Introduction à SVG statique

Code: svg-intro

Originaux

url: **http://tecfa.unige.ch/guides/tie/html/svg-intro/svg-intro.html**

url: **http://tecfa.unige.ch/guides/tie/pdf/files/svg-intro.pdf**

Auteurs et version

- Daniel K. Schneider
- Version: 1.5 (modifié le 25/10/07)

Prérequis

Module technique précédent: xml-dom

Module technique précédent: xml-tech

Modules suivants

Module technique suivant: **svg-dyn** (SVG Dynamique)

Module technique suivant: **svg-xslt** (Visualisation SVG avec XSLT)

Autres modules

Module technique suppl.: xml-xslt

Module technique suppl.: visu-gen (ainsi que les divers preréquis Php !)

Abstract

Introduction à SVG statique

Objectifs

- Philosophie de SVG, sa place sur le Web
- Graphismes SVG de base (formes, couleurs, positions, etc.)
- Groupage et transformations d’éléments

1. Table des matières détaillée

2. Scalable Vector Graphics - un nouveau paradigme

2.1 Origine et but

- SVG traduit: Graphiques Vectoriels Adaptables
- SVG est un standard W3C (en XML) et s'intègre bien aux autres formats W3C.

  url: **http://www.w3.org/TR/SVG/**

  De la spécification: "SVG est un langage de description de graphiques bi-dimensionnels en XML. SVG admet trois types d'objets graphiques : des contours graphiques vectoriels (par exemple, des tracés consistant en lignes droites et courbes), des images et du texte. Les objets graphiques peuvent être regroupés, stylés, transformés et composés dans des objets précédemment rendus. L'ensemble de fonctions comprend des transformations imbriquées, des tracés de rognage, des masques basés sur la couche alpha et des objets de gabarit.

  Les dessins SVG peuvent être interactifs et dynamiques. On peut définir et déclencher des animations, soit déclarativement (i.e., en incorporant les éléments d'animation SVG dans un contenu SVG), soit via un script.

Utilisations les plus intéressantes (actuellement):

- visualisation de contenus (économiques, processus, cartes, etc.)
- interface utilisateurs pour certains types d'applications Internet
- dessins statiques, animés ou même interactifs dans le monde de l'éducation
- animations multimédia de contenus formalisables (chimie, maths, etc.)

2.2 Pourquoi SVG ?

Avantages de "vector graphics"

- rendering correct dans multiples média et à différentes tailles (adaptation)
- possibilité d'appliquer des styles (adaptation 2)
- possibilité d'indexer le texte qui fait partie du graphisme
- taille de l'image (après compression en tout cas)
- facilités d'édition: les éléments sont des objets, hierarchies, etc.

Avantages particuliers de SVG (par rapport à Flash et similaires)

- insertion dans le monde XML/XHTML
  - génération de SVG avec XSLT/PHP etc. à partir de données XML
  - Intégration dans XHTML, viewers SMIL etc.
  - utilisation de CSS
  - scriptable avec JavaScript via DOM (standard)
- possibilité de partager du code, de travailler directement avec le format
- modèle de couleurs sophistiqué, filtres comme dans photoshop
- une spécification assez claire
- meilleurs capacités graphiques dans l'ensemble, voir:

  url: **http://www.carto.net/papers/svg/comparison_flash_svg.html**

2.3 Outils

Validation on-line:

url: http://validator.w3.org/

Viewers:

- Navigateurs Firefox 1.5+, Opéra 9+ ou Safari 3+
- On conseille le viewer 6a de Adobe pour faire du SVG dynamique/SMIL

  url: **http://www.adobe.com/svg/viewer/install/beta.html (V. 6a difficile à trouver !)**
  - Ne prenez pas la version "officielle" 3, car elle ne marche pas avec Mozilla !

Editeurs SVG statique et dynamique

- Pas de logiciel gratuit, on conseille Webdraw ou son successeur (commercial)

  url: **http://tecfa.unige.ch/guides/svg/pointers.html**

Editeurs SVG statique et exporteurs

- Prenez Inkscape (gratuit) pour faire des dessins.
- Note: OpenOffice "Draw", Illustrator etc. permettent de sauver du SVG

SVG avec un éditeur XML

- La DTD est à la fin de la spécification officielle

  url: **http://tecfa.unige.ch/lib/xml/dtd/svg10.dtd (copie locale de SVG 1.0)**

  url: **http://www.w3.org/TR/SVG/ (SVG 1.1 en oct 2007).**
- On conseille d'installer la DTD dans votre éditeur XML si c'est pas fait.

Installation d'une DTD dans Xemacs (pas nécessaire pour Exchanger lite !)

- Il faut copier le fichier avec la DTD qq part sur votre système (de préférence au même endroit que le CATALOG.
- Ensuite éditer le fichier *...XEmacs/xemacs-packages/etc/psgml-dts/CATALOG* qui se trouve dans l'installation Xemacs.
- Ajouter une ligne qui associe un identificateur public avec un fichier

```
Exemple Unix: PUBLIC "-//W3C//DTD SVG 1.0//EN" /web/lib/xml/dtd/svg10.dtd
Exemple Dos:  PUBLIC "-//W3C//DTD SVG 1.0//EN" svg10.dtd
```

- N'oubliez pas de dire à Emacs que *.svg est associé à XML, éditez le fichier **~/.xemacs/init.el** et ajoutez:

```
(setq auto-mode-alist
   (append '(("\.svg"  . xml-mode)) auto-mode-alist))
```

  Note: Votre home (~) sous windows est qq. part dans c:\Documents and Settings\xxxx\. Alternativement on peut aussi insérer cette instruction dans ...*XEmacs\site-packages\lisp\site-start.el*

2.4 Ressources

Spécification (SVG 1.1 Specification, W3C recommendation Jan 2003)

url: **http://www.w3.org/TR/SVG/**

url: **http://www.yoyodesign.org/doc/w3c/svg1/index.html (SVG 1.0 en français)**

Page ressource @ TECFA:

http://tecfa.unige.ch/guides/svg/pointers.html

3. SVG de base

3.1 Viewer et serveur

A. SVG avec un viewer ou plugin

- Pour regarder les exemples il vous faut un viewer SVG ou un navigateur avec SVG natif
- SVG dans Firefox:
  - SVG est inclu par défaut dans Firefox 1.5, Opéra 9 et Safari 3.
  - Désavantage: Il manque les animations (!), certains filtres, fonctionalités pour des fontes, etc.
  - Avantage: On peut combiner XHTML et SVG
- Pour arrêter/mettre en marche SVG natif dans FireFox
  - entrez l'URL: *about:config*
  - Tapez "svg" dans la fenêtre "filter"
  - Double-clic sur *svg.enabled*
- On conseille éventuellement d'installer le plug-in de Adobe (fait du SVG dynamique !)
  - ... mais sur ma machine ca plante souvent depuis la version 2.0 (DKS)

