DVI2SVG: Using LaTeX layout on the Web

Adrian Frischauf and Paul Libbrecht
German Research Center for Artificial Intelligence
Stuhlsatzenhausweg 3
66123 Saarbrücken
Germany
adrianf (at) activemath dot org, paul (at) activemath dot org
http://www.activemath.org/~adrianf

Abstract

The problem of presenting mathematical formulas on the Web is non-trivial. Current systems offer only partial answers to such requirements as the guaranteed layout on the client side or the availability of font glyphs. We describe DVI2SVG, a system to convert TeX's output into Scalable Vector Graphics. This approach responds to the requirements above and several others. We also present how it has been put to use in ActiveMath, a learning environment on the Web which presents mathematical documents personalized to each learner.

1 Different approaches to supporting mathematics on the Web

Classically, learning content is presented on the Web using the HTML format. This format, however, is unable to provide rich graphical constructs that are needed to render normal mathematical expressions. HTML's layout capabilities are limited, unable to fully render such constructs as the square-root or a fraction with proper baseline alignment. Mathematical formulas also often use characters which may be unavailable on some operating systems while HTML offers no method to ensure that a given font glyph will be available.

Using images for formulas solves these two issues but introduces several other problems. The resulting formula has no way to align properly inside a line of text, is not scalable, and does not adapt to changing text size. Moreover the separate parts of the formula can not be addressed as different objects by interactive scripts running on the client.

Presentation MathML [3] has the potential to realize a full featured mathematical presentation but current browser support has several drawbacks. As with HTML, the font glyphs have to be available to display the special characters correctly.

Another possibility for the delivery of mathematical content over the Web is to use PDF documents. This has no problem with fonts or layout but the PDF document does not allow much interactivity. As an "E-Paper" it is an offline resource and not an online presentation format.

Our DVI2SVG implementation has thus tried to answer the following requirements:

- provide layout quality as high as that of LaTeX, for text, formulas, and mixtures of both;
- deliver the content with guaranteed availability of font glyphs when presented to the client;
- present the content on a platform which can be dynamically scripted.

Moreover, we wished to integrate such a solution in the ActiveMath learning environment which combines and caches individual paragraphs before being personalized and delivered to the clients.

The specification of clients for the Scalable Vector Graphics provides an answer to all these requirements and was chosen for this reason.

2 The Scalable Vector Graphics format

The SVG file format [4] is an XML [1] language for describing two-dimensional vector graphics. It was issued as a recommendation by the W3C in 2003. This format allows for easy editing by hand, as well as easy generation, because of the many libraries available for manipulating XML. SVG allows font glyphs to be embedded within the document presented, and supports and specifies document object model access through a scripting API. Because of its graphical nature, the SVG format is able to display a complete layout faithfully even though it is unable to compute the layout itself. We thus put to use the well-known quality of LaTeX layout to create a document rendered using the modern SVG specification.

3 DVI2SVG

DVI2SVG is a converter for DVI files, the format output by TeX and LaTeX. It is written in Java and processes streams of DVI tokens. It parses the DVI

input file and generates events, each event representing a command of the input file and holding the current state of the page (position, font, etc.). The Writer interprets the commands and produces the vector graphics XML.

The DVI format, in contrast to SVG, is a document format with multiple pages. For each page of the input, DVI2SVG produces a separate SVG file. Each of these pages contains a header with the font definitions for this specific page. Since the fonts tend to be large, only the font glyphs used in the page are actually included. Especially with fonts used for the formulas, partial glyph embedding saves a large amount of space, since typically only a few characters of each math font are used in a page.

DVI2SVG makes use of the classical TEX fonts as translated to SVG by Michel Goossens [5], whose script converts entire fonts into their SVG glyph equivalents.

The TEX character encoding is used within this conversion. This poses a problem since the TEX fonts contain character codes which are invalid in an XML document. The '`&#06;`' (the Sigma character Σ) is an example of such. The solution used is to map the TEX character codes to the Unicode private area above `0xE000`. No special characters which break the document occur. The resulting SVG source document is not human readable and is also unusable for Web-robots. This issue is only temporary, as implementations such as the Hermes translator[1] show that it is possible to extract good Unicode text from DVI output.

