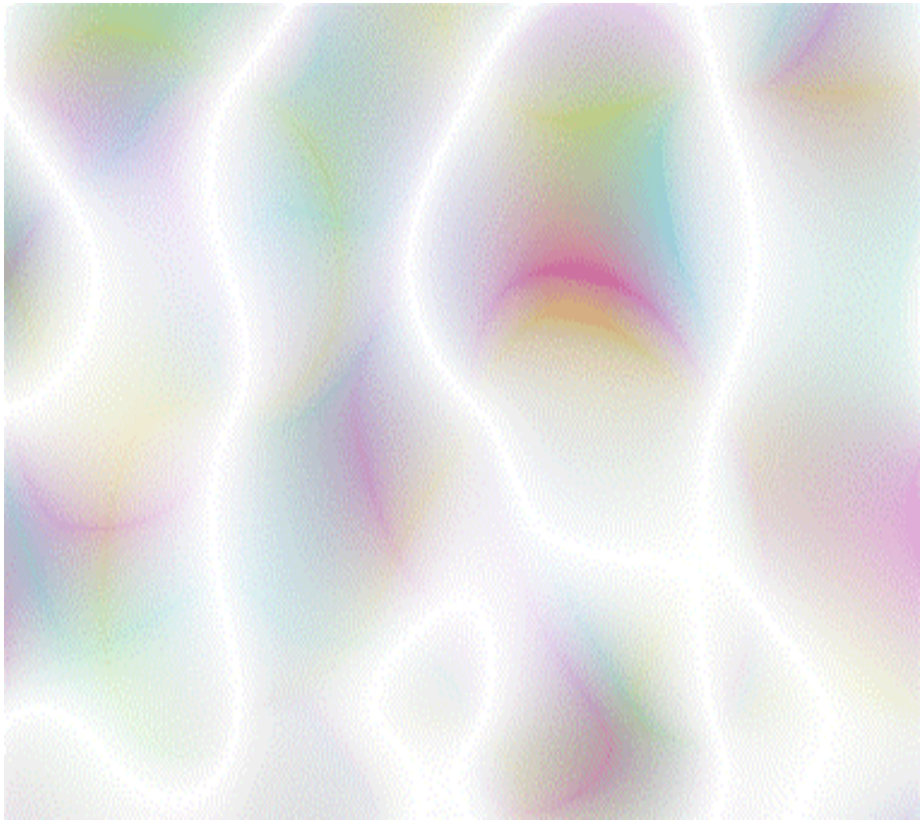
Scalable Vector Graphics



Outils, ressources, logiques d’utilisation, syntaxe et génération (php, xml)

Sommaire

- Objectifs
- Intro
- Outils et ressources
- Template SVG
- Logique SVG
- Syntaxe de base
- Animations
- Générer du SVG avec SVG et XSLT

Objectifs

- Vous montrer comment SVG marche
- Présentation grossière de la syntaxe
- « Défricher le terrain »
- Vous montrer ou chercher de quoi vous débrouillez

Introduction

- La nécessité d'avoir des graphiques vectoriels et retaillables a conduit le W3 Consortium a l'élaboration de la spécification SVG.
- SVG est un langage de description de graphiques 2D en XML.
- 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(2)

- Scalable = zoomable et permet un grand nombre d'utilisateurs et un grand nombre d'utilisations (par ré-utilisation de fragments)
- XML : possibilité d'utiliser tous les outils XML : parser, outil de transformation, base de données.
- La conformité aux espaces de nommage permet 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, ...

Outils SVG

- On peut valider le code on-line : http://validator.w3.org
- Il commence à y avoir des viewers. Le plus simple est le plug-in Adobe : http://www.adobe.com/svg/ (marche avec IE 5/6, NS 4.x et Mozilla < 0.9.0). Click droit pour voir source (menu)
- Il existe des éditeurs, le plus courant étant Webdraw (de Jasc) : http://www.jasc.com ou XMLwriter : http://www.xmlwriter.com ou Mayura Draw : http://www.mayura.com

Ressources

- Spécifications : http://www.w3.org/TR/SVG/
- Tutorials :
  http://www.adobe.com/svg/tutorial/intro.html
  http://www.xml.com/pub/a/2001/03/21/svg.html
  http://www.euroclid.fr/Cours_SVG/plan.htm
  (français)
- SVG reference with examples :
  http://www.zvon.org/xxl/svgReference/Output/index.html

Template SVG de base (1)

```
<?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>
```

Template SVG de base (2)

<?xml version="1.0" standalone="no"?> :déclaration XML standard

<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN" "http://www.w3.org/TR/2001/PR-SVG-20010719/DTD/svg10.dtd"> : indication du DTD pour un document non standalone

<svg width="400" height="250" xmlns="http://www.w3.org/2000/svg"> : déclaration du namespace et des dimensions du document SVG. On est ici à la racine du contenu SVG.