B. Mime-types que votre serveur doit définir (!)

- Lorsque votre navigateur n'affiche pas un fichier svg provenant d'un serveur, il faut configurer les serveur web avec les mime-types suivants.

```
*.svg   image/svg+xml
*.svgz  image/svg+xml
```

3.2 Structure d'une simple page SVG

- Une déclaration XML standarde, par exemple
```
<?xml version="1.0" standalone="no"?>
```
- Pour un document non "standalone", il faut indiquer le DTD:
```
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN"
          "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
```
- La racine d'un contenu SVG est "svg":
```
<svg>
 .....
</svg>
```
- Il faut déclarer un name space dans la racine (si ce n'était pas fait dans un parent):
```
<svg xmlns="http://www.w3.org/2000/svg">
....
</svg>
```

Template SVG 1.0 à copier (pour le plugin Adobe 3.0)

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN"
          "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="400" height="250"
     xmlns="http://www.w3.org/2000/svg">

</svg>
```

Template SVG 1.0 à copier (pour Opéra 9 et Firefox 2.x)

...contient déjà un peu de code SVG que vous pouvez enlever

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="5cm" height="4cm" version="1.1"
     xmlns="http://www.w3.org/2000/svg">

 <desc>Four separate rectangles  </desc>
    <rect x="0.5cm" y="0.5cm" width="2cm" height="1cm"/>
    <rect x="0.5cm" y="2cm" width="1cm" height="1.5cm"/>
    <rect x="3cm" y="0.5cm" width="1.5cm" height="2cm"/>
    <rect x="3.5cm" y="3cm" width="1cm" height="0.5cm"/>
  <!-- Show outline of canvas using 'rect' element -->
  <rect x=".01cm" y=".01cm" width="4.98cm" height="3.98cm"
        fill="none" stroke="blue" stroke-width=".02cm" />

</svg>
```

Exemple 3-1: Hello SVG - la structure d'un fichier SVG

url: **http://tecfa.unige.ch/guides/svg/ex/svg-intro/hello-svg.svg**

Ignorez les détails (rectangle + texte) pour le moment

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN"
          "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg>

 <!-- un petit rectangle avec des coins arroundis  -->
 <rect x="50" y="50" rx="5" ry="5" width="200" height="100"
       style="fill:#CCCCFF;stroke:#000099"/>

 <!-- un texte au meme endroit>
 <text x="55" y="90" style="stroke:#000099;fill:#000099;font-size:24;">
  HELLO cher visiteur
 </text>

</svg>
```

3.3 Imbrication d'un fichier SVG dans HTML

- Note: A moyen terme on devrait pouvoir insérer directement du code SVG dans du code XHTML. Firefox 1.5 implémente cette fonctionalité pour un sous-ensemble de SVG 1.1

Version canonique pour HTML et pseudo XHTML

- Marche pour Firefox et le plugin Adobe

```
<embed src="..." with=".." height="..." type="image/svg+xml" />
```

Version "iframe"

- conseillée si vous utilisez la combinaison plugin Adobe SVG/Mozilla 1.x ou plus ancien

Exemple 3-2: Hello avec SVG et un iframe

url: **http://tecfa.unige.ch/guides/svg/ex/svg-intro/hello-svg-iframe.html**

```
<iframe src="hello-svg.svg" height="540" width="99%" frameborder="0">
... Sorry you need an SVG plugin ...
</iframe>
```

Version XHTML pour la combinaison IE/Adobe Viewer

```
<object id="AdobeSVG" classid="clsid:78156a80-c6a1-4bbf-8e6a-3cd390eeb4e2"> </object>
<?import namespace="svg" urn="http://www.w3.org/2000/svg" implementation="#AdobeSVG"?>
 <svg:svg width="300" height="200">
     <svg:circle cx="150" cy="100" r="50" />
</svg:svg>
```

Balises mixtes SVG / XHTML

Exemple 3-3: Balises SVG / XHTML mixtes (document non-validé)

url: **http://tecfa.unige.ch/guides/svg/ex/svg-intro/hello-svg-within-xhtml.xhtml**

- marche dans les navigateurs avec support SVG natif (genre Firefox 2.0)
- Attention: Le contenu doit être servi comme xhtml (xml) et pas html par le serveur.
- Il faut aussi une déclaration d'un namespace **xmlns:svg** qq. part (étrange ...)

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd" >
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:svg="http://www.w3.org/2000/svg">
<head> <title>SVG within XHTML Demo</title> </head>
<body>
 <p> You can embed SVG into XHTML, provided that your browser natively implements
     SVG. E.g. Firefox 1.5 supports most of static SVG. </p>
  The SVG part starts below <hr />
<svg xmlns="http://www.w3.org/2000/svg" version="1.1" width="400" height="300">
   <!-- un petit rectangle avec des coins arroundis  -->
   <rect x="50" y="50" rx="5" ry="5" width="300" height="100"
         style="fill:#CCCCFF;stroke:#000099"/>
   <text x="55" y="90" style="stroke:#000099;fill:#000099;font-size:24;">
    HELLO cher visiteur </text>
  </svg>
  <hr />
  The SVG part ended above
</body> </html>
```

4. Eléments graphiques de base

SVG définit un certain nombre d'éléments graphiques de base. Voici la liste des éléments les plus importants:

1. "Rectangles <rect>" [20]
2. "Le cercle <circle> et l'ellipse <ellipse>" [22]
3. "Lignes <line> et poli-lignes <polyline>" [24]
4. "Polygones" [26]
5. "Formes arbitraires avec <path>" [27]
6. "Images <image>" [44]
7. Du texte

- Chaque élément graphique est représenté par un élément XML qui est paramétrable avec des attributs XML et qui hérite d'attributs de ses parents.
- Comme dans d'autres langages vectoriels (par ex. VRML), il existe des formes géométriques de base (rectangle, ellipse, cercle, lignes, poly-lignes et polygone). Ensuite il existe une construction pour produire des formes complexes.

4.1 Mécanismes principaux

A. Attributs

- Il faut se référer à la spécification pour connaître tous les détails. Ici, nous ne montrons en règle générale qu'un petit extrait, car leur nombre est énorme !
- La plupart des éléments se partagent un nombre commun d'attributs comme par exemple l'attribut "id" (identificateur) ou encore "style" (styles CSS2)
- La plupart des valeurs d'attributs sont assez intuitifs (pour ceux qui connaissent un peu CSS). Par contre pour certains il existe des véritables sousgrammaires (voir "Formes arbitraires avec <path>" [27] par exemple)

B. Positionnement

- Les objets SVG se positionnent dans un système de coordonnées qui commence en haut et à gauche (pratique standarde en graphisme informatique)
- Il est possible de travailler avec des coordonnées locales

C. Transformations

- Chaque objet peut être translaté, orienté et changé de taille. Il hérite des transformations de l'objet parent, voir "Système de coordonnées, transformations et unités" [46].

