```
…
  <g transform="translate(50,50)">
    <g style="fill:none; stroke:red; stroke-width:3">
      <!-- Draw lines of length 50 user units along
           the axes of the new coordinate system -->
      <line x1="0" y1="0" x2="50" y2="0" style="stroke:red"/>
      <line x1="0" y1="0" x2="0" y2="50" />
    </g>
    <text x="30" y="30" style="font-size:20 font-family:Verdana">
      ABC (translated coord system)
    </text>
  </g>
</svg>
```





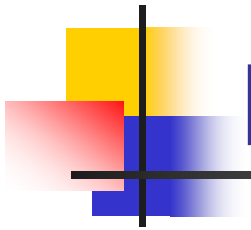
SVG

Le rendu



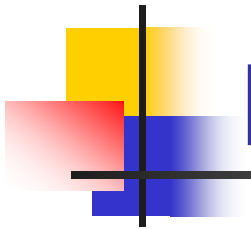
Le rendu

- Définition
  - Les éléments <path>, <text> et les formes de base peuvent être remplis et coloriés (c.à.d. peints sur les bords)
  - On appellera cette opération *le rendu*
  - En SVG, on peut *rendre* avec :
    - une couleur simple
    - un gradient (linéaire ou radial)
    - un motif (vecteur ou image)
    - des motifs personnalisés disponibles par extension



Le rendu

- Pour effectuer ce rendu, on utilise
  - les propriétés *fill* et *stroke*
  - *fill* : s'occupe du remplissage
    - précise le degré d'opacité de la couleur du remplissage
  - *stroke* : s'occupe du dessin, il précise :
    - épaisseur
    - jonction de lignes
    - arrondi des angles pointillés



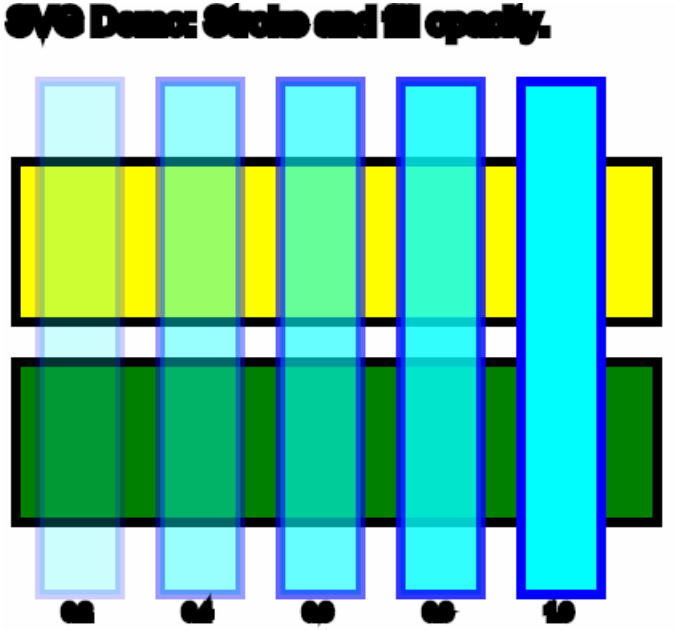
Le rendu

- Exemple : rendu1.svg

```
<rect x="10"  y="100" width="400" height="100" style="fill:yellow"  />
<rect x="10"  y="225" width="400" height="100" style="fill:green"  />
<g style="stroke:blue;fill:cyan">
 <rect x="25"  y="50"  width="50"  height="320" style="stroke-opacity:0.2;fill-opacity:0.2"  />
 <rect x="100" y="50"  width="50"  height="320" style="stroke-opacity:0.4;fill-opacity:0.4"  />
 <rect x="175" y="50"  width="50"  height="320" style="stroke-opacity:0.6;fill-opacity:0.6"  />
 <rect x="250" y="50"  width="50"  height="320" style="stroke-opacity:0.8;fill-opacity:0.8"  />
 <rect x="325" y="50"  width="50"  height="320"   />
</g>
```

