LaTeX, SVG, Fonts

Michel Goossens and Vesa Sivunen

Abstract

After giving a short overview of SVG, pointing out its advantages for describing in a portable way the graphics content of electronic documents, we show how we converted TeX font outlines (the Type 1 variant) into SVG outlines and explain how these SVG font glyphs can be used in SVG instances of documents typeset with TeX.

1 Introduction

The increasing affordability of the personal computer drastically reduces the production cost of electronic documents. The World Wide Web makes distributing these documents worldwide cheap, easy, and fast. Taken together, these two developments have considerably changed the economic factors controlling the generation, maintenance, and dissemination of electronic documents. More recently, the development of the XML family of standards and the ubiquity of the platform-independent Java language make it possible to have a unified approach to handle the huge amount of information stored electronically and to transform it into various customizable presentation forms.

Given the severe financial constraints in many parts of the world, where it is often out of the question to even consider printing multiple copies of a (highly technical) document, electronic dissemination via the Web is the only way to publish. Thus, the Web is not only an additional medium for the traditional publishing industry, but a necessary complement in large parts of the world to participate in sharing scientific and technical information and benefit from the wealth and progress it creates.

Various techniques are now available to transform LaTeX documents into PDF, HTML (XHTML), or XML so that the information can be made available on the Web. Thus, LaTeX will continue to play a major role in the integrated worldwide cyberspace, especially in the area of scientific documents. However, it is clear that LaTeX's greatest impact will remain in the area of typesetting, with TeX remaining an important intermediate format for generating high-quality printable PDF output.

The present article explores ways of transforming LaTeX-encoded information globally into a Scalable Vector Graphics (SVG) format, in particular by exploiting the use of the SVG font machinery. A further step, to be described in a forthcoming article, will be to transform the LaTeX document into various XML vocabularies, thus saving as much semantic information as possible.[1] Such a modular approach makes optimal document reuse possible.

2 SVG for portable graphics on the Web

As the Web has grown in popularity and complexity, users and content providers wanted ever better and more precise graphical rendering, as well as dynamic Web sites. Today, only drop shadows, rudimentary animations, and low-resolution GIF or PNG images are commonly used in Web pages. Moreover, that technology is not really scalable.

The publication of the SVG Recommendation was the result of more than two years of collaborative effort by major players in the computer industry[2] to find a workable cross-platform solution to Web imaging. Version 1.0 of the SVG specification was published as a W3C Recommendation on 4 September 2001 and it represents a genuine advance for portable graphics on the Web. The current version of SVG is 1.1, and it became a W3C Recommendation on 14 January 2003.[3]

Nowadays many software vendors support SVG in their products, while more and more free viewing and editing tools capable of handling SVG become available.

SVG is an open-standard vector graphics language for describing two-dimensional graphics using XML syntax. It lets you produce Web pages containing high-resolution computer graphics.

SVG has the usual vector graphics functions. Its fundamental primitive is the *graphics object*, whose model contains the following:

- graphics paths consisting of polylines, Bézier curves, etc.:
  - simple or compound, closed or open;
  - (gradient) filled, (gradient) stroked;
  - can be used for clipping;
  - can be used for building common geometric shapes;
- patterns and markers;
- templates and symbol libraries;

[1] For instance, the hierarchical structure of the document is encoded in XML by using one of DocBook, TEI or, to a certain extent, XHTML, while other specific XML vocabularies are used for their given application domain, such as MathML for mathematics, SVG for two-dimensional graphics, CML for chemistry, BSML for bioinformatics, GeneXML or GEML for gene expression, and many others.

[2] Among the companies represented on W3C's SVG committee were IBM, Microsoft, Apple, Xerox, Sun Microsystems, Hewlett-Packard, Netscape, Corel, Adobe, Quark, and Macromedia.

[3] *Scalable Vector Graphics (SVG) 1.1 Specification*, available at http://www.w3.org/TR/SVG11/.

- transformations:
  - default coordinate system: x is right, y is down, one unit is one pixel;
  - viewport maps an area in world coordinates to an area on screen;
  - transformations alter the coordinate system (2×3 transformation matrix for computers; translate, rotate, scale, skew for humans);
  - can be nested;
- inclusion of bitmap or raster images;
- clipping, filter and raster effects, alpha masks;
- animations, scripts, and extensions;
- groupings and styles;
- SVG fonts (independent from fonts installed on the system).

SVG consists of Unicode text in any XML namespace.[4] The use of Unicode throughout enhances searchability and accessibility of the SVG graphics.

SVG drawings can be dynamic and interactive. The Document Object Model (DOM) for SVG allows for efficient vector graphics animation via scripting, which can be performed on SVG elements and other XML elements from different namespaces simultaneously within the same Web page. Event handlers can be assigned to any SVG graphical object.

A major source of information on SVG is the W3C SVG site, which describes the latest developments in the area of SVG, the status of current implementations. It also has a reference list of articles, books, software announcements, and pointers to other interesting SVG sites.[5]

2.1 Inside an SVG document

As described earlier, the basis of SVG is a Unicode text *document*, usually identified with a file extension `.svg` and a mime-type (for the server) `image/svg-xml`. Thus it is rather straightforward to create and edit SVG documents with your favorite text editor.

The top part of Figure 1 shows a small SVG file `svgexa.svg`, which is an example of the static graphics possibilities of SVG. After the comment on line 1 we declare that we work in the SVG namespace (line 2) and define the size of the display area (line 3). We write a title (lines 5 and 6), draw a row of four rectangles (lines 7–14), followed by a row of four rectangles with rounded corners (lines 17–23), and finally a row of four ellipses (lines 25–29). The origin of SVG's x-y coordinate system is the upper left hand side of the display area. The semantics of the various arguments of the SVG elements should be rather easy to guess (the SVG Specification contains detailed definitions). Notice the similarity between the PostScript and SVG languages.

If we want to view our file `svgexa.svg` we must use an application that understands the SVG language, such as Apache's Batik[6] or Adobe's Browser plugin `svgview`.[7] W3C's browser Amaya,[8] which provides an interesting development environment for viewing and editing MathML and HTML code, also supports part of SVG. The recent default distributions (1.3 or later) of the Mozilla browser[9] have built-in support for presentation MathML but to be able to interpret SVG natively you need to download a special SVG-enabled executable.[10] The middle row of Figure 1 represents at the left our example file as displayed by Batik, with a zoom on part of the graphics at its right. This shows that regions of an image can be magnified without loss of quality (this is because of the vector nature of SVG's graphics model).[11] The bottom row of Figure 1 shows at the left the same file as displayed by Microsoft's Internet Explorer using Adobe's `svgview` plugin installed (left) and at the right as shown by the Amaya browser.

3 Generating SVG instances from TeX fonts

Many scientific documents, especially in physics and mathematics, are marked up with LaTeX. On the other hand XML has become a *lingua franca* of the Internet and XML-aware tools are becoming ubiquitous. Therefore, it becomes important to integrate LaTeX and XML in an optimal way.

[4] The target application must be able to interpret the specific XML vocabulary to make this useful. A forthcoming article will show how one can use XHTML, MathML and SVG together.

[5] The W3C SVG site is at `http://www.w3.org/Graphics/SVG/`. An SVG Tutorial site is at `http://www.svgtutorial.com/`. The Batik distribution (`http://xml.apache.org/batik`) comes with many SVG examples, including quasi 3D scenes, animations, complex languages, such as Arabic, etc. The Adobe SVG site (`http://www.adobe.com/svg`) features some interesting SVG files. Interactive geometry, statistical charts, cartographic material and much more is at Michel Pilat's site (`http://pilat.free.fr/english`).

[6] See `http://xml.apache.org/batik`.

[7] Available from `http://www.adobe.com/svg`.