D. Style

SVG définit quelques dizaines d'attributs-propriétés applicables à certains éléments. En ce qui concerne les éléments graphiques, voilà les 2 plus importants:

- **`stroke`**, définit comment le bord d'un objet est paint
- **`fill`**, définit comment le contenu d'un objet est paint

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
  - on peut mettre des styles dans un fichier externe comme autres fichiers xml
- Pour chaque style, il existe aussi un attribut de présentation SVG, cela simplifie la génération de contenus SVG avec XSLT.
- Exemple:

  Les attributs de présentation SVG

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

  ont le même effet qu'un déclaration de type CSS2 dans un attribut style:

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

- Voir le "property index" de la spécification pour plus de détails:

  url: **http://www.w3.org/TR/SVG/propidx.html**

E. Attributs XML vs attributs CSS

- Vous avez souvent le choix d'utiliser soit l'attribut style, soit un attribut SVG pour la mise en forme d'un objet.
- Avantage de CSS: vous pouvez transférer vos connaissances CSS/HTML
- Avantage des attribut "xml": plus faciles à générer avec XSLT, PhP etc.
  - Note: on a constaté que le CSS ne marche pas très bien pour le SVG natif de Firefox 1.5/2.0

Exemple 4-1: Deux façons pour faire la mise en forme

Les 2 exemples devraient s'afficher pareillement ...

Méthode CSS (à éviter dans Firefox):

url: **http://tecfa.unige.ch/guides/svg/ex/svg-intro/hello-svg.svg**

```
<rect x="50" y="50" rx="5" ry="5" width="200" height="100"
      style="fill:#CCCCFF;stroke:#000099"/>
<text x="55" y="90" style="stroke:#000099;fill:#000099;font-size:24;">
 HELLO cher visiteur
</text>
```

Méthode attributs

url: **http://tecfa.unige.ch/guides/svg/ex/svg-intro/hello-svg-attribs.svg**

```
<rect x="50" y="50" rx="5" ry="5" width="300" height="100"
      fill="#CCCCFF" stroke="#000099"/>
<text x="55" y="90" stroke="#000099" fill="#000099" font-size="24">
 HELLO cher visiteur
</text>
```

4.2 Rectangles <rect>

- permet de définir des rectangles y compris des coins arrondis

Attributs de base de <rect>:

x = "<coordonné>" et y = "<coordonné>"

indiquent la position du coin supérieur gauche.
direction vers la droite et le bas du canveas

```
x="15"
y="15mm"
```

Par défaut: x et y sont 0, les unités par défaut sont hérités ou sont des pixels

width = "<longeur>" et height = "<longeur>"

définissent la taille du rectangle

```
width = "100"
height = "100"
```

rx = "<length>" et ry = "<length>"

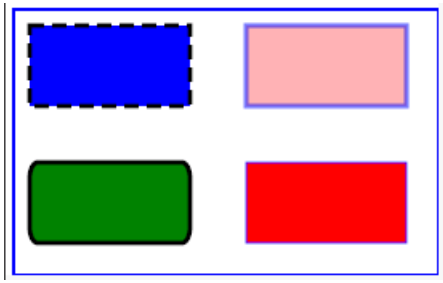
Axe x et y de l'ellipse utilisée pour arrondir, pas de nombre négatif,
ne doit pas dépasser la moité des longeurs respectifs

```
rx = "5"
ry = "5"
```

Exemple 4-2: Rectangles avec style:

url: **http://tecfa.unige.ch/guides/svg/ex/svg-intro/shapes/rectangles1.svg**

```
<?xml version="1.0" standalone="no"?>
<svg width="270" height="170" xmlns="http://www.w3.org/2000/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>
```



4.3 Le cercle <circle> et l'ellipse <ellipse>

Attributs de base pour le cercle et l'ellipse

cx = "<coordonné>" et cy = "<coordonné>"

```
cy="10" cy="20"
```

définissent la position du centre ("c" = circle)

r = "<longeur>"

```
r="10"
définit le radius du cercle
```

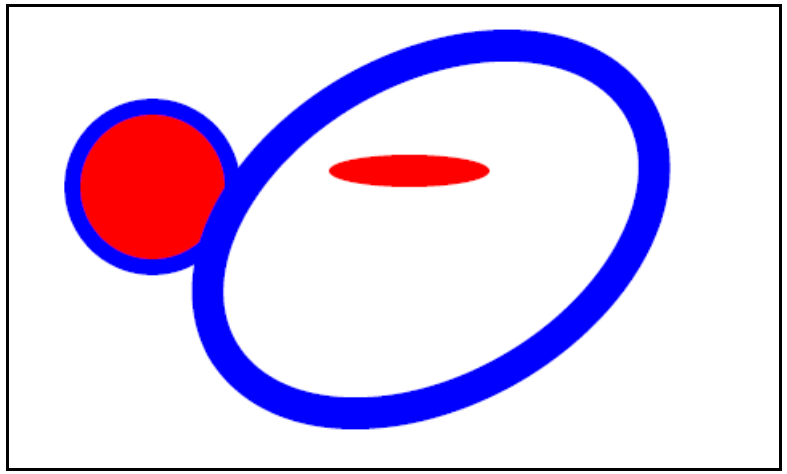
rx = "<longeur>" et ry = "<longeur>"

```
rx="10" ry="20"
définit les radius des axes x et y de l'ellipse ("r"=radius)
```

Exemple 4-3: Ronds figés

url: **http://tecfa.unige.ch/guides/svg/ex/svg-intro/shapes/ronds.svg**

```
<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"  />
```



- Voir 8. “Système de coordonnées, transformations et unités” [46] pour l’explication de la rotation de l’ellipse bleue: **`transform="rotate(-30)"`**

4.4 Lignes <line> et poli-lignes <polyline>

Attributs de base pour <line>:

x1 = "<coordinate>" et y1 = "<coordinate>"

Point de départ
`x1="100" y1="300"`

x2 = "<coordinate>" et y2 = "<coordinate>"

Point de d'arrivé
x2="300" y2="500"

Attributs de base pour <polyline>:

points = "<chemin de points>"

```
points = "10,100,10,120,20,20,........"
Séries de points x,y qui seront liés
```

Exemple 4-4: Lignes

url: **http://tecfa.unige.ch/guides/svg/ex/svg-intro/shapes/lignes.svg**

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

 <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"  />
```

4.5 Polygones

- Un polygone est une forme fermée, la bordure une polyline "fermée"

Attributs de base pour <polygone>:

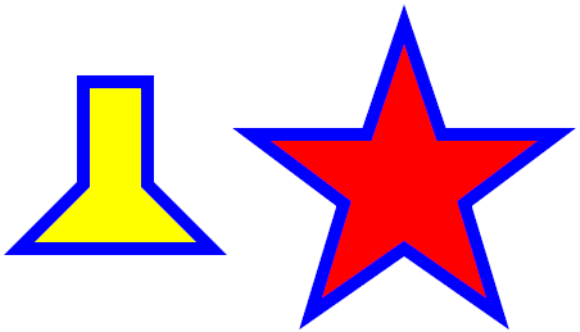
points = "<chemin de points>"

