Scalable Vector Graphics – XML Solutions for Designing Visual Components in a Web Age

Zvonko R. Milosevic[1] and Goran Janackovic[2]

Abstract - **SVG (Scalable Vector Graphic) is an open-standard vector graphics language that lets you design Web pages with high-resolution graphics including such sophisticated elements as gradients, embedded fonts, transparency, animation, and filter effects, using plain text commands. The Scalable Vector Graphic format is based on XML.**

Keywords - **SVG, Scalable Vector Graphic, XML.**

I. Introduction

The need to display quantitative data stored in XML files is quite common, even when transforming the most basic documents [1]. For example, consider the following cases:

- Number and type of hits registered in a server log;
- Percentage of sales by an individual on an annual sales report;
- Number of technical books vs. the total book count in a book list (almost every XML book in the world has that example);

II. What is SVG?

SVG is a language for describing two-dimensional graphics in XML. SVG allows for three types of graphic objects: vector graphic shapes (paths consisting of straight lines and curves), images and text [2]. Graphical objects can be grouped, styled, transformed and composited into previously rendered objects. Text can be in any XML namespace suitable to the appplication, which enhances searchability and accessibility of the SVG graphics. The feature set includes nested transformations, clipping paths, alpha masks, filter effects, template objects and extensibility. SVG drawings can be dynamic and interactive. The Document Object Model (DOM) for SVG, which includes the full XML DOM, allows for straightforward and efficient vector graphics animation via scripting. A rich set of event handlers such as onmouseover and onclick can be assigned to any SVG graphical object [3]. Because of its compatibility and leveraging of other Web standards, features like scripting can be done on SVG elements and other XML elements from different namespaces simultaneously within the same Web page.

[1]Zvonko R. Milosevic is with the Faculty of Electronic Engineering, Beogradska 14, 18000 Nis, Yugoslavia, E-mail: zvonko@ptt.yu

[2]Goran Lj. Janackovic is with the Faculty of Occupational Safety, Carnojevica 10a, 18000 Nis, Yugoslavia, E-mail: jagor@ptt.yu

There are many advantages of using SVG as the following short feature list demonstrates:

- Scalable Server Solutions
- Compatibility with other mediums such as wireless devices
- Small file sizes for faster Web page downloads
- Zoomable graphics and images
- Scripting control for custom interactive events and animation
- Clean, high-resolution printing from Web browsers
- Text-based format easily integrates with other Web technologies
- Built in International Language Support
- Reduced Maintenance Costs
- Easily Updated

III. SVG Main Features

Flash and SVG are often compared because the two have similar features. The reality is that SVG has some distinct advantages over its main competitor Flash. Perhaps chief among them is the compliance with other standards. SVG can utilize CSS and the DOM, where as Flash relies on proprietary technology that is not open source, at least not in the sense that we can right click on the page and see what is happening behind the scenes. SVG by contrast is open source and developers can readily learn from other developer's efforts in this area. While SVG has not yet reached the popularity level of Flash, times are changing quickly. Mozilla plans to fully support SVG, Microsoft has similar plans, and Adobe GoLive 5 also supports SVG. Additionally, SVG editors are now surfacing on the Web.

There are however some drawbacks and one of the major drawbacks at the moment is that no browser fully supports SVG currently. As a consequence, SVG has to be displayed through the use of a plug-in such as the Adobe SVG plug-in. While it is a good plug-in it does not currently support all the SVG specifications, it is a heavy download, and perhaps the biggest barrier is that it is CPU intensive. Still, despite these drawbacks it does allow for cross-browser implementation of SVG and the use of the plug-in is likely to increase dramatically in the years to come.

Essentially, SVG is a bridge between design and programming because unlike traditional methods of creating graphics, graphics in SVG are created through a programming language. This programming language is XML based and consequently integrates well with other W3C standards such as the DOM. A good way to think of SVG is to think of the browser as being a blank canvas that is defined by a multitude of x and y points. Each point in the canvas can then be utilized to create a shape via a mathematical formula. For example, absolute positioning of CSS layers uses a mathematical equation that allows developers to position a layer where they would like, by assigning left and top property values. Many of the equations used by SVG work on these same underlying principles.

