CS297 Report

Stylesheet Translations of SVG to VML

Julie Nabong
(jnabong2001@yahoo.com)

Advisor: Dr. Chris Pollett

December 2003

Abstract

Scalable Vector Graphics (SVG) is an XML-based language useful for producing two-dimensional graphics such as shapes, lines, and text. It was created by the World-Wide Web Consortium. Currently, SVG can be viewed in web-browsers using a plug-in such as the Adobe SVG plug-in. An earlier language for creating vector graphics on the web called, Vector Markup Language, was developed by Microsoft. Vector Markup Language (VML), just like SVG, is an XML-based language to generate two-dimensional graphics. It can be viewed in Microsoft IE version 5.0 and higher. Although SVG is generally considered a superior format as the base of browser technology is slowing, it does not look like it will be natively supported by Microsoft anytime soon. The goal of this master's project is to develop stylesheets and JavaScript technology to translate SVG to VML within Internet Explorer in as transparent a manner as possible. There are several challenging aspects to this project which will be explored. One is that VML does not fully support some of the project features of SVG such as gradients. The second difficulty is to make SVG's JavaScript document object model to work seamlessly from the VML setting. A last issue is to try to get the speed of the translation as close to that of the plug-in based solution as possible.

Table of Contents

I. Introduction

Some of the most popular image formats available on the Web today are JPEG and GIF. In addition to those image formats, web developers can also use SVG and VML, which are vector graphic formats. Vector graphic formats present some benefits over JPEG and GIF. One advantage is that vector graphic files download faster on the Web because of their smaller size. Another benefit is that vector graphic images can be enlarged without loss of quality, as explained by Watt, et al.

Two open standards are available for the creation of vector graphics for the Web: SVG and VML. Scalable Vector Graphics (SVG) and Vector Markup Language (VML) are both XML-based language for generating vector graphics. SVG can be viewed in browsers using a plug-in, and VML can be viewed in Microsoft's Internet Explorer version 5.0 and higher.

The purpose of this master's project is to develop stylesheets and JavaScript technology to translate SVG to VML within Internet Explorer in as transparent a manner as possible. The first semester part of this project contains four deliverables completed for the goal of learning the following:

- XML
- SVG
- VML

- XSLT (Extensible Stylesheet Language Transformation)
- XPath
- Microsoft's XML Parser
- XMLDOM and JavaScript
- DTD (Document Type Definition)

This paper is organized in several sections. The next section gives an overview of SVG, VML, XSLT, and XPath. The succeeding sections contain the deliverable descriptions, conclusion, references, and appendices consisting of codes for the deliverables.

II. Overview of SVG, VML, XSLT, and XPath

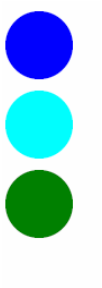
A. SVG (Scalable Vector Graphics)

The World Wide Web Consortium (W3C) defines Scalable Vector Graphics (SVG) as a language for describing two-dimensional graphics. SVG is based in XML (Extensible Markup Language). SVG graphics can be shapes, lines, or texts. According to W3C, users can include SVG into their Web sites because of SVG's compatibility with other W3C standards such as:

- XML
- Cascading Style Sheets (CSS)
- Namespaces in XML (XML-NS)
- Document Object Model (DOM)

- XSL Transformation (XSLT)
- HTML4 and XHTML

SVG has several advantages over existing graphics standards, as explained by Watt, et al. Some of them are: smaller file size, zooming images without reloading, open source code displayed by the SVG viewer, SVG text accessibility to search engines, and many more. There are also limitations to the use of SVG, and one of them is the plug-in download required to view SVG graphics. Below is a snapshot of an SVG image.



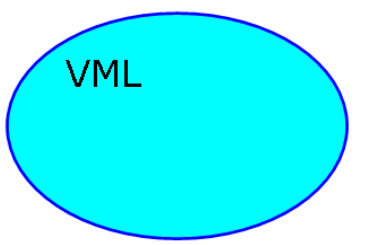
Different organizations, including Adobe, Sun Microsystems, IBM, and Apple, contributed to the development of SVG.

B. VML (Vector Markup Language)

The W3C defines VML as an XML-based language that describes and formats vector graphics for the Web. As explained by Microsoft, VML was designed to help developers overcome problems with binary graphic files that include large file size, hard to modify files and graphics that are external to the HTML file.

Currently, only Internet Explorer 5.0 and higher support VML according to Microsoft.

Just like SVG, VML can create graphics consisting of shapes, lines, and texts. Below is a snapshot of a VML image.



Several companies, including Microsoft, Hewlett-Packard, Autodesk, and Visio, contributed to the development of VML.

C. XSLT (Extensible Stylesheet Language Transformation)

The W3C defines XSLT as a language that describes rules for transforming a source tree into a result tree. The source tree is separate from the result tree. The source tree is an XML file, and the result tree can be another XML file or an HTML file. Transformation is achieved through the use of templates. The structure of the source tree can be completely different from the structure of the result tree, as explained in the W3C specification.

According to the W3C specification, a transformation expressed in XSLT is called a stylesheet, which contains template rules. A template rule has two parts:

- pattern - matched against nodes in the source tree
- template - forms the result tree

XSLT uses another language, XPath, for selecting nodes from the source tree to be processed. Below is an example of an XSLT taken from the W3C site:

```
<xsl:stylesheet version="1.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns="http://www.w3.org/TR/xhtml1/strict">
<xsl:template match="/">
<html>
  <head>
    <title>Expense Report Summary</title>
  </head>
  <body>
    <p>Total Amount: <xsl:value-of select="expense-report/total"/></p>
  </body>
</html>
</xsl:template>
</xsl:stylesheet>
```

D. XPath

The W3C defines XPath as a language for addressing parts of an XML document to be used by both XSLT and XPointer. XPath also provides support for manipulating strings, numbers, and booleans. XPath models an XML document as a tree of nodes, according to the W3C specification.

The expression is the primary construct in XPath. An expression yields an object such as a set of nodes or a number. One kind of expression is a location path,

which can be unabbreviated or abbreviated, as described in the W3C specification. Below are some examples of unabbreviated location path:

- child::text() - selects all text node children
- attribute::name - selects the name attribute

Below are the equivalent abbreviated location paths for the examples shown above:

- text() - selects all text node children
- @name - selects the name attribute

How is XPath used within XSLT? The example below shows an XSLT element with a “select” attribute selecting an employee name through an XPath abbreviated location path:

```
<xsl:value-of select="budget/employee/@name"/>
```

XPath gets its name from its use of a path notation as in URLs, as further explained in the W3C specification.

III. Deliverable 1: The 18th Green in SVG

The purpose of the first deliverable was to get familiar with SVG. Instead of using an application to drag and drop shapes, the picture was drawn by typing codes. The free Adobe SVG viewer plug-in from the Adobe website was

downloaded to be able to see SVG images on Microsoft's Internet Explorer browser.

The image was created by combining SVG elements such as lines, ovals, paths, and rectangles. As a reference, the definitions and examples from the World Wide Web Consortium site were consulted. The backgrounds were drawn first.

The shade of blue was chosen as the main background for the image. To draw the second background, SVG's path element was used. Using a single color for the shapes rendered the image flat, so the appeal of gradients was explored. The code below draws the second background with gradient color:

```
<linearGradient id="MyGradient">
    <stop offset="10%" stop-color="#0F3"/>
    <stop offset="90%" stop-color="#CE1"/>
  </linearGradient>
<path d="M 0 500 0 200 C 200 50 600 100 600 500"
fill="url(#MyGradient)"/>
```

The next four elements used were polygon, line, path, and ellipse. Again to add sparkle to the images, gradients were used as fill colors. Then to give the image a title, SVG's text element was used to write strings on top of the image. (Appendix A provides the complete code.) A snapshot of the final result is shown below as Figure 1.