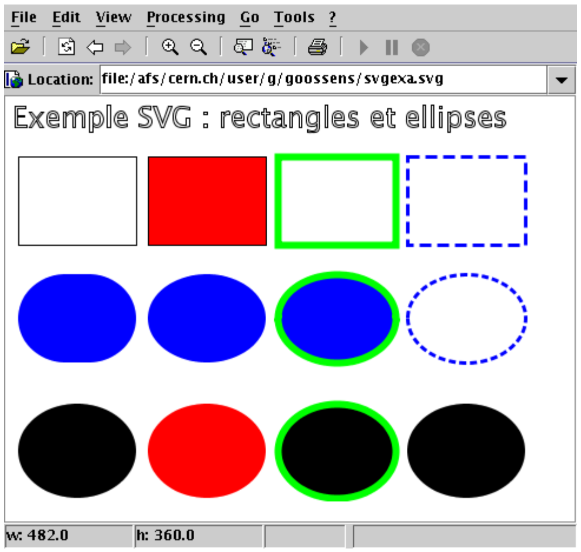
[8] See `http://www.w3.org/Amaya/`.

[9] See `http://www.mozilla.org`.

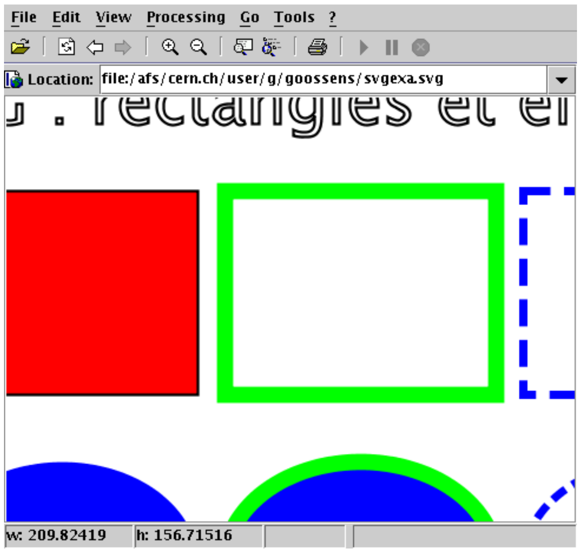
[10] Details at `http://www.mozilla.org/projects/svg/`.

[11] The Batik Squiggle SVG viewer and Adobe's Browser plugin `svgview` offer convenient ways to navigate an image. You can zoom in and out, move in the two-dimensional plane (in `svgview` hold down the `Alt` or in Squiggle the `Shift` key and move the mouse with the left button pressed) or rotate over a given angle.

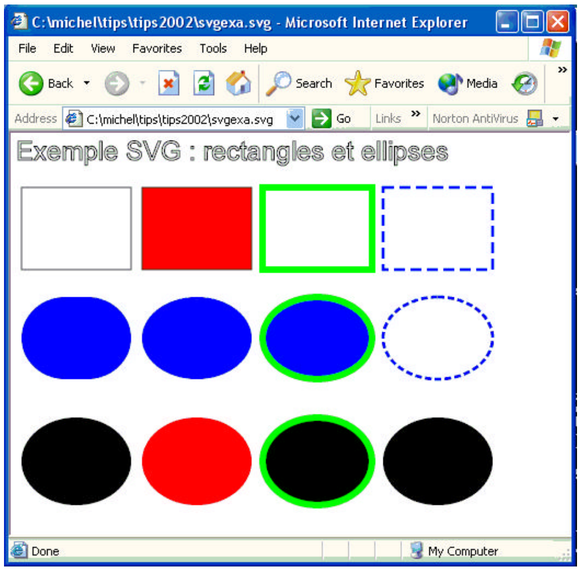
```
<!-- svgexa.svg ++ A small SVG Example ++ -->
<svg xmlns="http://www.w3.org/2000/svg"
     width="450pt" height="350pt">
 <g style="stroke:black; fill:none">
  <text x="5" y="25" style="font-size:24">
    Exemple SVG : rectangles et ellipses</text>
     <rect x="10"  y="50"  width="100" height="75"/>
     <rect x="120" y="50"  width="100" height="75"
           style="fill:red"/>
     <rect x="230" y="50"  width="100" height="75"
           style="stroke:lime; stroke-width:6"/>
     <rect x="340" y="50"  width="100" height="75"
           style="stroke:blue; fill:none; stroke-width:3;
           stroke-dasharray:10 5;stroke-linejoin:miter"/>
 </g>
 <g style="stroke:none; fill:blue">
  <rect x="10"  y="150" width="100" height="75" rx="40" ry="40"/>
  <rect x="120" y="150" width="100" height="75" rx="50" ry="50"/>
  <rect x="230" y="150" width="100" height="75" rx="60" ry="60"
        style="stroke:lime; stroke-width:6"/>
  <rect x="340" y="150"  width="100" height="75" rx="70" ry="70"
        style="stroke:blue; fill:none; stroke-width:3;
        stroke-dasharray:6 3;stroke-linejoin:miter"/>
 </g>
 <ellipse cx="60"  cy="300" rx="50" ry="40"/>
 <ellipse cx="170" cy="300" rx="50" ry="40" style="fill:red"/>
 <ellipse cx="280" cy="300" rx="50" ry="40"
          style="stroke:lime; stroke-width:6"/>
 <ellipse cx="390" cy="300" rx="50" ry="40" angle="45"/>
</svg>
```


File Edit View Processing Go Tools ?
Location: file:/afs/cern.ch/user/g/goossens/svgexa.svg
Exemple SVG : rectangles et ellipses
w: 482.0
h: 360.0

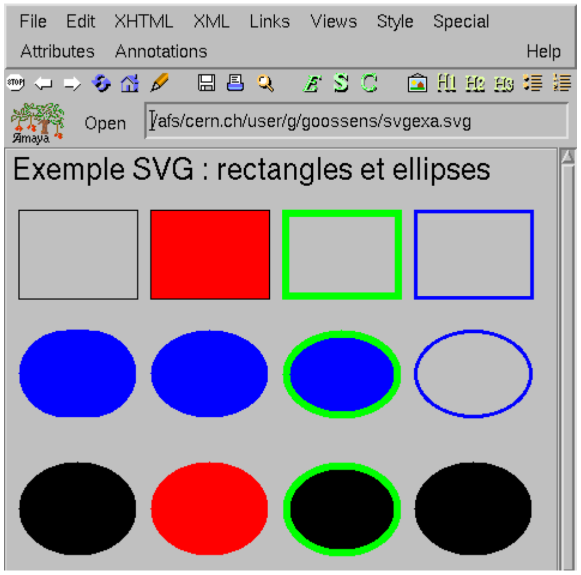
(a) Batik Squiggle SVG viewer


File Edit View Processing Go Tools ?
Location: file:/afs/cern.ch/user/g/goossens/svgexa.svg
w: 209.82419
h: 156.71516

(b) Zooming with Batik Squiggle


C:\michel\tips\tips2002\svgexa.svg - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address C:\michel\tips\tips2002\svgexa.svg Go Links Norton AntiVirus
Exemple SVG : rectangles et ellipses
Done
My Computer

(c) Microsoft Explorer


File Edit XHTML XML Links Views Style Special
Attributes Annotations Help
Amaya
Open /afs/cern.ch/user/g/goossens/svgexa.svg
Exemple SVG : rectangles et ellipses

(d) W3C's Amaya

Figure 1: Browsing an SVG file

SVG is a static declarative XML vocabulary; it provides only a final-form two-dimensional representation of a graphics image.[12]. Thus paragraphs or pages must be formatted by upstream application programs (e.g., TEX, drawing tools, Java) or by inline escaping to scripting (in Perl, Python, Ruby, JavaScript, etc.)

Various tools exist to translate EPS files to SVG, e.g., Adobe's `Illustrator` (commercial) or Wolfgang Glunz' `pstoedit`[13] (the SVG translator option is shareware). For direct translation from DVI there is Adrian Frischauf's `dvi2svg`.[14]

The translations work quite well as long as one uses standard fonts, such as Times, Helvetica and Courier. However, these applications have problems with TEX's non-standard character font encodings.