</svg>

Exemple simple

```
<?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>
<text x="80" y="90" style="stroke:#000099;fill:#000099;fontsize:30;">
    Hello world !</text>
</svg>
```

Exemple 1

- ***X et y : position***
- ***stroke* , définit comment le bord d'un objet est paint**
- ***fill* , définit comment le contenu d'un objet est paint**

Styles

On peut appliquer deux syntaxes différentes de mise en forme :

- soit par CSS2 (Cascading Style Sheet Level 2) : l'attribut *style* reprend la syntaxe et les styles de CSS2
- soit par XSL (XML Style Language)
- soit par l'attribut *style* commun à beaucoup de balises SVG
- Il est recommandé de ne pas mélanger les styles CSS et XSL dans le même document SVG

Mise en forme différente

- Exemple d’attributs de présentation SVG, les deux ont le même effet :
- SVG <rect x="200" y="100" width="60" height="30" *fill="red" stroke="blue" stroke-width="3" />*
- CSS2 **<**rect x="200" y="200" width="60" height="30" *style="fill:red;stroke:blue;stroke-width:3" />*

Types des données de base

- Angle : un entier suivi de *deg* (degrés), *grad* (grades), *rad* (radians)
- Couleur : spécifié dans le modéle sRGB ; spécification d'une couleur RGB comme en HTML : *#rrggbb* ou un mot-clé dans la liste : *aqua, black, fuchsia, gray, green,l ime, maroon, navy, olive, purple, red, silver, teal, white, yellow*
- Entier : entre - 2147483648 et 2147483647
- Réel : soit en notation décimale soit en notation scientifique (x.yyye(ou E)±nn) entre - 3.4e+38 et 3.4e+38
- Longueur : nombre suivi d'un identifiant CSS (*mm,cm,m*)
- Temps : nombre suivi de *ms* ou de *s.*
- URI : Uniform Ressource Identifier

Logique SVG : structure

- Un document SVG se compose d'un ou plusieurs fragments délimités par la balise <svg>
- Il peut y avoir plusieurs structures <svg> emboîtées dans le même document ou dans des documents composites résultants de plusieurs espaces de noms
- La balise <svg> définit l'espace utilisateur
- Attributs de la balise <svg>
  - x = "x0" ; position en x du coin supérieur gauche (pour les structures internes)
  - y = "y0" ; position en y du coin supérieur gauche (pour les structures internes)
  - width = "w0" ; largeur en pixels de l'espace (pour les structures externes)
  - height = "h0" ; hauteur en pixels de l'espace (pour les structures externes)

Logique SVG :mécanismes

- **Positionnement :** les objets SVG se positionnent dans un système de coordonnées qui commence en haut et à gauche. Il est possible de travailler avec des coordonnées locales
- **Attributs :** 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 intuitives (proche de CSS).
- **Transformations :** les objets peut être translaté, orienté et changé de taille. Il hérite des transformations de l'objet parent,

Formes de base SVG

- On va pas tout vous montrer :)
- Eléments de base : rectangle, cercel, lignes/poly-lignes, polygones et formes arbitraires
- Les formes prédéfinies (chemins particuliers) permettent de s'affranchir de la description complète d'un chemin (path)

Rectangles

<rect x="coord" y="coord" width="longueur" height="longueur"
rx="longueur" ry="longueur" />

- x : coordonnée x du coté du rectangle de plus bas x
- y : coordonnée y du coté du rectangle de plus bas y
- Par défaut x et y = 0
- width : largeur du rectangle
- height : hauteur du rectangle
- rx : pour des rectangles à coins arrondis
  rayon en x de l'ellipse assurant le raccord
- rx : pour des rectangles à coins arrondis
  rayon en y de l'ellipse assurant le raccord