```
points = "10,100,10,120,20,20,........"
Séries de points x,y qui seront reliés (également le dernier au premier point)
```

Exemple 4-5: 2 Polygones

url: **http://tecfa.unige.ch/guides/svg/ex/svg-intro/shapes/polygones.svg**

```
<polygon fill="yellow" stroke="blue" stroke-width="10"
       points="50,250,100,200,100,120,150,120,150,200,200,250"  />
<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" />
```



4.6 Formes arbitraires avec <path>

L'élément <path> permet de définir des formes arbitraires (shapes). Elles peuvent avoir un contour (stroke) et être utilisé comme "clipping path".

A. Attributs de base:

d = "path data"

```
d="M 100 100 L 300 100 L 200 300 z"
d="M100,100 L300,100 200,300 z"    (commande identique)
```

"path data" est un construction assez complexe dont on présentera seulement un extrait ci-dessous

- Note pour la syntaxe des "path data": On peut insérer des virgules et des fins de lignes quand on veut, on peut éliminer l'espace blanc entre une commande (lettre) et les chiffrent qui suivent.

B. Commandes "path data" de base:

M et m

M et **m** sont des commandes "moveto". Il faut s'imaginer le déplacement sans dessiner du crayon qui dessine. **M** indique des coordonnées absolues, **m** des coordonnées relatives par rapport au point de départ. Un M ouvre toujours un "sous-chemin" (voir aussi la commande Z).

```
Syntaxe: M|m (x y)+
M100 100 200 200
```

L et l

L et l dessinent des lignes du point courant vers le(s) point(s) indiqué(s). Cela ressemble donc à l'instruction polyline

```
Syntaxe: L | l
L 200,300 100,200
```

Z et z

Z et z ferment le sous-chemin courant. Autrement dit, on dessine une ligne depuis le point courant vers le début du chemin (défini avec un M ou m)

```
Syntaxe: Z|z
```

H et h, V et v

dessinent des lignes verticales et horizontales.

```
h100
```

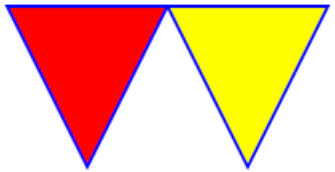
Exemple 4-6: 2 Simple path: un triangle

url: **http://tecfa.unige.ch/guides/svg/ex/svg-intro/shapes/path1.svg**

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

- 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>`

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



C. Commandes "path data" supplémentaires:

- voir la spécification pour plus d'explications

C et c, S et s

Permet de dessiner avec des courbes Bézier

Q et q, T et t

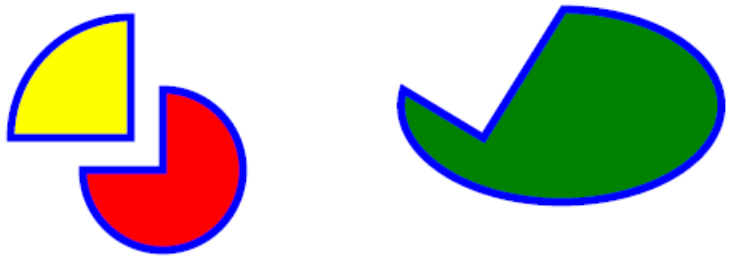
Courbes Bézier quadratiques

A et a

Arc elliptiques (bouts d'ellipse ou de cercle lorsque rx=ry):

```
Syntaxe: A|A (rx ry x-axis-rotation large-arc-flag sweep-flag x y)+
```

Voici un exemple (le code est ci-après)



Exemple 4-7: 2 Simple gateau avec <arc>

url: **http://tecfa.unige.ch/guides/svg/ex/svg-intro/shapes/path2.svg**

```
<path d="M180,180 v-75 a75,75 0 0,0 -75,75 z"
      fill="yellow" stroke="blue" stroke-width="5" />
```

- Pour faire la part jaune, on se positionne à 180, 180 (**`M 180,180`**)
- on dessine une ligne verticale vers y=-75 (**`v-75`**). Cela devien le point de départ pour l'arc.
- on dessine un arc (**a**) avec radius x=75 et radius y=75 (**`75,75`**), sans rotation (0). Le 2ème **`0`** indique l'arc se trouve du côté "petit", le 3ème **`0`** indique une direction de dessin négative. Les **`-75,75`** indiquent l'arrivé de l'arc.
- on ferme le tout (**z**)

```
<path d="M200,200 h-50 a50,50 0 1,0 50,-50 z"
      fill="red" stroke="blue" stroke-width="5" />
```

- La part rouge se fait similairement, on dessine l'arc dans le même sens (négatif), mais du côté "large" (**1**) de l'angle implicitement définit par le départ, les radius et l'arrivé.

```
<!-- some baking that has gone wrong :) -->
<path d="M400,180 L350,150 a100,60 0 1,0 100,-50 z"
      fill="green" stroke="blue" stroke-width="5" />
```

5. Le texte

- Le support texte en SVG 1.0 est assez médiocre
- En SVG 1.1 (Adobe Viewer 6a/Firefox 2.0?) et SVG 1.2 bon support

Eléments les plus importants pour SVG 1.0:

texte

- Chaque élément ’text’ entraîne le rendu d’une seule chaîne de texte.
- Un texte est un objet graphique comme un autre: On peut appliquer les fonctions de transformation de système de coordonnées, de peinture, de rognage et de masquage ainsi qu’un style CSS.
  `<text x="55" y="90" style="stroke:#000099;fill:#000099;font-size:18;">`
- Par défaut un text est affiché sur une ligne, mais on peut aussi le rendre au long du contour d’un élément ’path’.

tspan

- Pour afficher un texte sur plusieurs lignes, on doit pré-calculer les retours à la ligne et employer un seul élément ’text’ avec un ou plusieurs éléments enfants ’tspan’ et les valeurs adéquates pour les attributs x, y, dx et dy

```
 <text x="55" y="90"  style="stroke:#000099;fill:#000099;font-size:18;">
   Hello. Let’s show: <tspan x="55" dy="20"> a blue rectangle that pops up and goes
again</tspan> <tspan x="55" dy="40">and a yellow that slowly arrives and then stays!</
tspan> </text>
```

Exemple 5-1: text, tspan et textPath

url: **http://www.yoyodesign.org/doc/w3c/svg1/text.html (original)**

url: **http://tecfa.unige.ch/guides/svg/ex/svg-w3c/toap02.svg**

