Chapitre 12 Développement Web et fractales

"Your focus determines your reality."
- Star Wars: Episode II

"Do not be afraid to love everyone."
- Wesley Eads (1930-)

Objectifs

- Application statistique avec ASP.Net
- Utiliser XSLT pour passer de XML à SVG
- Interactivité avec XHTML et ECMAScript
- Données en temps réel et SVG
- Fractales en SVG

Aperçu

Dans ce chapitre, nous allons voir des utilisations avancées de SVG.

Nous examinerons une application statistique avec ASP.Net. Nous utiliserons HTML, des données XML, des feuilles de style XSLT, ECMAScript, les contrôles ASP et C# pour une application en temps réel.

Partie I: Publier avec SVG

Dans cette section nous allons créer une application statistique qui peut être déployée sur ASP.Net. Nous allons explorer les possibilités que nous apporte la génération de documents SVG sur le serveur.

L'interface

Nous avons une interface HTML + SVG + ECMAScript qui permet à l'utilisateur de choisir le type de représentation (ligne, barre, colonne, et flux) la taille du graphique (largeur et hauteur). Nous appliquerons à l'une de ces représentations des données en temps réel. L'utilisateur a également accès aux fichiers source XML, XSLT et SVG.

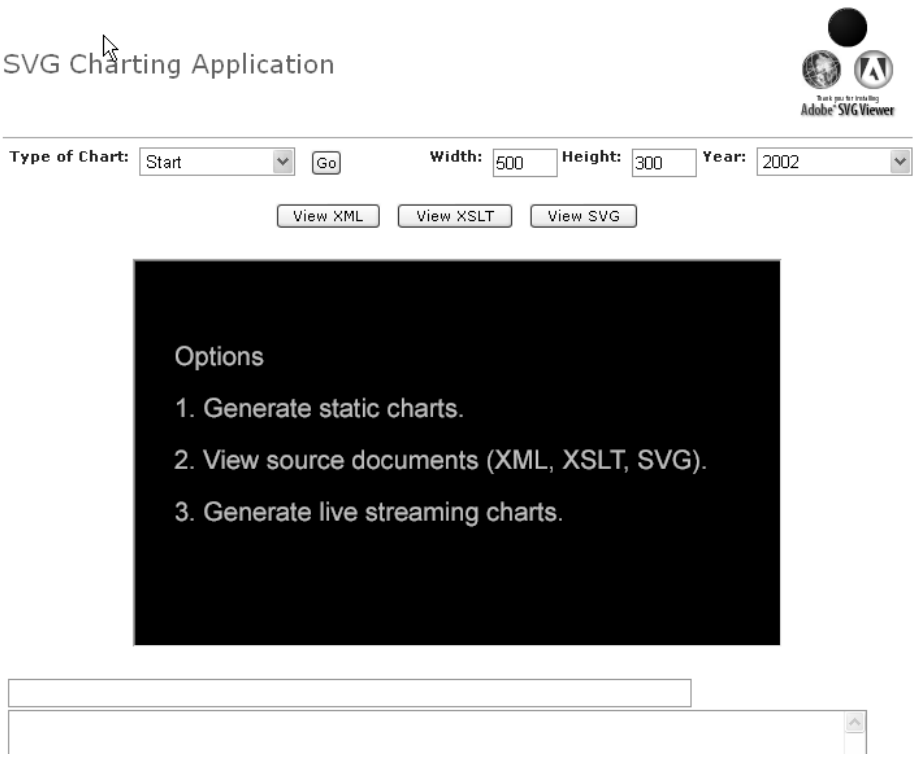

Figure 12-1. Interface utilisateur

L'interface utilise XHTML, SVG et ECMAScript mais est le résultat du mélange de plusieurs langages XML, XSLT, ASP.Net, C#, HTML, ECMAScript, et enfin SVG.

Développement ASP.Net

J'utilise Visual Studio.Net pour créer cette application mais tout ce qui est nécessaire est dans ".Net Framework SDK" s'exécutant dans un environnement serveur basique.

Nous utiliserons C# mais nous ne discuterons pas ce langage en profondeur dans ce livre. Comme ECMAScript, C# est un vrai langage orienté objet qui supporte classes, héritage, polymorphisme, etc. Comme vous le verrez, C# est plus pratique à utiliser que son ancêtre C++.

Le code

C'est le code ASP.Net et XHTML pour générer l'interface de notre application.

```
<%@ Page language="c#" Codebehind="Default.aspx.cs" AutoEventWireup="false"
Inherits="SVGChart08.SVGGenerator" %>
<%@ Import Namespace="SVGChart08" %>
<HTML>
<HEAD>
<link href='<%= Request.ApplicationPath + "/style/Main.css" %>' type=text/css
rel=stylesheet>
</HEAD>
<body bottomMargin="0" leftMargin="0" topMargin="0" rightMargin="0" marginwidth="0"
marginheight="0">
<form id="Form1" runat="server">
<span id="svgFileName" runat="server" style="DISPLAY:none">Start.svg</span>
<div align="center">
<table cellSpacing="0" cellPadding="4" width="100%" border="0">
<tr vAlign="top">
<td align="middle" width="*">
```

```
<table cellSpacing="0" cellPadding="0" width="700">
<tr>
<td class="Head" align="left">SVG Charting Application</td>
<td class="Head" align="right">
<iframe marginWidth="0" marginHeight="0" src="svg/svgtest.svg" frameBorder="0"
width="100" height="100">
<embed src="svg/svgtest.svg" name="SVGEmbed" width="100px" height="100px"
type="image/svg+xml" pluginspage="http://www.adobe.com/svg/viewer/install/" />
</iframe></td>
</tr>
<tr>
<td colSpan="2">
<hr noShade SIZE="1">
</td>
</tr>
</table>
<table cellSpacing="0" cellPadding="0" width="700">
<tr vAlign="top">
<td class="SubHead" align="right">Type of Chart:  </td>
<td align="left">
<asp:dropdownlist id="XslTransformSrc" runat="server" width="120"
AutoPostBack="True">
<asp:listitem>Start</asp:listitem>
<asp:listitem>Line</asp:listitem>
<asp:listitem>Bar</asp:listitem>
<asp:listitem>Column</asp:listitem>
<asp:listitem>Streaming</asp:listitem>
</asp:dropdownlist>
 
<asp:button class="CommandButton" id="generateButton" runat="server"
CausesValidation="False" Text="Go"></asp:button>  
</td>
<td class="SubHead" align="right" width="100">Width:  </td>
<td align="left" width="50"><asp:textbox id="Width" width="50"
Runat="server">500</asp:textbox></td>
<td class="SubHead" align="right">Height:  </td>
<td align="left"><asp:textbox id="Height" width="50"
Runat="server">300</asp:textbox></td>
<td class="SubHead" align="right">Year:  </td>
<td align="left" width="50">
<asp:dropdownlist id="DropdownYear" runat="server" width="120" AutoPostBack="True">
<asp:listitem>2002</asp:listitem>
<asp:listitem>2001</asp:listitem>
<asp:listitem>2000</asp:listitem>
</asp:dropdownlist></td>
</tr>
<tr vAlign="top">
<td colSpan="8"> 
</td>
</tr>
<tr vAlign="top">
<td class="SiteLink" align="middle" colSpan="8">
<asp:button class="CommandButton" id="viewXml" runat="server"
CausesValidation="False" Text="View XML"></asp:button>  
<asp:button class="CommandButton" id="viewXslt" runat="server"
CausesValidation="False" Text="View XSLT"></asp:button>  
<asp:button class="CommandButton" id="viewSvgSource" runat="server"
CausesValidation="False" Text="View SVG"></asp:button></td>
</tr>
</table>
```

```
</td>
</tr>
</table>
<p>
<span id="embedSvg" runat="server">
<iframe src="svg/Start.svg" frameBorder="1" width="500" height="300">
<embed src="svg/Start.svg" name="SVGChart" width="500px" height="300px"
type="image/svg+xml" pluginspage="http://www.adobe.com/svg/viewer/install/" />
</iframe>
</span>
</p>
<table cellSpacing="0" cellPadding="4" width="700" border="0">
<tr vAlign="top">
<td align="left" width="*"><asp:textbox id="output1" runat="server"
Width="526px"></asp:textbox><br>
<textarea id="output2" rows="30" cols="80" runat="server" align="left"></textarea>
<P></P>
</td>
</tr>
</table>
</div>
</form>
</SPAN>
</body>
</HTML>
```

Voyons quelques points de notre fichier ASPX en détail. Notre fichier ASPX 'default.aspx' fait appel au SVG avec la balise <EMBED>. Comme les navigateurs ne supportent pas SVG sans plugin, c'est la solution qui permettra une exécution dans le plus de configurations.

```
<span id="embedSvg" runat="server">
      <iframe src="svg/Start.svg" frameBorder="1" width="500" height="300">
            <embed src="svg/Start.svg" name="SVGChart" width="500px" height="300px"
type="image/svg+xml" pluginspage="http://www.adobe.com/svg/viewer/install/" />
      </iframe>
</span>
```

En plaçant notre balise 'embed' dans une balise 'iframe', nous pourrons tester cette application dans Mozilla sans trop de problèmes.

