Using SVG as the Rendering Model for Structured and Graphically Complex Web Material

Julius C. Mong and David F. Brailsford
Electronic Publishing Research Group
School of Computer Science & IT
University of Nottingham
Nottingham NG8 1BB, UK
{jxm,dfb}@cs.nott.ac.uk

ABSTRACT

This paper reports some experiments in using SVG (Scalable Vector Graphics), rather than the browser default of (X)HTML/CSS, as a potential Web-based rendering technology, in an attempt to create an approach that integrates the structural and display aspects of a Web document in a single XML-compliant envelope.

Although the syntax of SVG is XML based, the semantics of the primitive graphic operations more closely resemble those of page description languages such as PostScript or PDF. The principal usage of SVG, so far, is for inserting complex graphic material into Web pages that are predominantly controlled via (X)HTML and CSS.

The conversion of structured and unstructured PDF into SVG is discussed. It is found that unstructured PDF converts into pages of SVG with few problems, but difficulties arise when one attempts to map the structural components of a Tagged PDF into an XML skeleton underlying the corresponding SVG. These difficulties are not fundamentally syntactic; they arise largely because browsers are innately bound to (X)HTML/CSS as their default rendering model. Some suggestions are made for ways in which SVG could be more totally integrated into browser functionality, with the possibility that future browsers might be able to use SVG as their default rendering paradigm.

Categories and Subject Descriptors

E.1 [Data]: Data Structures — *Trees*; I.7.2 [Document and Text Processing]: Document Preparation — *Markup languages*; I.7.4 [Document and Text Processing]: Electronic Publishing.

General Terms

Algorithms, Documentation, Experimentation.

Keywords

PDF, SVG, XML, vector graphics.

1. INTRODUCTION

Current Web browser technologies are routinely based around the HTML markup language (or, more recently, its cleaned-up XML-compliant version called XHTML).

FINAL DRAFT of paper accepted for
DocEng'03, November 20–22, 2003, Grenoble, France.
Copyright 2003 Mong and Brailsford.

For enhanced Web page styling that goes beyond the XHTML defaults there is the option to use the Cascading Stylesheets (CSS) facilities. CSS can also be used to style arbitrary XML-based documents within the current generation of Web browsers.

From the outset, Web browsers have supported only simple bitmap formats for graphical material, such as GIF and JPEG, despite the fact that raster files in these formats are large and the resulting bitmap graphics are resolution-dependent. The World Wide Web Consortium (W3C), aware of the need for a flexible vector graphics format for the Web, set up a working group in 1998, charged with drawing up draft proposals for Scalable Vector Graphics (SVG) to be expressed in XML-compliant syntax. The advantages of vector graphics in terms of graphical precision and scalability become very apparent in material such as maps, line-diagrams and block schematics. The involvement of Adobe Systems Inc on the SVG working group did much to ensure that although SVG was *syntactically* XML-compliant the actual graphical *semantics* of its behaviour resembled, fairly closely, those of its PostScript and PDF page description languages.

SVG has predefined graphics elements, such as lines, rectangles and circles, which can be combined to create simple diagrams. For more sophisticated designs, operations are available for specifying an individual object's fill rule, colour space, transformation matrix, etc. SVG is also capable of placing text strings at arbitrary positions within the display of material on the screen. Indeed there is little to prevent it being used as a page layout language, very much like PostScript itself. Furthermore, since SVG code is itself text based, it is possible for search engines to easily index and search within any SVG material.

2. INVESTIGATION

Despite the fact that SVG is an application of XML it is unusual in that its tags describe low-level graphic primitives rather than the abstract logical structure more generally modelled in other XML applications. Thus some other XML-based tagset has to be devised to delineate the higher-level document architecture that lies behind the presentational use of SVG.

With the help of all the characteristics and tools of XML, such as *well-formedness*, *namespaces*, *schemas* and *DOM*, document structure can be presented as a traversable tree, because the structure of XML itself is fundamentally tree-oriented. With this in mind we have performed several experiments to investigate the feasibility of using an SVG rendering mechanism, coupled with a simple XML-based tagset for document structure, to create an integrated and totally XML-based electronic document.

We have created an Acrobat plug-in to carry out five levels of conversion from PDF to SVG. The first level is a direct conversion of PDF (both structured and unstructured) content streams into plain SVG, with one SVG document corresponding to each page of the original PDF and with no attempt being made to map any PDF structure tree that might be present. The second level involves iterating through the PDF structure tree and converting the marked-content sequences at every leaf into SVG.