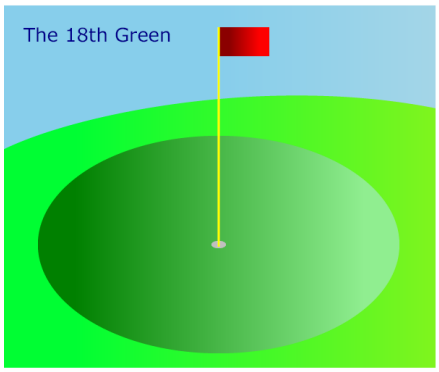


Figure 1. The 18th Green in SVG

Another goal of this deliverable was to draw a similar image using VML. Microsoft's website offers straightforward descriptions and plenty of examples on how to use VML. The elements used were rectangle, path, line, and oval. Notice that Microsoft calls the shape element "oval" instead of "ellipse". The code below shows how the second background was drawn using path and gradient color:

```
<v:shape
  style="position:relative;
  top:0px; left:0px;
  width:600px;
  height:500px"
  fillcolor="#CE1"
  strokecolor="#ADD8E6">
  <v:path id="myPath"
    v="m 0 200 c 200 50 600 100 600 500 l 0 500 0 200 x e"/>
  <v:fill    type="gradient"    color2="#0F3"    method="linear    sigma"
angle="400" focus="100%"/>
 </v:shape>
```

In SVG, the polygon element was used to draw the flag, but while working in VML it was later discovered that an easier alternative to draw the flag was to use the rectangle element. (Appendix A provides the complete code.) A snapshot of the final result is shown below as Figure 2.

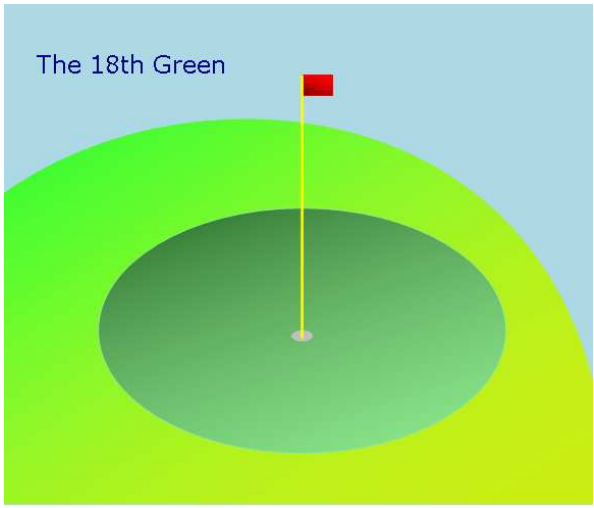


Figure 2. The 18th Green in VML

IV. Deliverable 2: XML Transformation

The World Wide Web Consortium defines the Document Object Model (DOM) as a programming interface for XML and HTML documents that can be used by different environments and applications. As explained by xmlfiles.com, Microsoft I.E. 5.0 comes with an XMLDOM parser that supports JavaScript, VBScript, Perl, Java, and other languages.

The goal of deliverable 2 was to learn how to use JavaScript and Microsoft's XMLDOM parser to load an XML file and XSLT files to translate a shape to different VML shapes. Figure 3 below depicts this goal:

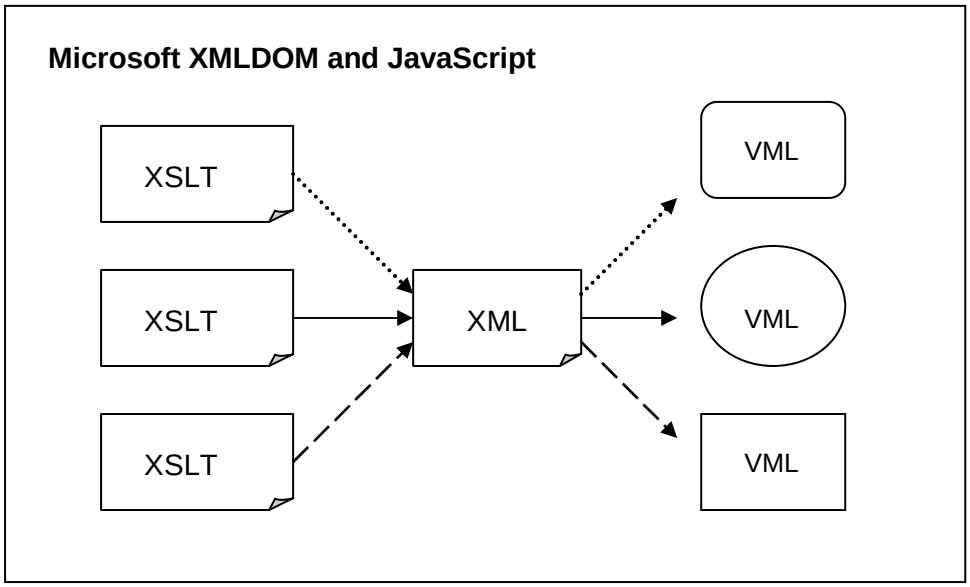


Figure 3. Transforming XML via XMLDOM and JavaScript

The first task was to create an XML file and three different XSLT files that will transform the XML file into three different VML shapes. (Appendix B shows the complete code for these files.) Then to show the shapes in I.E. browser, JavaScript and Microsoft XMLDOM were used to load the files. The code below loads the XML file into the XML parser:

```
//LOAD XML DOC
var xmldoc = new ActiveXObject("Microsoft.XMLDOM")
```

```
xmldoc.async = false
xmldoc.load("rect.xml")
```

As described by xmlfiles.com, the first line of the code above creates an instance of the Microsoft XML parser through the ActiveXObject interface, and then the second and third lines assure that the parser will wait until the document is completely loaded.

Once the XML file has been loaded into the memory, the XSLT files can be loaded and applied to transform the XML file. The code below loads an XSLT stylesheet that transforms the XML file:

```
//LOAD XSL DOC AND TRANSFORM
var oval = new ActiveXObject("Microsoft.XMLDOM")
oval.async = false
oval.load("oval.xsl")
document.write(xmldoc.transformNode(oval))
```

The XSLT stylesheet shows the result in an HTML file. Figure 4 below shows a snapshot of the final result of this deliverable:

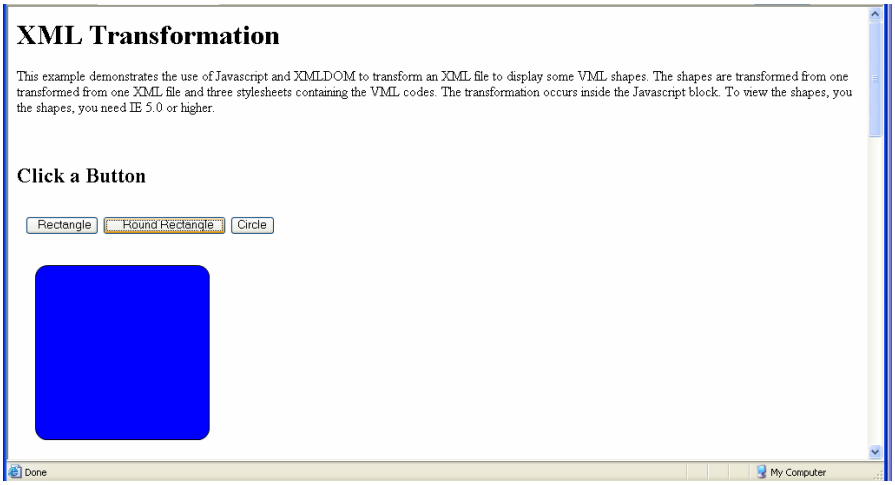


Figure 4. Deliverable 2

V. Deliverable 3: Experimenting with Two Stylesheet Transformations

A Document Type Definition (DTD) defines the document structure of an XML document, as explained by xmlfiles.com. A DTD contains a list of legal elements an XML file can use. A DTD can be a separate file or declared inside an XML document.

The purpose of deliverable 3 was to create an XML file that has a Document Type Definition, then using an XSLT stylesheet, transform this XML file into another XML file which has its own DTD. Figure 5 below shows this preliminary step.

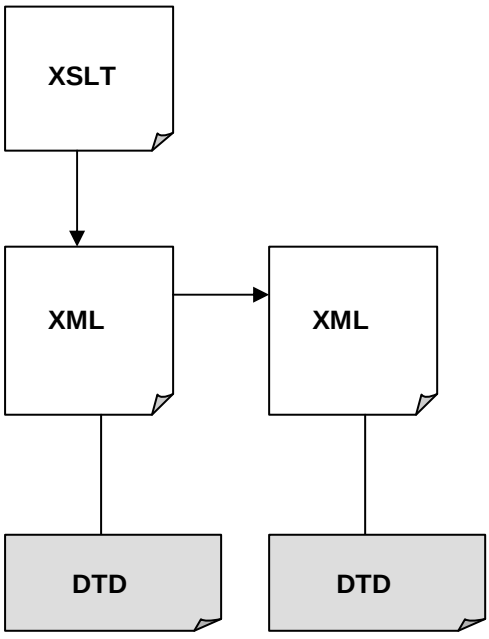


Figure 5. Transforming an XML File to Another XML File with DTD

After this initial transformation, the next goal was to apply another XSLT stylesheet to the resulting XML file that will calculate the sizes of the shapes and then draw the VML images. To accomplish this, JavaScript and Microsoft XMLDOM would be used again to load the XML and XSLT files, then instantiate a Microsoft ActiveXObject to store the result of the first transformation. (Appendix C shows the complete code for this deliverable.) The code below shows the first transformation:

```
//CREATE AN ACTIVEXOBJECT
var xml2 = new ActiveXObject("Microsoft.XMLDOM")

//LOAD INTO ACTIVEXOBJECT FIRST TRANSFORMATION
xml2.loadXML(xml1.transformNode(xsl1))
```

After transforming the first XML file and loading it into the ActiveXObject, the code below shows the second transformation.

```
//SECOND TRANSFORMATION
document.write(xml2.transformNode(xsl2))
```

Figure 6 below depicts the complete process.

