Introduction à SVG

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: 0.6 (modifié le 2/12/02)

Prérequis

Module technique précédent: xml-dom
Module technique précédent: xml-tech

Autres modules

Module technique suppl.: xml-xslt

Abstract

Introduction à SVG - lacunaire pour le moment

Objectifs

• Philosophie de SVG
• SVG graphiques de base
• Groupage
• Transformations
• SVG animé de base
• SVG interactif de base

1. Table des matières détaillée

2. Scalable Vector Graphics - un nouveau paradigme

2.1 Origine et but

- SVG est un langage XML et un standard W3C
- SVG a été fait pour s'insérer parfaitement au reste des langages du W3C

Utilisation

SVG est une concurrence presque directe à Flash, toutefois comme Flash a déjà sa "niche" (publicité et sites "branchés), on peut miser dans l'immédiat sur:

- visualisation de contenus (économiques, processus, cartes, etc.), implique du serveur-side scripting, du DOM scripting avec JS, des transformations avec XSLT, etc.
- interface utilisateurs pour certains types d'applications Internet
- dessin statiques, animés ou même interactifs dans le monde de l'éducation

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 particulières de SVG

- insertion dans le monde XML/XHTML
  - génération de SVG avec XSLT à partir de données XML
  - future intégration totale 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:

- les viewers se développent (sans être parfaits). On conseille celui de Adobe
  url: http://www.adobe.com/svg/

Editeur SVG

- WebDraw de Jasc qui est de bonne qualité
  url: http://www.adobe.com/svg/

SVG avec un éditeur graphique

- Illustrator permet de sauver du SVG (attention à la configuration)

SVG avec un éditeur XML

- Le DTD est à la fin de la spécification officielle
  url: http://tecfa.unige.ch/lib/xml/dtd/svg10.dtd **(copie locale)**
  url: http://www.w3.org/TR/2001/PR-SVG-20010719/DTD/svg10.dtd
- On conseille d'installer le DTD dans le système (voir Xemacs ci-dessous)

Installation d'un DTD dans Xemacs

- Il faut copier le fichier avec le DTD qq part sur votre système
- Ensuite éditer le fichier ..../`xemacs-packages/etc/psgml/CATALOG` qui se trouve dans l'installation Xemacs.
- Ajouter une ligne qui associe un identificateur public avec un fichier

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

- N'oublier pas de dire à Emacs que *.svg est associé à XML, éditez le fichier .emacs et ajoutez:

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

2.4 Ressources

Spécification (SVG 1.0 Specification, W3C recommendation 04 sept. 2001)

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

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 ces exemples il vous faut un viewer SVG.
- On conseille d'utiliser le plug-in de Adobe
  - marche avec IE 5/6, NS 4.x, Moz < 0.9.0.
  - Il existe une incompatibilité pour Moz 1.0RC1 qui empêche les "embedded" SVG de fonctionner et qui fait crasher Moz :(
  - Pour voir le source d'un SVG: clique-droit avec la souris et choisir

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/PR-SVG-20010719/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 pour fichiers SVG stand-alone à copier:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN"
          "http://www.w3.org/TR/2001/PR-SVG-20010719/DTD/svg10.dtd">
<svg width="400" height="250"
     xmlns="http://www.w3.org/2000/svg">

</svg>
```


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

url: http://tecfa.unige.ch/guides/tie/code/svg-intro/hello-svg.svg

```
Ignorez les détails (réctangle + 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/PR-SVG-20010719/DTD/svg10.dtd">
<svg>

 <!-- un petit rectangle avec des coins arrondis  -->
 <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;fontsize:24;">
  HELLO cher visiteur
 </text>

