Génération de graphismes vectoriels avec XSLT

Code: svg-xslt

Originaux

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

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

Auteurs et version

- Daniel K. Schneider

- Version: 0.4 (modifié le 27/11/06)

Prérequis

Module technique précédent: **xml-tech**

Module technique précédent: **svg-intro** (SVG statique de base)

Module technique précédent: **php-intro** (Pour XSLT server-side)

Module technique précédent: **php-xml** (Notions techniques de XML)

Module technique précédent: **xml-xslt**

Module technique précédent: **xml-xslt2**

Abstract

Introduction à la génération de graphismes SVG avec XSLT

Ce document est assez provisoire et ne donne que des pistes de départ ....

Objectifs

- Idées et techniques de base pour la visualisation
- Programmation XSLT plus avancée

1. Table des matières détaillée

2. Génération de SVG avec XSLT

2.1 Traitement batch ou server-side

- Il est assez difficile de générer du SVG avec XSLT client-side (sauf pour Firefox SVG natif)
- En conséquence, on montre d'abord la solution batch ou server-side, donc soit:
  - fabriquer le SVG avec un processeur XSLT (genre Saxon) et poser le fichier sur le serveur
  - utiliser un "service PHP" pour appliquer la feuille de style côté serveur

Solution 1: Fabrication de travaux.svg avec un processeur XSLT:

- Par exemple avec saxon (logiciel à installer chez vous)
- L'instruction suivante crée un fichier travaux.svg à partir de travaux.xml et travaux.xsl

```
saxon -o travaux.svg travaux.xml travaux.xsl
```

Solution 2: XSLT avec un service de traduction en ligne:

- Solution la plus simple
- Marche seulement depuis l'Université de Genève ou un mot de passe

url: **http://tecfa.unige.ch/guides/tools**

Solution 3: XSLT server-side avec PHP:

- Le fichier travaux.php génère du SVG à partir de travaux.xml et travaux.xsl
- La source (= travaux.text ou travaux.phps) montre comment utiliser le processeur XML
- Important : il faut absolument déclarer le mime type au début du fichier(!!)

```
<?php
header("Content-type: image/svg+xml");
```

Code PHP pour la solution 3

- il vous suffit de copier le fichier php et de changer les noms des 2 fichiers
- Depuis le Web, prendre travaux.text et ensuite renommer en travaux.php

```
header("Content-type: image/svg+xml");

error_reporting(E_ALL);

$xml_file = 'travaux.xml';
$xsl_file = 'travaux.xsl';

// load the xml file (and test first if it exists)
$dom_object = new DomDocument();
if (!file_exists($xml_file)) exit('Failed to open $xml_file');
$dom_object->load($xml_file);

// create dom object for the XSL stylesheet and configure the transformer
$xsl_obj = new DomDocument();
if (!file_exists($xsl_file)) exit('Failed to open $xsl_file');
$xsl_obj->load($xsl_file);
$proc = new XSLTProcessor;
$proc->importStyleSheet($xsl_obj); // attach the xsl rules

$html_fragment = $proc->transformToXML($dom_object);
print ($html_fragment);
```

Exemple 2-1: Simple visualisation d'une page travaux

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

Afficher travaux.xml (avec FireFox !) ou travaux.svg et lire ci-dessous

Code XSLT: travaux.xsl

- Ici on utilisera un style de programmation fonctionnelle
  - Donc pas le mécanisme habituel d'exécution de règles
  - Cela nous permettra de positionner plus facilement les éléments sur l'écran.
- Pour mieux comprendre comment cela marche:
  - commencez à lire la documentation de ce code depuis la fin (<xsl:template match="student">)
  - Tracez l'exécution du programme avec un processeur xslt comme saxon.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

- Ces déclarations indiquent qu'on produira du SVG

```
 <xsl:output method="xml" indent="yes"
  encoding="ISO-8859-1"
  standalone="no"
  doctype-public="-//W3C//DTD SVG 1.0//EN"
  doctype-system="http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd" />
```

- Ci-dessous on déclare des paramètres globaux pour définir qq. distances. Vous pouvez les modifier pour voir leur effet

```
 <!-- GLOBAL PARAMETERS - you may change this -->

 <xsl:param name="increase-y" select="60"/>
 <xsl:param name="element-length" select="150"/>
```

```
<xsl:param name="element-height" select="30"/>
<xsl:param name="space" select="10"/>

<!-- XSLT Functions -->
```