This problem of font encoding is closely related to the fact that SVG, being an XML language, uses Unicode as basic character encoding. So if one wants to go from TEX (DVI) to SVG the driver has to map each TEX character to its Unicode code-point and use a *large* corresponding SVG font that encodes all the needed characters.[15] Such a full mapping, although not impossible, is far from trivial. Hence, in coordination with Glunz, we have opted for a temporary hack, where we use a special *ad-hoc* option for `pstoedit`, where we map for each font instance the 256 hexadecimal codes `00` to `FF` in the TEX font encoding into Unicode's "Private Use Area" (PUA) using (hexadecimal) code positions `E000` to `E0FF`.

3.1 Producing SVG font instances

If we want to work with Computer Modern METAFONT sources we can use Szabó Péter's `TeXtrace` program.[16] It is a collection of Unix scripts that convert any TEX font into a Type 1 `.pfb` outline font[17] that is immediately suitable for use with `dvips`, `pdftex`, Adobe's `acroread`, etc. It now also has an option to generate SVG, but this did not directly do what we needed. In fact, we preferred to use `TeXtrace` only to generate Type 1 `pfb` files, and fall back on the second approach, that we describe next.

Working from Type 1 `pfa/pfb` font sources it is rather straightforward to generate the corresponding SVG font. A Perl script, `t1svg.pl`, was developed to achieve this translation, where the only (minor) difficulty is the correct handling of the position of the current point.[18]

Table 1 shows the correspondence between the PostScript Type 1 operators (left column) and the SVG equivalent (middle column), with the arguments for each command expressed in function of those of the corresponding Type 1 ones (`a1`, `a2`, etc.). The right column shows the x (`cx`) and y (`cy`) coordinates of the current point expressed in function of the Type 1 arguments.

3.2 Disassembling Type 1 font sources

Type 1 fonts are often commercial and should not be converted into SVG without the permission of the rights holder. TEX fonts, however, are publicly available. For TEX fonts that are not yet available in Type 1 format one can use Szabó Péter's `TeXtrace` program mentioned previously. Our Perl script needs the binary compressed `pfb` outline font to be disassembled with Lee Hetherington's and Eddie Kohler's `t1disasm` program into human-readable form.[19]

3.3 The conversion script

The Perl script `t1svg.pl`, which converts Type 1 fonts to SVG fonts, is relatively short and simple. It does not require any special modules and should run with any Perl version.

`t1svg.pl` reads the disassembled `pfb` file as input and translates the Type 1 operators into their SVG equivalents according to the correspondences of Table 1. Section A provides a detailed description of the SVG commands used in that Table.

The example of Table 2 shows the Type 1 source (left) and the SVG source (right) of the glyph of a contour integral. Line 1 of the SVG instance specifies the Unicode code position of the glyph (hexadecimal `E049`, i.e, in the PUA) and its name, which we copied from line 1 of the Type 1 source at the left (`contintegraldisplay`). Line 2 of the SVG specifies the horizontal width of the glyph (`555.6`). It was calculated by taking the second argument of the `hsbw` operator on line 2 of the Type 1 source, i.e., `5000 9 div`, which divides 5000 by 9.[20] Then, on

[12] This is unlike the PostScript language, which allows for inline computations.

[13] See `http://www.pstoedit.net/pstoedit`.

[14] See `http://www.activemath.org/~adrianf/dvi2svg/`.

[15] In fact, some mathematical TEX characters are still absent from and thus not yet encodable in Unicode 3.2.

[16] See `http://www.inf.bme.hu/~pts/textrace/`.

[17] See `http://partners.adobe.com/asn/developer/pdfs/tn/T1_SPEC.PDF` for a description of the Type 1 font format.

[18] This is needed since the SVG `z` operator sets the current point to the point that closes the path (after the operation), while the Type 1 `closepath` operator leaves it untouched, i.e., the current point remains at the value it had before the call. Therefore our Perl script has to keep track of the current point to restore it to the correct value with respect to the Type 1 coordinates after each `z` operator, at which point we insert an absolute `Moveto` (`M`) to the required (saved) current point coordinates before continuing.

[19] See `http://www.lcdf.org/ eddietwo/type/#t1utils`.

[20] Remember that PostScript uses reverse Polish notation, with arguments preceding the operator to which they belong.

Table 1: Correspondence between Type 1 and SVG commands

Type 1	*SVG*	*Current point*
`a1 a2 hsbw`	`horiz-adv-x = a2` (argument of `glyph`), `M a1 0`	`cx=a1`
`a1 hlineto`	`h a1`	`cx=cx+a1`
`a1 hmoveto`	`m a1 0`	`cx=cx+a1`
`a1 a2 a3 a4 hvcurveto`	`c a1 0 (a1+a2) a3 (a1+a2) (a3+a4)`	`cx=cx+a1+a2,` `cy=cy+a3+a4`
`a1 a2 rlineto`	`l a1 a2`	`cx=cx+a1,` `cy=cy+a2`
`a1 a2 rmoveto`	`m a1 a2`	`cx=cx+a1,` `cy=cy+a2`
`a1 a2 a3 a4 a5 a6 rrcurveto`	`c a1 a2 (a1+a3) (a2+a4) (a1+a3+a5) (a2+a4+a6)`	`cx=cx+a1+a3+a5,` `cy=cy+a2+a4+a6`
`a1 vlineto`	`v a1`	`cy=cy+a1`
`a2 vmoveto`	`m 0 a1`	`cy=cy+a1`
`a1 a2 a3 a4 vhcurveto`	`c 0 a1 a2 (a1+a3) (a2+a4) (a1+a3)`	`cx=cx+a2+a4,` `cy=cy+a1+a3`
`closepath`	`z`	

Table 2: Type 1 and SVG sources compared

```
 1 /contintegraldisplay {
 2 56 5000 9 div hsbw
 3 -2222 22 hstem -2173 94 hstem -1381 40 hstem
 4 -881 40 hstem -143 94 hstem -22 22 hstem
 5 0 97 vstem 195 40 vstem 694 40 vstem 790 97 vstem
 6 48 -2177 rmoveto 32 2 17 22 0 25 rrcurveto
 7 33 -25 16 -23 vhcurveto -24 -25 -15 -35 hvcurveto
 8 -51 50 -42 61 vhcurveto 155 0 58 242 76 322 rrcurveto
 9 15 61 35 154 14 62 rrcurveto 151 119 122 148 hvcurveto
10 0 76 -35 117 -132 57 rrcurveto 30 179 31 184 37 173 rrcurveto
11 22 100 44 203 49 0 rrcurveto 31 0 25 -19 4 -4 rrcurveto
12 -33 -2 -17 -22 0 -25 rrcurveto -33 25 -16 23 vhcurveto
13 24 25 15 35 hvcurveto 54 -54 39 -55 vhcurveto
14 -79 0 -55 -117 -53 -197 rrcurveto
15 -22 -81 -34 -144 -6 -25 rrcurveto
16 -15 -61 -35 -154 -14 -62 rrcurveto
17 -151 -119 -122 -148 hvcurveto
18 0 -76 35 -117 132 -57 rrcurveto
19 -42 -253 -35 -196 -30 -123 rrcurveto
20 -18 -74 -46 -193 -82 0 rrcurveto -36 -25 23 0 hvcurveto
21 closepath
22 320 857 rmoveto -79 37 -54 80 0 92 rrcurveto
23 0 113 84 109 137 8 rrcurveto
24 closepath
25 104 -21 rmoveto 78 -35 56 -80 0 -94 rrcurveto
26 0 -113 -84 -109 -137 -8 rrcurveto
27 closepath
28 endchar
29 }
```

```
 1 <glyph unicode="&#xE049;" glyph-name="contintegraldisplay"
 2        horiz-adv-x="555.6">
 3 <path style="fill:#000000; fill-rule=evenodd; stroke:none"
 4       d="M 56 0
 5          m 48 -2177
 6          c 32 2 49 24 49 49
 7          c 0 33 -25 49 -48 49 c -24 0 -49 -15 -49 -50
 8          c 0 -51 50 -93 111 -93 c 155 0 213 242 289 564
 9          c 15 61 50 215 64 277 c 151 0 270 122 270 270
10          c 0 76 -35 193 -167 250 c 30 179 61 363 98 536
11          c 22 100 66 303 115 303 c 31 0 56 -19 60 -23
12          c -33 -2 -50 -24 -50 -49 c 0 -33 25 -49 48 -49
13          c 24 0 49 15 49 50 c 0 54 -54 93 -109 93
14          c -79 0 -134 -117 -187 -314
15          c -22 -81 -56 -225 -62 -250
16          c -15 -61 -50 -215 -64 -277
17          c -151 0 -270 -122 -270 -270
18          c 0 -76 35 -193 167 -250
19          c -42 -253 -77 -449 -107 -572
20          c -18 -74 -64 -267 -146 -267 c -36 0 -61 23 -61 23
21          z
22          M 104 -2177 m 320 857 c -79 37 -133 117 -133 209
23          c 0 113 84 222 221 230
24          z
25          M 512 -881 m 104 -21 c 78 -35 134 -115 134 -209
26          c 0 -113 -84 -222 -221 -230
27          z"
28 />
29 </glyph>
```