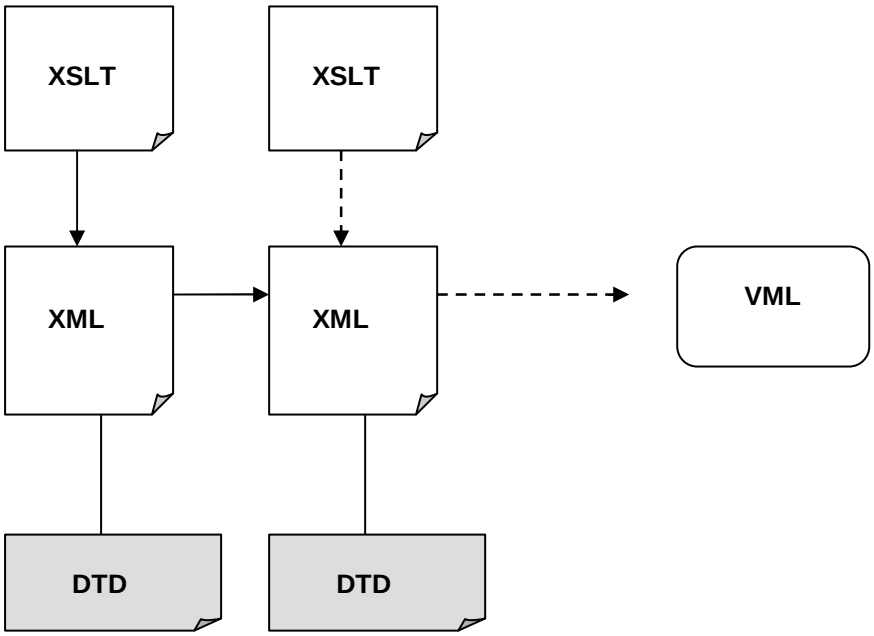


Figure 6. Two Stylesheets and Two XML files.

Figure 7 shows a snapshot of the first transformation and Figure 8 shows the result of the second transformation.

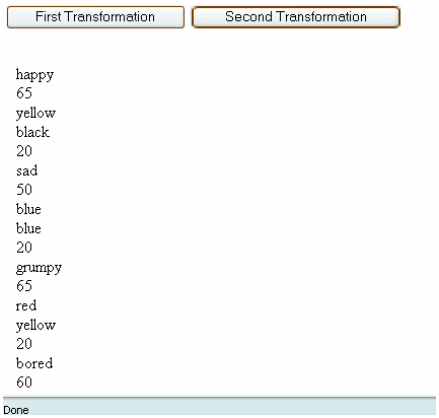


Figure 7. First Transformation

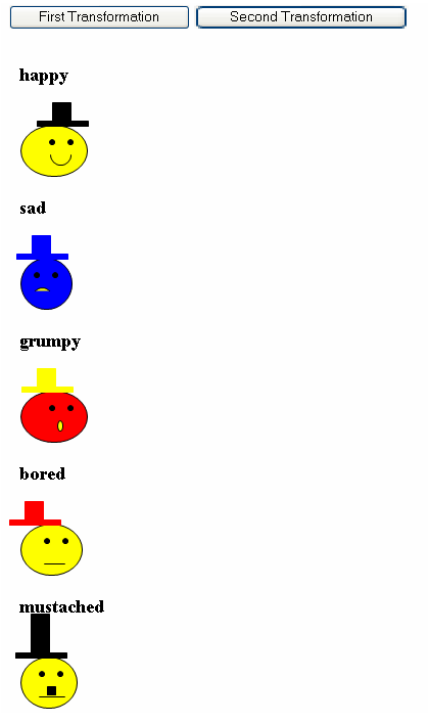


Figure 8. Second Transformation

VI. JavaScript Experiments

The second semester part of this project will contain intense JavaScript coding to translate an SVG stylesheet to its equivalent VML codes. Thus, before starting the last deliverable, the goal of the next task was to write JavaScript codes and measure the performance of each code in terms of time. The experiment compared:

- matrix multiplication using two-dimensional arrays with matrix multiplication using one-dimensional arrays
- drawing VML images without fill, with fill, and with gradient

Figure 9 below shows the result of the matrix multiplication performance comparison. Appendix D shows the complete JavaScript code for the matrix multiplication.

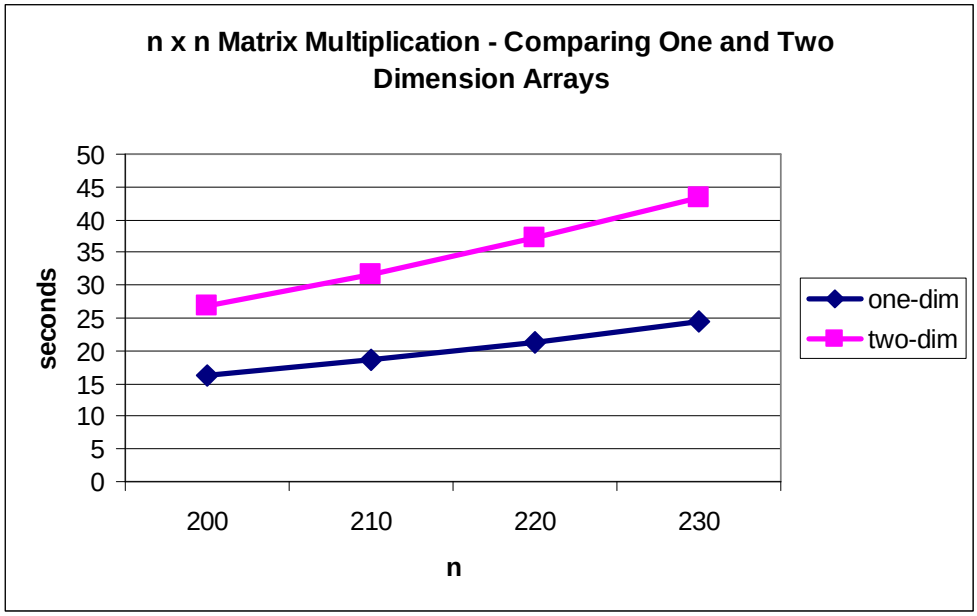


Figure 9. JavaScript Matrix Multiplication Performance

Figure 9 above shows that using one-dimensional arrays takes less time compared to using two-dimensional arrays. The next experiment compares the time taken for drawing VML images. Figure 10 shows the result.