As a command-line tool, DVI2SVG can be used to process static DVI files and publish scientific documents in SVG on the Web.

3.1 Support for additional LATEX packages

In addition to the basic DVI commands, DVI2SVG also supports additional features. LATEX packages such as `color`, `hyperref` or `graphicx` use special commands to enrich a document. They are also translated to SVG.

A TEX command such as `\special{abc}` is copied as ASCII text into the DVI file as a special event. DVI2SVG defines a custom language which it is able to interpret. Since SVG is an XML format, the language of the specials is closely related to that. Such a special command in the DVI file looks like:

```
svg: rect @x=0 @y=0 @width=10 @height=10 /rect
```

This is transformed into an XML fragment:

```
<rect x="0" y="0" width="10" height="10"/>
```

[1] Hermes is a translator from LATEX to XHTML+MathML, see `http://hermes.roua.org/`.

It is thus possible for an author with little knowledge of SVG to enrich the document with the graphics and interactivity that the SVG format supports.

Using this protocol, drivers for the `color` and `graphicx` packages have been designed for DVI2SVG. By compiling LATEX documents to DVI using these drivers, pictures, text in colors, rotated texts, etc., can be embedded using the same macros as those used, for example, for PDF documents. The supported image file formats include SVG, GIF, JPEG, and PNG.

The creation of links with the `hyperref` package is also supported. Since DVI2SVG produces one file per input page, in-document links will be translated into links to the SVG file for the corresponding page.

4 Integration with ActiveMath

ActiveMath is a Web-based intelligent learning environment [8]. It presents mathematical documents transformed from OMDoc [6], an XML language for semantically representing mathematical documents. The mathematical formulas in OMDoc are encoded in OpenMath [2].

The presentation engine of ActiveMath [10] has been designed to support the dynamic generation of content presentation from OMDoc fragments such as definitions or examples. It generates documents in different output formats such as HTML, XHTML + MathML, SVG, and PDF.

It first extracts the OMDoc fragments, injects related objects, and applies an XSLT transformation: this is done exactly once per fragment and per language, and is cached. Once queries from learners arrive, this cached result is interpreted to inject personalization parameters:

- special mathematical notations are chosen depending on the context and user;
- fragments are presented in an order that may be particular to the learner (the learner can edit his own books, and have them created by a course generator [9]);
- exercise links have to be generated with user information;
- traces of the learner-model are output within the presentation, which helps the learner's motivation and tracking of his progress.

The resulting SVG fragments are assembled using stream combination and the high-performance

template engine Velocity.[2] This architecture realizes an effective simultaneous delivery to classrooms of learners.

The resulting presentation is enriched with interactivity: for example, at the time of fragment extraction, each mathematical symbol used semantically in the OMDoc source is annotated with its title. The XSLT transformation outputs the necessary `\special` commands so that scripting code on the client brings up a tooltip-like layer above the presented symbol. Potentially, other interactive features made possible by the semantic nature of the source could be provided, for example, the formula sub-term highlighting, context menus, and drag-and-drop presented in [7].

4.1 Example conversion

We present the processing steps into SVG and see where caching is possible and where personalization happens. In the first step, the sources are fetched from the database. Such a source might look like:

```
<definition for="SVG">
  <metadata>
    <Title>
    Definition of SVG
    </Title>
  </metadata>
  <CMP xml:lang="en">
    SVG stands for Scalable Vector Graphics
    and is a Web graphics format.
    <OMOBJ>
      <OMA>
        <OMS name="divide"/>
        <OMI value="1"/>
        <OMA>
          <OMS name="sqrt"/>
          <OMI value="2"/>
        </OMA>
      </OMA>
    </OMOBJ>
  </CMP>
</definition>
```

Then, using XSLT, this source is transformed into the following LaTeX fragments:

```
...
\begin{fragment}
\section*{Definition of SVG}
SVG stands for Scalable Vector Graphics
and is a Web graphics format.
$\frac{1}{\sqrt{2}}$
\end{fragment}
...
```