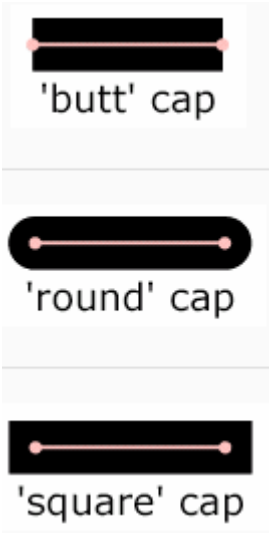
Le rendu

- Exemple : rendu1.svg,



Le rendu

- Exemple : rendu2.svg

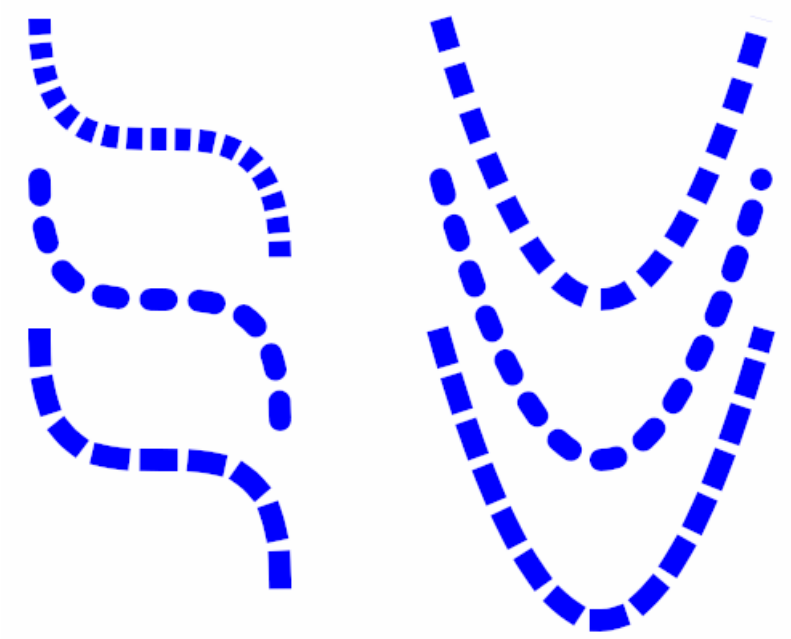


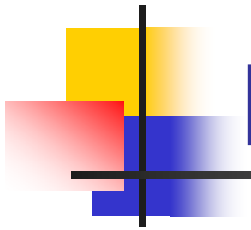

```
<g style="shape-rendering:default; stroke-width:14; stroke:#0000FF;
    fill:none" >
   <path d="M50,50C50,200,200,50,200,200" style="stroke-linecap:butt;
    stroke-dasharray:10 5" />
   <path d="M50,150C50,300,200,150,200,300" style="stroke-
    linecap:round; stroke-dasharray:10 20" />
   <path d="M50,250C50,400,200,250,200,400" style="stroke-
    linecap:square; stroke-dasharray:10 20" />
   <path d="M300,50Q400,400,500,50" style="stroke-linecap:butt;
    stroke-dasharray:20 10" />
   <path d="M300,150Q400,500,500,150" style="stroke-linecap:round;
    stroke-dasharray:10 20" />
   <path d="M300,250Q400,600,500,250" style="stroke-linecap:square;
    stroke-dasharray:10 20" />
</g>
```

stroke-dasharray:10 20 : pointillés : taille espace taille trait

Le rendu

- Exemple : rendu2.svg





Le rendu

- Gradients et motifs
  - En SVG, on peut remplir un objet ou en souligner le contour par :
    - une couleur
    - un gradient (de couleurs)
    - un motif
  - Un gradient de couleurs consiste en une transition *douce* entre deux couleurs selon un vecteur
  - Les gradients de couleur peuvent être :
    - linéaires
    - radiaux

Le rendu

Gradient linéaire : syntaxe

- Exemple :
  - Faire progresser un gradient d'un coin à l'autre d'un carré, en faisant évoluer le noir du coin en haut (point 0,0 ou 0%) au blanc à l'autre coin (point à 100%)

```
    <defs>
      <rect id = "r1" width = "350" height = "350" stroke =
       "black" stroke-width = "1"/>
      <linearGradient id = "g1" x1 = "0%" y1 = "0%" x2 =
        "100%" y2 = "100%">
        <stop stop-color = "black" offset = "0%"/>
        <stop stop-color = "white" offset = "100%"/>
      </linearGradient>
    </defs>
    <use x = "325" y = "25" xlink:href = "#r1" fill = "url(#g1)"/>
</svg>
```

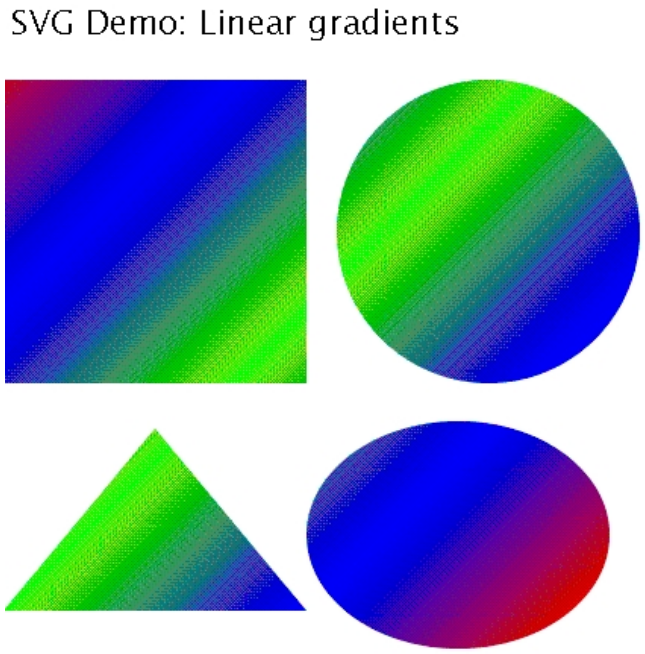
Le rendu

Gradient linéaire : syntaxe

- Exemple : gradient-lin.svg

```
<linearGradient id="grad1" x1="0%" y1="0%"
    x2="100%" y2="100%">
    <stop offset="0%" stop-color="#FF0000"/>
    <stop offset="25%" stop-color="#0000FF"/>
    <stop offset="50%" stop-color="#00FF00"/>
    <stop offset="75%" stop-color="#0000FF"/>
    <stop offset="100%" stop-color="#FF0000"/>
</linearGradient>
```