line 4 of the SVG we have an absolute `move` (`M 56 0`), which corresponds to the first argument of the `hsbw` operator on line 2 of the Type 1 source. We ignore all of hints in the Type 1 source (lines 3 to 5) and continue with the `rmoveto` operator and its two arguments (line 6) which we find back in the SVG instance on line 5. We leave it as an exercise to the reader to walk through the remaining lines of the Type 1 source and verify, referring to Table 1 as necessary, that you indeed obtain the result shown at the right. Note in particular how we have to reset the current point after each SVG `z` operator (i.e., the absolute move `M` on lines 22 and 25) to where it was before the `z` operator was executed.

Figure 2 shows the contour integral as viewed with `ghostview` (the PostScript Type 1 image) and with `Batik` (rendering the calculated SVG instance).

Rather than use the temporary hack to map the TeX code positions into the Unicode PUA it might

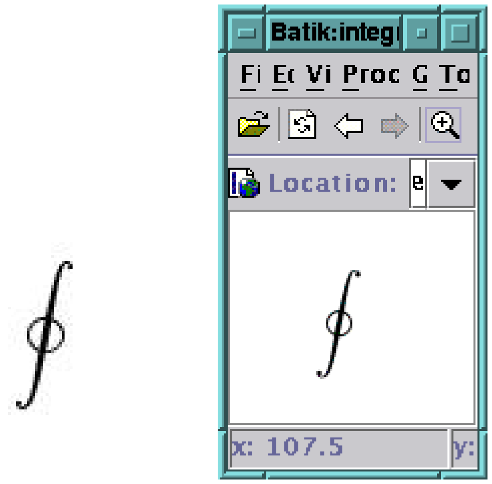


Figure 2: PostScript Type 1 and SVG rendering

be useful to map the TEX characters to their correct position in the Unicode character space to obtain a large Unicode-encoded font. This would allow other applications to use the available glyphs, but a straightforward translation from a DVI file would become impossible since TEX codes are hardwired in such a file.

4 Transforming a LATEX document into an SVG document

Note that SVG fonts are not needed if one is content with having the complete EPS figure, including the text, translated into SVG as pure paths descriptors (the text characters are treated merely as graphics paths). Use `pstoedit`'s `-dt` option in that case. This allows for fast rendering but loses the structural information about the graphics, making it more difficult to update. Moreover, when zooming, the quality of the rendering with SVG fonts (left part of Figure 3) is much better than when the TEX characters are simply translated into SVG paths (right part of Figure 3, where it is seen that the letters of the text are much coarser than at the left).

4.1 Using `pstoedit`

When using `pstoedit` to generate SVG from a PostScript source one needs to include the SVG font instances of the character glyphs referenced. For this we developed an XSLT stylesheet `ins-saxon6.xsl`.[21] In the following example we

[21] Versions of the stylesheet are available for Microsoft's `msxsl` (part of their MSDN XML Developer Center `http://msdn.microsoft.com/library/`) and for the Saxon (`http://saxon.sourceforge.net/`) and Xalan (`http://xml.apache.org/xalan-j`) Java XSLT processors.

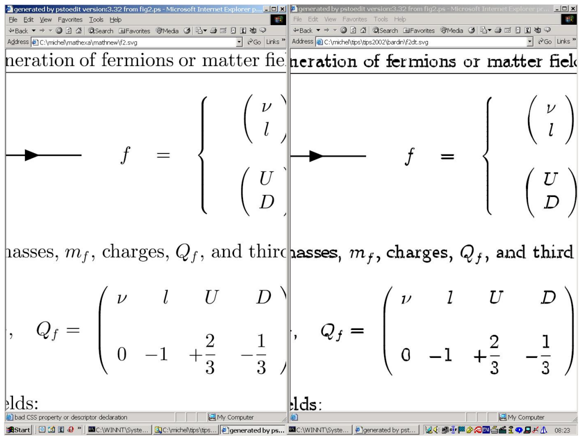

neration of fermions or matter fie
asses, m_f, charges, Q_f, and thir
lds:
neration of fermions or matter field
asses, m_f, charges, Q_f, and third
lds:

Figure 3: LATEX text rendered with SVG fonts or as graphics paths

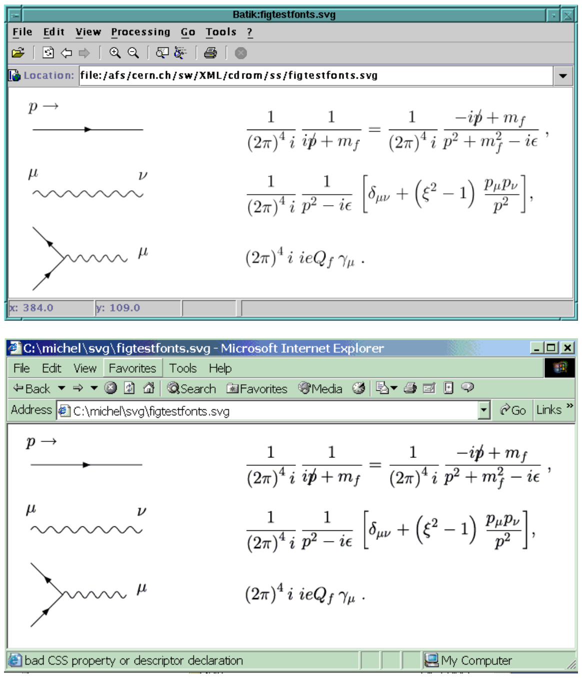

Batik:figtestfonts.svg
File Edit View Processing Go Tools ?
Location: file:/afs/cern.ch/sw/XML/cdrom/ss/figtestfonts.svg
x: 384.0
y: 109.0
C:\michel\svg\figtestfonts.svg - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address C:\michel\svg\figtestfonts.svg
bad CSS property or descriptor declaration
My Computer

Figure 4: Feynman diagrams and their propagators

transform the PostScript input file `in.ps` into the corresponding SVG instance `out.svg`. Since the latter file does not include the SVG source of the fonts we need to include these by transforming it with the help of an XSLT stylesheet to finally obtain `out1.svg`, which we can view with an SVG capable browser.

```
pstoedit -f svg:-texmode -nfr in.ps out.svg
saxon6.sh out.svg ~/www/svg/ins-saxon6.xsl out1.svg
```

Figure 4 shows a series of Feynman diagrams. The graphics at the left are generated with the help of PostScript commands interfaced to LaTeX, the right part shows their corresponding propagators and are prepared as LaTeX math. The source is run through LaTeX, turned into PostScript with `dvips`, before translating it into SVG with `pstoedit`. After including the SVG fonts with the XSLT script we display the result with Batik (top) and with Microsoft Internet Explorer and the Adobe `svgview` plugin (bottom).