Cercles et ellipses

- <circle cx="coord" cy="coord" r="longueur" />
  - cx : coordonnée x du centre du cercle
    0 par défaut
  - cy : coordonnée y du centre du cercle
    0 par défaut
  - r : rayon du cercle
- ellipse cx="coord" cy="coord" rx="longueur" ry="longueur" />
  - cx : coordonnée x du centre de l'ellipse
    0 par défaut
  - cy : coordonnée y du centre de l'elipse
    0 par défaut
  - rx : rayon de l'ellipse suivant l'axe des x
  - ry : rayon de l'ellipse suivant l'axe des y

Les lignes

<line x1="coord" x2="coord" y1="coord" y2="coord" />

- x1 : coordonnée x du point de départ
- y1 : coordonnée y du point de départ
- x2 : coordonnée x du point d'arrivée
- y2 : coordonnée y du point d'arrivée

Polyligne

- Une polyligne est un ensemble de lignes droites connectées entre elles
- Une polyligne définit une forme ouverte
- <polyline points="*liste de points*" />
- *liste de points* est une liste de coordonnées x , y séparées par des virgules
- Ces coordonnées sont exprimées dans l'espace utilisateur
- Exemple de polyligne

Polygones

- Un polygone est un ensemble de lignes droites connectées entre elles définissant une **forme fermée**
- <polygon points="liste de points" />
- *liste de points* est une liste de coordonnées x , y séparées par des virgules
- Ces coordonnées sont exprimées dans l'espace utilisateur
- Exemple de polygone

Formes arbitraires

- **L'élément <path> permet de définir des formes arbitraires pouvant avoir un contour (stroke).**
- **Exemple simple :**

  <path d="M 50 50 L 100 150 150 50 z" fill="red" stroke="blue" stroke-width="2" />
- Voir les spécifications pour plus de précisions
- En gros, il y a des commandes comme M (moveto) qui permettent de « déplacer » le crayon qui dessine ou Z qui peuvent fermer les chemins.

Texte (1)

- Le texte suit les recommandations générales des caractères XML (attention au codage des accents !)
- Il n'effectue ni retour à la ligne ni césure automatique
- La balise <text> est traitée comme un objet graphique. En tant que telle, elle subit l'influence :
  - des changements de coordonnées
  - du mode de rendu
  - du clipping

Texte (2) : syntaxe

- <text x="coord" y="coord">blabli</text>
- x représente l'abscisse de départ du texte
  - s'il n'est pas suivi d'unité, la valeur est calculée dans l'espace utilisateur
  - s'il est suivi d'une unité CSS ou de %, la valeur est calculée par rapport au point de vue
- y représente l'ordonnée de départ du texte
  - si elle n'est pas suivie d'unité, la valeur est calculée dans l'espace utilisateur
  - si elle est suivie d'une unité CSS ou de %, la valeur est calculée par rapport au point de vue

Texte (3) : éléments d'ajustement et attributs

- A l'intérieur d'un élément <text>, on peut ajuster la position du texte, la valeur du
texte ou la police du texte grace à l'élément <tspan>
- <tspan x="coord+" y="coord+" dx="coord+" dy="coord+" rotate="auto|nombre" />
- Par l'intermédiaire d'attributs de type CSS, on peut spécifier :
- la mise en page du texte : largeur des caractères, taille des caractères
- le sens d'écriture : de gauche à droite, de haut en bas, ...
- l'alignement du texte
- l'espacement, le retour à la ligne
- la décoration ...

Le texte (4) : lien avec les chemins

- On peut imposer au texte de *suivre* un chemin prédéfini par la balise <path>
- <textPath starOffset="longueur|pourcentage" xlink:href="uri" />
- Exemple
- starOffset est le décalage par rapport au début du texte
  - une longueur représente une distance le long du chemin mesurée selon la métrique de l'espace utilisateur
  - un pourcentage représente un pourcentage par rapport au chemin entier selon la métrique de l'espace utilisateur

Le rendu

- Les éléments <path>,<text> et les formes de base peuvent être remplis et coloriés (c'est à dire peints sur les bords). On parle donc de *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