Côté serveur, quand nous voulons actualiser les paramètres 'width', 'height' et 'src' notre code C# réécrit le code avec les nouvelles valeurs.

Dans ce code, le serveur exécute tout le code entre <asp:xxx> et </asp>, ainsi que les balises avec l'attribut runat="server" avant de retourner le XHTML avec ECMAScript au client.

Types de diagrammes

Il y a de très nombreuses possibilités de représentations statistiques que vous pouvez adapter à vos besoins. Voici une représentation pour un diagramme ligne brisée.

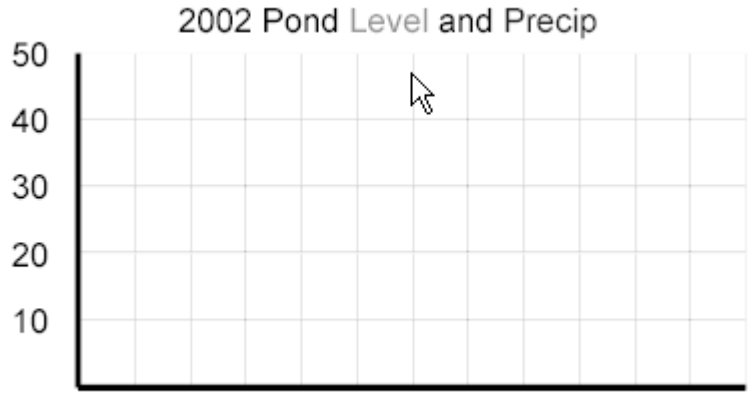

Figure 12-2. Présentation du diagramme

Voici le source SVG pour le créer:

```
<svg width="500" height="300" viewBox="0 0 110 90" xml:space="preserve">
        <g id="Chart" font-size="5"
                transform="translate( 10, 70 ) rotate( -90 )">
        <defs>
            <pattern id="gridPattern" width="10" height="8.33"
                patternUnits="userSpaceOnUse">
                        <rect x="0" y="0" width="10" height="8.33"
style="fill:none;stroke:rgb(128,128,128);stroke-width:.1" />
                </pattern>
        </defs>

        <text x="55" y="15" style="fill:black" writing-mode="tb">
                2002 Pond <tspan fill="darkorange">Level</tspan>
                 and <tspan fill="blue">Precip</tspan>
        </text>

        <text x="50" y="-10" writing-mode="tb">50</text>
        <text x="40" y="-10" writing-mode="tb">40</text>
        <text x="30" y="-10" writing-mode="tb">30</text>
        <text x="20" y="-10" writing-mode="tb">20</text>
        <text x="10" y="-10" writing-mode="tb">10</text>

        <rect x="0" y="0" width="50" height="100" fill="url(#gridPattern)" />

        <line x1="0" y1="0" x2="0" y2="100" style="stroke:black;" />
        <line x1="0" y1="0" x2="50" y2="0" style="stroke:black;" />
        <g id="ChartData">
        </g>
</g>
</svg>
```

Il n'y a pas de données représentées. Nous allons les insérer à partir de notre fichier XML.

Les données

Pour les données, j'ai collecté les hauteurs d'eau d'un étang proche et les précipitations mensuelles pour ma région et rassemblé les résultats dans un fichier XML

```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- ===================================================================== -->
<!-- Pond Wobegon Water Level and Precipitation Data                       -->
<!-- ===================================================================== -->
<pond id="Wobegon">
      <title> Pond Level and Precip</title>
      <year id="2002">
            <month id="January">
```

```
                <level>31.9</level>
                <precip>14.3</precip>
            </month>
            <month id="February">
                <level>27.6</level>
                <precip>11.8</precip>
            </month>
            <month id="March">
                <level>34.7</level>
                <precip>19.5</precip>
            </month>
            …and so on…
        </year>
        <year id="2003">
            <month id="January">
                <level>35.9</level>
                <precip>4.3</precip>
            </month>
            …and so on…
        </year>
</pond>
```

Quoique les données puissent provenir de n'importe quelle source, une base de données, un fichier texte, nous utiliserons un document XML. Ce qui nous permettra de transformer les données en SVG avec XSLT.

Pour créer le fichier XML, j'utilise un traitement de textes "Ultra-Edit" (www.ultraedit.com) et un "XML/XSLT IDE" nommé "Xselorator" (www.MarrowSoft.com).

Je n'ai besoin que d'une simple interface pour le diagramme. C'est très simple - les textes sont positionnés par les coordonnées en pixels et la dernière partie du SVG inclut les axes, le quadrillage et les données une fois chargé.

Le fichier XML est ensuite transformé avec XSLT pour produire une grande variété de diagrammes.

La feuille de style XSLT

La seconde étape concerne la feuille de style XSLT utilisée pour transformer les données XML en diagramme SVG. Chaque élément de notre diagramme, y compris sa taille, les légendes, les axes, le quadrillage peut-être modifié avec XSLT. Toutefois, dans notre exemple, nous nous contenterons d'ajouter deux lignes brisées pour le niveau de mon étang et les précipitations.

La feuille de style pour le diagramme "ligne brisée"

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!-- ====================================================================== -->
<!-- Generate a line graph                                                  -->
<!-- ====================================================================== -->

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

```
<xsl:param name="width" select="'500'"/>
<xsl:param name="height" select="'500'"/>
<xsl:param name="year" select="'2002'"/>

<xsl:template match="//pond">
<svg
    width="{$width}"
    height="{$height}"
    viewBox="0 0 110 90"
    xml:space="preserve"
    onload="Init(evt); InitContextMenu();">

<script>
<xsl:comment>
<![CDATA[
var SVGDoc=null;
var L1=null;
function Init( evt ) {
      SVGDoc = evt.getTarget().getOwnerDocument();
}
]]>
</xsl:comment>
</script>

<!-- chart -->
<g id="Chart" font-size="5" transform="translate( 10, 70 ) rotate( -90 )">
      <defs>
            <pattern id="gridPattern" width="10" height="8.33"
patternUnits="userSpaceOnUse">
                  <rect x="0" y="0" width="10" height="8.33"
style="fill:none;stroke:rgb(128,128,128);stroke-width:.1"/>
            </pattern>
      </defs>
      <!-- labels -->
      <text x="55" y="15" style="fill:black" writing-mode="tb"><xsl:value-of
select="$year"/> <!--<xsl:apply-templates select="title"/>-->Pond <tspan
fill="darkorange">Level</tspan> and <tspan fill="blue">Precip</tspan></text>

      <xsl:call-template name="monthLabels"/>
      <text x="50" y="-10" writing-mode="tb">50</text>
      <text x="40" y="-10" writing-mode="tb">40</text>
      <text x="30" y="-10" writing-mode="tb">30</text>
      <text x="20" y="-10" writing-mode="tb">20</text>
      <text x="10" y="-10" writing-mode="tb">10</text>
      <rect x="0" y="0" width="50" height="100" fill="url(#gridPattern)"/>
      <line x1="0" y1="0" x2="0" y2="100" style="stroke:black;"/>
      <line x1="0" y1="0" x2="50" y2="0" style="stroke:black;"/>
      <g id="grid">
            <path id="pondLevelData" stroke="darkorange" fill="none"><xsl:attribute
name="d"><xsl:text>M </xsl:text> <xsl:call-template
name="monthlyLevel"/></xsl:attribute></path>
            <path id="pondPrecipData" stroke="blue" fill="none"><xsl:attribute
name="d"><xsl:text>M </xsl:text> <xsl:call-template
name="monthlyPrecip"/></xsl:attribute></path>
      </g>
</g>
</svg>
</xsl:template>
```

```
<xsl:template match="title">
      <xsl:apply-templates/>
</xsl:template>

<xsl:template name="monthLabels">
      <xsl:for-each select="//year[@id=$year]//month">
            <text x="-2" writing-mode="rl"><xsl:attribute name="y"><xsl:value-of
select = "position() * 8.33" /></xsl:attribute><xsl:value-of select="substring( @id,
1, 3 )"/></text>
      </xsl:for-each>
</xsl:template>

<xsl:template name="monthlyLevel">
      <xsl:for-each select="//year[@id=$year]//month//level[.!='']">
            <xsl:choose >
                  <xsl:when test = "position()=1" >
                        <xsl:value-of select="."/>
                        <xsl:text> 0</xsl:text>
                  </xsl:when>
                  <xsl:when test = "position()!=1" >
                        <xsl:text> L </xsl:text>
                        <xsl:value-of select="."/>
                        <xsl:text> </xsl:text>
                        <xsl:value-of select = "position() * 8.33" />
                  </xsl:when>
            </xsl:choose>
      </xsl:for-each>
</xsl:template>

<xsl:template name="monthlyPrecip">
      <xsl:for-each select="//year[@id=$year]//month//precip[.!='']">
            <xsl:choose >
                  <xsl:when test = "position()=1" >
                        <xsl:value-of select="."/>
                        <xsl:text> 0</xsl:text>
                  </xsl:when>
                  <xsl:when test = "position()!=1" >
                        <xsl:text> L </xsl:text>
                        <xsl:value-of select="."/>
                        <xsl:text> </xsl:text>
                        <xsl:value-of select = "position() * 8.33" />
                  </xsl:when>
                  <xsl:when test = "lang('de')" >Germany</xsl:when>
            </xsl:choose>
      </xsl:for-each>
</xsl:template>

<xsl:template match = "*" >
      <xsl:value-of select = "*" />
      <xsl:text> </xsl:text>
</xsl:template>

</xsl:stylesheet>
```