```
<defs>
  <path id="MonTrace" 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>
 <desc>Exemple toap02 - un 'tspan' dans un 'textPath'</desc>
 <use xlink:href="#MonTrace" fill="none" stroke="red"  />
 <text font-family="Verdana" font-size="42.5" fill="blue" >
  <textPath xlink:href="#MonTrace">
   En
   <tspan dy="-30" fill="red" > haut </tspan>
   <tspan dy="30"> , </tspan>
   puis en bas, et encore en haut
  </textPath>
 </text>

 <!-- Montre le contour du canvevas avec un élément 'rect' -->
 <rect x="1" y="1" width="998" height="298"
  fill="none" stroke="blue" stroke-width="2" />
```

- Rappel: en SVG 1.1 on peut faire beaucoup plus, par exemple remplir une zone définie par un contour graphique.

6. Les liens

SVG permet de faire des liens vers d'autres ressources (URLs)

- La balise <a > delimite la zone sensible
  - dans l'exemple suivant le rectangle vert et le texte
  - le lien est indiqué avec un attribut XLink (xlink:href="....")
- Attention: Il faut déclarer le name space pour XLink

Exemple 6-1: Liens vers d'autres ressources avec la balise a

url: **http://tecfa.unige.ch/guides/svg/ex/links/external-link.svg**

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg height="900" width="900"
  xmlns:xlink="http://www.w3.org/1999/xlink"
  xmlns="http://www.w3.org/2000/svg">

  <desc>Un lien se crée simplement avec la balise a.</desc>

  <a xlink:href="http://tecfa.unige.ch">
    <rect style="fill:#00FF00;stroke:#00FF00" width="300" height="40" ry="5" rx="5"
y="80" x="50"/>
    <text x="100" y="110" style="stroke:#000099;fill:#000099;fontsize:24;">TECFA POWER
1 click away</text>
  </a>
</svg>
```

7. Structuration: éléments de groupage et références

- Chaque langage informatique de haut niveau doit permettre de regrouper des objets dans des blocs, de les nommer et de les réutiliser. SVG possède plusieurs constructions intéressantes.
- Il est aussi intéressant de noter que les objets SVG (comme les objets HTML) héritent le style de leurs parents ! Autrement dit: les styles sont "cascading".

  Voici la liste des éléments les plus importants. Notez qu'il faut se référer à la spécification pour connaître tous les attributs de ces éléments. Ici, nous ne montrons qu'un petit extrait !

1. "Le fragment d'un document SVG: <svg>" [36]
2. "Groupage d'éléments avec <g>" [37]
3. "Objets abstraits (chablons) <symbol>" [38]
4. "Section de définition <def>" [38]
5. "Utilisation d'éléments <use>" [39]
6. "Titre <title> et description <desc>" [43]

7.1 Le fragment d'un document SVG: <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 des coordonnées
- Voir aussi: Utilisation d'un viewport et "Système de coordonnées, transformations et unités" [46]

Exemple 7-1: Hello SVG 2

url: **http://tecfa.unige.ch/guides/svg/ex/svg-intro/hello-svg2.svg**

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN"
          "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg>
 <rect x="50" y="50" rx="5" ry="5" width="200" height="100"
       style="fill:#CCCCFF;stroke:#000099"/>
 <text x="55" y="90" style="stroke:#000099;fill:#000099;font-size:14;">
      HELLO cher visiteur  </text>
 <svg with="200" height="200" x="200" y="100">
    <rect x="50" y="50" rx="5" ry="5" width="200" height="100"
        style="fill:#CCCCFF;stroke:#000099"/>
    <text x="55" y="90" style="stroke:#000099;fill:#000099;font-size:14;">
         HELLO cher visiteur </text>
 </svg> </svg>
```

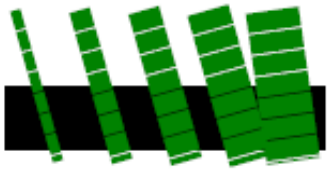
7.2 Groupage d'éléments avec <g>

- L'élément <g> sert à regrouper des éléments "qui vont ensemble":
  - Les enfants de <g> héritent les propriétés, le groupe est documenté avec <title> et <desc>

Exemple 7-2: Un simple groupe de bambous

url: **http://tecfa.unige.ch/guides/svg/ex/svg-intro/grouping/group1.svg**

```
<g stroke="green" stroke-dasharray="9 1" >
 <title content="structured text">Mon plus beau dessin </title>
 <line x1="5" y1="80" x2="155" y2="80" stroke-width="30"
       stroke="black" stroke-dasharray="none" />
 <line x1="10" y1="30" x2="30" y2="100" stroke-width="5" />
 <line x1="40" y1="30" x2="60" y2="100" stroke-width="10" />
 <line x1="70" y1="30" x2="90" y2="100" stroke-width="15" />
 <line x1="100" y1="30" x2="120" y2="100" stroke-width="20" />
 <line x1="130" y1="30" x2="140" y2="100" stroke-width="25" />
</g>
```



- Notez comme la couleur verte et le "dashing" **`stroke-dasharray="9 1`**" sont hérités.
- La ligne noire par contre utilise des "overrides", par exemple `stroke="`**`black`**"

7.3 Objets abstraits (chablons) <symbol>

- <symbol> permet de définir un objet graphique réutilisable avec <use>
- <symbol> ressemble à <g>, sauf que l'objet lui-même n'est pas dessiné
- <symbol> possède les attributs viewBox et preserveAspectRatio en plus

```
<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>
```

Voir "Utilisation d'éléments <use>" [39]

7.4 Section de définition <def>

- <defs> ressemble un peu à <symbol>, mais est plus simple
- Tout élément à l'intérieur de <defs> est défini mais pas dessiné.
- On peut utiliser arbitrairement chaque élément qui possède une identité

```
<defs>
 <rect id="redsquare" fill="red" width="10" height="10"/>
 <rect id="yellowsquare" fill="yellow" width="10" height="10"/>
</defs>
```

Voir "Utilisation d'éléments <use>" [39]

7.5 Utilisation d'éléments <use>

- <use> permet de réutiliser les objets suivants: <svg>, <symbol>, <g>, éléments graphics et <use>
- <use> se comporte légèrement différemment selon le type d'objet définit (voir la spécification !):
- Il s'agit donc d'un instrument de base pour éviter de répéter du code.

A. Objets réutilisables

- Les objets doivent avoir un identificateur XML

```
<rect id="redsquare" fill="red" width="10" height="10"/>
```

- xlink est le simple XLink standard, il faut donc ne oublier de définir son namespace (normalement vous le faites dans la racine)