Such a fragment is translated to SVG as follows. For readability, the private Unicode-range text parts were replaced by their ASCII representation. In this SVG document one can see the fine control over the horizontal positioning of each glyph (in the precise `x` values), one of the major ingredients of LaTeX's layout's quality.

```
<rect x="0" y="0" width="468" height="690"
  stroke="none" fill="none" />
<g>
  <text font-family="CMBX" font-size="16.97">
    <tspan y="99.30" x="61.69 76.32 85.02
     95.63 106.24 111.54 118.96 124.26 133.81
     150.77 160.31 172.51 183.11 197"
     >Definition of SVG</tspan>
  </text>
  <text font-family="CMR" font-size="11.78">
    <tspan y="125.20" x="61.69 68.1 76.43
     89.32 93.88 98.36 104.13 110.54 116.95
     125.34 128.87  134.64 142.97 149.38
     154.50 160.27 163.48 169.25  175.65
     178.86 187.83 195.52 200.65 205.77
     210.26 216.03 224.36 233.41 237.90
     243.66 250.07 256.48 259.69 264.81
     273.21 278.98 285.39 295.64 298.85
     307.24 316.86 327.75 332.87 343.13
     348.90 353.38 359.15 365.56 371.97
     375.17 380.30 388.70 392.22
     397.99 402.48 412.09 417.86 422.34"
     >SVG stands for Scalable
      Vector Graphics and is a Web
      graphics format.</tspan>
  </text>
  <rect x="431.85" y="122.062"
    width="11.13" height="0.39"
    fill="Black" stroke="Black"
    stroke-width="0.1" />
  <text font-family="CMR" font-size="7.85">
    <tspan y="120.56" x="435.33">1</tspan>
  </text>
  <rect x="438.81" y="123.20" width="4.17"
     height="0.35" stroke-width="0.1"
     fill="Black" stroke="Black" />
  <text font-family="CMSY" font-size="7.85">
    <tspan x="431.9" y="123.6"
      >&#0xE017;</tspan>
  </text>
  <text font-family="CMR" font-size="7.85">
    <tspan y="130.08" x="438.81">2</tspan>
  </text>
</g>
```

These fragments are embedded into a document and are then processed into DVI using LaTeX. The conversion to SVG files using DVI2SVG is invoked. Similar to caching of HTML fragments (for serving HTML to clients), the SVG fragments are now cached.

In the last step, the SVG fragments are assembled to build a page using Velocity. The Velocity

[2] See http://jakarta.apache.org/velocity/

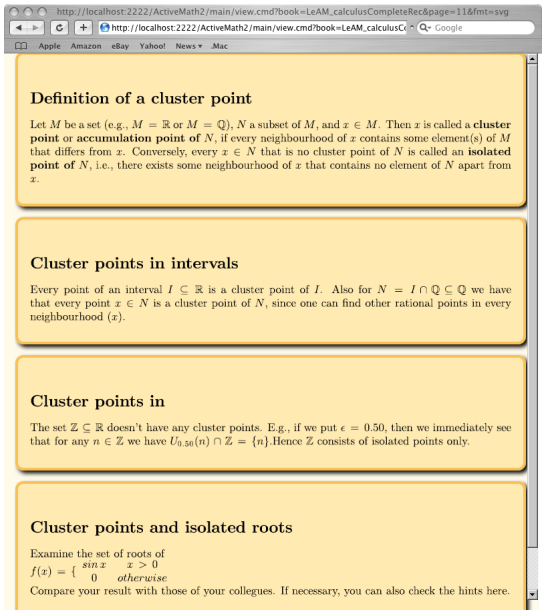

Figure 1: An ActiveMath page using SVG.

page provides an SVG skeleton, includes the fonts, and embeds the other SVG fragments. This step not only collects all the items for the page but also determines the final appearance of the page and the personalized appearance of the items. In the situation shown in Figure 1, for example, yellow shaded rounded rectangles are put around each item.

5 Conclusion

We have presented the DVI2SVG processor and how it has been integrated into ActiveMath.

Compared to other approaches of putting mathematics on the Web, DVI2SVG appears to offer interesting promises:

- Compared to PDF-based solutions, DVI2SVG offers richer interactivity, being based on standard scripting features.
- Compared to other solutions to convert TEX-based files to HTML or MathML, DVI2SVG ensures available fonts and the highest quality of layout provided by the classical (LA)TEX algorithms.
- compared to approaches which make use of the Flash player,[3] DVI2SVG is more open due to using its XML format; moreover, fragments are easier to combine. A feature of the Flash player format that we have not yet found in SVG, however, is the ability to embed an SVG document within another SVG document and preserve all interactivity in the embedded document; this could have avoided the repeated delivery of the SVG fonts for TEX.