4.2 Using `dvi2svg`

We interfaced the `dvi2svg` Java library via a small Unix script `dvi2svg.sh`, whose use is as follows:

```
> dvi2svg.sh
Usage: dvi2svg.sh [options] [DVIFILE]
Options:
 -o [FILENAME] : Specify an output filename prefix. If not
                 set, dvi2svg will take the input filename.
 -d : set the debug mode to on(1)/off(0 default)
```

An example of the use of the `dvi2svg` program is the translation of the font table of the American Mathematical Society font `msbm10` from the DVI format into SVG. In contrast to `pstoedit`, the `dvi2svg` program includes itself the SVG font outlines for the needed characters (font sub-setting is used). It is however impossible to deal with non-TeX material, such as EPS or PDF graphics, in which case `pstoedit` should be used.

Below we first generate the DVI file using the fonttable utility `nfssfont.tex` that comes with the LaTeX distribution. Then we run `dvi2svg.sh` on the generated `nfssfont.dvi` DVI file and obtain the SVG file `msbm1.svg`.

```
> latex nfssfont
This is TeX, Version 3.14159 (Web2C 7.3.7x)
(/TeXlive/tl7/texmf/tex/latex/base/nfssfont.tex
LaTeX2e <2001/06/01>
...
***********************************************
* NFSS font test program version <v2.0e>
*
* Follow the instructions
***********************************************
Name of the font to test = msbm10
Now type a test command (\help for help):)
*\table
*\bye
[1]
Output written on nfssfont.dvi (1 page, 5940 bytes).
> dvi2svg.sh nfssfont.dvi -o msbm
DEBUG from converter.DviToSvg => Converting file: nfssfont.dvi
DEBUG from converter.DviToSvg => Writing result to: msbm
DEBUG from converter.DviToSvg => Reader has been created
DEBUG from converter.DviToSvg => Writer has been created
Converting ................FINISHED
> ls -l msbm*.svg
-rw-rw-r--    1 goossens  161252 Jan  2 10:08 msbm1.svg
```

Figure 5 shows in its bottom part the font table of the font `msbm10` as viewed with `xdvi`, while its top part shows the SVG rendering with Batik of the SVG file `msbm1.svg` prepared with `dvi2svg` from the original DVI file, as shown above.

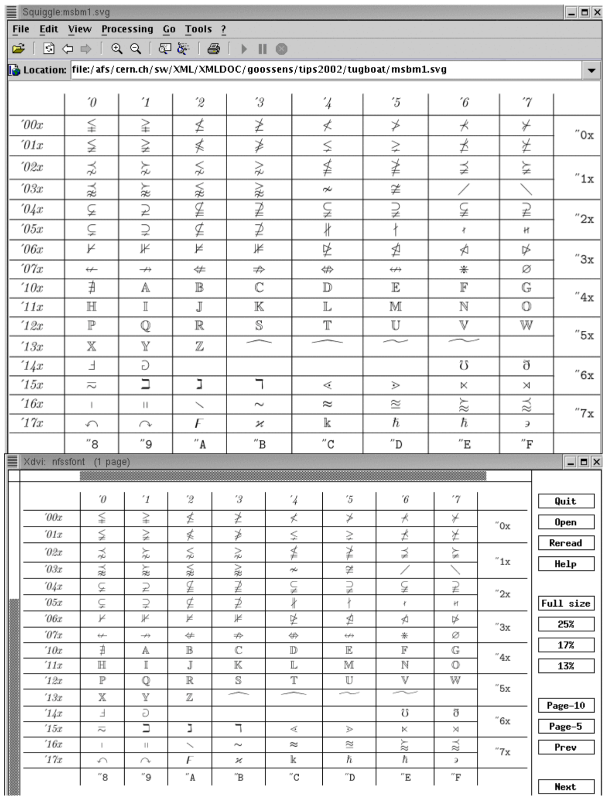

Squiggle:msbm1.svg
File Edit View Processing Go Tools ?
Location: file:/afs/cern.ch/sw/XML/XMLDOC/goossens/tips2002/tugboat/msbm1.svg
Xdvi: nfssfont (1 page)
Quit
Open
Reread
Help
Full size
25%
17%
13%
Page-10
Page-5
Prev
Next

Figure 5: TeX's msbm10 font table

4.3 More complex examples

Figure 6 displays the result of translating LaTeX text and a complex use of its `picture` environment into SVG in various ways. At the left we show the file as viewed with `dvips` (PostScript version), in the middle the SVG file created with the `pstoedit` program from the PostScript output, at the right the SVG file as generated directly from the DVI file with `dvi2svg`. Both SVG files were displayed with the help of Microsoft's Internet Explorer (with Adobe's `svgview` plugin). Figure 7 shows part of the SVG source file that was generated by `dvi2svg`. We shall study its basic structure later (see Section B.2).

We also looked at LaTeX sources that use non-standard fonts, such as the MusiXTeX and XY-pic packages. We first generated the SVG instances for the Type 1 versions of the needed fonts and then selected a few examples[22] that are typical for these packages. For completeness we also include examples of chemistry and algebra. We generated the

[22] We took our inspiration from chapters 5 "The XY-pic package" and 7 "Preparing music scores" of *The LaTeX Graphics Companion, Illustrating Documents with TeX and PostScript*, by Michel Goossens, Sebastian Rahtz, and Frank Mittelbach, Addison-Wesley, 1997, ISBN 0201854694.