➔ Ici, le dégradé de gradient de couleur se fait tous les 25%

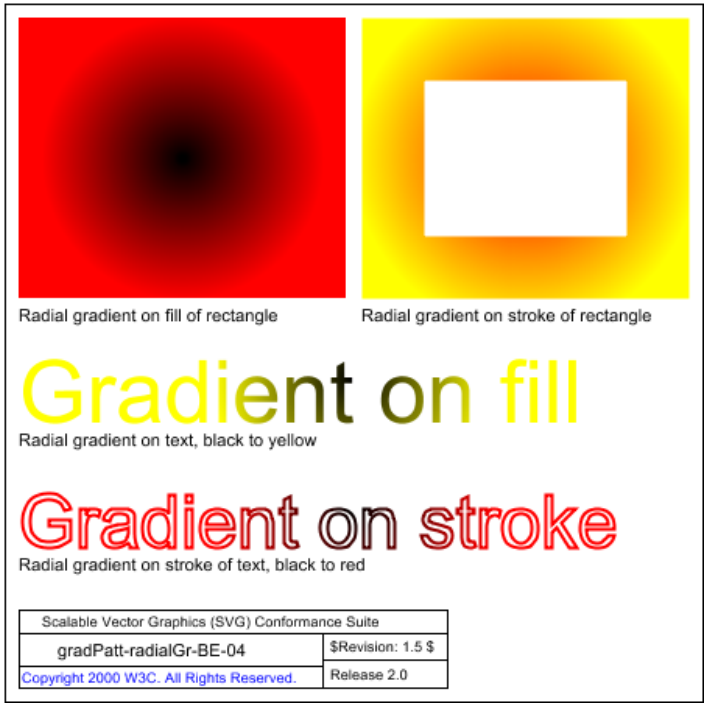


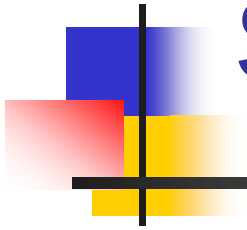
Le rendu

Gradient radial : syntaxe

- Exemple : gradPatt-radialGr-BE-05.svg

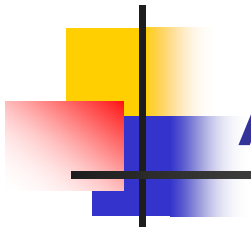
```
<radialGradient gradientUnits="userSpaceOnUse" id="Grad1" cx="115" cy="100"
    r="90" fx="115" fy="100">
      <stop style="stop-color:black" offset="0"/>
      <stop style="stop-color:red" offset="1"/>
</radialGradient>
```





SVG

Animation



Animation

- Concepts de base
  - On peut pratiquement animer chaque attribut
  - Les animations SVG sont basées sur SMIL 2.0
  - Principe : indiquer
    - l'attribut à animer
    - le début et la fin de l'animation
    - la transformation à appliquer avec valeur de début et de fin
    - le comportement une fois le temps d'animation écoulé

Animation

- Exemple
  - Rectangle animé : anim-rectangle.svg

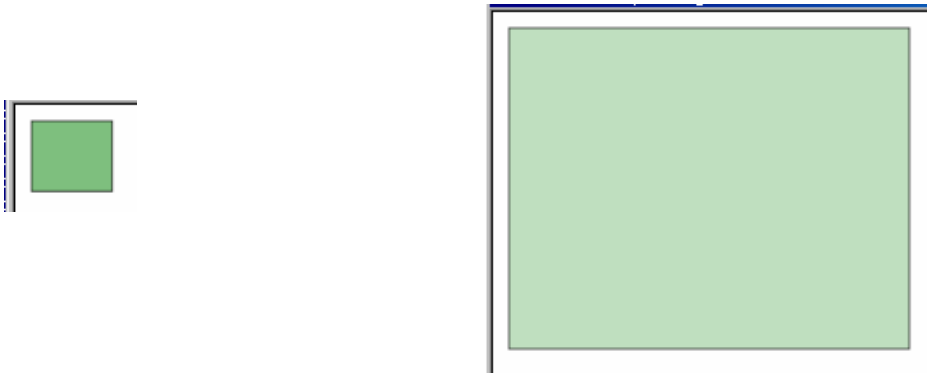
```
<?xml version="1.0" standalone="no"?>
<svg width="8cm" height="3cm" viewBox="0 0 800 300">
   <rect x="10" y="10" width="200" height="20" stroke="black"
   fill="none">
            <animate
            attributeName="width"
            attributeType="XML"
            from="200" to="20"
            begin="0s" dur="5s"
            fill="freeze" />
   </rect>
</svg>
```

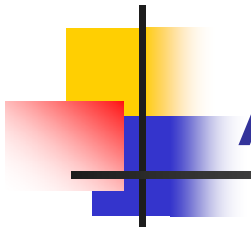
Début de l'animation

Fin de l'animation

Animation

- Animation multiple : anim-multiple.svg
  - Faire grossir un carré vert d'une taille de 20x20 pixels à une taille de 250x200 pixels sur une durée de 8s
  - Pendant les 3 premières secondes l'opacité du carré croît puis elle décroît pendant les 3s suivantes
  - Fill-opacity fait parti du style CSS



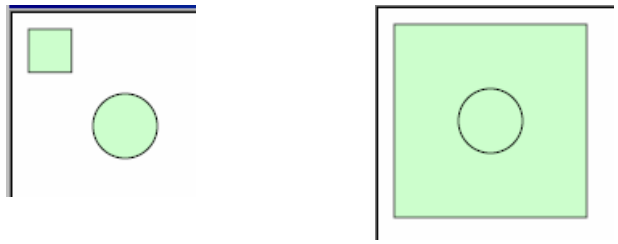


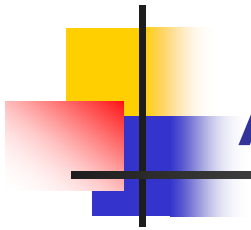
Animation

```
<rect x="10" y="10" width="20" height="20" style="stroke: black; fill: green;
        style: fill-opacity: 0.25;">
    <animate
      attributeName="width" attributeType="XML"
      from="20" to="250" begin="0s" dur="8s" fill="freeze" />
    <animate
      attributeName="height" attributeType="XML"
      from="20" to="200" begin="0s" dur="8s" fill="freeze" />
    <animate
      attributeName="fill-opacity" attributeType="CSS"
      from="0.25" to="1" begin="0s" dur="3s" fill="freeze" />
    <animate
      attributeName="fill-opacity" attributeType="CSS"
      from="1" to="0.25" begin="3s" dur="3s" fill="freeze" />
</rect>
```