</svg>
```

- A faire: imbrication dans HTML et XHTML

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>" [16]
2. "Le cercle <circle> et l'ellipse <ellipse>" [18]
3. "Lignes <line> et poli-lignes <polyline>" [20]
4. "Polygones" [22]
5. "Formes arbitraires avec <path>" [23]

- 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>" [23] 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" [39].

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
- 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

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>"

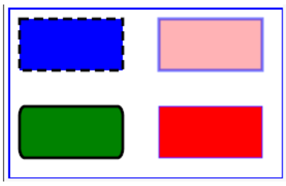
Axe x et y de l'ellipse utilisée pour arrondir, pas de nombre négatif,
ne doit pas dépasser la moité des longeurs respectifs

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

Exemple 4-1: Rectangles avec style:

url: http://tecfa.unige.ch/guides/tie/code/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>"

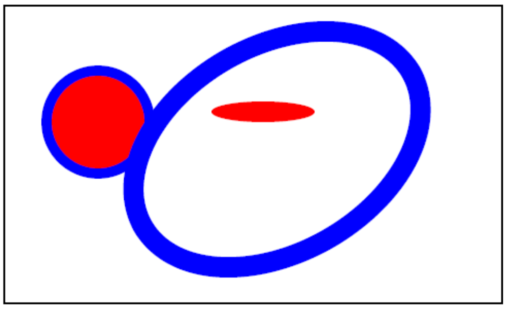
`rx="10" ry="20"`

définit les radius des axes x et y de l'ellipse ("r"=radius)

Exemple 4-2: Ronds figés

http://tecfa.unige.ch/guides/tie/code/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 6. "Système de coordonnées, transformations et unités" [39] 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 4-3: Lignes

http://tecfa.unige.ch/guides/tie/code/svg-intro/shapes/ronds.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>"

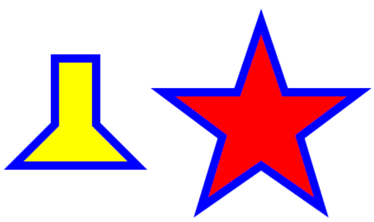
`points = "10,100,10,120,20,20,........"`

Séries de points x,y qui seront reliés (également le dernier au premier point)

Exemple 4-4: 2 Polygones

http://tecfa.unige.ch/guides/tie/code/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
```

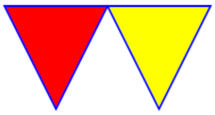
Exemple 4-5: 2 Simple path: un triangle

http://tecfa.unige.ch/guides/tie/code/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

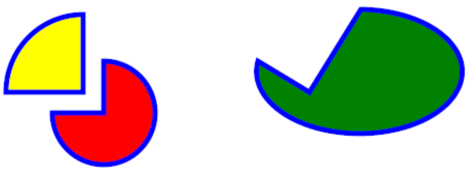
Courbes Bézier quadratiques

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 4-6: 2 Simple gateau avec <arc>

http://tecfa.unige.ch/guides/tie/code/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. 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 !

5.1 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" [39]

Exemple 5-1: Hello SVG 2

url: http://tecfa.unige.ch/guides/tie/code/svg-intro/hello-svg2.svg

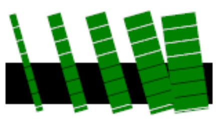
```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN"
          "http://www.w3.org/TR/2001/PR-SVG-20010719/DTD/svg10.dtd">
<svg>
 <rect x="50" y="50" rx="5" ry="5" width="200" height="100"
       style="fill:#CCCCFF;stroke:#000099"/>
 <text x="55" y="90" style="stroke:#000099;fill:#000099;fontsize:24;">
      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;fontsize:24;">
         HELLO cher visiteur </text>
 </svg> </svg>
```


5.2 Groupage d'éléments avec <g>

- L'élément <g> sert à regrouper des éléments "qui vont ensemble":
  - Les enfants de <g> héritent les propriétés
  - On peut documenter le group avec <title> et <desc>

Exemple 5-2: Un simple groupe de bambous

```
<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" est hérité. La ligne noir par contre utilise des "overrides".

5.3 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>" [32]

5.4 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>" [32]

5.5 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 5-3: Réutilisation d’un rectangle

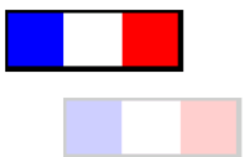
url: http://tecfa.unige.ch/guides/tie/code/svg-intro/grouping/use1.svg