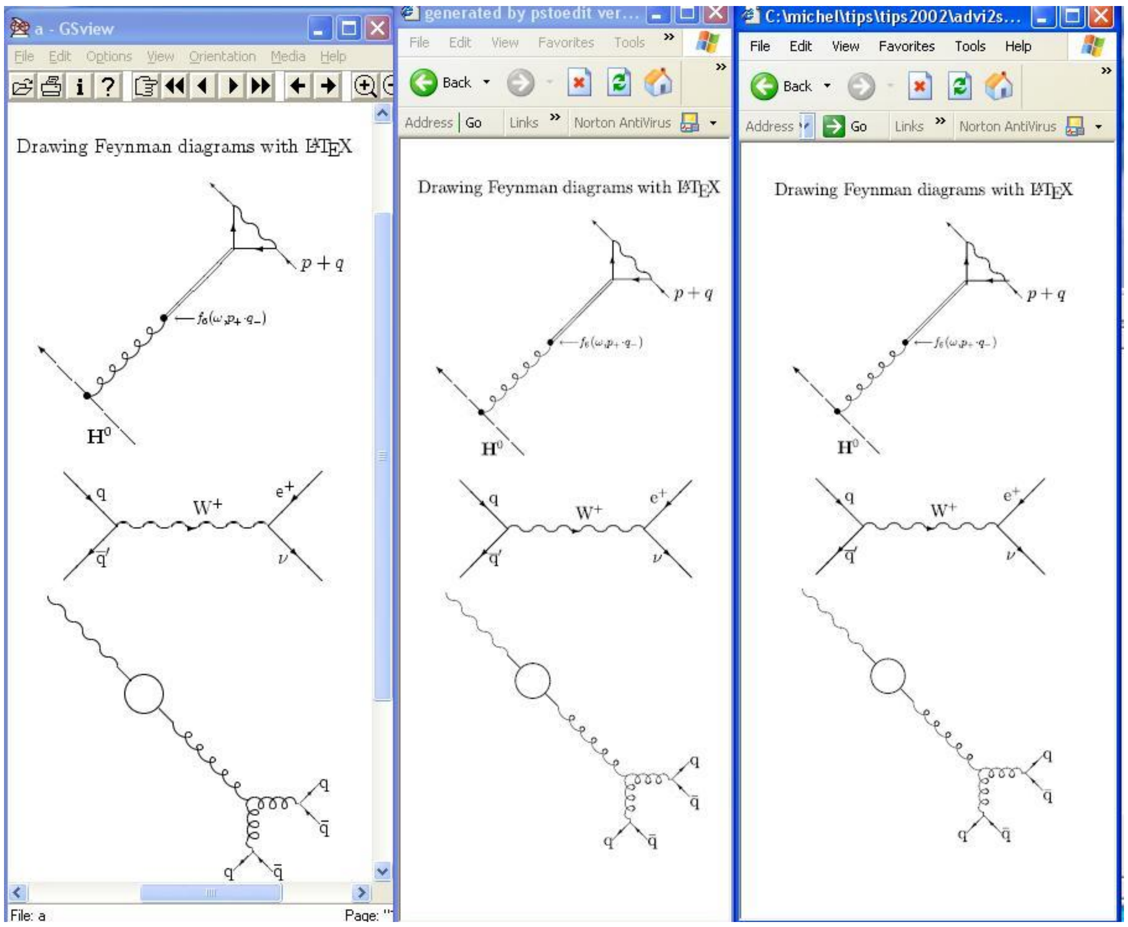

a - GSview
File Edit Options View Orientation Media Help
Drawing Feynman diagrams with LaTeX
$p+q$
$f_6(\omega, p_+ q_-)$
H^0
q
W+
e+
q'
ν
q
q̄
File: a
Page:
generated by pstoedit ver...
File Edit View Favorites Tools
Back
Address Go Links Norton AntiVirus
C:\michel\tips\tips2002\advi2s...
File Edit View Favorites Tools Help

Figure 6: LaTeX and its `picture` environment: PostScript and SVG renderings

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!--This file was automatically generated by dvi2svg-->
<svg width="504" height="800" viewBox="0 0 504 800">
 <defs>
  <font id="CMR">
   <font-face font-family="CMR"/>
   <glyph unicode="&#57412;" glyph-name="D"
                             horiz-adv-x="763.9">
    <path style="fill:#000000; fill-rule=evenodd; stroke:none"
          d="..."/>
   </glyph>
   <glyph unicode="&#57458;" glyph-name="r"
                             horiz-adv-x="391.7">
    <path style="..." d="..."/>
   </glyph>
     ...
  </font>
  <font id="LCIRCLEW">
   <font-face font-family="LCIRCLEW"/>
   <glyph unicode="&#57352;" glyph-name="a8"
                             horiz-adv-x="1200">
    <path style="..." d="...’’/>
   </glyph>
     ...
  </font>
  <font id="LCIRCLE"><font-face font-family="LCIRCLE"/>...</font>
  <font id="LINEW"><font-face font-family="LINEW"/>...</font>
  <font id="LINE"><font-face font-family="LINE"/>...</font>
  <font id="CMBX"><font-face font-family="CMBX"/>...</font>
  <font id="CMSY"><font-face font-family="CMSY"/>...</font>
  <font id="CMMI"><font-face font-family="CMMI"/>...</font>
 </defs>
 <g>
  <text fill="black" font-family="CMR" font-size="11.7871">
   <tspan y="101.88267" x="...">...</tspan>
  </text>
  <rect x="322.78577" y="145.65428" width="0.872714"
        height="0.26910424" fill="black" stroke="black"
        stroke-width="0.1"/>
  ...
 </g>
</svg>
```

Figure 7: LaTeX and its `picture` environment: Generated SVG instance

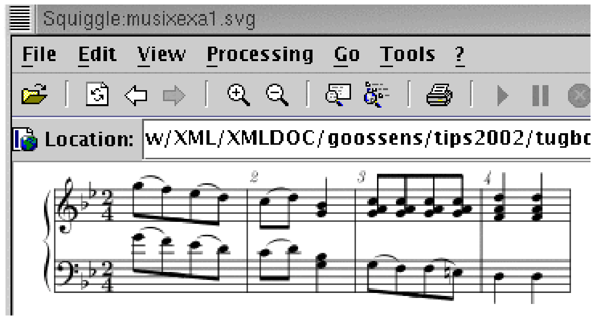

Squiggle:musixexa1.svg
File Edit View Processing Go Tools ?
Location: w/XML/XMLDOC/goossens/tips2002/tugbc

Figure 8: SVG instance of a MusiXTEX example viewed with Batik Squiggle

SVG from the DVI files as follows (we show only one run, the others are similar).

```
dvi2svg.sh xytest.dvi

DEBUG from converter.DviToSvg => Converting file: xytest.dvi
DEBUG from converter.DviToSvg => Writing result to: xytest
DEBUG from converter.DviToSvg => Reader has been created
DEBUG from converter.DviToSvg => Writer has been created