Animation

- Exemple encore plus ambitieux : anim-rect-circle.svg
  - Anime un carré et un cercle
  - Le carré grossit depuis une taille de 20x20 à une taille de 120x120 pixels sur une durée de 8s
  - 2s après, le début de l'animation le rayon du cercle commence à grossir de sa taille initiale de 20 pixels jusqu'à une taille de 50 pixels sur une durée de 4s





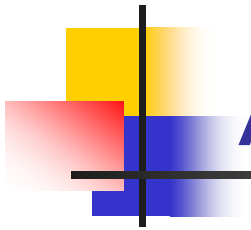
Animation

```
<rect x="10" y="10" width="20" height="20" style="stroke: black; fill:
    #cfc;">
    <animate
        attributeName="width" attributeType="XML"
        from="20" to="120" begin="0s" dur="8s" fill="freeze" />
    <animate
        attributeName="height" attributeType="XML"
        from="20" to="120" begin="0s" dur="8s" fill="freeze" />
</rect>
<circle cx="70" cy="70" r="20" style="fill: #cfc; stroke:black;">
    <animate
        attributeName="r" attributeType="XML"
        from="20" to="50" begin="2s" dur="4s" fill="freeze" />
</circle>
```

Animation

- Animation synchronisée : anim-synchro.svg
  - L'une commence quand l'autre s'arrête

```
<circle cx="60" cy="60" r="30" style="fill: #9f9; stroke:gray;">
    <animate id="c1"
      attributeName="r" attributeType="XML"
      from="30" to="10" begin="0s" dur="4s" fill="freeze" />
</circle>
<circle cx="120" cy="60" r="10" style="fill: #9f9; stroke:gray;">
    <animate
      attributeName="r" attributeType="XML"
      from="10" to="30" begin="c1.end" dur="4s" fill="freeze" />
</circle>
```



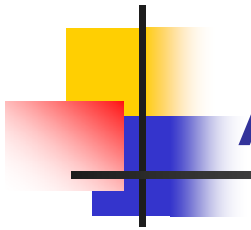
Animation

- Arrêt d'une animation : attribut end
  - Ici, l'animation commence 6s après le chargement de la page et dure au moins 12s ou jusqu'à ce que l'animation autreAnim s'arrête

```
<animate
    attributeName="r" attributeType="XML"
    begin="6s" dur="12s" end="autreAnim.end"
    from="10" to="100" fill="freeze" />
```

  - Celle-ci commence à 6s et devrait durer au moins 12s mais est interrompue après 9s

```
<animate
    attributeName="r" attributeType="XML"
    begin="6s" dur="12s" end="9s"
    from="10" to="100" fill="freeze" />
```



Animation

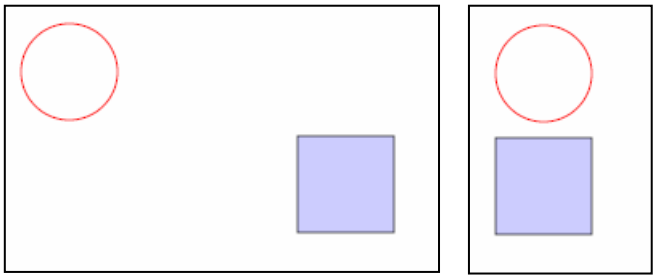
- Synchronisation d'une animation avec une répétition
  - Jusqu'alors l'animation n'est lancée qu'une fois
    - ➔ fill est positionné à freeze
  - Si on veut que l'objet revienne à son état initial :
    - ➔ positionner fill à remove (valeur par défaut)
  - 2 attributs permettent de répéter une animation
    - repeatCount :
      - donne le nb d'animations d'une animation
    - repeatDur :
      - donne la durée d'une répétition
  - Répétition infinie
    - L'animation dure jusqu'à ce que l'utilisateur quitte la page
    - Positionner repeatDur ou repeatCount à indefinite

Animation

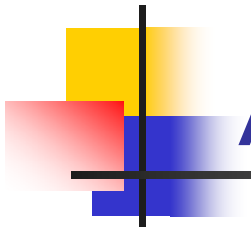
- Exemple : anim-repeat-synchro.svg

```
<circle cx="60" cy="60" r="30" style="fill: none; stroke:red;">
    <animate id="cercleAnim"
      attributeName="cx" attributeType="XML"
      from="60" to="260" begin="0s" dur="5s" fill="freeze" />
</circle>
<rect x="230" y="100" width="60" height="60" style="stroke: black; fill:
    #ccf;">
    <animate
      attributeName="x" attributeType="XML" from="230" to="30"
    begin="cercleAnim.repeat(1)+2.5s" dur="5s" fill="freeze" />
</rect>
```

Avant



Après



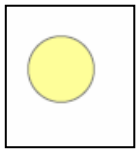
Animation

- L'élément <set>
  - C’est un raccourci pour affecter des valeurs d'animation, des attributs et propriétés non numériques :
  - Exemple :
    - faire apparaître un texte qui n'était pas visible où les attributs from to n'ont pas de sens
    - <set> permet d'obtenir cet effet en n'ayant à spécifier que l'attribut à modifier avec to et l'information de durée souhaitée

Animation

- L'élément <set> : exemple : anim-set.svg

```
<circle cx="60" cy="60" r="30" style="fill: #ff9; stroke:gray;">
    <animate id="c1"
        attributeName="r" attributeType="XML"
        from="30" to="0" begin="0s" dur="4s" fill="freeze" />
</circle>
<text text-anchor="middle" x="60" y="60" style="visibility:hidden;">
        <set attributeName="visibility" attributeType="CSS"
        to="visible" begin="4.5s" dur="1s" fill="freeze"/>
        Plus rien !
</text>
```