- Le template ci-dessous affichera chaque exercice
  - tous les exercices d'un cours sont affichés sur une même ligne, décalés vers la droite
  - le contexte d'exécution est l'élément "exercise".

```
<!-- Display a single exercise -->
<xsl:template name="render-exercise">
 <xsl:param name="position-x"/>
 <xsl:param name="position-y"/>

 <rect x="{$position-x * ( $element-length + $space ) }"
       y="{$position-y * $increase-y }"
       width="{$element-length}"
       height="{$element-height}"
       style="fill:yellow; stroke:black; stroke-width:3"/>

 <text x="{$position-x * ( $element-length + $space ) }"
       y="{$position-y * $increase-y + 15 }"
       style="stroke:#000099;fill:#000099;font-size:10;">
  <xsl:value-of select="title"/>

 </text>
</xsl:template>

<!-- Display a course (Text + each exercise a box on the same line  -->
```

- Render-course est un template qui a pour boulot d'afficher les éléments "exercice"
  - Le contexte d'exécution est l'élément "course"

- En fonction de sa position dans la liste, cet élément est affiché un peu plus bas ($position-y)
- On affiche le titre du cours plus tous les exercices associés, mais pour cela on utilise de nouveau un autre template

```
<xsl:template name="render-course">
 <xsl:param name="position-y"/>

 <text x="{$space}" y="{$position-y * $increase-y - $space }"
       style="stroke:#000099;fill:#000099;font-size:12;">
  <xsl:value-of select="title"/>
 </text>

 <xsl:for-each select="exercise">
  <xsl:call-template name="render-exercise">
   <xsl:with-param name="position-x" select="position()"/>
   <xsl:with-param name="position-y" select="$position-y"/>
  </xsl:call-template>
 </xsl:for-each>

</xsl:template>
```

- Ce template est appelé en premier en tant que règle.
  - Il contient une boucle qui, pour chaque élément-enfant "course", fait appel au template "render-course" en lui passant le contexte de l'élément "course" en question plus sa position dans la liste des enfants (select="position")

```
<xsl:template match="student">

 <svg width="1000" height="900">
  <!-- xmlns="http://www.w3.org/2000/svg"> -->
  <text x="{$space}" y="90" style="stroke:#000099;fill:#000099;font-size:24;">
   Example visualisation of a student's work page</text>
```

```
    <svg y="3cm">
     <xsl:for-each select="//course">
      <xsl:call-template name="render-course">
        <xsl:with-param name="position-y" select="position()"/>
      </xsl:call-template>
     </xsl:for-each>
    </svg>

   </svg>

  </xsl:template>
</xsl:stylesheet>
```

2.2 SVG avec XSLT client-side

- Pas facile, car la situation n'est pas totalement stabilisée. Ca manque d'exemples sur le Web.
- Je ne sais pas si les solutions présentées ci-dessous sont justes !! Use with care !

A. Solution qui marche dans Firefox avec SVG natif (Firefox 1.5)

Exemple 2-2: Visualisation client-side avec un navigateur SVG natif

url: **http://tecfa.unige.ch/guides/svg/ex/svg-xslt/travaux-client-side.xml**

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

- On génère une page HTML avec le SVG intégré
- La feuille de style ressemble à celui de l' exemple 2-1 "Simple visualisation d'une page travaux" [6]
- Ici on montre juste le template exécuté en premier

```
<xsl:template match="/">
  <html xmlns:svg="http://www.w3.org/2000/svg">
   <head><title>Client-side XHTML / SVG generation</title></head>
   <body>
    <p> .....   </p>
   <svg width="1000" height="900" xmlns="http://www.w3.org/2000/svg">
    <text x="{$space}" y="90" style="stroke:#000099;fill:#000099;font-size:24;">Example
visualisation of a student's work page</text>
     <svg y="3cm">
       <xsl:for-each select="//course">
         <xsl:call-template name="render-course">
           <xsl:with-param name="position-y" select="position()"/>
         </xsl:call-template>
```

```
        </xsl:for-each>
      </svg>
     </svg>
    </body>
</html>
</xsl:template>
```