Converting ................
```

We display the resulting SVG files with Batik's Squiggle SVG viewer. Figure 8 corresponds to example 7–2–5 of *The LATEX Graphics Companion*. It represents a moderately complex part of a music piece typeset with MusiXTEX. Figure 9 is the structural formula of adonitoxin as typeset with the XΥMTEX package (see page 221 of the same book). XΥMTEX is an advanced application based on LATEX's `picture` environment. Figure 10 which combines complex mathematical formulae with a graphical representation of a Dynkin diagram generated with the help of the `picture` environment, is based on an entry in a dictionary of Lie superalgebras.[23]

Figure 11 shows complex examples from the "Links and knots" Section 5.5.8 of *The LATEX Graphics Companion*. In particular the display is an enlarged view of the bottom parts of examples 5–5–28 and 5–5–29, as well as of examples 5–5–34 and 5–5–35. The display shows that SVG can be nicely scaled for better viewing and is thus a perfect complement for PDF output.

5 Conclusion

We have explained how SVG is a truly scalable two-dimensional XML-based graphics language for the Web. Since graphics representations are at the heart of many scientific, technical and other documents,

[23] The example is taken from the tables in the Appendix of *Dictionary On Lie Algebras And Superalgebras* by Luc Frappat, Antonino Sciarrino, and Paul Sorba, Harcourt Publishers 2000, ISBN 0122653408.

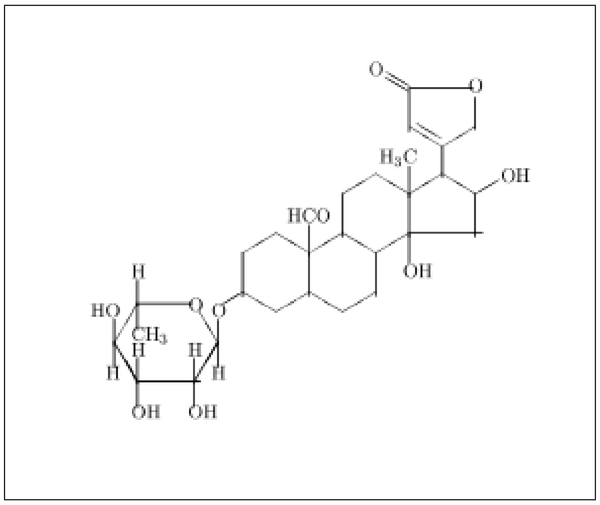
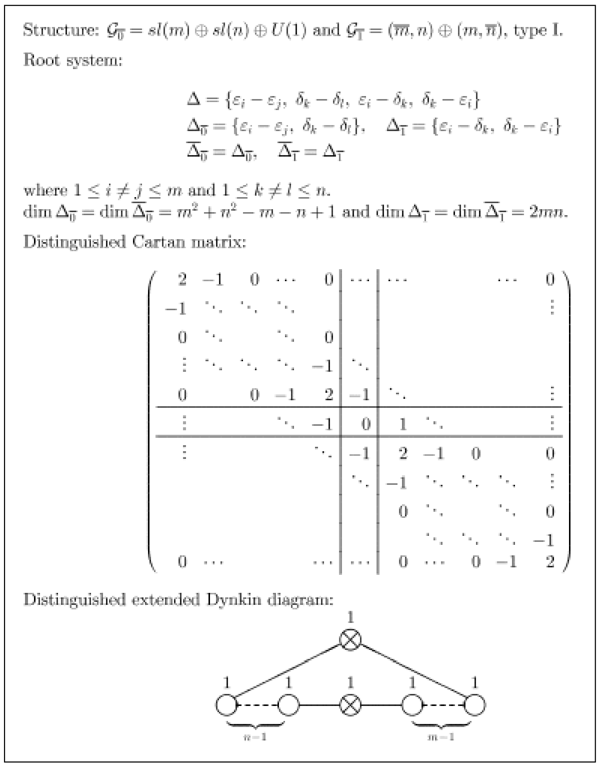


Figure 9: SVG instance of a XΥMTEX example viewed with Batik Squiggle


Structure: $\mathcal{G}_{\overline{0}} = sl(m) \oplus sl(n) \oplus U(1)$ and $\mathcal{G}_{\overline{1}} = (\overline{m}, n) \oplus (m, \overline{n})$, type I.

Root system:

$$\begin{aligned}
&\Delta = \{\varepsilon_i - \varepsilon_j,\ \delta_k - \delta_l,\ \varepsilon_i - \delta_k,\ \delta_k - \varepsilon_i\}\\
&\Delta_{\overline{0}} = \{\varepsilon_i - \varepsilon_j,\ \delta_k - \delta_l\}, \quad \Delta_{\overline{1}} = \{\varepsilon_i - \delta_k,\ \delta_k - \varepsilon_i\}\\
&\overline{\Delta}_{\overline{0}} = \Delta_{\overline{0}}, \quad \overline{\Delta}_{\overline{1}} = \Delta_{\overline{1}}
\end{aligned}$$

where $1 \le i \ne j \le m$ and $1 \le k \ne l \le n$.
$\dim \Delta_{\overline{0}} = \dim \overline{\Delta}_{\overline{0}} = m^2 + n^2 - m - n + 1$ and $\dim \Delta_{\overline{1}} = \dim \overline{\Delta}_{\overline{1}} = 2mn$.

Distinguished Cartan matrix:

$$\left(\begin{array}{ccccc|c|ccccc}
2 & -1 & 0 & \cdots & 0 & \cdots & \cdots & & & \cdots & 0\\
-1 & \ddots & \ddots & \ddots & & & & & & & \vdots\\
0 & \ddots & & \ddots & 0 & & & & & & \\
\vdots & \ddots & \ddots & \ddots & -1 & \ddots & & & & & \\
0 & & 0 & -1 & 2 & -1 & \ddots & & & & \vdots\\
\hline
\vdots & & & \ddots & -1 & 0 & 1 & \ddots & & & \vdots\\
\hline
\vdots & & & & \ddots & -1 & 2 & -1 & 0 & & 0\\
 & & & & & \ddots & -1 & \ddots & \ddots & \ddots & \vdots\\
 & & & & & & 0 & \ddots & & \ddots & 0\\
 & & & & & & & \ddots & \ddots & \ddots & -1\\
0 & \cdots & & & \cdots & \cdots & 0 & \cdots & 0 & -1 & 2
\end{array}\right)$$

Distinguished extended Dynkin diagram:

Figure 10: SVG instance of math and a graph viewed with Batik Squiggle

we are convinced that most applications will be able to generate in the foreseeable future SVG output from input data marked up in their specific vocabularies. Similarly, with the help of the Computer Modern family (and other) SVG font sets, which we have explained how to generate, it is now possible to

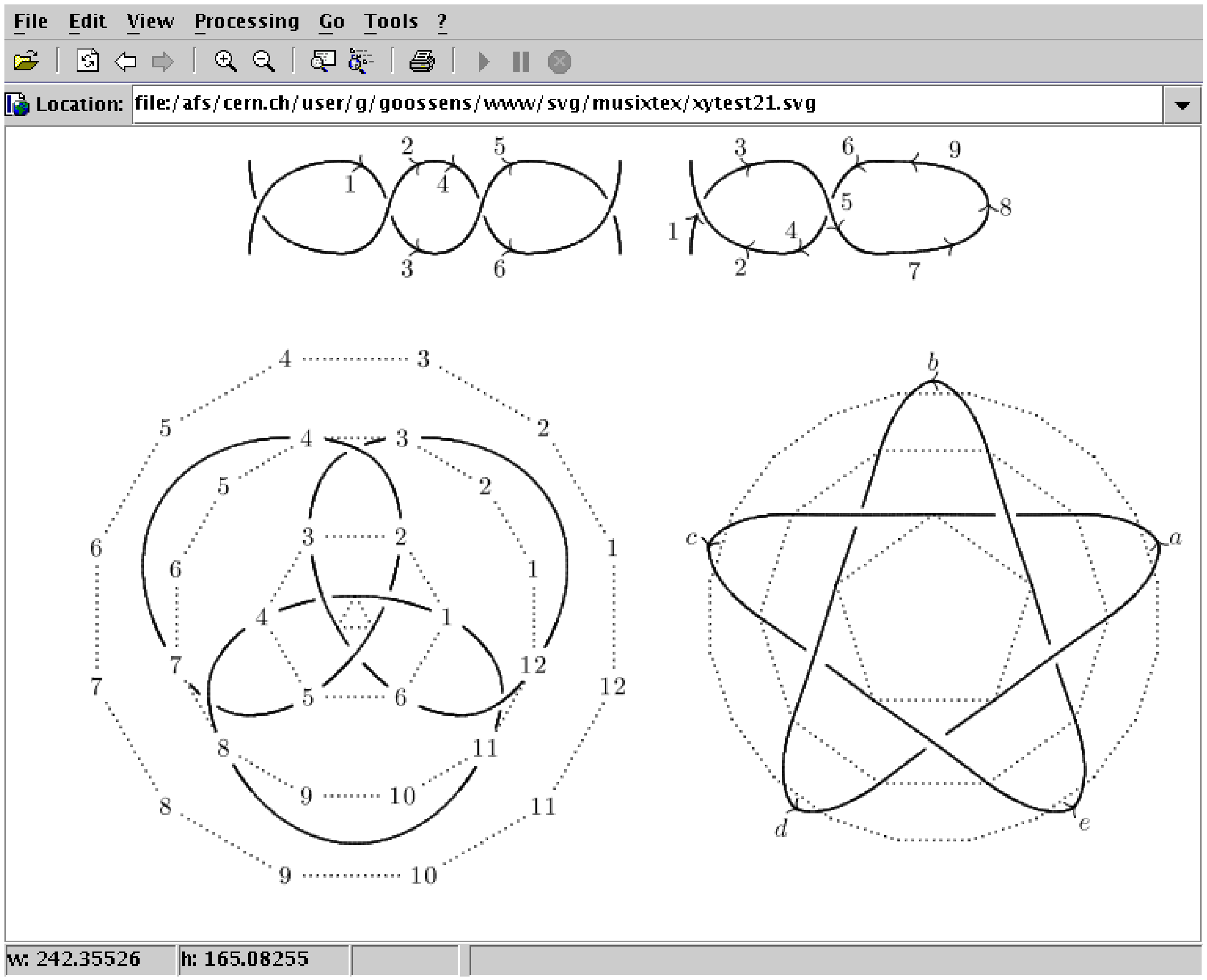

File
Edit
View
Processing
Go
Tools
?
Location: file:/afs/cern.ch/user/g/goossens/www/svg/musixtex/xytest21.svg
w: 242.35526
h: 165.08255

Figure 11: XY-pic example viewed with Batik Squiggle

transform complete LaTeX documents into SVG. Although this may prove useful in itself, the existence of PDF and its use inside browsers on the Web makes SVG somewhat redundant in this respect.

However, the more important use of SVG with LaTeX will be to translate the graphics images contained in a LaTeX document or parts of single pages of LaTeX documents from EPS, etc. into SVG so that they can be included in XML instances generated from LaTeX sources with the help of LaTeX to XML converters, such as `tex4ht`.[24] This will become extremely useful once the major browsers are able to handle XML namespaces, making it possible to combine different XML vocabularies. We look forward to the not too distant future when we will be able to generate and edit XHTML (DocBook), MathML and SVG directly and have the result displayed correctly.[25]

6 Acknowledgments and distribution

One of us (VS) would like to acknowledge the funding of a fellowship to work at CERN that he received from the "Tools for Innovative Publishing in Science" (TIPS) Project, part of the Information Society Technologies Programme of the 5th Framework of the European Union.[26]

The latest version of the utilities developed for this project is available as a ZIP file on the Web.[27] The fonts also come with the `dvi2svg` distribution.

[24] See `http://www.cis.ohio-state.edu/~gurari/TeX4ht` and references therein.

[25] See `http://www.w3.org/TR/XHTMLplusMathMLplusSVG/` for some work that is been done in this area.

[26] See `http://www.cordis.lu/ist/ist-fp5.html`.

[27] See `http://home.cern.ch/goossens/svgfonts.html`.

A SVG graphics path commands

Paths (defined in SVG using the `path` element) specify the geometry of the outline of an object. Path operators can set the current point (`moveto`), draw a straight line (`lineto`), draw a cubic Bézier curve (`curveto`), and close the current shape (`closepath`). Details on a subset of the SVG path commands that are used in SVG fonts follow. Command names in lowercase are for relative coordinates, uppercase names for absolute coordinates.

`[m|M] (x y)+`
: Start a new sub-path at the given `(x,y)` coordinate. A relative `moveto` (`m`) appearing as the first element of a path is treated as a pair of absolute coordinates. If a `moveto` is followed by multiple pairs of coordinates, the subsequent pairs are treated as implicit `lineto` commands.

`[z|Z]`
: Closes the current subpath by drawing a straight line from the current point to the current subpath's initial point.[28]

The various `lineto` commands draw straight lines from the current point to a new point:

`[l|L] (x y)+`
: Draws a line from the current point to the given `(x,y)` coordinate which becomes the new current point. A number of coordinate pairs may be specified to draw a polyline. At the end of the command, the new current point is set to the final coordinate provided.

`[h|H] x+`