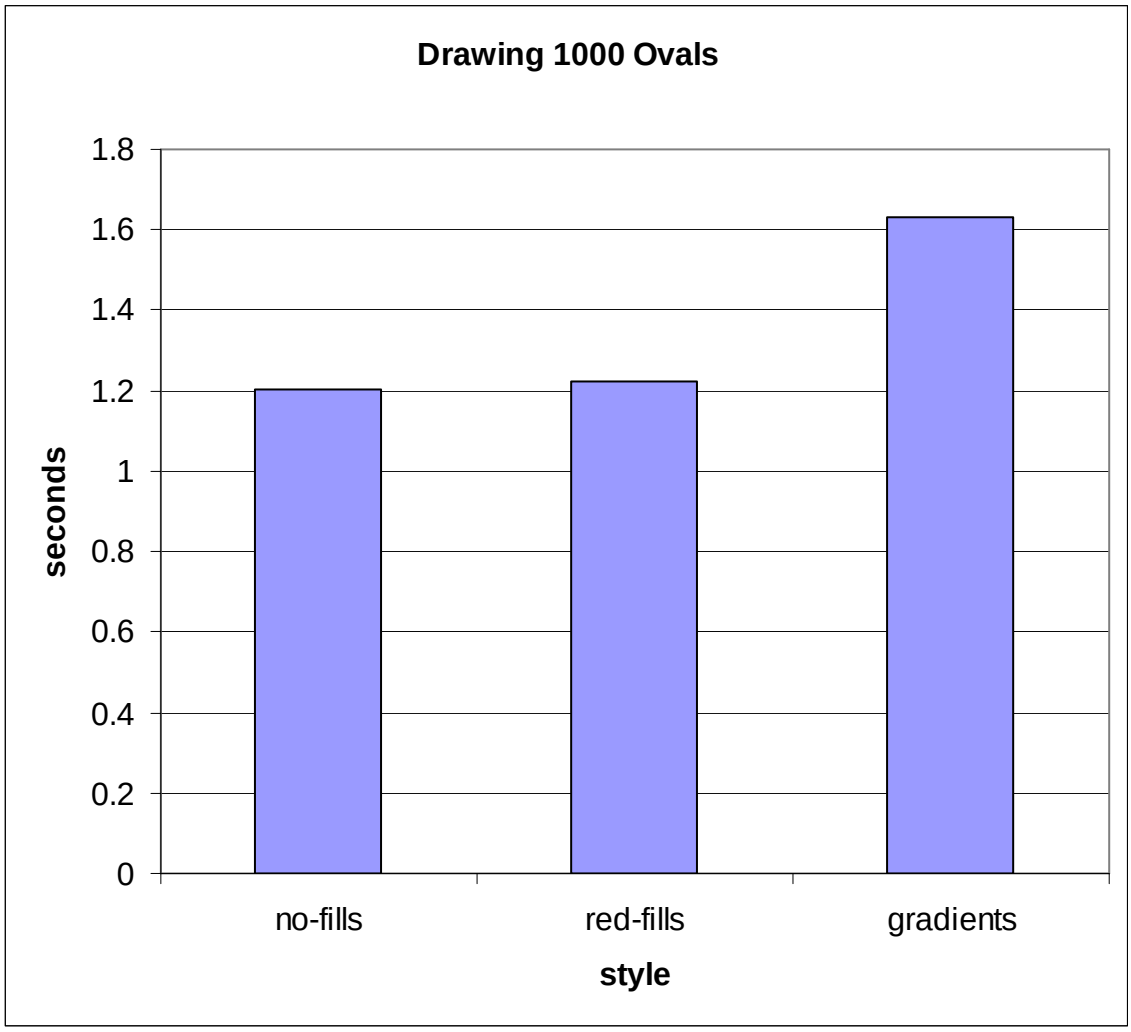


Figure 10. Drawing 1000 Ovals.

Figure 10 confirms that drawing VML ovals containing gradients takes the longest amount of time. This experiment shows that if a stylesheet contains images with gradients, the browser will take longer time to render the graphics.

Appendix D shows the complete JavaScript code for this experiment. The next section continues with the last deliverable for this semester.

VII. Deliverable 4: Stylesheet Translation of an SVG Shape to VML

The purpose of this deliverable was to translate an SVG rectangle that contains a gradient fill to a VML rectangle with an equivalent gradient using an XSLT stylesheet. Figure 11 shows the process.

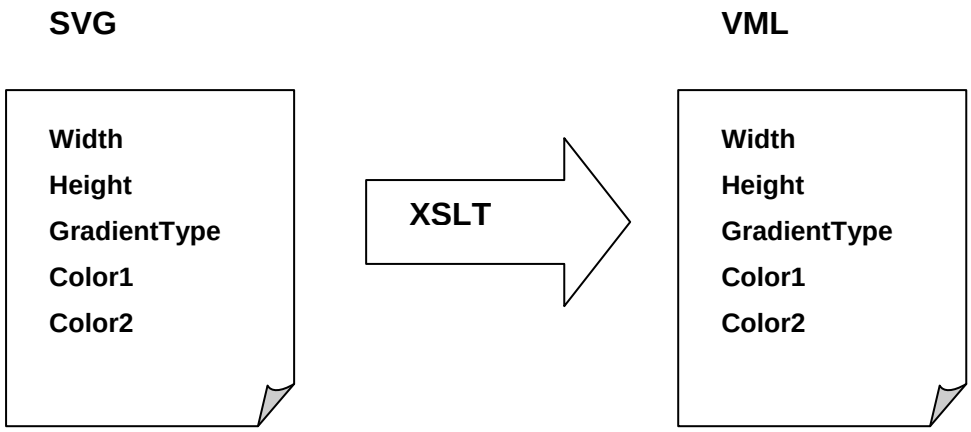


Figure 11. SVG Transformation to VML

There are two types of gradients in SVG, and they are known as radial gradient and linear gradient as defined in W3C's SVG specification. A linear gradient can be either a horizontal gradient or a vertical gradient. A gradient can also contain more than two colors.

As in SVG, a VML gradient can have more than two colors. A gradient can also be linear or radial as defined in W3C's VML specification.

The SVG file in this deliverable is very simple in that it contains only one rectangle and one linear gradient with two colors, red and green. To transform this SVG image into a VML image, the elements and attributes in the SVG file should be mapped into the equivalent VML elements and attributes. To accomplish this task requires enough knowledge of XPath to be able to extract the XML elements and attributes from the SVG file.

Using XSLT elements, the next step was to store the SVG features in variables that will be used later to render the VML image. The process can be described as follows:

- extract SVG element (or attribute)
- store in a variable
- use variable to draw VML

Appendix E shows the complete code for deliverable 4. Figure 12 below shows the two images.

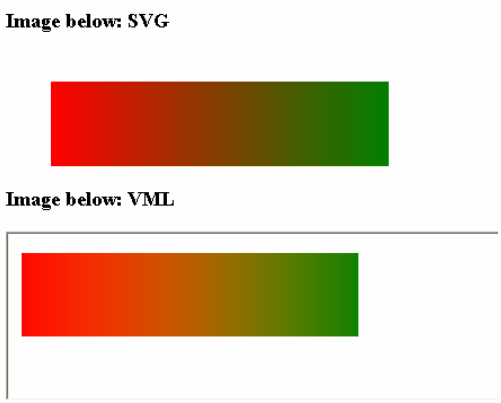


Figure 12. SVG and VML Images

An interesting observation in deliverable 4 was the use of stroke color around the VML's rectangle shape. To be able to match the SVG image, a stroke color of white was added to the stylesheet; otherwise, a black border will appear as a default to the VML image.

One of the challenges of the second semester's final project is how to represent an SVG gradient in VML. As shown in Figure 12, in spite of using the same colors, the gradient of the VML image appears lighter in the middle as compared to the SVG. The second semester part of this project will address that issue.

VIII. Conclusion.

The author started the first semester part of this master's project without any knowledge of SVG, VML, XMLDOM, and DTD, so completing the deliverables required a lot of time and study. Luckily, XML and XPath were covered very briefly in one of the author's previous classes, which helped a little. After completing all the deliverables, familiarity with the acronyms mentioned above was gained.

Deliverable 1 was fun and exciting. Deliverable 2 was very challenging because it was the first XML transformation. With very little exposure to XML, XPath, JavaScript, and XMLDOM, it was quite challenging to carry out this task. After reading books, tutorials from the web, and repeated explanation from the project's advisor, three stylesheets were finally created.

It was expected that working on deliverable 3 was to be as demanding as the previous deliverable; it was about transforming a stylesheet, only doing it twice. In this assignment, DTD was introduced. In addition, the knowledge on how to transform an XML to a valid XML based on a DTD was needed.

The additional difficulty of this deliverable stemmed from performing the second transformation. In this task, the result of the first transformation had to be stored into the memory so it can be used for the second transformation. In addition,

some useful XSLT math functions were introduced to be able to use radius data and rectangle width to calculate the VML shapes.

Before moving on to the last deliverable, the JavaScript experiments provided extra hands-on and additional insight as to how fast the language performs calculations and graphics rendering. The re-coding of two-dimensional matrix multiplication into one-dimension arrays took considerable amount of time.

The last deliverable offered a chance to get more familiar with XSLT and XPath. The approach that was used to transform the SVG into VML is the use of variables in XSLT. It was learned that once you have assigned a value in a variable, there is no provision in XSLT to change or modify that value, as explained by devguru.com. This deliverable also supplemented the author's knowledge of XPath, which is vital in stylesheet translations.

The author looks forward to gaining more proficiency in stylesheet translations, learning more about SVG and VML features, XPath and XSLT functions, and JavaScript optimizations in the second semester part of this master's project.

IX. References for All the Deliverables and this Report

Cagle, K. SVG Programming. APress. 2002.