Solution qui marche aussi avec un plugin SVG et IE 6

url: **http://tecfa.unige.ch/guides/svg/ex/svg-xslt/travaux-client-side2.xml**

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

- Cet exemple est plus compliqué, par sûr qu'il marche avec toutes les variantes ...
- Il faut générer le code qui associe le name space "svg" au plugin Adobe
- Il faut utiliser partout des préfix "**`svg:`**" pour les balises SVG
  (je n'ai pas trouvé comment faire autrement ....)
  - Cf. les déclarations **`xmlns:svg="http://www.w3.org/2000/svg`**"
  - Donc pas **`xmlns`**="http...", mais **`xmlns:svg`**="...." !!
- Cet exemple marche aussi avec un navigateur natif comme Firefox.
  - Il lui suffit d'une déclaration du namespace svg dans la balise html:

```
<html xmlns:svg="http://www.w3.org/2000/svg">
```

Voici le template pour la racine:

```
<xsl:template match="/">
  <html xmlns:svg="http://www.w3.org/2000/svg">
  <object id="AdobeSVG" CLASSID="clsid:78156a80-c6a1-4bbf-8e6a-3cd390eeb4e2"> </object>
   <xsl:processing-instruction name="import">namespace="svg"
implementation="#AdobeSVG"</xsl:processing-instruction>
```

```
   <head><title>Client-side XHTML / SVG generation</title></head>

   <body>
    <p> Read this article:
     <a href="http://surguy.net/articles/client-side-svg.xml">http://surguy.net/
articles/client-side-svg.xml</a>.
     It explains how to generate SVG plugin code within XHTML.
   </p>

   <svg:svg width="1000" height="900" xmlns:svg="http://www.w3.org/2000/svg">
     <svg:text x="{$space}" y="90" style="stroke:#000099;fill:#000099;font-
size:24;">Example visualisation of a student's work page</svg:text>
     <svg:svg y="3cm">
       <xsl:for-each select="//course">
         <xsl:call-template name="render-course">
           <xsl:with-param name="position-y" select="position()"/>
         </xsl:call-template>
       </xsl:for-each>
     </svg:svg>
   </svg:svg>
 </body>
</html>
</xsl:template>
```

3. Faire des cercles

3.1 Trigonométrie

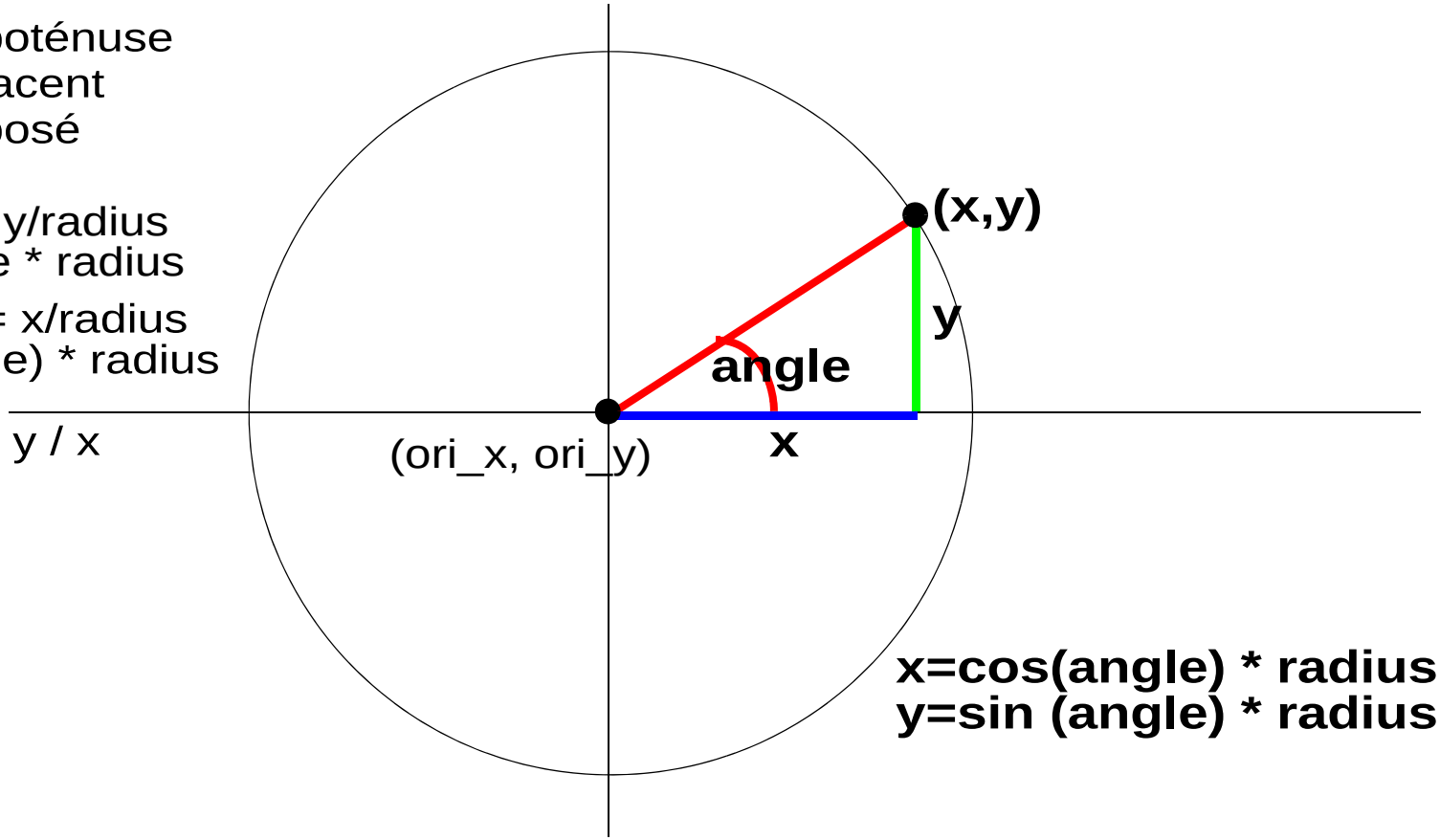
- Pour placer des éléments autour d'un cercle il faut faire un peu de trigonométrie