The third level traverses the PDF structure tree and incorporates into the SVG document all the PDF structure elements, mapped as XML tags and named according to the PDF Standard Structure Types (SST) [2], while extracting content sequences at every leaf. The resulting SVG document has custom tags at various places which mark the logical structure using an XML version of the SST (see example below). We have written a Schema for this custom tagset to be used as a validation reference by browsers and XML parsers. Note that this XML custom tagset has its own namespace in order to avoid interference with the SVG tagset when the two are intermixed in an SVG document.

```
<svg xmlns="svg-URI" …>
  <sst:H1>
    <text …>
       <tspan …>T</tspan>
       <tspan …>est Do</tspan>
       <tspan …>cument</tspan>
    <text …>
 </sst:H1>
 …
</svg>
```

The fourth level of our conversion experiments involved writing an XML structure tree layout out the original PDF document structure into a "structured" SVG document labelled with the SST tagset, while using XLink pointers at the leaves of the XML tree as the referencing mechanism for relating logical structure to actual contents stored in an external "unstructured" SVG document obtained directly from the level 1 conversion.

In the final level, not only is structure information incorporated into the output document, individual SVG elements are also presented as separate graphical inserts. This level extends the previous levels further by inserting the "unstructured" contents as `<defs>` elements into the "logically-structured" output document, imitating the PDF architecture of having the logical structure separate from the content streams.

```
<svg xmlns="svg-URI"
  xmlns:sst="sst-URI"
  xmlns:xsi="xmlschema-URI" … >
 <sst:page number="1">
  <sst:H1>
   <svg viewBox="0 0 595 842">
     <use xlink:href="#txt0">
   </svg>
  </sst:H1>
  …
 </sst:page>
 …
 <defs>
  <text id="txt0" … >
    <tspan …>T</tspan>
    <tspan …>est Do</tspan>
    <tspan …>cument</tspan>
  </text>
</svg>
```

The intention of this approach is to propose SVG as a *final-form* graphic format, and as a potential Web rendering mechanism. In this way an entire document, containing single or multiple pages of very high quality content, can be self-inclusive as a single document file. This mechanism would be particularly useful if it were to work in Web browsers. It would enable, for example, SVG graphical inserts to be placed within a logically structured and paginated single XML document. Each of the inserts would have its own graphics state and could be scaled, animated, viewed independently and reused elsewhere in the document, provided that viewer/browser support is available.

3. FINDINGS

We carried out a number of conversion experiments on both unstructured and structured PDF documents. These conversions involve direct extraction of content streams, iteration through the PDF structure tree, and the extraction of marked-content sequences at each tree node. It was found that a PDF document, with or without logical structure, can be converted into plain SVG, with very few problems at the content stream level. However a structured (Tagged) PDF may present some difficulties when converted at levels 2 and above (as defined in the previous section). For instance, a traversal of the structure tree in a correctly tagged PDF document should result in the correct reading order for the textual content, but this is not always aligned with the painting, or rendering, order of the various components.

By exploiting the differences between rendering and reading orders a PDF document can achieve certain graphic effects (e.g. object *A* partly superimposed on top of object *B*) while still asserting, via the PDF structure tree ordering, that *A* should be read out *before B*. In an XML/SVG equivalent to this PDF the reading order and the rendering order are necessarily the same since everything is controlled by a top-down descent of the XML tree. Thus, if the PDF to XML/SVG translation is guided by the PDF structure tree, rather than the ordering of *A* and *B* in the PDF file, then *B* will be superimposed on *A*, instead of the other way around.

3.1. Painting Order

Apart from occasional problems of content ordering, as outlined above, it is generally the case that a direct conversion of the content streams of a Tagged PDF document — which may contain text, line diagrams, hyperlinks (converted from PDF annotations into SVG `<a>` elements), tables (treated as text and line diagrams) and raster images — will result in an SVG rendering which closely matches that of the PDF.

SVG 1.2 [1] proposes adding the ability to specify the *Z-order* (painting order) of individual elements. Since the painting order can be obtained implicitly from the PDF content streams, it would be possible to specify the Z-order in the converted SVG once SVG 1.2 becomes a W3C Recommendation. However, with the current version of PDF and SVG, and with the painting order being determined implicitly from the ordering of content sequences in a content stream, synchronisation of the reading order specified by the structure tree and the painting order of marked-content sequences must be enforced in order for structured PDF to be converted with maximum precision.

3.2. The Page Concept

Pages in PDF are currently imitated in SVG by creating an individual SVG document for every page of the original PDF.

There is however a proposal in SVG 1.2 for a page model so that SVG drawings can be defined on its own `<page>`, each enclosed in a `<pageset>` collection [5]. This feature will further extend the use of SVG as a Web presentation language.

3.3. Content Reflow

It has also been proposed that SVG 1.2 should provide functionality to wrap SVG elements so that they can be reflowed as necessary, for example, in the case when a `flowRegion`, containing shapes, or a `flowText`, containing text is transformed via transformation matrices.

The proposed "flow" elements in SVG 1.2 can be used to wrap text in a shape, or shapes in a region, so that they can be reflowed individually when their wrappers are transformed, but an SVG document containing a collection of shapes, text and miscellaneous elements cannot be reflowed as a whole according to the display screen size or resolution. Thus the reflow proposals, as currently formulated, would still encounter problems if SVG were displayed on, say, a Personal Digital Assistant (PDA).