Avant  

Après

Animation

- L'élément <animateColor> : anim-color.svg
  - Comme <animate> ne permet pas de faire varier les couleurs, alors on utilise <animateColor> avec ses attributs from et to

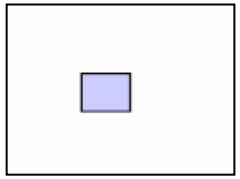
```
<circle cx="60" cy="60" r="30" style="fill: #ff9; stroke: gray; stroke-
   width: 10;">
   <animateColor attributeName="fill"
            begin="2s" dur="4s" from="#ff9" to="red"  fill="freeze" />
   <animateColor attributeName="stroke"
            begin="2s" dur="4s" from="gray" to="blue"  fill="freeze" />
</circle>
```

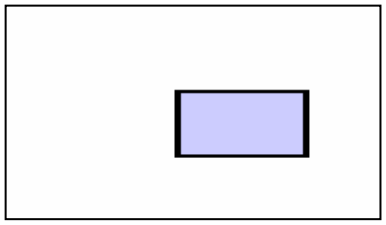
avant

après

Animation

- L'élément animateTransform : anim-transform.svg
  - permet d'animer la rotation, la translation, le changement d'échelle… en utilisant l'attribut type :
  - Exemple :
    - Etirer un rectangle de sa taille normale à 4 fois cette taille horizontalement et 2 fois verticalement
    - Noter que le rectangle est centré autour de l'origine afin qu'il ne se déplace pas et il est encapsulé dans un <g> de façon à pouvoir être placé à une position quelconque

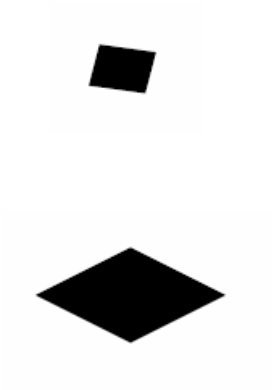




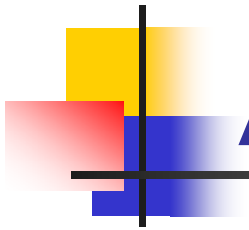
```
<g transform="translate(120,60)">
   <rect x="10" y="10" width="20" height="20" style="fill: #ccf; stroke:
            black;">
            <animateTransform
                     attributeName="transform" attributeType="XML"
                     type="scale"          from="1" to="4 2" begin="0s"
                     dur="4s" fill="freeze"
            />
   </rect>
</g>
```

Animation

- Transformations multiples : anim-multipl-transform.svg
  - Attribut : additive
    - Sa valeur par défaut est replace ce qui remplace la transformation de l'objet par celle spécifiée



```
<g transform="translate(120,60)">
   <rect x="20" y="20" width="20" height="20" style="fill: #ffg; stroke:
            black;">
      <animateTransform
            attributeName="transform" attributeType="XML"
            type="scale" from="1" to="4 2" additive="sum" begin="0s"
            dur="4s"   fill="freeze" />
      <animateTransform
            attributeName="transform" attributeType="XML"
            type="rotate" from="0" to="45" additive="sum" begin="0s"
            dur="4s"   fill="freeze" />

   </rect>
</g>
```



Animation

- L'élément animateMotion : anim-motion.svg
  - <translate> anime l'objet selon une droite
  - <animateMotion> autorise l'animation le long d'un chemin arbitraire
  - Exemple1 : animation le long d'une droite :

```
<svg width="8cm" height="8cm">
<g transform="translate(120,60)">
   <rect x="0" y="0" width="30" height="30" style="fill: #ccc;"/>
   <circle cx="30" cy="30" r="15" style="fill: #cfc; stroke:
   green;"/>
   <animateMotion from="0,0" to="60,30"  dur="4s"
   fill="freeze" />
</g>
</svg>
```

Animation

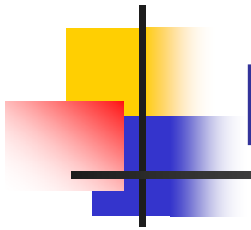
- L'élément animateMotion
  - Exemple2 : animation le long d'un chemin : anim-motion-path.svg
    - Utiliser l'attribut path : on déplace un triangle le long d'une courbe de Bézier cubique

```
<svg width="8cm" height="8cm">
<path d="M50,125 C 100,25 150,225, 200, 125" style="fill: none;
   stroke: blue;"/>
<path d="M-10, -3 L10,-3 L0, -25z" style="fill: yellow; stroke:
   red;">
   <animateMotion
      path="M50,125 C 100,25 150,225, 200, 125" dur="6s"
      fill="freeze"/>
</path>
</svg>
```

Animation

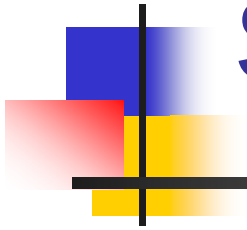
- L'élément animateMotion
  - Exemple3 : animation le long d'un chemin avec rotation automatique : anim-motion-path-rotate.svg
    - Utiliser l'attribut path : on déplace un triangle le long d'une courbe de Bézier cubique

```
<svg width="8cm" height="8cm">
<path d="M50,125 C 100,25 150,225, 200, 125" style="fill: none;
   stroke: blue;"/>
<path d="M-10, -3 L10,-3 L0, -25z" style="fill: yellow; stroke:
   red;">
   <animateMotion
   path="M50,125 C 100,25 150,225, 200, 125" rotate="auto"
   dur="6s" fill="freeze"/>
</path>
</svg>
```