Demcsak, D. Introduction to the SharpVectorGraphics Project SVG Open 2003: 2nd Annual Conference on Scalable Vector Graphics.

DevGuru XML DOM Introduction. http://www.devguru.com/Technologies/xmldom/quickref/xmldom_intro.html

DevGuru XSLT Introduction. http://www.devguru.com/Technologies/xslt/quickref/xslt_intro.html

Goodman, D. JavaScript Examples Bible. Hungry Minds, Inc. 2001.

Harold, E. R. XML Bible (2nd Ed). John Wiley & Sons. 2001.

Introduction to Vector Markup Language. http://msdn.microsoft.com/library/default.asp?url=/workshop/author/vml/. MSDN 1998.

Kay, M. XSLT. 2nd Edition. Wrox Press Ltd. 2001.

Scalable Vector Graphics (SVG) 1.1 Specification. http://www.w3.org/TR/SVG/

Shah, B. Scalable Vector Graphics. 3rd Annual CM316 Conference on Multimedia Systems. 2003.

SVG Document Object Model (DOM). http://www.w3.org/TR/SVG/svgdom.html

Vector Markup Language (VML). http://www.w3.org/TR/NOTE-VML

W3C Document Object Model. http://www.w3.org/DOM/

Watt A, Lilley C., Ayers, D., George, R., Wenz, C., Hauser T., Lindsey K., Gustavsson, N. SVG Unleashed. Sams Publishing. Indianapolis, Indiana. 2003

XML Path Language. http://www.w3.org/TR/xpath

XMLFiles.com. http://xmlfiles.com

XSL Transformations (XSLT). http://www.w3.org/TR/xslt

Appendix A: Source Code for Deliverable 1

SVG Source Code

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

<!-- Author: Julie Nabong-->
<!-- Date:   4 July 2003-->
<!-- Title:  The 18th Green in SVG-->

<!--THE BACKGROUND-->
<g>
  <defs>
    <linearGradient id="BackGradient">
      <stop offset="30%" stop-color="skyblue"/>
      <stop offset="70%" stop-color="lightblue"/>
    </linearGradient>
  </defs>

  <rect style="fill:url(#BackGradient);" width="600" height="500"/>
</g>

<!--THE FRINGE-->
<g>
  <defs>
    <linearGradient id="MyGradient">
      <stop offset="10%" stop-color="#0F3"/>
      <stop offset="90%" stop-color="#CE1"/>
    </linearGradient>
  </defs>

  <path   d="M   0   500   0   200   C   200   50   600   100   600   500"
fill="url(#MyGradient)"/>
</g>

<!--THE GREEN-->
<g>
  <defs>
    <linearGradient id="GreenGradient">
      <stop offset="10%" stop-color="green"/>
      <stop offset="90%" stop-color="lightgreen"/>
     </linearGradient>
  </defs>

  <ellipse style="fill:url(#GreenGradient);" cx="300" cy="330" rx="200"
ry="120"/>
</g>

<!--THE HOLE-->
  <ellipse style="fill:silver;" cx="300" cy="330" rx="10" ry="5"/>

<!--THE FLAG-->
<g>
  <defs>
   <linearGradient id="FlagGradient">
```

```
    <stop offset="20%" stop-color="darkred"/>
    <stop offset="80%" stop-color="red"/>
   </linearGradient>
  </defs>

  <polygon   fill="url(#FlagGradient)"   points="300,60   330,60   330,80
300,80"/>
</g>

<!--THE FLAGSTICK-->
<g stroke="yellow">
  <line x1="300" y1="333" x2="300" y2="60" stroke-width="3"/>
</g>

<!--THE TITLE-->
<text     x="30"     y="50"     font-family="Verdana"     font-size="26"
fill="darkblue">
  The 18th Green
</text>

</svg>
```

VML Source Code

```
<!-- Author: Julie Nabong-->
<!-- Date:   10 July 2003-->
<!-- Title:  The 18th Green in VML-->

<v:group id="18thGreen"
  style="position:relative; top:0px; left:0px; width:600px;
height:500px"
  coordorigin="0,0" coordsize="600,500">

<!--THE BACKGROUND-->
<v:rect
  style="position:relative;
  top:0px; left:0px;
  width:600px;
  height:500px;"
  fillcolor="#ADD8E6"
  border="0"
  strokecolor="white">
</v:rect>

<!--THE TITLE-->
<v:rect
 strokecolor="#ADD8E6"
 fillcolor="#ADD8E6"
 style="position:relative;
 top:50;left:30;width:220;height:40">
  <v:textbox id="mytitle">
   <font size="+3" face="Verdana" color="darkblue">
   The 18th Green
   </font>
```

```
  </v:textbox>
</v:rect>

<!--THE FRINGE-->
<v:shape
  style="position:relative;
  top:0px; left:0px;
  width:600px;
  height:500px"
  fillcolor="#CE1"
  strokecolor="#ADD8E6">
  <v:path id="myPath"
    v="m 0 200 c 200 50 600 100 600 500 l 0 500 0 200 x e"/>
  <v:fill type="gradient" color2="#0F3" method="linear sigma"
angle="400" focus="100%"/>
 </v:shape>

<!--THE GREEN-->
<v:oval
  strokecolor="#90EE90"
  fillcolor="#006400"
  style="position:relative;
  top:210px; left:100px;
  width:400;
  height:240">
  <v:fill type="gradient" color2="#90EE90" method="linear sigma"
angle="30" focus="100%"/>
</v:oval>

<!--THE HOLE-->
<v:oval
  strokecolor="silver"
  fillcolor="silver"
  style="position:relative;
  top:330px; left:290px;
  width:20;
  height:10">
</v:oval>

<!--THE FLAG-->
<v:rect
  style="position:relative;
  top:80px; left:300px;
  width:30;
  height:20"
  fillcolor="red"
  border="0"
  strokecolor="red">
  <v:fill type="gradient" color2="#8B0000" method="linear sigma"
angle="330" focus="100%"/>
</v:rect>

<!--THE FLAGSTICK-->
<v:line
  from="300,338"
  to="300,80"
  strokecolor="yellow"
```

```
  strokeweight="2pt">
</v:line>

</v:group>
```

Appendix B: Source Code for Deliverable 2

Stylesheet Source Code for the Blue Rectangle

```
<!--Author:  Julie Nabong-->
<!--Date:  August 2003-->
<!--Title: Blue Rectangle Stylesheet-->

<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">
   <xsl:template match="/">
      <html xmlns:v="urn:schemas-microsoft-com:vml">
        <head>
            <style type="text/css">
              v\:*{behavior:url(#default#VML);}
            </style>
        </head>
        <body>
          <xsl:element name="v:roundrect">
            <xsl:attribute name="fillcolor">blue</xsl:attribute>
            <xsl:attribute
name="style">width:200;height:200</xsl:attribute>
          </xsl:element>
        </body>
      </html>
    </xsl:template>
 </xsl:stylesheet>
```

Stylesheet Source Code for the Red Rectangle

```
<!--Author:  Julie Nabong-->
<!--Date: August 2003-->
<!--Title: Red Rectangle Stylesheet-->

<xsl:stylesheet         xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">
   <xsl:template match="/">
      <html xmlns:v="urn:schemas-microsoft-com:vml">
        <head>
            <style type="text/css">
              v\:*{behavior:url(#default#VML);}
            </style>
        </head>
        <body>
          <xsl:element name="v:rect">
            <xsl:attribute name="fillcolor">red</xsl:attribute>
            <xsl:attribute
name="style">width:180;height:180</xsl:attribute>
```

```
          </xsl:element>
        </body>
      </html>
    </xsl:template>
 </xsl:stylesheet>
```

Stylesheet Source Code for the Yellow Oval

```
<!--Author:  Julie Nabong-->
<!--Date:  August 2003-->
<!--Title:  Yellow Oval Stylesheet-->

<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">
   <xsl:template match="/">
      <html xmlns:v="urn:schemas-microsoft-com:vml">
        <head>
            <style type="text/css">
              v\:*{behavior:url(#default#VML);}
            </style>
        </head>
        <body>
          <xsl:element name="v:oval">
            <xsl:attribute name="fillcolor">yellow</xsl:attribute>
            <xsl:attribute
name="style">width:180;height:180</xsl:attribute>
          </xsl:element>
        </body>
      </html>
    </xsl:template>
 </xsl:stylesheet>
```

Source Code for the XML

```
<!--Author:  Julie Nabong-->
<!--Date:  August 2003-->
<!--Title: XML File for Transformation-->

<?xml version="1.0" encoding="ISO-8859-1"?>
<x xmlns:v="urn:schemas-microsoft-com:vml">
    <v:rect>
    </v:rect>
</x>
```

Source Code for HTML & JavaScript