SVG is much like a vector based graphics program; with the exception that it is void of a graphical program interface that one may typically associate with the creation of images. Instead, vector images are created through text based commands that are formatted to comply with XML specifications. In this instance, the code is literally the art, and the brush used to paint the art is XML based. SVG can display 24 bit color with the additional benefit of producing lower weight graphics. The graphics produced by SVG don't lose any quality if they are zoomed or resized. Best of all every element and every attribute of an element can be animated. These are compelling reasons to utilize SVG.

Integrating an SVG is free, and we use Adobe’s plug-in. This is an early release of the next version of the Adobe SVG Viewer, which adds support for more of the W3C's SVG Recommendations. The Adobe SVG Viewer now supports external as well as inline and embedded style sheets. Adobe Illustrator 9.0 automatically exports SVG images with embedded styles, but you can easily design your SVG to work with external style sheets, just like with HTML. Additional arguments are the reference to the SVG file, and the pixel dimensions of the object in the browser page. The Adobe SVG viewer supports US-ASCII, ISO-8859-1, UTF-16, and UTF-8 encodings. To display the characters, the correct font needs to be installed, using a small "CEF" font so that anyone can view the SVG - even if they don't have the font installed.

Conventions have it that SVG files are saved with an .svg extension, and .svgz if compressed for better delivery through Web servers. Of course you are technically free to choose any file name you like. ".svgz" is the appropriate designation for .svg files that have been compressed with "GZIP". Adobe SVG Viewer decompresses .svgz files automatically with no significant difference in performance. When someone uses .svgz files on its site, it is necessary to set the mime type for it as well in the Web server.

IV. SVG Document Structure

There are a couple of ways SVG can be defined in a Web document: as a standalone SVG page, as an embedded element, or it can be utilized in an XHTML document with a namespace declaration [3]. Let us begin by taking a look at the standalone example:

```
1. <?xml version="1.0" standalone="no"?>
2. <!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN"
   "http://www.w3.org/TR/2001/REC-SVG-
20010904/DTD/svg10.dtd">
3. <svg width="100" height="100" x="0" y="0">
4. <!—User defined SVG content -->
5. </svg>
```

Since SVG is an application of XML it must include the initial XML declaration. SVG must be identified by a standard set of rules. These rules are stored in a separate document called a *Document Type Declaration* or *DTD* and is utilized to validate the accuracy of the SVG document structure [4]. The purpose of a DTD is to describe in precise terms the language and syntax allowed in SVG. The <svg> tag denotes to the browser that this is a SVG document. The canvas of the SVG document is defined by the width and height properties. For example, increasing width and height to 500 respectively increases the size of the canvas where the content will be contained. Not defining the width and the height properties cause the SVG canvas to fill the browser dimensions. The x and y properties denote where the canvas will be placed in the browser window. The x property equates to the top position of the browser and the y property equates to the left position of the browser. All the SVG content is placed between the <svg> </svg> tags. Since SVG is an application of XML, all tags must be closed. The </svg> tag closes the document.

This method, while useful for providing standalone examples, has some shortcomings particularly in regard to search engine placement. There is a provision for Meta tags in the SVG specifications, but since SVG is XML based most of the popular search engines will not pick up a standalone SVG page. RDF enabled search engines will however pick up SVG.

Nonetheless, for most of us, being listed by search engines is important and to overcome this problem we can use a combination of HTML/XHTML and SVG. SVG can be also embeded within a HTML or XHTML document by using the following structure:

```
1.  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0
Transitional//EN">
2.  <html>
3.  <head>
4.  <title>SVG</title>
5.  </head>
6.  <body>
7.  <object data="test.svg" width="500" height="500"
type="image/svg+xml">
8.  <embed src="test.svg" width="500" height="500"
type="image/svg+xml" />
9.  </object>
10. </body>
11. </html>
```

The document is a straightforward HTML document. The important tags in the above example are the object and embed tags. If we wanted to adhere strictly to standards based coding then we would only use the object tag, but using this tag *only* causes the SVG file to not appear in Netscape version browsers. As a consequence it is best to use both the object and embed tags or just the embed tag. Lines 7 through 9 contain the appropriate object and embed tags. It is important to note that the object tag uses the data property to specify the url of the SVG document while the embed tag employs the src property.