```
<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 5-4: Utilisation d’un objet <symbol>

url: http://tecfa.unige.ch/guides/tie/code/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 5-5: Utilisation de 2 <defs> carrés

url: http://tecfa.unige.ch/guides/tie/code/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 :)
```


5.6 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')

5.7 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" [39]
- Une image est affiché par défaut selon les dimensions que vous donnez !

Exemple 5-6: Inclusion d'une image à plusieurs sauces

url: http://tecfa.unige.ch/guides/tie/code/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>
```

6. Système de coordonnées, transformations et unités

Au menu:

6.1 Le canevas, les viewports et les unités

A. Le canveas 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.

6.2 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

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

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

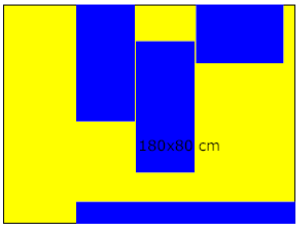
  Exemples:

  ```
  <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 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 6-1: Redimensions d'un bureau avec ViewPort

url: http://tecfa.unige.ch/guides/tie/code/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 !!

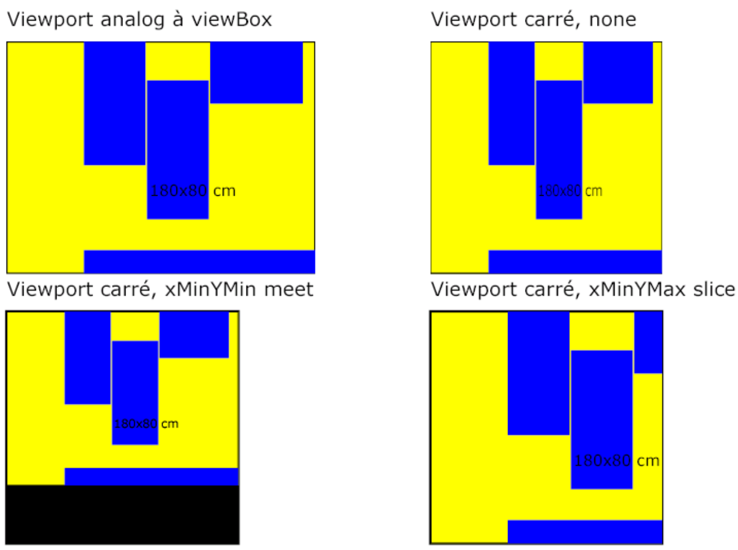
Exemple 6-2: Adaptations d'un bureau à des ViewPort

url: http://tecfa.unige.ch/guides/tie/code/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>
```

- 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 !



6.3 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" [13] et "Structuration: éléments de groupage et références" [28]) 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

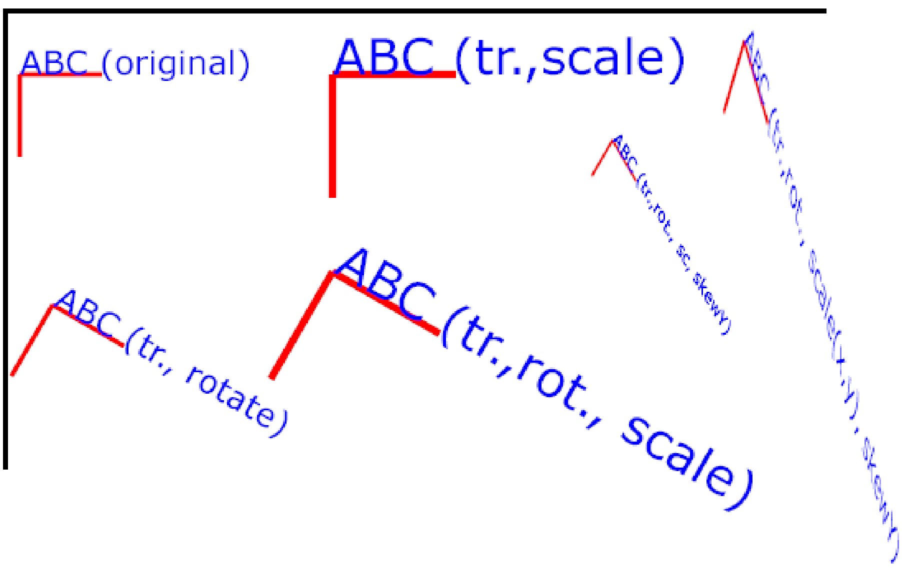
- l'ordre des opérations est séquentielle ! A chaque transformation on obtient un nouveau système de coordonnées !
- Les 2 fragments suivants font la même chose:

```
<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 6-3: Simples transformations

url: http://tecfa.unige.ch/guides/tie/code/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.).

7. Animation SVG

Les possibilités d'animation sont fortes:

- on peut animer pratiquement chaque attribut avec les éléments d'animation SVG (un peu comme en VRML)
- on peut utiliser le SVG DOM, chaque attribut et " style sheet setting" est accessible selon DOM1 & 2. En plus, il existe un jeu d'interfaces DOM additionnelles.
- L'animation SVG étend celle de SMIL avec quelques extensions.
- L'animation est "time-based" (vs. "frame-based") ce qui donne beaucoup de flexibilité, on peut animer des attributs de façon indépendant.

Attributs communs:

xlink:href

- SVG utilise xlink pour designer le "animation target" et il faut donc indiquer son namespace qq part.

attributeName

indique le nom de l'attribut qu'il faut animer

attributType

`Syntaxe: attributeType = "CSS | XML | auto"`
indique le type d'attribut animé (pour des raisons d'efficacité ?)

7.1 animate et set

- [à faire]

7.2 AnimateMotion

- [à faire]

7.3 AnimateColor

url: http://tecfa.unige.ch/guides/tie/code/svg-intro/anim-colors/anim-color1.svg