```
<!--Author: Julie Nabong-->
<!--Date:  August 2003-->
```

```
<!--Title: XML Transformation-->

<html xmlns:v="urn:schemas-microsoft-com:vml">
<head><title>Shapes</title>
<style type="text/css">
v\:*{behavior:url(#default#VML);}
</style>
</head>

<body onload="showYellow()">

<script type="text/javascript">

  //LOAD XML DOC
  var xmldoc = new ActiveXObject("Microsoft.XMLDOM")
  xmldoc.async = false
  xmldoc.load("rect.xml")

  //LOAD AND TRANSFORM XSL DOC FOR CIRCLE
  function showYellow()
  {
      frame.document.open()
      var oval = new ActiveXObject("Microsoft.XMLDOM")
      oval.async = false
      oval.load("oval.xsl")
      frame.document.write(xmldoc.transformNode(oval))
      frame.document.close()
  }

  //LOAD AND TRANSFORM XSL DOC FOR RED RECT
  function showRed()
  {
      frame.document.open()
      var redRect = new ActiveXObject("Microsoft.XMLDOM")
      redRect.async = false
      redRect.load("redRect.xsl")
      frame.document.write(xmldoc.transformNode(redRect))
      frame.document.close()
  }

  //LOAD AND TRANSFORM XSL DOC FOR BLUE ROUNDRECT
  function showBlue()
  {
      frame.document.open()
      var blueRect = new ActiveXObject("Microsoft.XMLDOM")
      blueRect.async = false
      blueRect.load("blueRect.xsl")
      frame.document.write(xmldoc.transformNode(blueRect))
      frame.document.close()
  }

</script>

<form>
  <input type="button" value="Rectangle" onclick="showRed()"></input>
  <input          type="button"          value="Round          Rectangle"
onclick="showBlue()"></input>
```

```
  <input type="button" value="Circle" onclick="showYellow()"></input>
</form>

<iframe  name="frame"  frameborder="0"  src="rect.htm"  scrolling="no"
width="300" height="300"></iframe>
</body>
</html>
```

Appendix C: Source Code for Deliverable 3

Source Code for the XML File

```
<!--Author:  Julie Nabong-->
<!--Date:  October 2003-->
<!--Title: The Smiley XML File-->

<?xml version="1.0" ?>
<!DOCTYPE smileys SYSTEM "smileys.dtd">
<smileys>
  <face type="graphical" filetype="jpg">
    <emotion>happy</emotion>
    <width>130</width>
    <color>yellow</color>
    <hatcolor>black</hatcolor>
    <hatheight>20</hatheight>
   </face>

   <face type="graphical" filetype="gif">
      <emotion>sad</emotion>
      <width>100</width>
      <color>blue</color>
      <hatcolor>blue</hatcolor>
    <hatheight>20</hatheight>
   </face>

   <face type="graphical" filetype="bmp">
    <emotion>grumpy</emotion>
    <width>130</width>
    <color>red</color>
    <hatcolor>yellow</hatcolor>
    <hatheight>20</hatheight>
   </face>

   <face type="text" filetype="txt">
    <emotion>bored</emotion>
    <width>120</width>
    <color>yellow</color>
    <hatcolor>red</hatcolor>
    <hatheight>20</hatheight>
   </face>

   <face type="graphical" filetype="jpg">
    <emotion>mustached</emotion>
    <width>110</width>
    <color>yellow</color>
    <hatcolor>black</hatcolor>
    <hatheight>40</hatheight>
   </face>
</smileys>
```

Source Code for the First Stylesheet

```
<!--Author:  Julie Nabong-->
<!--Date:  October 2003-->
<!--Title: First Stylesheet-->

<?xml version="1.0" ?>
<xsl:stylesheet                                                    version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output    method="xml"
                    standalone="yes"
                    omit-xml-declaration="yes"
                    doctype-system="images.dtd"/>


  <xsl:template match="/">
      <xsl:element name="images">
            <xsl:apply-templates />
      </xsl:element>
  </xsl:template>


  <xsl:template match="face">
      <image>
        <xsl:apply-templates />
      </image>
  </xsl:template>

  <xsl:template match="emotion" >
      <xsl:element name="name">
            <xsl:value-of select="." />
      </xsl:element>
  </xsl:template>

   <xsl:template match="width" >
      <xsl:element name="radius">
            <xsl:variable name="r" select="." />
                  <xsl:value-of select="$r div 2" />
      </xsl:element>
  </xsl:template>

 <xsl:template match="color">
      <xsl:element name="fillcolor">
            <xsl:value-of select="." />
      </xsl:element>
 </xsl:template>


 <xsl:template match="hatcolor">
      <xsl:element name="secondcolor">
            <xsl:value-of select="." />
      </xsl:element>
 </xsl:template>

 <xsl:template match="hatheight">
      <xsl:element name="rec-height">
```

```
            <xsl:value-of select="." />
      </xsl:element>
 </xsl:template>

</xsl:stylesheet>
```

Source Code for the Second Stylesheet

```
<!--Author:  Julie Nabong-->
<!--Date:  October 2003-->
<!--Title: Second Stylesheet-->

<?xml version="1.0" ?>
<xsl:stylesheet                                                    version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                                          xmlns:v="urn:schemas-
microsoft-com:vml">

      <xsl:template match="/">

      <html xmlns:v="urn:schemas-microsoft-com:vml">
      <head>
      <style type="text/css">
            v\:*{behavior:url(#default#VML);}
      </style>
      </head>
      <body>

      <xsl:for-each select="//image">
            <xsl:variable name="look">
                  <xsl:value-of select="name" />
      </xsl:variable>
      <h3><xsl:value-of select="$look" /></h3>
      <br></br>

      <xsl:variable name="width">
            <xsl:value-of select="radius" />
      </xsl:variable>

      <xsl:variable name="fillcolor">
            <xsl:value-of select="fillcolor" />
      </xsl:variable>

<!-- DRAW CIRCLES -->

            <xsl:element name="v:oval" >
        <xsl:attribute name="style">width:
            <xsl:value-of select="$width"/>;height:50px</xsl:attribute>
        <xsl:attribute name="fillcolor">
            <xsl:value-of select="$fillcolor"/></xsl:attribute>
        </xsl:element>

      <!-- LEFT EYE -->
        <xsl:element name="v:oval">
```

```
        <xsl:attribute   name="style">position:relative;top:-20px;left:-
40px;width:5px;height:5px</xsl:attribute>
        <xsl:attribute name="fillcolor">black</xsl:attribute>
        </xsl:element>

   <!-- RIGHT EYE -->
            <xsl:element name="v:oval">
        <xsl:attribute   name="style">position:relative;top:-20px;left:-
30px;width:5px;height:5px</xsl:attribute>
        <xsl:attribute name="fillcolor">black</xsl:attribute>
        </xsl:element>

 <!-- DRAW THE MOUTHS -->
                  <xsl:if test="$look='bored'">
                        <xsl:element name="v:line">
                              <xsl:attribute               name="from">-
35,40</xsl:attribute>
                              <xsl:attribute                 name="to">-
55,40</xsl:attribute>
                        </xsl:element>
                  </xsl:if>

                  <xsl:if test="$look='happy'">
                        <xsl:element name="v:arc">
                              <xsl:attribute
name="style">position:relative;top:-5;left:-
55;width:20;height:20</xsl:attribute>
                              <xsl:attribute
name="startangle">90</xsl:attribute>
                              <xsl:attribute
name="endangle">270</xsl:attribute>
                              <xsl:attribute
name="fillcolor">yellow</xsl:attribute>
                        </xsl:element>
                  </xsl:if>

                  <xsl:if test="$look='grumpy'">
                        <xsl:element name="v:arc">
                              <xsl:attribute
name="style">position:relative;top:-5;left:-
48;width:5;height:10</xsl:attribute>
                              <xsl:attribute
name="startangle">0</xsl:attribute>
                              <xsl:attribute
name="endangle">360</xsl:attribute>
                              <xsl:attribute
name="fillcolor">yellow</xsl:attribute>
                        </xsl:element>
                  </xsl:if>

                  <xsl:if test="$look='sad'">
                        <xsl:element name="v:arc">
                              <xsl:attribute
name="style">position:relative;top:-5;left:-
55;width:20;height:20</xsl:attribute>
                              <xsl:attribute         name="startangle">-
45</xsl:attribute>
```