Quelques passages sont signalés en gras, ils sont importants.

```
<xsl:call-template name="monthLabels"/>
```

Ceci exécutera:

```
<xsl:template name="monthLabels">
      <xsl:for-each select="//year[@id=$year]//month">
            <text x="-2" writing-mode="rl"><xsl:attribute name="y"><xsl:value-of
select = "position() * 8.33" /></xsl:attribute><xsl:value-of select="substring( @id,
1, 3 )"/></text>
      </xsl:for-each>
</xsl:template>
```

L'étape suivante est la sélection des données du document XML en utilisant les déclarations Xpath. La syntaxe de XPath est utilisée pour naviguer aisément dans le document XML pendant le processus de transformation XSLT.

```
"//year[@id=$year]//month"
```

Le résultat est douze éléments texte avec les trois premières lettres des mois de l'année.

```
<text x="-2" writing-mode="rl" y="8.33">Jan</text>
<text x="-2" writing-mode="rl" y="16.66">Feb</text>
<text x="-2" writing-mode="rl" y="24.990000000000002">Mar</text>
<text x="-2" writing-mode="rl" y="33.32">Apr</text>
<text x="-2" writing-mode="rl" y="41.65">May</text>
<text x="-2" writing-mode="rl" y="49.980000000000004">Jun</text>
<text x="-2" writing-mode="rl" y="58.31">Jul</text>
<text x="-2" writing-mode="rl" y="66.64">Aug</text>
<text x="-2" writing-mode="rl" y="74.97">Sep</text>
<text x="-2" writing-mode="rl" y="83.3">Oct</text>
<text x="-2" writing-mode="rl" y="91.63">Nov</text>
<text x="-2" writing-mode="rl" y="99.960000000000008">Dec</text>
```

Code côté serveur

L'utilisation de données côté serveur est de plus en plus utilisée pour rendre disponibles des flots de données sur les sites Web. Un document SVG peu contenir de très nombreuses données mais nous sommes limités par la taille des fichiers. Très souvent, l'utilisateur n'a besoin que d'une faible partie des données et il est plus pratique de générer le SVG sur le serveur et de n'envoyer que le nécessaire. L'interactivité permet cette sélection des données par l'utilisateur. .Net est une manière de générer des images SVG dynamiques.

L'intérêt d'utiliser .Net côté serveur est que nous ne sommes plus dépendants du navigateur et ceci nous ouvre d'autres possibilités. Nous pouvons à partir de données sur le serveur créer cartes, diagrammes, graphes, navigation interactive et bien d'autres choses utilisant des données

Le code C#

La première ligne du document ASPX référence quelque code externe:

```
<%@ Page language="c#" Codebehind="Default.aspx.cs" AutoEventWireup="false"
Inherits="SVGChart08.SVGGenerator" %>
<%@ Import Namespace="SVGChart08" %>
```

Voyons ce code externe, le fichier ‘Default.aspx.cs’ en entier:

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Web;
```

```
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.HtmlControls;
using System.IO;
using System.Xml;
using System.Xml.Xsl;
using System.Xml.XPath;

namespace SVGChart08
{

#region SVGGenerator
public class SVGGenerator : System.Web.UI.Page
{
protected System.Web.UI.WebControls.Button generateButton;
protected System.Web.UI.WebControls.DropDownList XslTransformSrc;
protected System.Web.UI.WebControls.DropDownList DropdownYear;
protected System.Web.UI.WebControls.TextBox Width;
protected System.Web.UI.WebControls.TextBox Height;
protected System.Web.UI.WebControls.Button viewXml;
protected System.Web.UI.WebControls.Button viewXslt;
protected System.Web.UI.WebControls.Button viewSvgSource;
protected System.Web.UI.WebControls.TextBox output1;
protected System.Web.UI.HtmlControls.HtmlTextArea output2;
protected System.Web.UI.HtmlControls.HtmlGenericControl embedSvg;
protected System.Web.UI.HtmlControls.HtmlGenericControl svgFileName;

#region Web Form Designer generated code
override protected void OnInit(EventArgs e)
{
      //
      // CODEGEN: This call is required by the ASP.NET Web Form Designer.
      //
      InitializeComponent();
      base.OnInit(e);
}

/// <summary>
/// Required method for Designer support - do not modify
/// the contents of this method with the code editor.
/// </summary>

private void InitializeComponent()
{
      this.XslTransformSrc.SelectedIndexChanged += new
System.EventHandler(this.XslTransformSrc_SelectedIndexChanged);
      this.DropdownYear.SelectedIndexChanged += new
System.EventHandler(this.DropdownYear_SelectedIndexChanged);
      this.generateButton.Click += new
System.EventHandler(this.generateButton_Click);
      this.viewXml.Click += new System.EventHandler(this.viewXml_Click);
      this.viewXslt.Click += new System.EventHandler(this.viewXslt_Click);
      this.viewSvgSource.Click += new System.EventHandler(this.viewSvgSource_Click);
      this.Load += new System.EventHandler(this.Page_Load);
}
#endregion

private void Page_Init(object sender, EventArgs e)
{
      //
```

```
      // CODEGEN: This call is required by the ASP.NET Web Form Designer.
      //
      InitializeComponent();
}

public SVGGenerator()
{
      //
      // Initializes all event handlers
      //
      Page.Init += new System.EventHandler(Page_Init);
}

//*****************************************************************
//
// The Page_Load event is currently not being used.
//
//*****************************************************************
private void Page_Load(object sender, System.EventArgs e)
{
}

//*****************************************************************
//
// The XslTransformSrc_SelectedIndexChanged and GenerateBtn_Click
// event handlers on this Page are used to generate a new SVG
//  chart based on user settings.
//
//*****************************************************************
private void XslTransformSrc_SelectedIndexChanged(object sender, System.EventArgs e)
{
      String[] args1 = { Server.MapPath( "data/wobegon.xml" ), Server.MapPath(
"style/" + XslTransformSrc.SelectedItem.Value + ".xslt" ) };
      if ( XslTransformSrc.SelectedItem.Value != "Start" )
      {
            GenerateSVG(args1);
      }
      else
      {
            //Rewrite the 'iframe' and 'embed' tags.
            embedSvg.InnerHtml = "<iframe src='svg/Start.svg' frameBorder='0'
width='500' height='300'>"
                  + "<embed src='svg/Start.svg' name='SVGChart' width='500px'
height='300px' type='image/svg+xml'
pluginspage='http://www.adobe.com/svg/viewer/install/' />"
                  + "</iframe>";
      }
}

private void DropdownYear_SelectedIndexChanged(object sender, System.EventArgs e)
{
      String[] args1 = { Server.MapPath( "data/wobegon.xml" ), Server.MapPath(
"style/" + XslTransformSrc.SelectedItem.Value + ".xslt" ) };
      if ( XslTransformSrc.SelectedItem.Value != "Start" )
      {
            GenerateSVG(args1);
      }
}

void generateButton_Click(object sender, System.EventArgs e)
```

```
{
      String[] args1 = { Server.MapPath( "data/wobegon.xml" ), Server.MapPath(
"style/" + XslTransformSrc.SelectedItem.Value + ".xslt" ) };
      if ( XslTransformSrc.SelectedItem.Value != "Start" )
      {
            GenerateSVG(args1);
      }
      else
      {
            //Rewrite the 'iframe' and 'embed' tags.
            embedSvg.InnerHtml = "<iframe src='svg/Start.svg' frameBorder='0'
width='500' height='300'>"
                  + "<embed src='svg/Start.svg' name='SVGChart' width='500px'
height='300px' type='image/svg+xml'
pluginspage='http://www.adobe.com/svg/viewer/install/' />"
                  + "</iframe>";
      }
}

//*************************************************************
//
// The viewXml, viewXslt and viewSvgSource event handlers on this
// Page read and output the source of the XML, XSLT and SVG files.
//
//*************************************************************
private void viewXml_Click(object sender, System.EventArgs e)
{
      output1.Text = "XML source document (wobegon.xml):";

      StringWriter writer = new StringWriter();
      Console.SetOut(writer);

      String[] args1 = { Server.MapPath( "data/wobegon.xml" ) };
      ReadXML(args1);

      output2.InnerHtml = writer.ToString();