```
<svg height="900" width="900">
<rect style="fill:#000000;" height="900" width="900" y="0" x="0">
 <animate fill="freeze" dur="5s" begin="0.1s"
  to="#FFFFFF" from="#000000" calMode="linear" attributeName="fill"/>
</rect>
<rect style="fill:#CC9933;stroke:#CC9933" width="20" height="500"
      ry="5" rx="5" y="100" x="400">
<animate dur="5s" begin="1s" to="#000000"
         from="#CC9933" calMode="linear" attributeName="fill"/>
<animate dur="5s" begin="6s" from="#000000"
         to="#CC9933" calMode="linear" attributeName="fill"/>
</rect>
```

7.4 AnimateTransform

Exemple 7-1: Simples transformations

url: http://tecfa.unige.ch/guides/tie/code/svg-intro/anim-trans/rotation.svg

```
<title>Simple rotation example</title>
<desc> Rotation with a grouping node </desc>
<g>
  <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;fontsize:24;">
   Hello. Let's rotate</text>
  <animateTransform attributeName="transform" type="rotate"
            values="0 150 100; 360 150 100"
            begin="0s" dur="5s" />
</g>
```


8. Interactivité

- ressemble au principe des GUI modernes (par exemple JavaScript)

  url: http://tecfa.unige.ch/guides/tie/code/svg-intro/mouse-over/mouse-over1.svg

```
<!-- LIEN HTTP -->
<a xlink:href="http://tecfa.unige.ch">
   <rect style="fill:#00FF00;stroke:#00FF00" width="200" height="26" ry="5"
rx="5" y="100" x="100"/>
</a>
<!-- Mouse Over -->
<g transform="translate(300,200)">
  <g style="display:none">
  <animate fill="freeze" dur="0.1s" begin="bouton.mouseover" to="block"
from="none" attributeName="display"/>
  <animate fill="freeze" dur="0.1s" begin="bouton.mouseout" to="none"
from="block" attributeName="display"/>
  <circle style="fill:yellow;" r="80" cy="25" cx="300"/>
  <text style="font-size:12;" y="25" x="240">COUCOU</text>
  </g>
<ellipse stroke-width="2" stroke="black" fill="yellow" ry="12" rx="5"
id="bouton">
<animateColor fill="freeze" dur="0.1s" to="blue" from="yellow"
attributeName="fill" begin="mouseover"/>
<animateColor fill="freeze" dur="0.1s" to="yellow" from="blue"
attributeName="fill" begin="mouseout"/>
</ellipse>
</g>
```

9. Scripting

- [à faire]