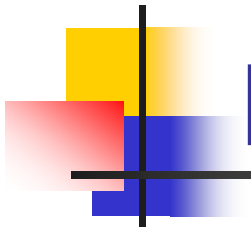
Exercice

- Énoncé
  - Reprendre le dessin de la voiture avec les courbes de Bézier
  - Faire avancer la voiture suivant un chemin à dessiner



SVG

Interaction



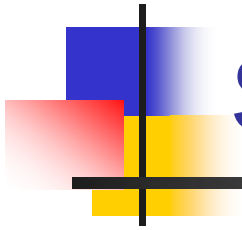
Balise <script>

- Comment ?
  - On ajoute à l'objet d'interaction :
    - Un attribut onnomEvenement
  - Exemples :
    - <circle … onmouseover="élargir_cercle(evt)" …
    - <circle … onmouseout="réduire_cercle(evt)" …
  - Comme pour JavaScript
    - On inclut les fonctions de gestion entre <script></script> : ceci indique que l'on quitte le monde SVG pour celui d'ECMA Script (*European Computer Manufacturers Association*)

Exemple

script-attribute.svg

```
<svg width="8cm" height="8cm">
    <script type="text/ecmascript">
        function élargir_cercle(evt){
        var cercle = evt.getTarget();
        cercle.setAttribute("r", 50);
        }
        function réduire_cercle(evt){
        var cercle = evt.getTarget();
        cercle.setAttribute("r", 25);
        }
    </script>
    <circle cx="150" cy="100" r="25" fill="red"
    onmouseover="élargir_cercle(evt)"
    onmouseout="réduire_cercle(evt)"/>
    <text x="150" y="175" style="text-anchor:
        middle;">Passer la souris sur le cercle
        pour changer sa taille.
    </text>
</svg>
```

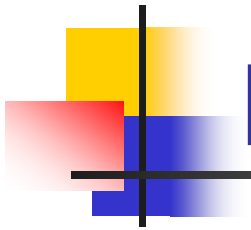


SVG

Filtres

http://www.svgopen.org/2002/papers/hirtzler_using_svg_filters/

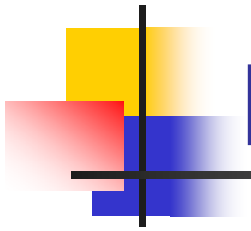
http://pilat.free.fr/filtres/etude.htm



Filtres

- Objectif
  - Agir sur les images pour créer des effets
  - Pour déclarer un filtre en SVG,
    - on utilise le tag <filter> dans une <defs>

```
<defs>
  <filter id="MyFilter"
     <!-- Espace utilisé pour les coordonnées -->
    filterUnits="userSpaceOnUse" x="0" y="0"
    width="400"       height="400">
     <!-- Valeurs du filtre -->
    <feColorMatrix …>
  </filter>
</defs>
```



Filtres

- Filtre colorimétrique
  - permet de changer l'aspect d'une image
  - Ceci peut se faire dans <feColorMatrix > par :
    - filtrage matriciel :
      - application d'un masque de coefficients de transformations des valeurs des pixels
    - ou par modification des propriétés de l'image
      - en agissant sur la saturation, la luminance, etc.

Filtrage matriciel

- Modification de la couleur : filtre-feColorMatrix.svg

```
<defs>
   <!– Annonce du filtre -->
   <filter id="MyFilter" filterUnits="userSpaceOnUse" x="0"
     y="0" width="400" height="400">
       <!– Valeurs du filtre -->
       <feColorMatrix id='color' type="matrix"
             values="1 0 1 1 1
                     0 1 0 0 0
                     0 0 1 0 0
                     0 0 0 1 0"/>
   </filter>
   <!– Image sur laquelle s'applique le filtre -->
   <image id="MyImage" width='400' height='400'
     xlink:href='puzzle.jpg'/>
</defs>
<!– Utilisation du filtre -->
<use filter="url(#MyFilter)" xlink:href='#MyImage' x='0'
   y='0'/>
```

← masque

Filtrage matriciel

- Résultat en appliquant un filtre unité :
  - values="1 0 0 0 0
    0 1 0 0 0
    0 0 1 0 0
    0 0 0 1 0"/>

Filtrage par action sur les propriétés de l'image

- Saturate

```
<feColorMatrix id='color'  type="saturate"
  values="0"/>
```

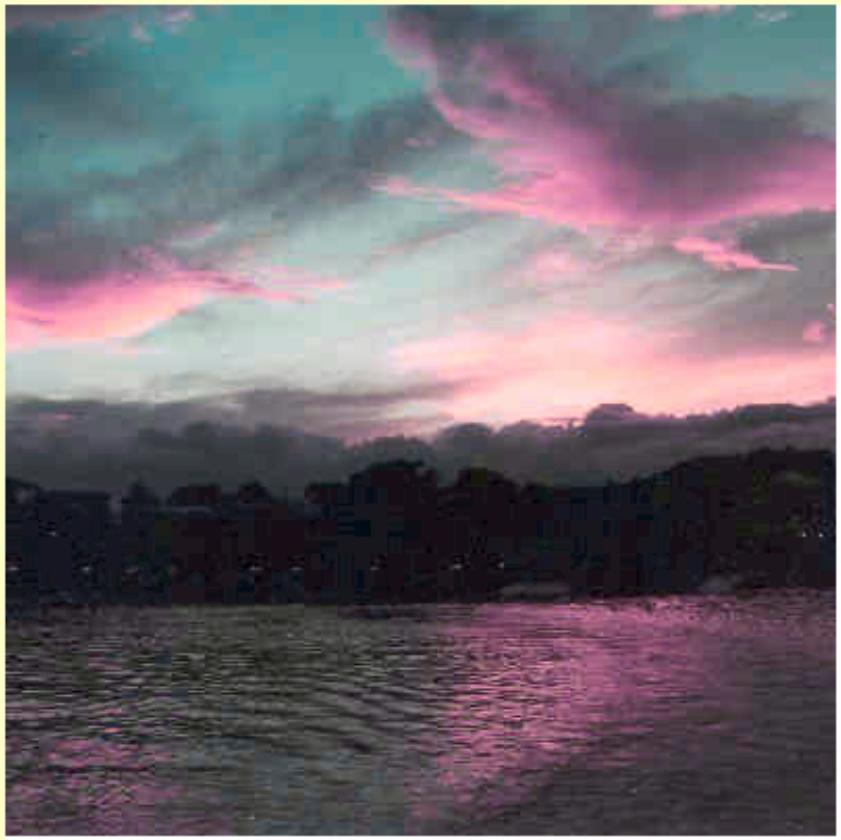
- hueRotate

```
<feColorMatrix id='color' type="hueRotate"
        values="90" />
```