Spécifications pour le rendu

- Deux propriétés *fill* et *stroke* se partagent les attributs suivants :
  - couleur
  - uri de la couleur ou du gradient
- Propriétés de *fill* (remplissage)
  - opacité
- Propriétés de *stroke* (dessin)
  - épaisseur
  - jonction de lignes
  - arrondi des angles
  - pointillés
- Exemple
- A noter que l'on peut aussi remplir avec des gradients et des motifs (cf. spécifications)

Insertion d’une image

- Formats bitmap supportés: png et jpeg
- <image> permet également d'insérer un fichier svg
- Attributs utilisés:
  - x et y : coordonnées
  - width = "largeur" et height = "hauteur"
  - xlink:href = "uri" définit l'URI où se trouve l'image
- Possibilité de définir des masques et des chemins de découpe (*clipping path*)
- Exemple :

```
<image x="10" y="50" width="200" height="100"
    xlink:href="image.jpg">
```

Petit exercice

Afficher une image et en dessous une légende dans un rectangle bleu avec un contour noir

Liens

- On distingue les liens extra et intra document SVG
- Liens extra document : ils sont pris en charge par l'élément <a> analogue à l'élément correspondant de HTML ou SMIL
- Utilise la syntaxe de XLink (en cours de spécification)
- Exemple :

```
<a xlink:href="http://www.w3.org">
  <path d="M 0 0 L 200 0 L 100 200 z"/>
</a>
```

- Mailto possible :

```
<a xlink:href="mailto:nnova@dokidenki.com">...</a>
```

Lien interne

- Nécessite de spécifier un fragment SVG
  - Analogue à HTML :
    MyDrawing.svg#MyView
  - Référence compatible avec XPointer :
    MyDrawing.svg#xptr(id('MyView'))
  - Spécification d'une vue SVG :
    MyDrawing.svg#svgView(viewBox(0,200,1000,1000))

Référencement

- Nécessité d’avoir un référencement pour les liens internes (« ancre »)
- Référencement relatif : la référence est locale
- <linearGradient id="myGradient"> ….
  </linearGradient>
  …
  <rect style="fill:url(#myGradient) />

Le document SVG

- <svg> est la racine d'un document SVG
- On peut imbriquer des éléments SVG
- L'intérêt est que l'on crée ainsi un nouveau système de coordonnées.
- On peut alors réutiliser des fragments graphiques sans devoir modifier les coordonnées.

Métadonnées (1)

- Les balises <title> et <desc> permettent de documenter le code SVG.
- Ce sont des métadonnées
- Ca permet de mieux comprendre le code et d'indexer le code SVG
- On peut les mettre dans les éléments suivants :
  - 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')

Métadonnées (2)

- La structure <desc> autorise l'insertion de commentaires *non rendus* (ne s'affichant pas)
- La structure <title> autorise un titre pouvant être rendu par les *viewers* dans le bandeau par exemple
- Exemple:

```
<g>
<title> Mon image
</title>
<desc> Cette image ne contient qu'un rectangle </desc>
<rect ..... />
</g>
```

Eléments de structuration

- SVG autorise le regroupement des objets dans des blocs.
- Cette modularité permet la réutilisation des objets
- Les objets crées peuvent définir des hiérarchies dans laquelle les blocs enfants héritent des attributs des objets parents.
- Racine svg, groupage d'éléments, objets abstraits, section de définition, utilisation des éléments et description

Groupement d'éléments

- L'élément <g> sert à grouper des éléments qui se partagent des attributs communs : couleur, style, ...
- Les « enfants » de <g> héritent des propriétés
- On peut documenter le group avec <title> et <desc>

```
<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>
```

Objet abstrait

- Avec la balise <symbol>, on peut définir des objets graphiques réutilisables dans les cas suivants
  - Objet à instancier de multiples fois
  - Objet classique référencé par de nombreux éléments
  - Définition d'un élément de police textuel
  - ...
- La différence avec <g> est que l'objet lui-même n'est pas dessiné, il faut l'appeler avec <use>
- Exemple :

```
<symbol id="s1">
<rect x="0" fill="blue" width="10" height="10"/>
<rect x="10" fill="white" width="10" height="10"/>
</symbol>
```

Section de définition

- La balise <defs> autorise la définition d'objets référencés plus tard dans le même fichier.
- Il est requis que toutes les définitions d'objets devant être référencés plus tard soient faites dans la même structure <defs>
- Il n'y a donc qu'une structure <defs> par fichier SVG
- Exemple :
- <defs>
  <linearGradient id="Gradient01"> .... </linearGradient >
  </defs>
  ...
  <rect style="fill:url(#Gradient01) ..../>