Since there is no native method for accessing system environments in SVG, the reflow of whole SVG drawings can, at present, be done only via scripting; a `<script>` element can contain JavaScript to explicitly transform the viewport and all the coordinates and dimensions of elements within the SVG document. Although W3C has produced *SVG Tiny* and *SVG Basic* [4] to address some issues of displaying SVG on mobile devices; these do not yet provide a solution to the reflow limitation.

3.4. Browser Limitations

To view SVG documents, either a Web browser plugin or a standalone viewer application can be used. Adobe has developed an SVG viewer plugin (currently version 3) that relies heavily on the Web browser's CSS-oriented XML rendering mechanism. For instance, Internet Explorer displays general XML documents in an expandable tree form whenever it does not recognise the tagset as one of its built-in document types (e.g. (X)HTML); the Adobe SVG plugin builds on this rendering mechanism with the result that SVG rendering is turned off for any tree branches that are found not to be pure SVG. Thus, if SVG tags are intermixed with other XML tags then the entirety of the coding is hierarchically displayed but interpretation of the SVG portions is disabled. For this reason the SVG output documents produced by conversion levels 2 to 4, as defined in the previous section, will have all or part of the document ignored by the Adobe SVG viewer.

The behaviour of the Adobe SVG Viewer, in the way it handles unrecognised tagsets, follows the 'letter of the law' as set out in the SVG 1.1 specification. However, the spirit of what we need seems to be better addressed in W3C's Amaya browser; this interprets SVG natively and is designed to render all W3C technologies, even though it started life as an HTML/CSS editor-browser. When given the scenario described above, Amaya simply ignores any tags that are not SVG and displays the drawing as a pure SVG document. The results therefore seem promising in that the logical structure tagset we incorporated does not affect the appearance aspects of the SVG, but does contain the essential document structure information should it be required.

W3C provides an SVG Test Suite [3] that includes a test utilising XLink in `tref` elements (similar to `use` in our conversion). Both the Adobe SVG Viewer and Amaya do not display the `tref` elements referencing text stored in a separate file; the text elements are simply ignored. However, we understand that this shortcoming has now been addressed in the most recent release of the Adobe SVG viewer.

4. CONCLUSION

On the purely syntactic front the `svg` namespace, if used correctly, can happily co-exist with others. The difficulties we have encountered arise because SVG is more than "just another XML-based tagset". It is one that fundamentally affects the way in which Web browsers render documents, and SVG is very different in this regard from (X)HTML/CSS.

It must be remembered that SVG's original place in the scheme of things was to provide high-quality vector-graphic inserts into material that was predominantly (X)HTML/CSS based. If we expand the bounding box of the "SVG insert" so that it takes over the rendering of the whole screen then its geometrically bounded nature prompts us at once into thinking of "pages" just as the original PDF material we have translated was also page based. But that, in turn, leads us to the fact that the Web, at the moment, has no capabilities for understanding the concept of "page".

However, we are greatly encouraged by the success of our plug-in in being able to translate a wide variety of PDF features directly into SVG graphic equivalents. It is also easy to appreciate the attraction of mapping the PDF structure, as we have done, into a supporting abstract XML structure behind the SVG.

The results from the Amaya browser also begin to show how this combination could work well provided the SVG 1.2 specification moves in the right direction. It should be possible to validate any custom (i.e. non-SVG) XML tags against a DTD or Schema, while having the SVG viewer ignore them so that it can concentrate on the SVG content alone.

From our investigations into combining SVG and XML, and the extent to which such a combination can be used as the default rendering mechanism in Web browsers, we are confident that, as technologies such as SVG, XLink and implementations of SVG rendering gain further stability and maturity, SVG will attract increasing attention and serious consideration of its possible role as a high quality alternative for many Web presentations that are currently dominated by HTML and CSS.

5. ACKNOWLEDGMENTS

We thank Adobe Systems Inc. for taking an interest in Julius Mong's Ph.D. studies. In particular we would like to thank Jon Ferraiolo, Jim King and Steve Zilles for answering technical questions on PDF and SVG.

6. REFERENCES

[1] *W3C Working Draft of Scalable Vector Graphics 1.2*
http://www.w3.org/TR/SVG12/

[2] *Tagged PDF – Standard Structure Types*
PDF Reference (Third Edition) version 1.4 (Chapter 9.7.4), Adobe Systems Incorporated,
ISBN 0-201-75839-3, Addison-Wesley, December 2001.

[3] *W3C SVG Test Suite*
http://www.w3.org/Graphics/SVG/Test/

[4] *Mobile SVG Profiles: SVG Tiny and SVG Basic*
W3C Recommendations 14 January 2003
http://www.w3.org/TR/SVGMobile/

[5] SVG 1.2 – Multiple Pages
http://www.w3.org/TR/SVG12/#multipage