- luminanceToAlpha:

```
<feColorMatrix id='color' type="luminanceToAlpha"
  values="ok" />
```

- Exemple plus complet : filtre_fcm.htm
  - Cet exemple vous montre en même temps comment insérer du SVG dans du HTML
    - en utilisant le DOM



Filtre feColorMatrix

Type matrix

Cliquez + ou - pour chaque paramètre

0.92	0.715	-0.635	0	0
-0.038	0.817	0.221	0	0
0.618	-0.296	0.677	0	0
0	0	0	1	0

Type saturate (Essayez 0 pour le noir et blanc)

1

Type hueRotate

-45

Type luminanceToAlpha

OK

Supprimer le filtre

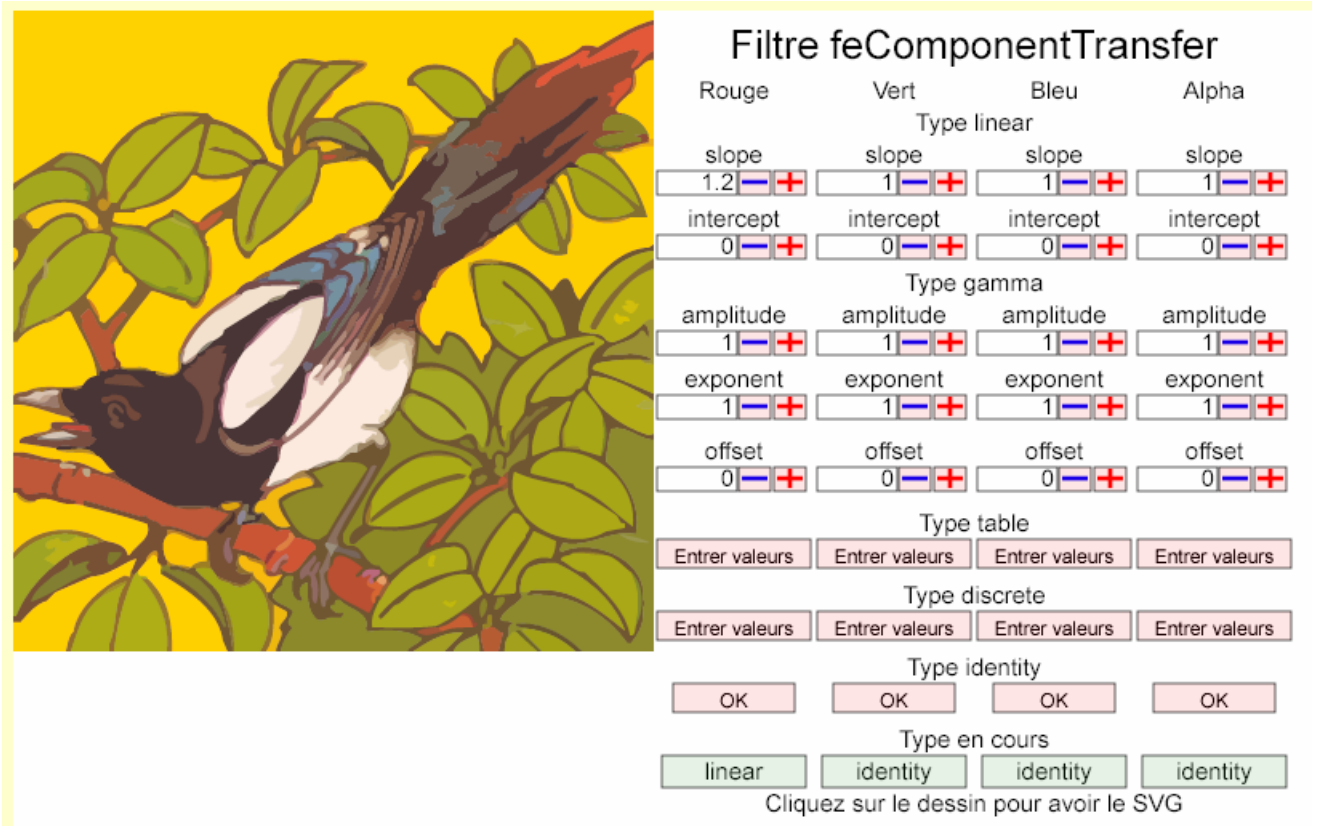
OK

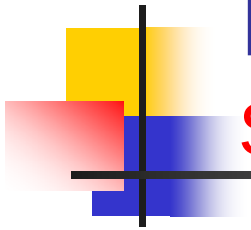
Cliquez sur le dessin pour avoir le SVG

Filtres

Modification de la couleur d'une image

- Exemple complet : filtre_fct.htm





Filtres

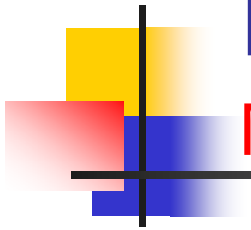
Superposition d'images

- Filtre : filtre_fme.htm
  - Permet de créer une nouvelle image à partir d'autres sources
  - Dans l'exemple :
    - on importe une image de ciel nuageux afin de l'utiliser comme arrière-plan pour le bateau