One of the benefits of utilizing this method is that it is able to combine the advantages of HTML and XHTML with SVG.

For example, the chances of your pages being picked up by search engines are considerably enhanced when employing a mixture of HTML / XHTML and SVG. It is also easier to integrate sound and music capabilities utilizing this method (note: The Adobe plug-in provides support for MP3 format and WAV files). There is another way to allow SVG to be displayed by browsers and that is through the use of XML namespaces.

V. Graphics Structure

Similar to most vector drawing packages, SVG has some predefined basic shapes that can be utilized by developers. These shapes are elements, much like a table can be an element of a HTML document [3]. The following shape elements are defined by the SVG specifications:

- Rectangle <rect>
- Circle <circle>
- Ellipse <ellipse>
- Line <line>
- Polyline <polyline>
- Polygon <polygon>

The <rect> tag allows for the drawing of a rectangle and variations of a rectangle shape, for example squares and rectangles or squares with rounded corners. For example, a basic rectangle shape can be created by the following code:

```
<rect x="80" y="53" width="189" height="52"
style="fill:rgb(39,44,231);
stroke:rgb(0,0,128);
stroke-width:1"/>
```

The tag begins by first declaring the <rect> tag and then declaring The style declaration allows developers to define CSS properties that are supported by the SVG recommendations. There are quite a number of CSS properties specific to SVG. The fill property (fill:rgb(39,44,231);) defines the fill color of the rectangle; in this particular instance rgb format is used to express the color. SVG also allows for the expression of colors by using named or hexadecimal color formats as such: fill: red or fill: #ffff00. The stroke property provides Web developers with a mechanism to create an outline for a rectangle. In SVG it is assumed that an outline does not have any width. In other words the width of an outline is set to 0 by default. This is true of all SVG shapes and as a consequence if an outline is needed it needs to be defined by using the following syntax: stroke:rgb(0,0,128); which defines the color to be used for the stroke. stroke-width:1 defines the width of the outline. Increasing the width value to 5 intuitively would produce a larger outline.

The opacity property demonstrates a very important facet of SVG, which often is not immediately noticed. It is not only the element itself that can be altered, but its sub-components as well - which makes them ideally suited to dynamic manipulation.

Another way of thinking about this is by comparing it to Flash. In Flash you have the ability to alter a symbol's transparency and color values, but in order to achieve the same effect in Flash, each of a rectangle's strokes must be converted to symbols as well as the fill itself, thereby increasing the file size of the movie. In SVG the same effect is achieved with minimal code and consequently does not greatly impact the file size. There is, however, the ability to do far more than create basic lines in the SVG specifications.

VI. Filter Effects

In SVG there exists the ability to add effects directly to shapes and text. The best way to think of filters is to think of the filters used by a graphics editing and creation program like *Adobe Photoshop* or *Macromedia Fireworks*. In these programs one has the option to apply a Drop Shadow effect or Gaussian Blur effect on a graphic. SVG has the same capabilities, thus bringing with it the same features typically associated with bitmap images (gif, jpg, png, etc). Filter effects can consist of any combination of the above. In other words it is possible to use multiple filters on a vector image. It easy to realize the many possible permutations involved when using filter effects.

An SVG filter effect must be nested within the <defs> element. The <defs> tag is short for definitions, and, as the name suggests, its purpose is to allow for the definition of special elements, such as a filter. The filter itself is defined through the <filter> tag. A requirement of the filter tag is that it also must contain an id attribute. With SVG filters, the id attribute is used to identify which filter will be applied to the graphic.

The purpose of giving a filter a unique id is then to be able to use that same filter repeatedly on many elements. In other words, the <filter id="Gaussian_Blur1"> acts as a template that can be repeatedly used throughout a SVG document. To apply the filter to an element an xlink is used. An xlink is just an expression used in SVG that is the equivalent of a link in HTML e.g., <a href="#">. In fact the syntax to link to other files at another url is very similar to HTML and should not present too many problems. To link the element to a filter, the filter:url(#Gaussian_Blur1) property is used. The # character must be used when linking back to the filter's id. Linking is an important concept to understand in SVG as it will save many hours of repetitive coding and development.