      Console.WriteLine();
}

private void viewXslt_Click(object sender, System.EventArgs e)
{
      output1.Text = "XSLT source document (" + XslTransformSrc.SelectedItem.Value +
".xslt):";
      StringWriter writer = new StringWriter();
      Console.SetOut(writer);
      String[] args1 = { Server.MapPath( "style/" +
XslTransformSrc.SelectedItem.Value + ".xslt" ) };
      ReadXML(args1);
      output2.InnerHtml = writer.ToString();
      Console.WriteLine();
}

private void viewSvgSource_Click(object sender, System.EventArgs e)
{
      output1.Text = "XSLT transformation output source (SVG!):";
      StringWriter writer = new StringWriter();
      Console.SetOut(writer);
      String[] args1 = { Server.MapPath( "data/wobegon.xml" ), Server.MapPath(
"style/" + XslTransformSrc.SelectedItem.Value + ".xslt" ) };
```

```
      ReadTransformation(args1);     // ReadTransformation() currently does not use
'args1' parameter.
      output2.InnerHtml = writer.ToString();
      Console.WriteLine();
}

//*************************************************************
//
// The GenerateSVG function is used to generate a new SVG chart
// based on the users settings.
//
//*************************************************************
public void GenerateSVG(String[] args)
{
      //Generate random integer for new Svg chart name.
        //Create the randomGenerator.
      Random randomGenerator = new Random(DateTime.Now.Millisecond);
      //Generate random number that's between 0 and 1 trillion ;D
      long randomNum = randomGenerator.Next(0, 1000000000);

      //Write the new SVG file name so that we can view the new SVG source.
      svgFileName.InnerText = "chart" + randomNum + ".svg";

      //Rewrite the 'iframe' and 'embed' tags.
      embedSvg.InnerHtml = "<iframe src='svg/chart" + randomNum + ".svg'
frameBorder='0' width='" + Width.Text + "' height='" + Height.Text + "'>"
            + "<embed src='svg/chart" + randomNum + ".svg' name='SVGChart' width='"
+ Width.Text + "px' height='" + Height.Text + "px' type='image/svg+xml'
pluginspage='http://www.adobe.com/svg/viewer/install/' />"
            + "</iframe>";

      //Create and load the XmlTextWriter.
      XmlTextWriter writer = new XmlTextWriter( Server.MapPath( "svg/chart" +
randomNum + ".svg" ), null );

      //Create and load the XPathDocument and XslTransform.
      XPathDocument myXPathDocument = new XPathDocument ( args[0] );
      XslTransform myXslTransform = new XslTransform();
      myXslTransform.Load( args[1] );

      //Create the XsltArgumentList.
      XsltArgumentList xslArg = new XsltArgumentList();

      //Create a parameter which represents the current date and time.
      xslArg.AddParam( "width", "", Width.Text );
      xslArg.AddParam( "height", "", Height.Text );
      xslArg.AddParam( "year", "", DropdownYear.SelectedItem.Value );

      //Transform the file.
      myXslTransform.Transform( myXPathDocument, xslArg, writer );

      writer.Close();

      //Clear all the document source content.
      output1.Text = "";
      output2.InnerHtml = "";
}

//*************************************************************
//
```

```
// The GenerateSVG function is used to generate a new SVG chart
// based on the users settings.
//
//************************************************************
public void ReadXML(String[] args)
{

      StreamReader stream = null;
      try   {
            stream = new StreamReader (args[0]);
            Console.Write(stream.ReadToEnd());   }
      catch (Exception e)      {
            Console.WriteLine ("Exception: {0}", e.ToString());     }
      finally     {
            if (stream != null)
                  stream.Close();    }
}

//************************************************************
//
// The GenerateSVG function is used to generate a new SVG chart
// based on the users settings.
//
//************************************************************
public void ReadTransformation(String[] args)
{
      StreamReader stream = null;
      try   {
            ///Read svg source.  File name comes from 'svgFileName'.
            stream = new StreamReader ( Server.MapPath( "svg/" +
svgFileName.InnerText ) );
            Console.Write(stream.ReadToEnd());   }

      catch (Exception e)      {
            Console.WriteLine ("Exception: {0}", e.ToString());     }

      finally     {
            if (stream != null)
                  stream.Close();    }

}

}
#endregion
}
```

Quelques passages de ce code en détail.

Nous référençons les autres variables globales:

```
using System.Web.UI.WebControls;
using System.Web.UI.HtmlControls;
using System.IO;
using System.Xml;
using System.Xml.Xsl;
using System.Xml.XPath;
```

Enregistrement des contrôles ASP comme ceci.

```
protected System.Web.UI.WebControls.Button generateButton;
```

```
protected System.Web.UI.WebControls.DropDownList XslTransformSrc;
protected System.Web.UI.WebControls.DropDownList DropdownYear;
```

La référence ‘IO’ pour pouvoir lire et écrire dans les fichiers. Les références ‘Xsl’ et ‘XPath’ pour la transformation XSLT comme nous le verrons.

MSDN Library fournit une abondante documentation sur l'usage de l'interpréteur MSXML pour manipuler le DOM de documents XML.

La dernière étape est d'utiliser MSXML dans notre application C# pour faire la transformation et renvoyer le fichier au client. La transformation XML->SVG avec XSLT peut être initiée par trois commandes:

```
void generateButton_Click(object sender, System.EventArgs e)
{
      String[] args = { Server.MapPath( "data/wobegon.xml" ), Server.MapPath(
"style/" + XslTransformSrc.SelectedItem.Value + ".xslt" ) };

      if ( XslTransformSrc.SelectedItem.Value != "Start" )
      {
            GenerateSVG(args);
      }
}

private void XslTransformSrc_SelectedIndexChanged(object sender, System.EventArgs e)
{
      …function code here…
}

private void DropdownYear_SelectedIndexChanged(object sender, System.EventArgs e)
{
      …function code here…
}
```

L'événement ‘generateButton_Click()’ est le résultat d'un click sur le bouton ‘Go’. L'événement ‘XslTransformSrc_SelectedIndexChanged()’ survient quand l'utilisateur choisit un type de graphique. Le dernier, ‘DropdownYear_SelectedIndexChanged()’ survient quand l'utilisateur change d'année pour les données.

Données serveur et transformation XSLT

Chacun de ces événements appelle la fonction GenerateSVG() si le type de graphique n'est pas ‘Start’. Cette fonction GenerateSVG() coordonne le chargement, la manipulation et l'enregistrement de nos documents XML et XSLT comme documents SVG.

Quand l'utilisateur choisit un autre type de graphique, son nom est passé comme second argument à la fonction GenerateSVG(). Les noms de ces types de graphiques correspondent aux noms des feuilles de style dans le répertoire “/style” du projet.

```
public void GenerateSVG(String[] args)
{

//Generate random integer for new Svg chart name.
//Create the randomGenerator.
Random randomGenerator = new Random(DateTime.Now.Millisecond);
//Generate random number that's between 0 and 1 trillion ;D
```

```
long randomNum = randomGenerator.Next(0, 1000000000);

//Create and load the XmlTextWriter.
XmlTextWriter writer = new XmlTextWriter( Server.MapPath( "svg/chart" + randomNum +
".svg" ), null );

//Create and load the XPathDocument and XslTransform.
XPathDocument myXPathDocument = new XPathDocument ( args[0] );
XslTransform myXslTransform = new XslTransform();
myXslTransform.Load( args[1] );

//Create the XsltArgumentList.
XsltArgumentList xslArg = new XsltArgumentList();

//Create a parameter which represents the current date and time.
xslArg.AddParam( "width", "", Width.Text );
xslArg.AddParam( "height", "", Height.Text );
xslArg.AddParam( "year", "", DropdownYear.SelectedItem.Value );

//Transform the file.
myXslTransform.Transform( myXPathDocument, xslArg, writer );

writer.Close();
}
```

Comme toujours, les tableaux commencent avec l'indice 0 et le deuxième argument est référencé comme ‘args[1]’.

Les données XML sont chargées dans myXPathDocument:

```
XPathDocument myXPathDocument = new XPathDocument ( args[0] );
```

La feuille de style est chargée dans l'objet XslTransform() nommé ‘myXslTransform’.

```
XslTransform myXslTransform = new XslTransform();
myXslTransform.Load( args[1] );
```

Dans chaque feuille de style XSLT il y a trois paramètres en tête de document - ‘width’, ‘height’, et ‘year’.

```
<xsl:param name="width" select="'500'"/>
<xsl:param name="height" select="'500'"/>
<xsl:param name="year" select="'2002'"/>
```

Ces trois paramètres ‘Width’, ‘Height’ et ‘Year’ sont des choix de l'utilisateur, ces valeurs sont actualisées dans le code C# . Le code C# lit le DOM des documents XSLT, trouve les paramètres et les actualisent avec les valeurs choisies par l'utilisateur.

Voici le code qui fait ce travail:

```
//Create the XsltArgumentList.
XsltArgumentList xslArg = new XsltArgumentList();

//Update the XSLT stylesheet parameters.
xslArg.AddParam( "width", "", Width.Text );
xslArg.AddParam( "height", "", Height.Text );
xslArg.AddParam( "year", "", DropdownYear.SelectedItem.Value );
```

```
//Transform the file.
myXslTransform.Transform( myXPathDocument, xslArg, writer );
```

Enregistrer le SVG dans un fichier

Après l'application de la feuille de style, le dernier travail est de passer le document à l'objet 'writer'. Voici le code pour ce processus.