Utilisation d’éléments

- <use> sert à éviter de répéter du code en permettant de réutiliser les objets suivants: <svg>, <symbol>, <g>, éléments graphics et <use>
- Pour réutiliser un objet, ceux-ci doivent avoir un identificateur XML. Par exemple :

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

  <use x="20" y="10" fill="yellow" xlink:href="#MyRect" />

  <use x="20" y="20" fill="red" xlink:href="#MyRect" />
- x, y, width, height permettent de repositionner et de redimensionner l'objet
- xlink:href permet de référencer et instantier l'objet (avec son attribut "id")
- On pourrait aussi rajouter des attributs différents dans le <use> (comme opacity)

Le système des coordonnées

- A l'origine l'élément <svg> le plus externe établit un espace utilisateur
- Chaque élément <svg> interne redéfinit un nouvel espace utilisateur et un nouveau point de vue associé
- Le rectangle visible pour l'utilisateur est appelé « viewport ». Il possède un système de coordonné qui commence en haut à gauche du rectangle.
- Suivant votre client, vous pouvez vous déplacer ou zoomer dans cet « user-space ».
- Les dessins qui dépassent de cette zone sont coupés.
- On peut se créer des viewports (cf. spec)

Unités de longueur

- On indique les longueurs par une valeur en unités absolues ou relatives :
  - em, ex (largeur d'un "m" et hauteur d'un "x" de la fonte courante)
  - px (pixels, unité par défaut)
  - 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)

Transformations

Sans rentrer dans le détail (à vous de tester), SVG permet de faire avec la balise transform :

- Des rotations
- Des translations
- Des redimensionnements
- Des transformations suivant des matrices

Exercice 1

Faire un groupe d'objets simple (un carré avec du texte) et essayer d'appliquer des transformations : rotations, translations, redimensionnement

→ Chercher la syntaxe des transformations dans les spécifications

Exercice2

- Définir une ensemble d'objets comprenant un carré rouge, une légende et un cercle vert au contour bleu. Lui donner un ID et l'appeler dans le fichier SVG, le placer aux coordonnées x= 200 y= 200

Exercice 3

- Ecrire un texte sur un chemin représentant un pic et faire en sorte que ce texte soit un lien vers une page web quelconque

SVG et HTML

- Embedding SVG in HTML pages
- Utilisation du tag <embed> dans une page HTML (ou <object>
- <embed src="fichier.svg" width="200" height="170" type="image/svg+xml">
- Rajouter des scrollbars aux pages web SVG (pratique si l'image dépasse) :
- <embed src="fichier.svg" width="100%" height="100%" type="image/svg+xml">

A voir et à essayer

- Si on a le temps :
  - Webdraw
  - Mayura
  - XMLValidator
  - Feature sympa : les filtres

Exercice 4

Faire un système de navigation (barre) en utilisant des groupes d'objets, mettre le tout dans une page HTML

Faites pas ca :)

Animation avec SVG

- Bon potentiel de SVG pour l'animation en utilisant les modifications des attributs de position, dimension, couleurs, gradients, opacités...
- Animation avec : éléments SVG déclaratifs, ECMAScript (javascript) ou d'autres langages de manipulation du DOM
- Animation « time-based » (et pas « frame-based »)
- But ici : vous montrer un exemple pas tout ce qui est possible

Mécanisme d'animation simple

- Exemple
- Code :

```
<svg width="300" height="250" >
<rect x="100" y="100" width="10px" height="100px"
   style="stroke:red; fill:rgb(0,1,1)">
<animate attributeName="width" from="10px" to="100px"
   begin="0s" dur="10s" repeatCount="1" fill="freeze"/>
<animate attributeName="height" from="10px" to="100px"
   begin="0s" dur="10s" repeatCount="1" fill="freeze"/>
</rect>
</svg>
```

- Freeze : laisse le dessin tel qu'il s'affiche à la fin de l'animation

Animation (2) : chemin

- Autre exemple simple : <animateMotion> permet de créer une animation le long d'un chemin (path)
- Path définit la trajectoire de l'animation
- Exemple :

```
<svg>
<circle cx="0" cy="0" r="5" styles="fill:red; stroke:red;">
<animateMotion path="M50,50 150,50 150,100 50,100 z" dur="5s"
   repeatCount="4" />
</circle>
```