```
<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">
```

B. Attributs importants:

- x, y, with, height permettent de repositionner et de redimensionner l'objet
- xlink:href permet de référencer et instantier l'objet (avec son attribut "id")

Exemple 7-3: Réutilisation d'un rectangle

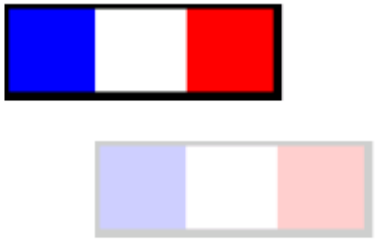
url: **http://tecfa.unige.ch/guides/svg/ex/svg-intro/grouping/use1.svg**

```
<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>
```

Exemple 7-4: Utilisation d'un objet <symbol>

url: **http://tecfa.unige.ch/guides/svg/ex/svg-intro/grouping/use2.svg**

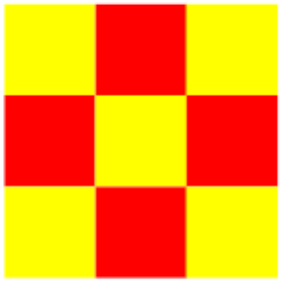
```
<symbol id="bleublancrouge">
 <rect x="0" fill="blue" width="10" height="10"/>
 <rect x="10" fill="white" width="10" height="10"/>
 <rect x="20" fill="red" width="10" height="10"/>
 <rect x="0" fill="none" width="30" height="10" stroke="black"/>
</symbol>

<use x="10" y="5" xlink:href="#bleublancrouge" />
<use x="20" y="20" xlink:href="#bleublancrouge" opacity="0.3" />
```



Exemple 7-5: Utilisation de 2 <defs> carrés

url: **http://tecfa.unige.ch/guides/svg/ex/svg-intro/grouping/use3.svg**

```
 <defs>
 <rect id="redsquare" fill="red" width="10" height="10"/>
 <rect id="yellowsquare" fill="yellow" width="10" height="10"/>
</defs>

<use x="20" y="0" xlink:href="#yellowsquare" />
<use x="30" y="0" xlink:href="#redsquare" />
<use x="40" y="0" xlink:href="#yellowsquare" />

 <use x="20" y="10" xlink:href="#redsquare" />
 <use x="30" y="10" xlink:href="#yellowsquare" />
 <use x="40" y="10" xlink:href="#redsquare" />

 <use x="20" y="20" xlink:href="#yellowsquare" />
 <use x="30" y="20" xlink:href="#redsquare" />
 <use x="40" y="20" xlink:href="#yellowsquare" />
```



Les couleurs de l’Espagne sont plus gaies que celle de la France :)

7.6 Titre <title> et description <desc>

<title> et <desc> permettent de documenter le code. Ces éléments ne sont pas affichés tel quels, par contre un client peut décider de les afficher comme "tooltips" par exemple

2 raisons pour bien documenter:

- Comprendre mieux le code !
- Aider l'utilisateur (à explorer ...)
- Aider les engins de recherche à indexer votre SVG (SVG c'est du texte !)

Eléments qui peuvent avoir <title> et <desc>

- Les conteneurs ('svg', 'g', 'defs' 'symbol', 'clipPath', 'mask', 'pattern', 'marker', 'a' et 'switch')
- les éléments graphiques ('path', 'text', 'rect', 'circle', 'ellipse', 'line', 'polyline', 'polygon', 'image' et 'use')

7.7 Images <image>

- Formats bitmap supportés: png et jpeg
- <image> permet également d'insérer un fichier svg (avec un nouveau viewport)

Attributs importants:

x = "<coordinate>" et y ="<coordinate>"

définit l'emplacement (comme pour <rect>, <svg>, <g>, etc.)

width = "<longeur>" et height = "<longeur>"

définit hauteur et largeur de l'image (comme pour <rect>, <svg>, <g>, etc.)
Des valeurs positives indiquent la taille à afficher
(Note: une valeur de 0 empêche l'affichage, les valeurs négatifs sont interdits)

xlink:href = "<uri>"

définit l'URI où se trouve l'image

Adaptation de la taille de l'image

- Voir preserveAspectRatio et viewBox dans "Système de coordonnées, transformations et unités" [46]
- Une image est affiché par défaut selon les dimensions que vous donnez !

Exemple 7-6: Inclusion d'une image à plusieurs sauces

url: **http://tecfa.unige.ch/guides/svg/ex/svg-intro/images/images1.svg**

```
<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>
```

8. Système de coordonnées, transformations et unités

Au menu:

8.1 Le canevas, les viewports et les unités

A. Le canevas SVG

- Le canevas SVG est un espace infini ou s'affiche le contenu SVG

B. Le viewport SVG

- Le "viewport" SVG est le rectangle visible pour l'utilisateur
- Le viewport possède un système de coordonnés qui commence en haut à gauche du rectangle (***viewport coordinate system***, aussi appelé ***viewport space***)
- On peut définir un viewport dans un viewport (par exemple avec l'élément <svg>)
- Le dessins se reférent par rapport à un système de coordonnées d'utilisateur (***user coordinate system*** ou ***user space***) qui au départ est identique au viewpoint coordinate system
- Certains clients permettent à l'utilisateur de bouger et "zoom" le "user space" (Alt-drag pour bouger dans le plugin Adobe)
- Tout dessin qui dépasse est tronqué (clipped)

C. Longeurs

- Les longeurs sont indiqués soit par un nombre, soit par les unités absolues ou relatives habituelles:
  - em, ex (largeur d'un "m" et hauteur d'un "x" de la fonte courante)
  - px (pixels, unités définies par le device)
  - pt, pc (points, et ??). Normalement le client indique à combien de pixels correspond pt ou pc, pareils pour les cm, mm et in.
  - cm, mm, in
  - pourcentages (par rapport au viewport)
- La signifcation de nombres sans unités s'établit par rapport au unités utilisés pour définir le viewport (pixels par défaut)
- Note: On le choix d'ignorer les subtilités du système des longeurs, mais suivant la tâche il faut les maîtriser et relire la spécification.

8.2 Création de viewports

A. Eléments qui créent un nouveau viewport

- <svg>, <symbol> (instantié par <use>), <image>
- et <foreignObject> (par ex une image X3D dans l'avenir)

B. L'attribut viewBox

(explications à revoir un jour, c'est pas clair du tout .... désolé)

- Tous les éléments qui créent un nouveau viewport + <marker>, <pattern> et <view> permettent d'adapter les dimensions d'un graphisme à celles d'un conteneur

  `Syntaxe: viewBox = "<min-x> <min-y> <width> <height>"`

  Attention: Ne pas utiliser autre chose que des simples nombres ! donc 1500 et pas 15mm ou 150px !!

  Exemples:

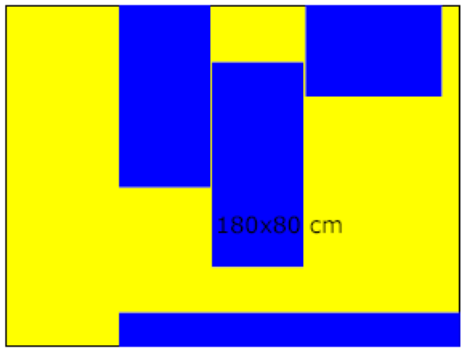
```
<svg width="300px" height="200px" viewBox="0 0 1500 1000">
<svg width="300px" height="200px" viewBox="0,0,1500,1000">
```

- viewBox permet de greffer un système de coordonnés sur les dimensions réelles d'un viewport.
- C'est très utile lorsqu'on a par exemple des dessins en mètres et qui doivent quand-même s'afficher à l'écran.
- On peut aussi créer des distorsions (lorsque largeur et hauteur du viewBox n'ont pas les mêmes proportions que ceux du viewPort).

Exemple 8-1: Redimensions d'un dessin de bureau avec ViewPort

url: **http://tecfa.unige.ch/guides/svg/ex/svg-intro/viewports/viewports1.svg**