```
//Generate random integer for new Svg chart name.
//Create the randomGenerator.
Random randomGenerator = new Random(DateTime.Now.Millisecond);
//Generate random number that's between 0 and 1 trillion ;D
long randomNum = randomGenerator.Next(0, 1000000000);

//Create and load the XmlTextWriter.
XmlTextWriter writer = new XmlTextWriter( Server.MapPath( "svg/chart" + randomNum +
".svg" ), null );

//Transform the file.
myXslTransform.Transform( myXPathDocument, xslArg, writer );
```

Si tout se passe bien, le fichier est créé dans le répertoire 'svg'.

Permissions pour le serveur

Nous devons donner des droits d'écriture pour le répertoire, avec Windows 2000 vous aurez cette boîte de dialogue:

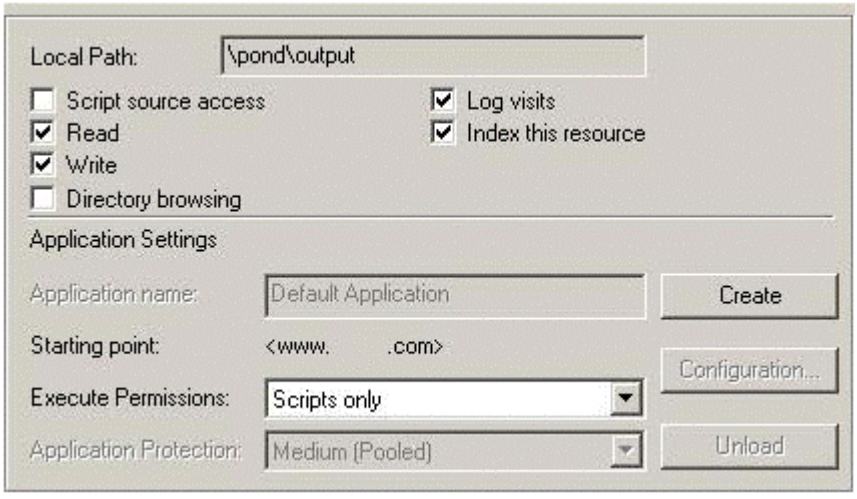

Figure 12-3. Configuration du serveur

Quand vous choisissez par exemple 'line' comme type de graphique et '2002' pour l'année, la transformation XSLT ajoute deux lignes brisées à notre dessin. Voici cet exemple.

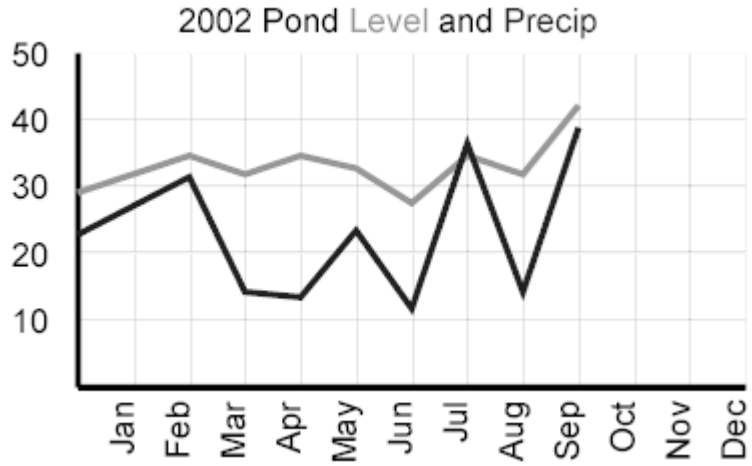

Figure 12-4. Graphique SVG avec 'line'

Voici ces deux éléments 'path' ajoutés.

```
<svg width="500" height="300" viewBox="0 0 110 90" xml:space="preserve">
        …SVG chart…
        <g id="ChartData">
                <path id="pondLevelData"
                        stroke="darkorange" fill="none" d=" M  29.2 0 L 34.7 16.66
L 31.9 24.990000000000002 L 34.7 33.32 L 32.8 41.65 L 27.6 49.980000000000004 L
34.7 58.31 L 31.9 66.64 L 42.2 74.97"></path>
                <path id="pondPrecipData"
                        stroke="blue" fill="none" d=" M  22.9 0 L 31.5 16.66 L
14.3 24.990000000000002 L 13.5 33.32 L 23.5 41.65 L 11.8 49.980000000000004 L
36.5 58.31 L 14.3 66.64 L 38.9 74.97"></path>
        </g>
</svg>
```

Ces éléments‘path’ représentent nos données pour le niveau de l'eau et les précipitations.

Nous avons couvert la transformation XML -> SVG, voyons comment utiliser des données en temps réel.

Données en temps réel

Pour exploiter ces données dynamiquement et actualiser un graphique, nous pouvons utiliser ECMAScript pour manipuler les objets du DOM SVG.

Nous pouvons créer l'image de base côté serveur et la manipuler côté client.

Comme nous l'avons vu dans ce livre ECMAScript a accès au SVG DOM depuis le fichier SVG ou depuis le fichier HTML si le SVG est inséré dans une page HTML.

Voyons notre fond d'image, l'image sans les données.

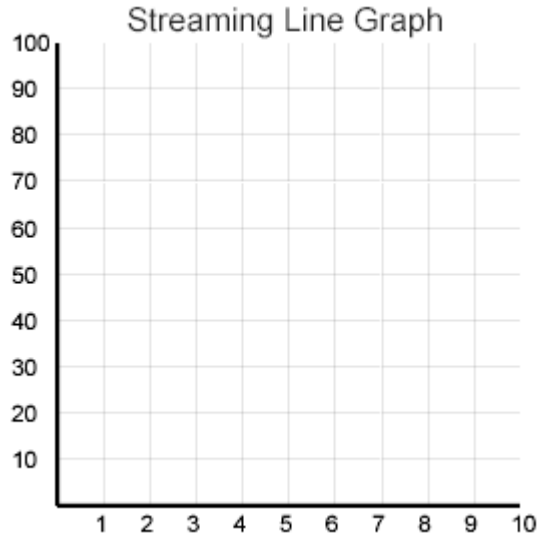

Figure 12-5. Fond d'image pour notre exemple

```
<svg width="500" height="300" viewBox="0 0 90 130"
        xml:space="preserve">
        <g id="Chart" font-size="5"
                transform="translate( 10, 110 ) rotate( -90 )">
        <defs>
        <pattern id="gridPattern" width="10" height="10"
                patternUnits="userSpaceOnUse">
        <rect x="0" y="0" width="10" height="10"
                style="fill:none;stroke:rgb(128,128,128);stroke-width:.1" />
        </pattern>
        </defs>

        <text x="105" y="15" writing-mode="tb" font-size="7">
                <tspan fill="darkred">Streaming Line Graph</tspan>
        </text>
        <text x="-4" writing-mode="tb" y="8">1</text>
        <text x="-4" writing-mode="tb" y="18">2</text>
        <text x="-4" writing-mode="tb" y="28">3</text>
        <text x="-4" writing-mode="tb" y="38">4</text>
        <text x="-4" writing-mode="tb" y="48">5</text>
        <text x="-4" writing-mode="tb" y="58">6</text>
        <text x="-4" writing-mode="tb" y="68">7</text>
        <text x="-4" writing-mode="tb" y="78">8</text>
        <text x="-4" writing-mode="tb" y="88">9</text>
        <text x="-4" writing-mode="tb" y="98">10</text>
        <text x="100" y="-10" writing-mode="tb">100</text>
        <text x="90" y="-10" writing-mode="tb">90</text>
        <text x="80" y="-10" writing-mode="tb">80</text>
        <text x="70" y="-10" writing-mode="tb">70</text>
        <text x="60" y="-10" writing-mode="tb">60</text>
        <text x="50" y="-10" writing-mode="tb">50</text>
        <text x="40" y="-10" writing-mode="tb">40</text>
        <text x="30" y="-10" writing-mode="tb">30</text>
        <text x="20" y="-10" writing-mode="tb">20</text>
        <text x="10" y="-10" writing-mode="tb">10</text>

        <rect x="0" y="0" width="100" height="100"
                fill="url(#gridPattern)" />
        <line x1="0" y1="0" x2="0" y2="100" style="stroke:black;" />
        <line x1="0" y1="0" x2="100" y2="0" style="stroke:black;" />
        <g id="grid"></g>
                <text id="CurrentTime" x="50" y="10"
                        style="fill:black;font-size:20;opacity:0.1"
                        writing-mode="tb">time</text>
        </g>
</svg>
```

Nous avons besoin d'un script pour récupérer les données et actualiser l'image.