Filtres

Création de motif et remplissage de forme

- Filtre : filtre_ftu.htm
  - Crée une texture ou crée du bruit (fonction de turbulence de Perlin)
  - Voir définition : http://www.yoyodesign.org/doc/w3c/svg1/filters.html#feTurbulence
  - Paramètres :
    - type: fractalNoise ou turbulence
  - Pour chaque type:
    - baseFrequency : deux nombres positifs pour x et y
    - numOctaves : entier positif
    - Seed : entier positif (change le départ du générateur aléatoire de couleurs)
  - En utilisant pattern, on peut remplir une forme avec la texture créée



Filtres

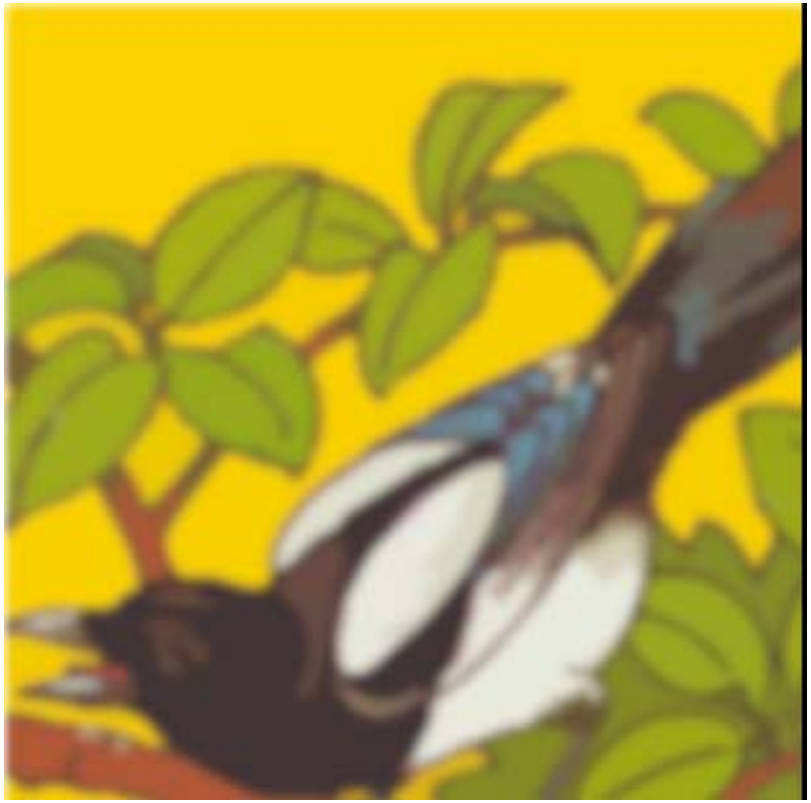
Modification d'une image

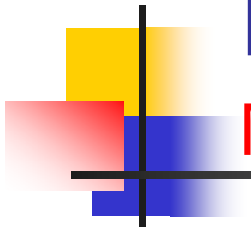
- Flou gaussien : filtre-feGaussianBlur.svg
  - Paramètres :
    - stdDeviation : deux entiers positifs pour x et y
    - in: image à modifier
- Effet
  - Flou gaussien
- L'attribut stdDeviation
  - Donne la puissance du flou
  - Si on donne deux valeurs de stdDeviation, alors :
    - le premier indique la force du flou dans la direction des x
    - le deuxième dans la direction des y

Filtres

Modification d'une image

- Autre exemple : filtre-feGaussianBlur2.svg





Filtres

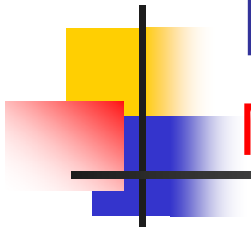
Mixage d'images

- Principe
  - Ce filtre permet la fusion de couches en une seule image
  - Il compose les couches d'image en entrée l'une sur l'autre en utilisant l'opérateur *in* avec *l'Entrée1* (correspondant au premier élément 'feMergeNode' enfant) en-dessous et la dernière entrée spécifiée, *l'EntréeN* (correspond au dernier élément 'feMergeNode' enfant), par-dessus
  - De nombreux effets produisent un certain nombre de couches intermédiaires pour la création de l'image de sortie finale

Filtres

Mixage d'images

- Exemple : filtre_fme.htm



Filtres

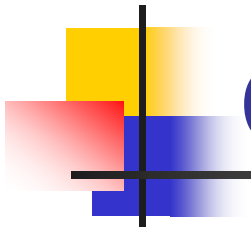
Mixage d'images

- Filtre feBlend
  - Principe :
    - Ce filtre compose deux objets à l'aide de modes de mélange couramment employés par les logiciels d'imagerie
    - Il effectue une combinaison au pixel de deux images en entrée

Filtres

Mixage d'images

- Filtre feBlend : filtre_fbl.htm



Conclusion

- SVG
  - Alternative crédible à Flash
  - Standard W3C : fortement couplé aux autres standard XML
  - Cumule les avantages du graphique vectoriel et du standard XML
  - Nombreux outils d'édition et de visualisation disponibles sur le marché
  - Possibilités graphiques étendues :
    - Richesse des éléments
    - Possibilité de structuration et groupage
    - Nombreuses possibilités de transformation et nombreux effets
    - Animation avec SMIL
    - Interactivité avec l'utilisateur