Animation (3)

- Dans les exemples précédents, les animations commencent tout de suite. On peut les retarder en mettant un attribut begin= " x " de x secondes comme ici.
- Scrolling Text : texte animé

```
<text style="font-family: Arial, sans-serif; stroke:blue; fill:blue; font-size:24;">Ceci n'est pas un pamplemousse
<animateMotion path="M 400 90 L -300 90" begin="0s" dur="12s" repeatCount="indefinite" />
</text>
```

Après on peut faire des logos avec des trucs plus compliqués comme ca

Animation (4)

- <animateColor> : permet de changer la couleur d'un élément
- Exemple

  <text x="50" y="80" style="font-family: Arial, sans-serif; stroke:green; fill:green; font-size:36;">Je suis un cameleon

  <animateColor attributeName="fill" attributeType="CSS" from="rgb(255,0,0)" to="rgb(0,0,0)" begin="2s" dur="2s" fill="freeze" /></text>

Exercice 5

- Faire une barre de défilement qui se comporte comme cela

Exercice 6

- Dessiner un chemin noir (path) fermé et un cercle rouge qui se déplace du début à la fin. Faire varier la couleur du cercle (de rouge à bleu) le long du parcours

Exercice 7

- Faire une bannière animée, en combinant par exemple des transformations (rotations) et des animations (on fait alors varier l'angle de rotation au cours du temps)
- Rien que ca, c long :(
- Bien sur, il y a des possibilitées d'animations beaucoup plus complexes !! Cf spec

Evènements interactifs

- Bon potentiel également pour les interactions.
- Il s’agit de définir des évènements
- Exemple simple
- On décrira pas

Générer du SVG

- Jusqu’à maintenant, on a vu du SVG statique.
- En fait, véritable gros intérêt/potentiel de SVG = dynamique
- Surtout lien avec monde XML

Moyen de générer du SVG

- Server-side scripting : Perl, PHP : facile
- XSLT : plus complexe mais plus trendy et plus d'avenir
- Processeur : xslate par exemple
- But ici : vous montrer des exemples **simples** pour générer du SVG

Générer SVG avec PHP

- On peut partir d'un fichier php simple ou d'une BD (mySQL)
- On pourrait aussi écrire un script php qui parse les données XML et les renvoient en SVG
- Pourquoi faire ?
  - Utiliser des variables pour les attributs par ex : pour les dimensions, les durées d'animations...
  - Générer des tags
  - Utilisation de formulaires et script de visualisation directe de données

Comment faire ?

- Exemple simple

```
<?php
$dim1 = 100; //Definition des variables
$dim2 = 200;
$colorBody = 'blue';
header("Content-type: image/svg-xml"); //Header XML
print('<?xml version="1.0" encoding="iso-8859-1"?>' . "\n");
print('<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN"
    "http://www.w3.org/TR/2001/PR-SVG-20010719/DTD/svg10.dtd"> ' . "\n");
print("<svg xml:space=\"preserve\" " . "width=\"800\" height=\"600\" " . "viewBox=\"0 0
    $ChartWidth $ChartHeight\">\n");
print("<rect x=\"100\" y=\"100\" " ."width=\"$dim1\" height=\"$dim2\" "
    ."style=\"fill:$colorBody;\" />\n");
print('</svg>' . "\n");
?>
```

Exercice 8

Ecrire un script php qui génère une barre de défilement comme ca avec utilisation de variables (couleur, dimension et texte « voici une barre de défilement »)

XML → SVG : why ?