```
<svg width="500" height="300" viewBox="0 0 90 130"
        xml:space="preserve"
        onload="Init(evt); ">
<script>
<!--
var SVGDoc=null;
var count=0;

function AddData(x,y) {
        var Path=SVGDoc.getElementById('pathData');
        Path.setAttribute( 'd', Path.getAttribute('d')
                + ' L ' + x + ' ' + y );
        return this
}

function updateData( obj ) {
        AddData( ++count, +obj.content )
        if ( count < 100 )
                setTimeout( 'getData()', 1000 );
}

function getData() {
        getURL( '../randomData.aspx', updateData )
}

function Init( evt ) {
        SVGDoc = evt.target.ownerDocument;
        var iW = SVGDoc.documentElement.getAttribute( 'width' );
        var iH = SVGDoc.documentElement.getAttribute( 'height' );
        setTimeout( 'getData()', 1000 )
}

-->
</script>
…  objets SVG  …
</svg>
```

Nous retrouvons la fonction setTimeout qui appelle la fonction getData() toutes les 1000 millisecondes (à chaque seconde donc).

La fonction essentielle est getURL() appelée depuis la fonctiont getData().

```
getURL( '../randomData.aspx', updateData )
```

Cette fonction getURL() est la seule qui nous permette d'actualiser notre SVG depuis une source externe sans avoir à recharger le document. Il n'y a pas de limite de taille pour les données à transférer. Voici la syntaxe de cette fonction:

```
getURL( url, callback)
```

L'argument ‘url’ donne la localisation de la source pour les données. Cette source doit être sur le même serveur que le SVG.

L'argument ‘callback’ est le nom de la fonction qui va traiter les données récupérées.

La page ‘.aspx’ source des données, appelée par getURL() est nommée ‘randomData.aspx’, est localisée dans le répertoire racine de notre application, voici son code.

```
<scrispt language="C#" runat="server">
```

```
void Page_Load(Object Src, EventArgs E) {

      //Create the randomGenerator.
      Random randomGenerator = new Random(DateTime.Now.Millisecond);

      //Generate random number.
      float randomNum = randomGenerator.Next(0, 11);

      //Return random number to client.
      Response.Write ( randomNum * 10 );
}
</script>
```

Quand ce fichier est appelé, il retourne un nombre aléatoire entre 0 et 100 au client. Ce nombre est envoyé à la fonction AddData() pour la suite du processus.

Détaillons cette fonction AddData().

```
var Path=SVGDoc.getElementById('pathData');
Path.setAttribute( 'd', Path.getAttribute('d')
    + ' L ' + x + ' ' + y );
```

Ce code actualise le SVG DOM avec la nouvelle valeur. Nous accédons au noeud 'pathData' que nous mettons dans la variable 'Path'. Nous ajoutons un nouveau 'lineto' aux commandes de tracé de l'attribut 'd', un segment est tracé du point précédent à ce nouveau point.

Voici une illustration du graphique obtenu.

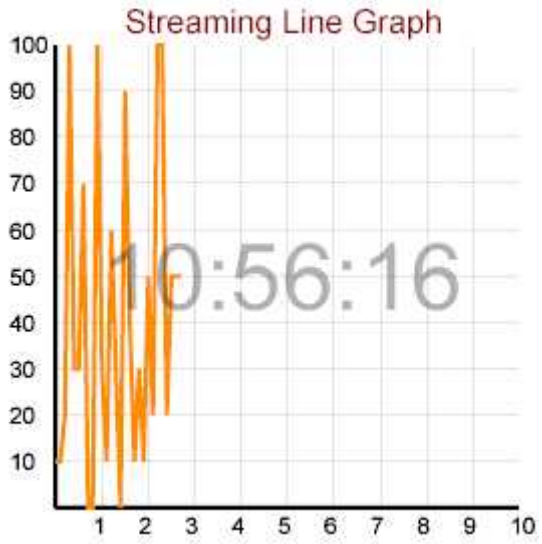

Figure 12-6. Graphe de données en temps réel

Pour cet exemple simple, vous pouvez penser que tout ceci aurait pû se faire dans le script du document SVG sans faire appel à un générateur aléatoire sur le serveur. Mais nous voulions un exemple plus général qui puisse être utilisé pour des données véritables issues de phénomènes concrets.

La fonction getURL() n'est pas encore une extension standard et n'est pas supportée par tous les visualiseurs, mais nous pouvons aussi utiliser XMLHTTP pour récupérer des données externes.

Compatibilité avec les navigateurs

Notre application fonctionne avec IE 5.x et +, Netscape 4.x, et Netscape 6.2, Mozilla 1.0. La plupart des documents SVG obtenus peuvent être vus dans Batik.

Quelques améliorations

Cette application que nous avons examinée pourrait être améliorée avec une gestion des erreurs de connection, la structure des fonctions, de nouveaux graphes avec les feuilles de style XSLT, des données plus nombreuses et variées, une légende pour les graphiques, des icônes pour le zoom …

Publier SVG sur un site Web

Pour un site Web, nous devons prendre en considération l'optimisation des fichiers, l'installation de scripts pour vérifier la présence du plugin et lancer son installation si nécessaire. Une autre considération peut être de sécuriser le plus possible le contenu SVG. Voyons brièvement ces aspects.

Optimisation de la taille des fichiers

La méthode principale pour réduire la taille des fichiers est la compression. HTTP/1.1 permet de passer du serveur au client des données compressées avec le format GZIP. Cette compression est efficace pour les fichiers SVG, voici une table prise dans les spécifications:

SVG Initial	Compression avec gzip	Taux de Compression
12,912	2,463	81%
12,164	2,553	79%
11,613	2,617	77%
18,689	4,077	78%
13,024	2,041	84%

Nous pouvons également en utilisant CSS comme nous l'avons vu au chapitre 4 optimiser nos fichiers. Illustrator 9/10 peut simplifier la géométrie des objets et supprimer des données inutiles pour réduire la taille du fichier.

Pour plus d'information sur ce sujet, http://www.w3.org/TR/SVG/minimize.html et http://www.adobe.com/svg/workflow/optimizing.html peuvent vous aider.

Sécuriser le contenu des documents SVG

Actuellement, il n'est pas possible de sécuriser SVG. Le fait que SVG soit texte ne facilite pas les choses. D'autre part ce problème n'est pas spécifique à SVG et les outils pour déplomber n'importe quel contenu abondent.

Désactiver “Voir Source” dans 'Adobe SVG Viewer'

L'option 'Voir Source' peut être désactivée dans la version 3 du plugin d'Adobe de deux manières. Vous pouvez ajouter la balise XML au début de votre document:

```
<?AdobeSVGViewer save="disable"?>
```

Vous pouvez également avec un script localiser “contextMenu” dans le DOM XML et supprimer ou rendre inaccessibles certaines options en donnant à la propriété “enabled” la valeur “no”. Vous pouvez également personnaliser ce menu contextuel et même l'adapter à l'objet sur lequel le pointeur de l'utilisateur est positionné.

Ces deux méthodes peuvent peut-être décourager l'utilisateur mais il y a bien d'autres méthodes pour voir le fichier source du SVG.

Ce point a provoqué beaucoup de débats sur les droits d'auteur, il est effectivement très facile de copier et réutiliser un logo, une animation, un script. Mais ce débat est sans fin et rejoint les autres sur les contenus musicaux et vidéo sur le Web. Quelques mesures pourraient cependant protéger la propriété intellectuelle.

Les lois dans ce domaine ont changé et continueront à évoluer pour traiter tous les contenus numériques, y compris SVG. Nous pouvons cependant mettre un commentaire clair en tête de nos documents pour expliquer notre position quant à la réutilisation du contenu de ces pages.

Sommaire

La création de documents SVG côté serveur nous ouvre de nouvelles possibilités. Avec notre exemple, nous avons créé une application .Net utilisant XHTML + SVG + ECMAScript pour transformer des données XML en graphiques SVG grâce à XSLT et C#.

Publier SVG sur le Web est simple, mais l'attente principale est l'implémentation complète de SVG dans les applications Web.

Partie II: La beauté des fractales

Les fractales peuvent être créées de manière élégante avec SVG et elles n'en sont que plus belles.

Les fractales de type ***"Iterated Function Systems"*** (IFS) sont un type qui se prête bien à SVG car elles sont définies vectoriellement. La théorie est relativement simple, nous appliquons de manière récursive un ensemble de transformations affines à une forme de départ

Nous avons vu un exemple au chapitre 6 avec le triangle de Sierpinski. Nous allons détailler la méthode pour un autre exemple, le flocon de neige de Von Koch.

Le flocon de Von Koch

Nous partons d'un segment.
Nous appliquons un groupe de quatre transformations:
- une homothétie de rapport 1/3
- l'homothétie suivie d'une rotation de –60 degrés et une translation pour amener ce segment à la suite du précédent
- l'homothétie suivie d'une rotation de 60 degrés et une translation pour amener ce segment à la suite du précédent
- l'homothétie suivie d'une translation pour amener ce segment à la suite du précédent

.

Figure 12-7. Von Koch étapes 0 et 1