```
<svg width="6cm" height="4.5cm" viewBox="0 0 400 300" >
 <rect x="0" y="0" width="400" height="300"
       fill="yellow" stroke="blue" stroke-width="4"  />
....
 <g>
  <title>le gros bureau qui pose problème</title>
  <rect x="182" y="50" width="80" height="180" fill="blue" />
  <text x="185" y="200" font-size="20">180x80 cm </text>
 </g>
 <rect x="264" y="0" width="120" height="80"
       fill="blue" />
</svg>
```



C. L’attribut preserveAspectRatio

- Cet attibut est disponible lorsque lorsqu’un noveau viewPort est crée et permet d’indiquer comment il faut préserver les ratios (x,y) dans le dessin.

  `Syntaxe: preserveAspectRatio="<align> [<meetOrSlice>]"`

Le paramètre "align"

Il existe pleins de différentes sortes de align. Ce paramètre indique ce qui doit se passer lorsque les rapports entre longeur et hauteur du viewBox ne correspondent pas au viewBox, autrement dit: comme "tirer" le graphisme.

- none - Tire le graphisme aux 2 bords
- xMinYMin - alignement sur x-min et y-min (coin du haut à gauche)
- xMinYMid - alignement sur x-min et centrage-y
- xMinYMax -alignement sur x-min et y-max (coin en bas à gauche)
- ... ainsi que les autres 6 combinaisons entre x* et Y*.

Le paramètre "meetOrSlice"

- meet (défaut) - garder la "aspect ratio", toute la viewBox est visible et elle est adaptée au dessin, le graphise sera toujours visible mais n’utilise peut-être pas tout le viewPort.
- slice - garder la "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.

Attention: respectez minuscules ou majuscules !!

Exemple 8-2: Adaptations d'un bureau à des ViewPort

url: **http://tecfa.unige.ch/guides/svg/ex/svg-intro/viewports/viewports2.svg**

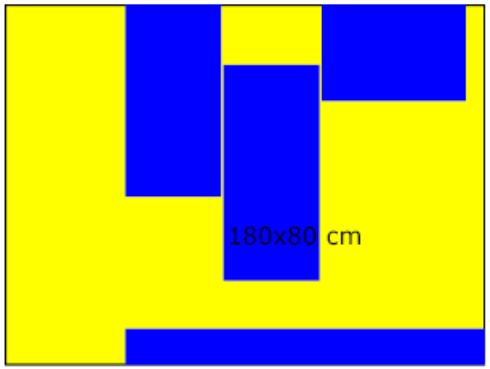
```
....
<symbol id="bureau">
.....
</symbol>
<svg x="0.5cm" y="0.5cm" width="4cm" height="3cm"
     viewBox="0 0 400 300">
  <use xlink:href="#bureau" />
 </svg>
 <svg x="6cm" y="0.5cm" width="3cm" height="3cm" viewBox="0 0 400 300"
     preserveAspectRatio="none"  >
  <use xlink:href="#bureau" />
</svg>
 <svg x="0.5cm" y="4cm" width="3cm" height="3cm" viewBox="0 0 400 300"
     preserveAspectRatio="xMinYMin meet"  >
.... </svg>
 <svg x="6cm" y="4cm" width="3cm" height="3cm" viewBox="0 0 400 300"
     preserveAspectRatio="xMinYMax slice" >
.... </svg>
```

(voir slide suivant pour un image)

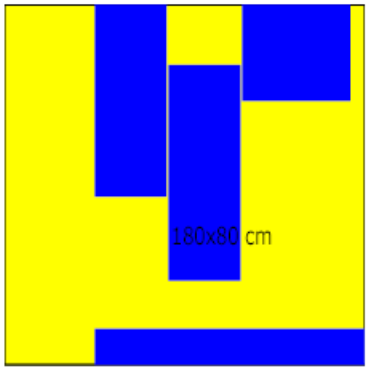
- Cet exemple montre quelques variations de "preserveAspectRatio par rapport à des viewBox qui ne possèdent pas les mêmes ratios que les viewPorts
- Dans la spécification on trouve plus de détails et d'exemples !

- Faites attention à correctement écrire les paramètres, votre client risque de ne pas se plaindre pour ce genre d'erreurs !

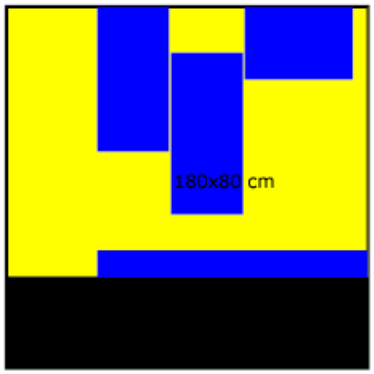
Viewport analog à viewBox



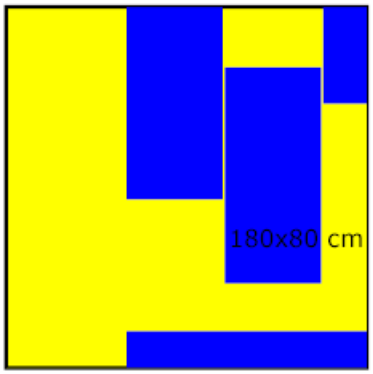
Viewport carré, none



Viewport carré, xMinYMin meet



Viewport carré, xMinYMax slice



8.3 Transformations avec l'attribut "transform"

- transform permet de définir des translations, scalings, rotations, transformations selon une matrice, skewX et skewY

A. Translations avec le paramètre "translate"

- On a déjà vu qu'il est possible de définir un nouveau système de coordonnées avec des nouveaux viewPorts et viewBox
- L'attribut "transform" disponible pour tous les éléments conteneurs ou graphiques (voir "Eléments graphiques de base" [16] et "Structuration: éléments de groupage et références" [35]) le permet aussi.

  `Syntaxe: translate(<tx> [<ty>])`
  Exemple:
  ```
  <g transform="translate(50,50)">
  ```

B. Redimensionnement (scaling) avec le paramètre "scale"

`Syntaxe: scale (<sx> [<sy>])`
lorsque sy n'est pas indiqué on assume que sy=sx
Exemples:
```
<g transform="scale(2)"> <rect .... /> <g>
<g transform="scale(2,3)"> <rect .... /> <g>
```

C. Rotations avec le paramètre "rotate"

`Syntaxe: rotate(<rotate-angle> [<cx> <cy>])`

- L'angle de rotation en degrées
- cx et cy définissent le point de rotation (par défaut il s'agit de l'orgine du système de coordonnées locale, PAS le centre de l'objet !)

D. Ordre des opérations

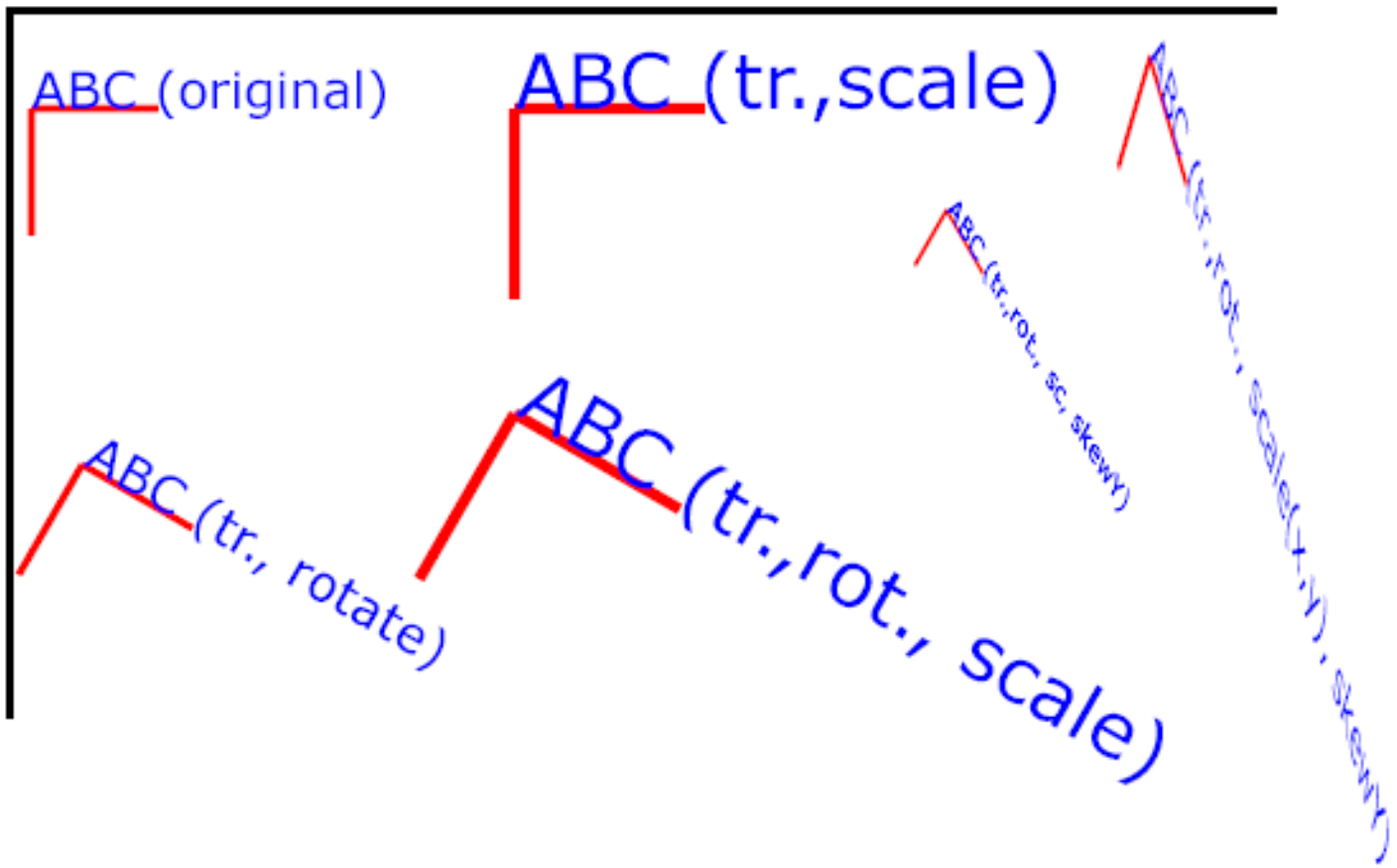
- l'ordre des opérations est séquentielle ! A chaque transformation on obtient un nouveau système de coordonnées !
- Les 2 fragments suivants font la même chose:

```
<g transform="translate(-10,-20) scale(2) rotate(45
   translate(5,10)">
  <!-- graphics elements go here -->
</g>

<g transform="translate(-10,-20)">
  <g transform="scale(2)">
    <g transform="rotate(45)">
      <g transform="translate(5,10)">
        <!-- graphics elements go here -->
      </g> </g>   </g>  </g>
```

Exemple 8-3: Simples transformations

url: **http://tecfa.unige.ch/guides/svg/ex/svg-intro/transforms/transforms1.svg**

```
<g transform="translate(10,40)">
 <g id="dessin" 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" font-size="20" font-family="Verdana" fill="blue" >
      ABC (original) </text>
</g>
<g transform="translate(200,40)">
 <g transform="scale(1.5)">
 <use xlink:href="#dessin" />
 <text x="0" y="0" font-size="20" font-family="Verdana" fill="blue">
    ABC (tr.,scale)</text>
</g> </g>
....
<g transform="rotate(30)"> ... </g>
...
<g transform="translate(200,160),scale(1.5),rotate(30)"> ...</g>
...
<g transform="translate(370,80),scale(0.5),rotate(30),skewY(30)">
....
<g transform="translate(450,20),scale(0.5,1),rotate(30),skewY(30)">
```



- Il n'est pas toujours facile de s'imager ce que donne une série de transformations (remember VRML ?)
- Les matrix transform permettent des opérations encore plus générales (et plus difficiles à comprendre, voir la spéc.).

9. Suite (rappel)

Module suivant: SVG dynamique

Il existe deux types de SVG dynamique

- Animation avec des balises "SMIL"
  - Marche uniquement avec Opera 9x et des plugins
- Animation avec DOM et JavaScript
  - plus puissant (on peut tout faire), mais plus difficile
  - marche avec les navigateurs actuels (sauf IE)

url: **http://tecfa.unige.ch/guides/tie/html/svg-dyn/svg-dyn.html**

Module suivant: SVG avec XSLT

Module technique suivant: **jhttp://tecfa.unige.ch/guides/tie/html/svg-xslt/svg-xslt**

Voir aussi: Visualisation avec PHP, XSLT et SVG

url: **http://tecfa.unige.ch/guides/tie/html/visu-gen/visu-gen.html**