SVG appears to be a real opportunity for Web-based presentation of TEX documents for the future. For now, some drawbacks remain: mainly that SVG players which have to be installed before one can use any SVG abilities; SVG support is emerging in developer versions of the Mozilla and Safari browsers,[4] but this support is incomplete; in particular, it is lacking the ability to render embedded fonts, a fundamental ingredient of the DVI2SVG approach and the only way to ensure that all font glyphs will be available on the client.

At present, the Java-based SVG viewer Batik[5] and the latest version of the Adobe SVG plugin,[6] which is available only for Windows, are the only players which work well with DVI2SVG.

We hope to see a renewal of the Adobe family of plugins following the merger of the two competing vector graphics leaders: Adobe and MacroMedia could bring the widely available Flash vector graphics plugins to fully support the SVG format.

References

[1] T. Bray, J. Paoli, and C. M. Sperberg-McQueen. Extensible Markup Language (XML). W3C Recommendation PR-xml-971208, World Wide Web Consortium, December 1997. http://www.w3.org/TR/PR-xml.html.

[2] Stephen Buswell, Olga Caprotti, David Carlisle, Mike Dewar, Marc Gaëtano, and Michael Kohlhase. The OpenMath standard, version 2.0. Technical report, The OpenMath Society, June 2004. Available from http://www.openmath.org/.

[3] D. Carlisle, P. Ion, R. Miner, and N. Poppelier. Mathematical markup language, version 2.0, 2001. http://www.w3.org/TR/MathML2/.

[4] Jon Ferraiolo, Jun Fujisawa, and Dean Jackson. Scalable vector graphics (SVG) 1.1 specification. Technical report, World Wide Web Consortium, 2003. Available from http://www.w3.org/TR/SVG11/.

[3] The Flash player presents animated vector graphics using a widespread plugin; see http://www.macromedia.com.

[4] For the current status, see http://www.mozilla.org/projects/svg/status.html and http://webkit.org/projects/svg/status.xml.

[5] The Batik SVG toolkit is available at http://xml.apache.org/batik; it is a Java library which supports downloaded fonts but with limited performance.

[6] The Adobe SVG plugin is available at http://www.adobe.com/svg.

[5] Michel Goossens and Vesa Sivunen. LaTeX, SVG, fonts. *TUGboat*, 22(4):269–281, 2001. Available from `http://tug.org/TUGboat/Articles/tb22-4/tb72goos.pdf`.

[6] M. Kohlhase. OMDoc: Towards an OpenMath representation of mathematical documents. Seki Report SR-00-02, Fachbereich Informatik, Universität des Saarlandes, 2000. See also `http://www.mathweb.org/omdoc`.

[7] Paul Libbrecht and Dominik Jednoralski. Drag and drop of formulae from a browser. In *Proceedings of MathUI'06*, August 2006. Available from `http://www.activemath.org/~paul/MathUI06/`.

[8] E. Melis, G. Goguadze, M. Homik, P. Libbrecht, C. Ullrich, and S. Winterstein. Semantic-aware components and services of ActiveMath. *British Journal of Educational Technology*, 37(3):405–423, May 2006.

[9] C. Ullrich. Tutorial planning: Adapting course generation to today's needs. In M. Grandbastien, editor, *Young Researcher Track Proceedings of 12th International Conference on Artificial Intelligence in Education*, pages 155–160, Amsterdam, The Netherlands, 2005.

[10] C. Ullrich, P. Libbrecht, S. Winterstein, and M. Mühlenbrock. A flexible and efficient presentation-architecture for adaptive hypermedia: Description and technical evaluation. In Kinshuk, C. Looi, E. Sutinen, D. Sampson, I. Aedo, L. Uden, and E. Kähkönen, editors, *Proceedings of the 4th IEEE International Conference on Advanced Learning Technologies (ICALT 2004), Joensuu, Finland*, pages 21–25, 2004.