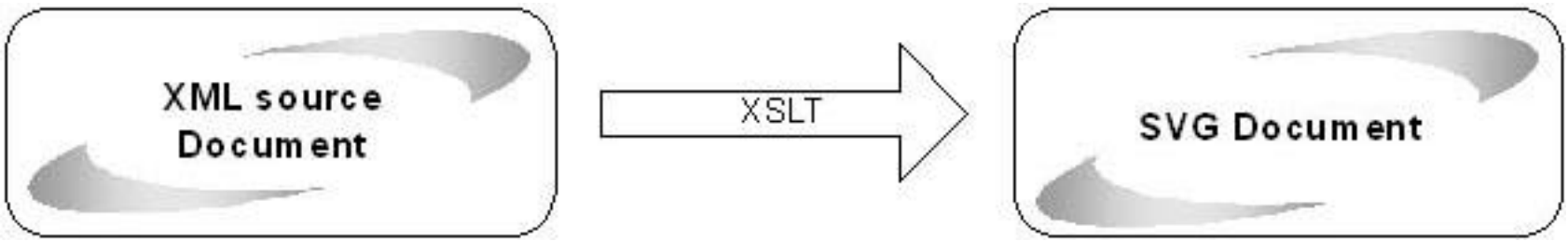
- Mission critical :
  - A partir de XML, on formatte comme on veut (pdf avec FO, XHTML, SVG...) et sur le support que l'on veut (mobile phone, PDA, ordinateur...)
  - Graphical Stylesheets are applicable to such diverse areas as Business Graphics, Flowcharting, Project Plans, Engineering Blueprints, CAD, GIS, AM/FM, Meteorology, Scientific Visualization, diagrams... → Système standardisé de représentation de données
  - Advantages of being able to vary the graphical styling separately from the data : on profite des avantages de XML

XML → SVG : avantages

- On peut faire XML → JPEG ou GIF mais SVG apporte des plus :
  - **SVG is searchable. Les éléments peuvent être indexés dans les moteurs de recherche**. Using a search engine, a user could look for a map that has a particular street ("Homer") on it. Then they could search that map for where the street ("Homer") is on the map. To do the equivalent with a raster format, you'd need optical character recognition software.
  - On a **accès au DOM** des images SVG donc on peut donner une dynamique au contenu avec JS (mousover event, mouseclick...).
  - On peut **zoomer et se déplacer sur l'image SVG** SANS perte de qualité de l'image. En plus, on n'a pas besoin d'attendre des heures que le serveur recharge une version zoomée de la page
  - **SVG files are much smaller** than their corresponding raster graphic representation, so they **download faster**.

Logique XML

- On retrouve la triade classique
  - DTD : définition/structure du document
  - Stylesheet : feuille de style (svg donc ici)
  - XML : les données
- XSLT : traduction en SVG



XML -> SVG

- Plusieurs solutions :
  - Un processeur : xslate, saxon...
  - Cocoon : marche pas avec la version dispo a tecfa
  - En client-side avec Mozilla

Exemple

- On définit un DTD simple :

```
<?xml encoding="ISO-8859-1"?>
<!ELEMENT high (name)>
<!ELEMENT name (#PCDATA)>
```

- On écrit le fichier XML correspondant :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet href="essai.xsl" type="text/xsl"?>
<?cocoon-process type="xslt"?>
<!DOCTYPE essai SYSTEM "essai.dtd">
<high>
   <name>STAF Students are so cool</name>
</high>
```

Exemple (2)

- Le fichier XSL :

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xsl:stylesheet version="1.0"  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="xml" indent="yes" standalone="no" doctype-public="-//W3C//DTD SVG
    1.0//EN" doctype-system="http://www.w3.org/TR/2001/PR-SVG-
    20010719/DTD/svg10.dtd" />
<xsl:template match="high">
 <svg>
  <rect x="50" y="50" rx="5" ry="5" width="150" height="70"
     style="fill:#CCCCFF;stroke:#000099" />
 <text x="55" y="90" style="stroke:#000099;fill:#000099;fontsize:24;">
  <xsl:value-of select="name" />
     </text>
 </svg>
 </xsl:template>
</xsl:stylesheet>
```

Exemple (3)

- Ensuite, on utilise un processeur qui va permettre de faire la transformation XML -> SVG
- Xslate ou saxon
- A Tecfa, xslate sur tecfasun5.unige.ch (taper xslate pour voir les options et les attributs à rentrer)

Commande xslate

- Syntaxe : `xslate -IN fichier.xml -XSl fichier.xsl -OUT fichier.svg`
- Le processeur xslate écrit un fichier svg en sortie.

Exercice à rendre

- Générer un fichier SVG avec la méthode que vous voulez (php ou xslt) pour représenter graphiquement des informations simples (par exemple un pie chart représentant des pourcentages, avec couleur, légende)
- + Rapport explicatif as usual :)
- Vous pouvez commencer à bosser dessus maintenant !
- Vous pouvez nous demander notre avis (faisabilité)