radius = hypoténuse
x = côté adjacent
y = coté opposé

sin(angle) = y/radius
y = sin(angle * radius

cos(angle) = x/radius
x = cos(angle) * radius

tan(angle) = y / x



Formules:

Les angles sont exprimés en radians basés sur la constante Pi (3.142)

- En informatique on utilise les radians (au lieu des degrés)
- Un cercle = 360 degrés = 2 Pi, 180 degrés = Pi, 90 degrés = Pi/2

  `Formule: rad = deg / 180 * Pi`

Calculer la position x d'un élément en connaissant l'angle et le radius:

- Pour chaque élément à placer on incrémente l'angle: i * angle
- Pour les dessins vectoriels, il faut ajouter la position du cercle (ori_x, ori_y)
- Pour rendre la formule plus flexible: on ajoute l'angle de départ, on travaille avec un arc (partie d'une cercle), et on utilise (i div n) qui permet de faire un 2ème tour.

  `x=ori_x + cos(arc / n_els * (i % n_els) + start_angle) * radius`

  `ori_x` = coordonné x du cercle
  `arc` = angle total de dessin, par ex. pour 10 éléments sur 360 degrés:
  `n_els` = nombre d'éléments
  `angle = arc/n_els = Pi * 2 / 10 = 6.283 / 10 = 0.63`
  `i` = numéro de l'élément (`i % n_els` est égal à `i` dans ce contexte)
  `start_angle` = angle de départ sur le cercle (où commencer à dessiner)

Pour calculer la position y d'un élément en connaissant l'angle et le radius:

`y=ori_y + sin(arc / n_els * (i % n_els) + start_angle) * radius`

3.2 Alignement en cercle avec XSLT et un brin de PhP

Exemple 3-1: Génération d'éléments SVG autour d'un cercle

url: http://tecfa.unige.ch/guides/svg/ex/objects-in-circles/ **(répertoire et source)**

url: **elements-on-circle-with-xslt.php (placer des rectangles autour d'un cercle)**

url: **elements-on-circle-with-xslt.xsl (feuille de style)**

Fonctions trigonométriques en XSLT

- Il n'y en a pas
  (donc il faudrait soit les programmer avec JS, Java, ou en XSLT avec des tables, etc.)
- Ici on a fait appel aux fonctions trigonométriques de PhP et on fait le rendering du côté serveur.
- Pour que cela marche, il faut enregistrer les fonctions PHP dans xslt:

```
$proc = new XSLTProcessor;
// This allows to access ALL php functions within XSLT
$proc->registerPHPFunctions();
```

Attention aux namespaces dans le fichier XSLT:

```
<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 xmlns:php="http://php.net/xsl"
 xmlns:xlink="http://www.w3.org/1999/xlink">
```

(autres explications: à faire)