```
<svg width="650" height="120">
   <defs>
     <path id="level_0" fill="none" stroke="black" d="M0 0h300" />
     <g id="level_1">
         <use xlink:href="#level_0" transform="scale(0.333)" />
         <use xlink:href="#level_0" transform="translate(100 0) rotate(-60)
scale(0.333)"/>
         <use xlink:href="#level_0" transform="translate(150 -86.6) rotate(60)
scale(0.333)"/>
         <use xlink:href="#level_0" transform="translate(200 0) scale(0.333)"/>
     </g>
   </defs>
   <use xlink:href="#level_0" stroke-width="1" transform="translate(20,100)"/>
   <use xlink:href="#level_1" stroke-width="3" transform="translate(340,100)"/>
</svg>
```

Une petite remarque, vous savez sans doute que dans la transformation 'scale', la largeur du tracé est également affectée, aussi nous modifions 'stroke-width' pour garder la même épaisseur. Nous réitérons cette opération et nous avons la courbe de *Von Koch.*

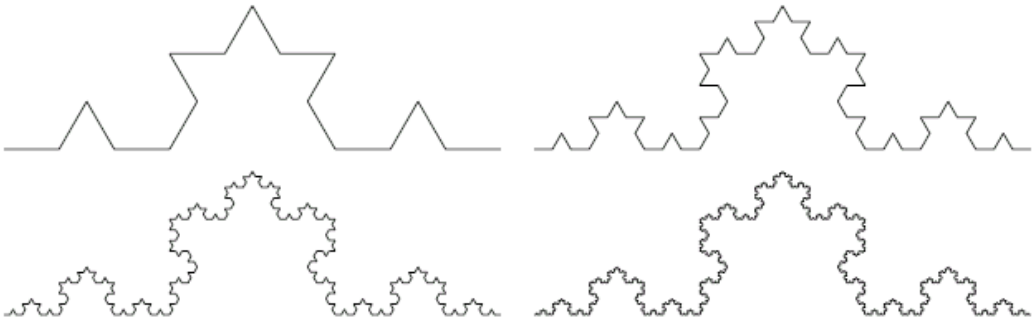

Figure 12-8. Etapes 2 3 4 et 5 de la courbe de Von Koch

Voici le code pour cette figure:

```
<svg width="650" height="220">
   <defs>
     <path id="level_0" fill="none" stroke="black" d="M0 0h300" />
     <g id="level_1">
         <use xlink:href="#level_0" transform="scale(0.333)" />
         <use xlink:href="#level_0" transform="translate(100 0) rotate(-60) scale(0.333)"
/>
         <use xlink:href="#level_0" transform="translate(150 -86.6) rotate(60)
scale(0.333)" />
         <use xlink:href="#level_0" transform="translate(200 0) scale(0.333)" />
     </g>
     <g id="level_2">
         <use xlink:href="#level_1" transform="scale(0.333)" />
         <use xlink:href="#level_1" transform="translate(100 0) rotate(-60) scale(0.333)"
/>
         <use xlink:href="#level_1" transform="translate(150 -86.6) rotate(60)
scale(0.333)" />
         <use xlink:href="#level_1" transform="translate(200 0) scale(0.333)" />
     </g>
     <g id="level_3">
```

```
         <use xlink:href="#level_2" transform="scale(0.333)" />
         <use xlink:href="#level_2" transform="translate(100 0) rotate(-60) scale(0.333)"
/>
         <use xlink:href="#level_2" transform="translate(150 -86.6) rotate(60)
scale(0.333)" />
         <use xlink:href="#level_2" transform="translate(200 0) scale(0.333)" />
     </g>
     <g id="level_4">
         <use xlink:href="#level_3" transform="scale(0.333)" />
         <use xlink:href="#level_3" transform="translate(100 0) rotate(-60) scale(0.333)"
/>
         <use xlink:href="#level_3" transform="translate(150 -86.6) rotate(60)
scale(0.333)" />
         <use xlink:href="#level_3" transform="translate(200 0) scale(0.333)" />
     </g>
     <g id="level_5">
         <use xlink:href="#level_4" transform="scale(0.333)" />
         <use xlink:href="#level_4" transform="translate(100 0) rotate(-60) scale(0.333)"
/>
         <use xlink:href="#level_4" transform="translate(150 -86.6) rotate(60)
scale(0.333)" />
         <use xlink:href="#level_4" transform="translate(200 0) scale(0.333)" />
     </g>
   </defs>
   <use xlink:href="#level_2" stroke-width="9" transform="translate(20,100)" />
   <use xlink:href="#level_3" stroke-width="27" transform="translate(340,100)" />
   <use xlink:href="#level_4" stroke-width="81" transform="translate(20,200)" />
   <use xlink:href="#level_5" stroke-width="243" transform="translate(340,200)" />
</svg>
```

Nous constatons au passage que les groupes et les éléments <use> supportent la récursivité.

Et le flocon de neige?

Nous utilisons trois lignes de niveau 5 pour former un triangle.

Avec la même section <defs> que dans notre code précédent, nous écrivons:

```
   <use xlink:href="#level_5" stroke-width="243" transform="translate(20,100)" />
   <use xlink:href="#level_5" stroke-width="243" transform="translate(320,100)
rotate(120)" />
   <use xlink:href="#level_5" stroke-width="243" transform="translate(170,360) rotate(-
120)" />
```

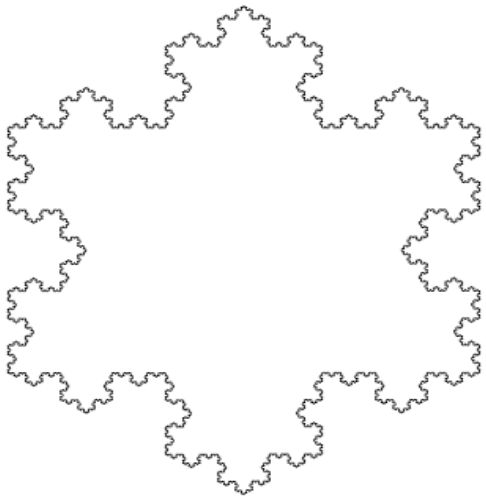

Figure 12-9. Le flocon de neige de Von Koch

Vous pouvez zoomer, augmenter le niveau de récursivité en construisant les groupes suivants, mettre quelques couleurs … Vous pouvez surtout utiliser d'autres transformations pour obtenir toute une série de fractales.
Il n'est pas difficile de créer ce code en utilisant un script exécuté au chargement du document.

Générateur d'IFS

Pour explorer ces IFS et voir le résultat sans codage, j'ai créé un outil où l'utilisateur définit les transformations à appliquer et ses formes de départ comme dans un outil de dessin, simplement à la souris.

Les transformations, affines pour que le résultat soit intéressant, sont représentées par une matrice. Dans ces transformations, un carré a pour image au minimum un parallélogramme. Si nous n'avons que des rotations, translations et homothéties ('scale' avec les deux paramètres égaux), le carré reste carré.

L'utilisateur définit donc une transformation par le dessin d'un parallélogramme à la souris, celui-ci étant l'image dans la transformation du carré quadrillé, espace de dessin. Il place trois points, le quatrième est calculé. L'ordre des points est important, il change la transformation. L'utilisateur peut déplacer ces points pour changer de transformation.

Sur cet exemple, nous voyons les carrés qui définissent les quatre transformations de la courbe de Von Koch, combinaisons de rotation, homothétie et translation. Nous définissons la forme de départ, ici un simple segment dessiné ici en noir. Nous choisissons un niveau de récursivité et nous obtenons la fractale dans la partie droite de l'espace de travail.

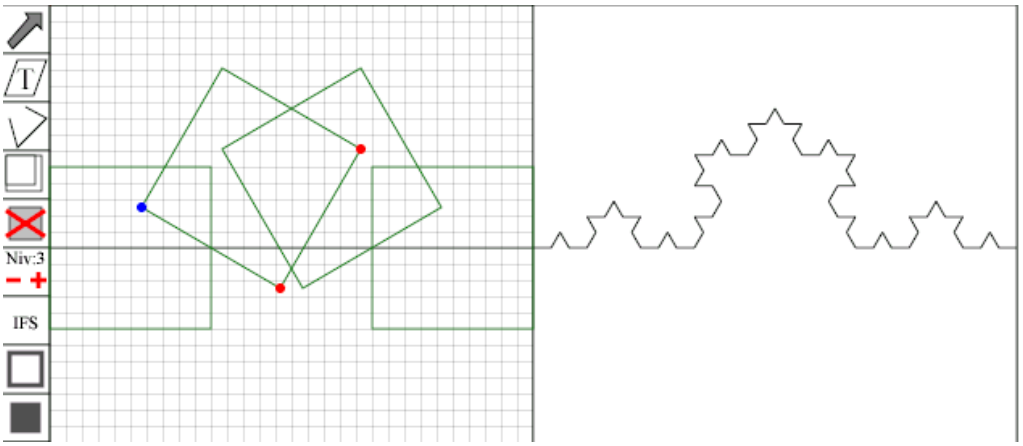

Figure 12-10. Générateur d'IFS et courbe de Von Koch

Pour cet exemple, la forme de niveau 0 n'a pas beaucoup d'importance, voyons avec un triangle:

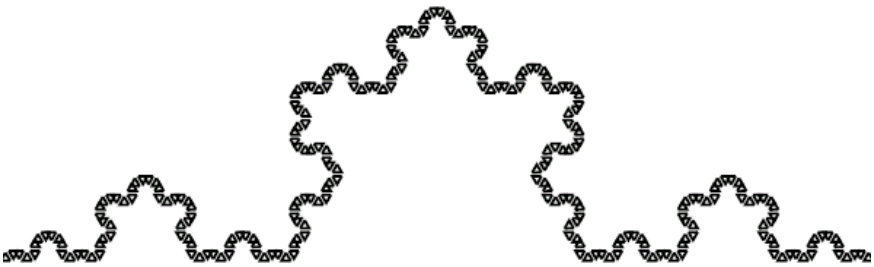

Figure 12-11. Un triangle comme forme de base

Quelques exemples créés avec ce générateur:

Tapis de Sierpinski

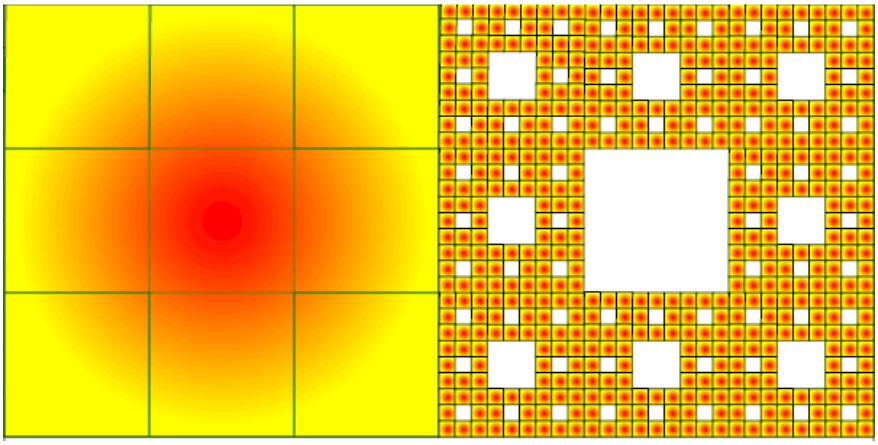

Figure 12-12. Tapis de Sierpinski

Avec ce générateur, nous pouvons créer et utiliser des gradients pour la forme de départ. Cette fractale nécessite huit transformations représentées par les carrés verts. Seul le carré central n'est pas une transformation. Nous obtenons le tapis de Sierpinski, nous pouvons faire un travail similaire en 3D pour obtenir l'éponde de Menger.

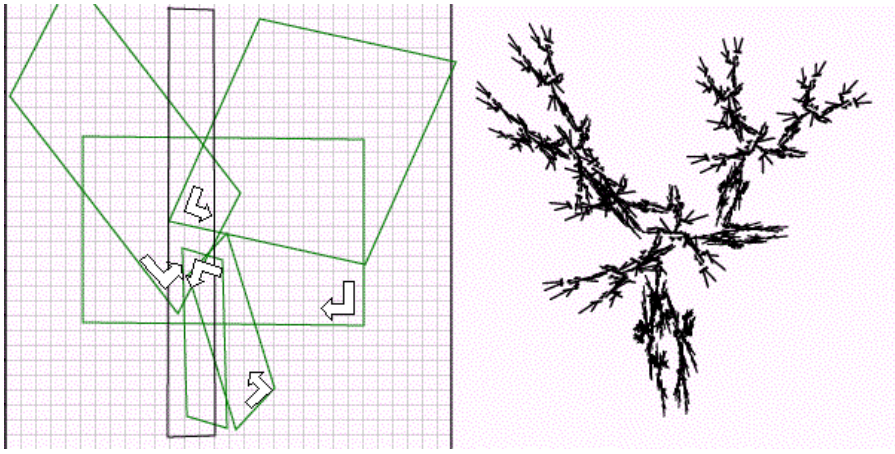

Figure 12-13. Arbre tourmenté

Pour ce malheureux arbre, nous utilisons cinq transformations et l'objet de départ est un rectangle.

Fougère de Barnsley

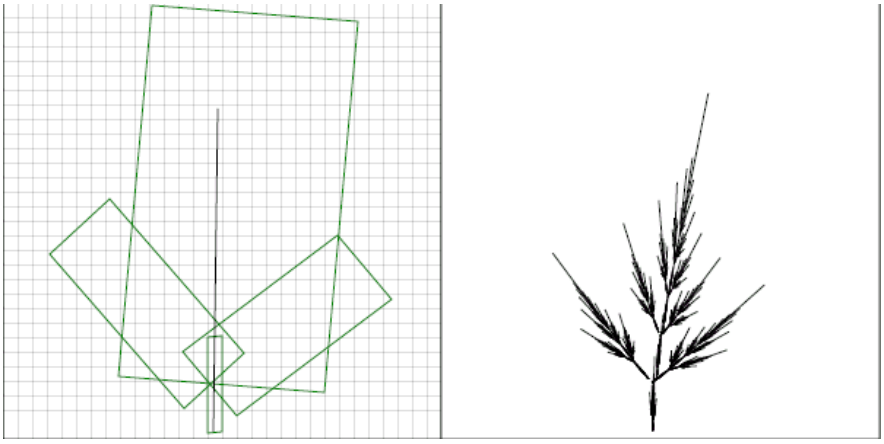

Figure 12-14. Un essai de fougère

Ici, nous n'avons besoin que de quatre transformations.

Autres exemples

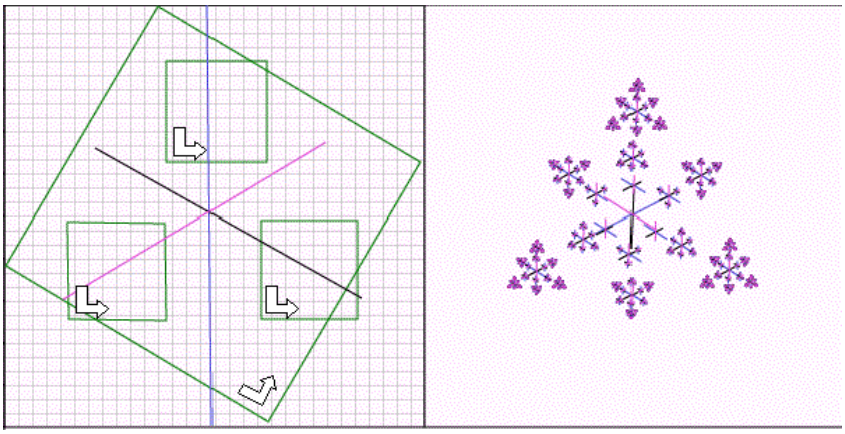

Figure 12-15. Cristal hexagonal

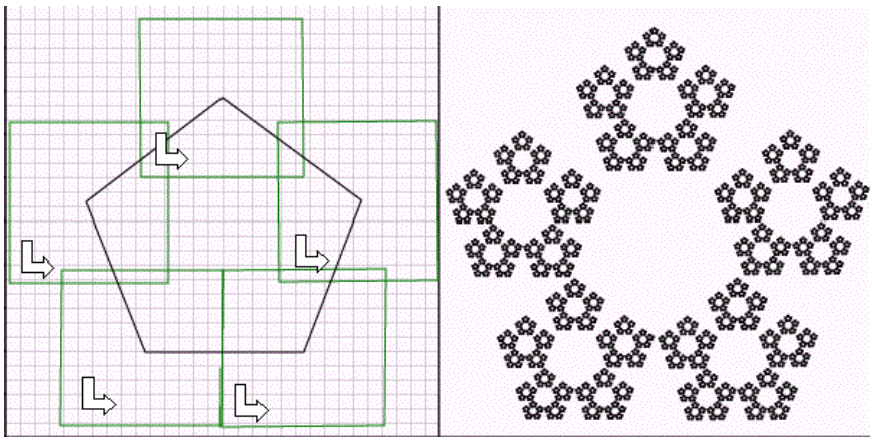

Figure 12-16. A partir d'un pentagone

Nous terminons ce livre par ces fractales. Il est intéressant de noter que notre générateur d'IFS est un exemple convaincant de l'utilisation de SVG dans des applications diverses. SVG n'est pas seulement un langage pour la création de pages Web, il donne à ECMAScript une sortie graphique vectorielle. Le couple SVG+ECMAScript permet la création de tout type d'application.