```
                              <xsl:attribute
name="endangle">45</xsl:attribute>
                              <xsl:attribute
name="fillcolor">yellow</xsl:attribute>
                        </xsl:element>
                  </xsl:if>

                  <xsl:if test="$look='mustached'">
                        <xsl:element name="v:line">
                              <xsl:attribute               name="from">-
30,40</xsl:attribute>
                              <xsl:attribute                 name="to">-
55,40</xsl:attribute>
                        </xsl:element>

                        <xsl:element name="v:rect">
                              <xsl:attribute
name="style">position:relative;top:-5;left:-
48;width:8;height:8</xsl:attribute>
                              <xsl:attribute
name="fillcolor">black</xsl:attribute>
                        </xsl:element>
            </xsl:if>

 <!-- GET OTHER VARIABLES -->

            <xsl:variable name="hat-color">
                  <xsl:value-of select="secondcolor" />
            </xsl:variable>

            <xsl:variable name="rec-height">
                  <xsl:value-of select="rec-height" />
            </xsl:variable>

<!-- DRAW THE HATS -->

            <xsl:element name="v:rect">
                  <xsl:attribute    name="style">position:relative;top:-
38;left:-90;width:50;height:5
                  </xsl:attribute>
                  <xsl:attribute name="fillcolor">
                        <xsl:value-of select="secondcolor" />
                  </xsl:attribute>
                  <xsl:attribute name="strokecolor">
                        <xsl:value-of select="secondcolor" />
                  </xsl:attribute>
            </xsl:element>

            <xsl:element name="v:rect">
                  <xsl:attribute    name="style">position:relative;top:-
52;left:-128;width:18;height:
                        <xsl:value-of select="rec-height"/>
                  </xsl:attribute>
                  <xsl:attribute name="fillcolor">
                        <xsl:value-of select="secondcolor" />
                  </xsl:attribute>
                  <xsl:attribute name="strokecolor">
```

```
                        <xsl:value-of select="secondcolor"/>
                    </xsl:attribute>
                </xsl:element>


        </xsl:for-each>


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

Source Code for HTML & JavaScript

```
<!--Author:  Julie Nabong-->
<!--Date:  October 2003-->
<!--Title:  Double Transformation-->

<html xmlns:v="urn:schemas-microsoft-com:vml">
<head><title>Double Transformation</title>
<style type="text/css">
v\:*{behavior:url(#default#VML);}
</style>
</head>
<body onload="showStart()">
<script type="text/javascript">

  //LOAD FILES
  var xml1 = new ActiveXObject("Microsoft.XMLDOM")
  xml1.async = false
  xml1.load("smileys.xml")

  //LOAD FIRST XSL FILE
  var xsl1= new ActiveXObject("Microsoft.XMLDOM")
  xsl1.async = false
  xsl1.load("xsl1.xsl")

  //CREATE AN ACTIVEXOBJECT
  var xml2 = new ActiveXObject("Microsoft.XMLDOM")
  xml2.async = false

  //LOAD INTO ACTIVEXOBJECT FIRST TRANSFORMATION
  xml2.loadXML(xml1.transformNode(xsl1))

  //LOAD SECOND XSL FILE
  var xsl2 = new ActiveXObject("Microsoft.XMLDOM")
  xsl2.async = false
  xsl2.load("xsl2.xsl")


//SHOW FIRST TRANSFORMATION
function showFirst()
{
```

```
    frame.document.open()
    var nodes = xml2.documentElement
    traverse(nodes)
    frame.document.close()
}

//SECOND TRANSFORMATION
function showSecond()
{
      frame.document.open()
      frame.document.write(xml2.transformNode(xsl2))
      frame.document.close()
}

function showStart()
{
     frame.document.open()
     frame.document.close()
}

function traverse(tree)
{
  if(tree.hasChildNodes())
  {
      for(var i=0; i < tree.childNodes.length; i++)
        traverse(tree.childNodes(i))
  }
  else
  {
      frame.document.write(tree.text)
      frame.document.write("<br>")
  }
}

</script>

<form>
  <input        type="button"        value="First        Transformation"
onclick="showFirst()"></input>
  <input        type="button"       value="Second        Transformation"
onclick="showSecond()"></input>
</form>

<iframe  name="frame"  frameborder="0"  src="display.htm"  scrolling="no"
width="600" height="800"></iframe>

</body>
</html>
```

Appendix D: Source Code for the JavaScript Experiments

Source Code for Two-Dimension Matrix Multiplication

```
<!--Author:  Julie Nabong-->
<!--Date: November 2003-->
<!--Title: Matrix Multiplication Using Two-Dimension Arrays-->

<html>
<head><title>Matrix-For-VarOut</title></head>
<body>
<form name="text" onSubmit="calc()">
<b>Type a number </b>
<input type="text" name="textbox" size="20"></input>
</form>

<script type="text/javascript">
function calc()
{
  var num = parseInt(window.document.text.textbox.value)
  var matrix1 = new Array(num)
  var matrix2 = new Array(num)
  var ans = new Array(num)

  //CREATE 2 DIMENSION ARRAYS
  for(var i=0; i<num; i++)
  {
      matrix1[i] = new Array(num)
      matrix2[i] = new Array(num)
      ans[i] = new Array(num)
  }

  //FILL IN MATRICES WITH RANDOM VALUES
  for (var i=0; i<num; i++)
  {
      for (var j=0; j < num; j++)
      {
          matrix1[i][j] = Math.floor(Math.random() * 10)
          matrix2[i][j] = Math.floor(Math.random() * 10)
      }
  }

//PRINT MATRIX A
document.write("MATRIX A" + "<br>")
  for(var i=0; i<num; i++)
  {
       for(var j=0; j<num; j++)
       {
         document.write(matrix1[i][j] + "  ")
       }
       document.write("<br>")
```

```
  }

//PRINT MATRIX B
document.write("MATRIX B" + "<br>")
  for(var i=0; i<num; i++)
  {
       for(var j=0; j<num; j++)
       {
         document.write(matrix2[i][j] + "  ")
       }
       document.write("<br>")
  }

var x = 0
var i=0
var j=0
var k=0
ans[i][j]=0
//SET START TIME
var startTime = new Date().getTime()
var endTime = 0
document.write("<br>" + "Start Time: " + startTime/1000)

//DO MATRIX MULTIPLICATION
for (i=0; i<num; i++)
{
    for(j=0; j<num; j++)
    {
        for (k=0; k<num; k++)
        {
            x += matrix1[i][k] * matrix2[k][j]
         }
         ans[i][j] = x
         x=0
     }
     x=0
 }

endTime = new Date().getTime()
document.write("<br>" + "End Time: " + endTime/1000)
document.write("<br>" + "Time Taken: " + ((endTime-startTime)/1000) + "
seconds")

//PRINT RESULT MATRIX
document.write("<br><br>" + "ANSWER" + "<br>")
for(var i=0; i<num; i++)
{
  for(var j=0; j<num; j++)
  {
    document.write(ans[i][j] + " ")
  }
  document.write("<br>")
}

alert("Time Taken: " + ((endTime-startTime)/1000) + " seconds")

}//END FUNCTION
```

```
</script>
</body>
</html>
```