: Draws a horizontal line from the current point `(cpx,cpy)` to `(x,cpy)`, which becomes the new current point.

`[v|V] y+`
: Draws a vertical line from the current point `(cpx,cpy)` to `(cpx,y)`, which becomes the new current point.

There are three groups of commands to draw curves. Here we only look at one of the cubic Bézier commands, since it is used in the translation of the Type 1 fonts. Further information on the other commands are in the SVG Specification.

`[c|C] (x1 y1 x2 y2 x y)+`
: Draws a cubic Bézier curve from the current point to `(x,y)` using `(x1,y1)` as the control point at the beginning of the curve and `(x2,y2)` as the control point at the end of the curve. Multiple sets of coordinates may be specified to draw a polybézier. The new current point is set to the final `(x,y)` coordinate pair used in the polybézier.

[28] At the end of the command, the new current point is set to the initial point of the current subpath, so that if a `closepath` is followed immediately by any other command, then the next subpath and the current subpath share their same initial point.

B SVG Fonts

Graphics designers creating SVG content using arbitrary fonts need to be sure that the same graphical result will be displayed when the content is viewed by all end users, even those who do not have the necessary fonts installed on their computers. Therefore, to guarantee reliable font delivery the SVG Specification defines a common "SVG font" format that all conforming SVG viewers must support.

B.1 Overview

SVG fonts contain unhinted font outlines. Because of this, on many implementations there will be limitations regarding the quality and legibility of text in small font sizes. For increased quality and legibility in small font sizes, or for faster delivery of Web pages (SVG fonts are expressed using SVG elements and attributes, so that they can be quite verbose compared to other formats) alternate font technology might be considered on some systems.[29]

SVG fonts and their associated glyphs do not specify bounding box information, so that it is up to the applications to calculate bounding box and overhang based on an analysis of the graphics elements contained within the glyph outlines.

B.2 The font element

An *SVG font* is defined using a `font` element.[30] The characteristics and attributes of SVG fonts follow closely the font model of the *Cascading Style Sheets (CSS) level 2* Specification.[31]

[29] The authors of the SVG Specification have realized that the absence of a hinting mechanism in the font format of current SVG 1.1 is a drawback. Indeed, Web developers embed fonts in other formats in their SVG documents in situations where the available resolution is insufficient for adequate rendering in native SVG. Therefore, SVG 1.2 (`http://www.w3.org/TR/SVG12`) plans to add hinting as an optional feature for SVG fonts, thus offering Web authors the choice of a pure SVG solution. The adopted approach is likely to be based on a free variant of PostScript Type 1 hinting.

[30] See `http://www.w3.org/TR/SVG/fonts.html`.

[31] See `http://www.w3.org/TR/REC-CSS2/fonts.html`. In that document font metrics are expressed in units that are relative to an abstract square whose height is the intended distance between lines of type in the same type size. This square is called the *em square* and it is the design grid on which the glyph outlines are defined. The value of the `units-per-em` attribute on the `font` element specifies how many units the em square is divided into. Common values are 1000 (Type 1) and 2048 (TrueType or OpenType).

An `font` element can contain the following elements: `font-face`[32] (provides further typographic information about the font, including the name of the font), `hkern` and `vkern` (kerning information between Unicode characters), `missing-glyph` (defines the representation to be used for all Unicode characters that have no explicit `glyph` element defining their outline in the present font), and finally `glyph`.

The `glyph` element defines the graphics for a given glyph. The coordinate system for the glyph is defined by the various attributes in the `font` element. The graphics that make up the `glyph` can be either a single *path data* specification within the `d` attribute (see below) or arbitrary SVG as content within the `glyph` element.

Important attributes of the `glyph` element are described below.

`unicode = "<string>"`
: If a single character is provided, then this glyph corresponds to the given Unicode character. If multiple characters are provided (e.g., for ligatures) then this glyph corresponds to the given sequence of Unicode characters. For example see line 1 of Table 2 and lines 7, 12, and 20 of Figure 7.

`glyph-name = <name> [,<name>]*`
: A glyph name should be unique within a font. Glyph names are used when Unicode character numbers do not provide sufficient information to access the correct glyph (e.g., when there are multiple glyphs per Unicode character). Glyph names are referenced in *kerning* definitions. For example see line 1 of Table 2 and lines 7, 12, and 20 of Figure 7.

`d = "path data"`
: Definition of the outline of a glyph. Uses the same syntax as the `d` attribute on a `path` element, which is often used instead. For example see lines 4–27 of Table 2 or lines 10, 14, 22 of Figure 7.

`horiz-adv-x = "<number>"`
: The default horizontal advance after rendering a glyph in horizontal orientation. Glyph widths must be non-negative, even if the glyph is rendered right-to-left, as in Hebrew and Arabic scripts. An attribute `horiz-adv-y` exists for specifying the vertical advance for glyphs rendered in vertical orientation. For example see line 2 of Table 2 and lines 8, 13, and 21 of Figure 7.

[32] Similar to CSS2's `@font-face` font descriptor, see `http://www.w3.org/TR/REC-CSS2/fonts.html`.

For increasing portability it is advisable to embed all SVG fonts that are referenced inside an SVG document. As an example, Figure 7 shows how all the fonts needed to render the given SVG graphics image are first included (inside a `defs` element, lines 4 to 32) and later referenced (e.g., line 34 calls for a character of font `CMR` whose definition is on lines 5–17).

On the other hand, it is also possible, e.g., for convenience, to save SVG font sources in external files and reference characters in these fonts via CSS style directives from inside SVG images. In this case one must, however, make sure that the needed fonts are installed on the client's system or are shipped together with the referencing SVG file to the client site.

◇ Michel Goossens
IT Division, CERN
CH1211 Geneva 23, Switzerland
`michel.goossens@cern.ch`

◇ Vesa Sivunen
ETT Division, CERN
CH1211 Geneva 23, Switzerland
`vesa.sivunen@cern.ch`