VII. Gradients

One of the most visually appealing facets of SVG comes through its ability to create gradients. A gradient is a smooth transition from one color to the next. In addition, several color transitions can be applied to the element making for some striking effects.

There are two main types of gradients available to SVG [3]. These are:

- Linear Gradients
- Radial Gradients

Linear gradients can be defined as horizontal, vertical or angular gradients. As is the case with all SVG elements, style attributes such as opacity can be applied to gradients. The other type of gradient allowed in SVG is the radial gradient. The tag to define a radial gradient is <radialGradient> and like the <linearGradient> tag it also must be nested within the <defs> tag.

VIII. SVG File Protection

Today, developers have tried to lock SVG files in a number of ways. Locking out the "show source" menu does not provide true security, because any file on the Web that can be seen by your browser can be downloaded to a machine. From there, opening an unencrypted file with any text editor is routine. Some developers might rely on legal protections. Others will look upon SVG as they do with HTML or JavaScript.

There are clearly some instances where people will want, or need to have, some form of Digital Rights Management (DRM). There are a variety of possible schemes for providing DRM for SVG files.

IX. Conclusion

The use of XSLT and SVG opens up exciting new ground for the presentation of XML data on the Web [5]. The correct use of these tools may improve vastly the quality and quantity of information your users can consume, as well as your process to present and create it. The creation of good visual representations of XML data using XSLT is governed by principles and best practices both on the programming and technical graphic design sides. In this article we have examined a few of them while providing an illustration of their implementation. There are a few tips for using SVG:

- Try to maximize space/information ratio, since the best graphics use fewer pixels to express more data.
- Create reusable graphic templates by induction. Start with a prototype of what you want to do in plain SVG, then create a basic XSLT template, possibly tightly coupled with the rest of the stylesheet, finally, and only if it is general enough, factorize into a reusable library.
- Try to provide representations that truthfully represent the nature and dimension of your input. Don't use n-dimensional representations for m-dimensional data (variations would be deceiving).
- Use the possibilities of SVG to elegantly escape "flatland". Transparencies, JavaScript tool tips, and dynamic insertion of elements are great tools to allow several layers of information to coexist without cluttering. Using these tools is harder than overloading your graph or using disruptive mechanisms like alert boxes, but the extra effort on your part will be reflected on the quality and usefulness of your graphic and surely appreciated by your clients.

SVG is a new graphical file format, based on XML, that flexibly incorporates vector graphics, bitmap graphics, text and style sheets. It's "scalable" in more than just the sense that the incorporated graphics can be scaled. It's also scalable in its flexibility. Because it's XML-based, SVG can be mixed with other formats — such as XHTML — and scripting languages — such as JavaScript. Entire Web pages or individual components such as graphics could be rendered with SVG.

The very presence of a format for vector graphics on the Web is significant. For all its promise as a graphical user interface and publishing medium, the Web has relied too heavily on bitmap graphical formats such as GIF and JPG. Bitmaps have inherent limitations; they tend to be static and difficult to reuse, and they often need to be optimized for the particular screen resolution of the displaying device. As a result, most Web sites are laden with single-use graphic files, and graphics are seldom used to personalize the presentation of material. Moreover, with the growth of non-PC devices such as PDAs and cell phones, the limitations of heavy bitmaps become even more pronounced.

The vector images that already exist on the Web largely have been rendered as Macromedia Flash illustrations and animations. Yet for all the, well, "flash" of Macromedia's authoring tools, the format has never proliferated all that widely. Internet Explorer has included support for its own vector format, VML. The compelling thing is that SVG is an entirely open, textual format. It can be easily generated from a database for applications such as dynamic page serving. The obvious solution is an open standard that can be created by many tools.

References

[1] Randall M. Rohrer and Edward Swing, “Web-Based Information Visualization”, *IEEE Computer Graphics and Applications*, pp. 52-59, July/Aug. 1997.
[2] L. Wood, “Programming the WEB”, *IEEE Internet Computing*, pp. 48-54, Jan/Feb 1999.

Web References

[3] SVG 1.0 W3C Recommendation , 2001. www.w3c.org
[4] SVG DTDs W3c Recommendation, 2001. www.w3c.org
[5] C.M. Sperberg- McQueen, XML-related Activities, 2001. www.xml.com