Source Code for One-Dimension Matrix Multiplication

```
<!--Author:  Julie Nabong-->
<!--Date:  November 2003-->
<!--Title: One-Dimension Matrix Multiplication-->


<html>
<head><title>One Dimension Matrix Multiplication</title></head>
<body>
<form name="text" onSubmit="calc()">
<b>Type a number </b>
<input type="text" name="textbox" size="20"></input>
</form>

<script type="text/javascript">
function calc()
{
  var num = parseInt(window.document.text.textbox.value)
  var matrix1 = new Array(num)
  var matrix2 = new Array(num)
  var ans = new Array(num*num)

  //CREATE 2 DIMENSION ARRAYS
  for(var i=0; i<num; i++)
  {
      matrix1[i] = new Array(num)
      matrix2[i] = new Array(num)
  }

  //FILL IN MATRICES WITH RANDOM VALUES
  for (var i=0; i<num; i++)
  {
      for (var j=0; j < num; j++)
      {
         matrix1[i][j] = Math.floor(Math.random() * 10)
         matrix2[i][j] = Math.floor(Math.random() * 10)
      }
  }

//PRINT MATRIX A
document.write("MATRIX A" + "<br>")
  for(var i=0; i<num; i++)
  {
       for(var j=0; j<num; j++)
       {
         document.write(matrix1[i][j] + "  ")
       }
       document.write("<br>")
  }
```

```
//PRINT MATRIX B
document.write("MATRIX B" + "<br>")
  for(var i=0; i<num; i++)
  {
       for(var j=0; j<num; j++)
       {
         document.write(matrix2[i][j] + "  ")
       }
       document.write("<br>")
  }

document.write("<br>")

//CREATE ONE-DIMENSION MATRIX A
var n=num*num
var mat1 = new Array(n)
var mat2 = new Array(n)
var c=0
var i=0
var j=0

while(c<n)
{
  for(i=0; i<num; i++)
  {
    for(j=0;j<num;j++)
    {
      mat1[c] = matrix1[i][j] //OK
      c++
    }
   }
 }

//CREATE ONE-DIMENSION MATRIX B
c=0
i=0
j=0

while(c<n)
{
  for(j=0; j<num;j++)
  {
    for(i=0;i<num;i++)
    {
      mat2[c] = matrix2[i][j] //OK
      c++
    }
   }
 }

//CREATE ONE-DIMENSION ANS MATRIX
for(var c=0;c<n;c++)
  ans[c]=0

//SET START TIME
var startTime = new Date().getTime()
```

```
var endTime = 0
document.write("<br>" + "Start Time: " + startTime/1000 + "<br>")

//MATRIX MULTIPLICATION
c=0
i =0
j=0
var k=0
var x=0

for(i=0; c<n; i+=num)
{
  for(j=0; j<n; j += num)
  {
     for(k=0; k<num; k++)
     {
        x += mat1[k+i]*mat2[k+j]
     }
      ans[c]=x
      x=0
      c++
   }
}

endTime = new Date().getTime()
document.write("<br>" + "End Time: " + endTime/1000)
document.write("<br>" + "Time Taken: " + ((endTime-startTime)/1000) + "
seconds")

//CREATE TWO-DIMENSION RESULT MATRIX
var resultMatrix = new Array(num)

for(var i=0; i<num; i++)
{
    resultMatrix[i] = new Array(num)
}

//STORE RESULT TO RESULT MATRIX
c=0
while(c<n)
{
  for(i=0; i<num; i++)
  {
    for(j=0; j<num; j++)
    {
      resultMatrix[i][j] = ans[c]
      c++
    }
  }
}

//PRINT RESULT MATRIX
document.write("<br><br>" + "ANSWER" + "<br>")
for(var i=0; i<num; i++)
{
  for(var j=0; j<num; j++)
  {
```

```
    document.write(resultMatrix[i][j] + " ")
  }
  document.write("<br>")
}

alert("Time Taken: " + ((endTime-startTime)/1000) + " seconds")

}//END FUNCTION
</script>
</body>
</html>
```

Source Code for Drawing 1000 Ovals without Fill Color

```
<!--Author:  Julie Nabong-->
<!--Date:  November 2003-->
<!--Title:  Drawing VML Ovals Without Fills-->

<html xmlns:v="urn:schemas-microsoft-com:vml">
<head>
<style type="text/css">
  v\:*{behavior:url(#default#VML);}
</style>
</head>

<body>

<script language=Javascript>
//SET START TIME
var startTime = new Date().getTime()
var endTime = 0

for(var i=0;i<1000; i++)
{
  document.write('<v:oval id="oval"
style="width:100;height:100"></v:oval>')
}

endTime = new Date().getTime()
alert("Time Taken: " + ((endTime-startTime)/1000) + " seconds")
</script>
</body>
</html>
```

Source Code for Drawing 1000 VML Ovals with Fillcolor

```
<!--Author:  Julie Nabong-->
<!--Date:  November 2003-->
<!--Title:  Drawing VML Ovals with Fillcolor-->

<html xmlns:v="urn:schemas-microsoft-com:vml">
```

```
<head>
<style type="text/css">
  v\:*{behavior:url(#default#VML);}
</style>
</head>

<body>

<script language=Javascript>
//SET START TIME
var startTime = new Date().getTime()
var endTime = 0

for(var i=0;i<1000; i++)
{
  document.write('<v:oval
style="width:100;height:100"fillcolor="red"></v:oval>')
}

endTime = new Date().getTime()
alert("Time Taken: " + ((endTime-startTime)/1000) + " seconds")
</script>
</body>
</html>
```

Source Code for Drawing 1000 VML Ovals with Gradient

```
<!--Author:  Julie Nabong-->
<!--Date:  November 2003-->
<!--Title:  Drawing VML Ovals with Gradient-->

<html xmlns:v="urn:schemas-microsoft-com:vml">
<head>
<style type="text/css">
  v\:*{behavior:url(#default#VML);}
</style>
</head>

<body>

<script language=Javascript>
//SET START TIME
var startTime = new Date().getTime()
var endTime = 0

for(var i=0;i<1000; i++)
{
  document.write('<v:oval
style="width:100;height:100"fillcolor="red"><v:fill type="gradient"
color2="#8B0000" method="linear sigma" angle="330"
focus="100%"/></v:oval>')
}

endTime = new Date().getTime()
alert("Time Taken: " + ((endTime-startTime)/1000) + " seconds")
```

```
</script>
</body>
</html>
```

Appendix E: Source Code for Deliverable 4

Source Code for the SVG File

```
<?xml version="1.0" ?>
<svg>
  <rect        x="40"        y="25"        width="300"        height="75"
style="fill:url(#gradient1)"></rect>
  <defs>
    <linearGradient id="gradient1">
      <stop offset="0%" stop-color="red"/>
      <stop offset="100%" stop-color="green"/>
    </linearGradient>
  </defs>
</svg>
```

Source Code for the Stylesheet

```
<!--Author:  Julie Nabong-->
<!--Date:  November 2003-->
<!--Title:  Stylesheet Transformation-->

<?xml version="1.0" ?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:v="urn:schemas-microsoft-com:vml">

<xsl:template match="/">
  <html xmlns:v="urn:schemas-microsoft-com:vml">
  <head>
  <style type="text/css">
    v\:*{behavior:url(#default#VML);}
  </style>
  </head>
  <body>
    <xsl:for-each select="/">
    <xsl:apply-templates select="svg"/>
  </xsl:for-each>
  </body>
  </html>
 </xsl:template>

  <xsl:template match="svg">
  <xsl:variable name="left">
      <xsl:value-of select="rect/@x" />
    </xsl:variable>

    <xsl:variable name="top">
      <xsl:value-of select="rect/@y" />
```

```
  </xsl:variable>

  <xsl:variable name="width">
    <xsl:value-of select="rect/@width" />
  </xsl:variable>

  <xsl:variable name="height">
    <xsl:value-of select="rect/@height" />
  </xsl:variable>

<xsl:variable name="color">
  <xsl:value-of select="defs/linearGradient/stop[1]/@stop-color" />
</xsl:variable>

<xsl:variable name="color2">
  <xsl:value-of select="defs/linearGradient/stop[2]/@stop-color" />
</xsl:variable>

<!--DRAW VML-->

<xsl:element name="v:rect">
    <xsl:attribute name="style">width:
        <xsl:value-of select="$width"/>;height:
        <xsl:value-of select="$height"/>;top:
        <xsl:value-of select="$top"/>;left:
        <xsl:value-of select="$left"/>
      </xsl:attribute>
    <xsl:attribute name="stroke">true</xsl:attribute>
    <xsl:attribute name="strokecolor">white</xsl:attribute>


    <xsl:element name="v:fill">
      <xsl:attribute name="type">gradient</xsl:attribute>
      <xsl:attribute name="angle">90</xsl:attribute>
      <xsl:attribute name="color"><xsl:value-of select="$color"/>
      </xsl:attribute>
      <xsl:attribute name="color2"><xsl:value-of select="$color2"/>
      </xsl:attribute>
      <xsl:attribute name="focus">100%</xsl:attribute>
    </xsl:element>
  </xsl:element>


</xsl:template>

</xsl:stylesheet>
```

Source Code for HTML & JavaScript

```
<!--Author:  Julie Nabong-->
<!--Date:  November 2003-->
<!--Title:  SVG Rectangle Transformation-->

<html xmlns:v="urn:schemas-microsoft-com:vml">
<head><title>SVG to VML</title>
<style type="text/css">
v\:*{behavior:url(#default#VML);}
</style>
</head>
<body onload="transform()">

<script type="text/javascript">

  //LOAD FILES
  var svg = new ActiveXObject("Microsoft.XMLDOM")
  svg.async = false
  svg.load("svg-gradient.svg")

  var xsl= new ActiveXObject("Microsoft.XMLDOM")
  xsl.async = false
  xsl.load("gradient-xsl.xsl")

//TRANSFORMATION
function transform()
{
      document.write(svg.transformNode(xsl))
}

</script>
</body>
</html>
```