SVG − Scalable Vector Graphics

A future cornerstone of the WWW−infrastructure.

1. What is SVG?

»Scalable Vector Graphics« (Acronym is SVG) is the new official vector graphics standard of the W3C (World Wide Web Consortium). As all new W3C standards it is XML−based, and works together with other XML based−standards, such as DTD, Schema, XSL, XQL, SMIL, XHTML, XFORMS, XQL, RDF, Namespaces, etc. Furthermore it is DOM−transparent and allows the use of ECMA−Javascript, which means that every single SVG−element and all attributes may be changed with scripts. SVG allows for the first−time to use an open standard for the display of high−quality vector graphics within web−applications. Before, one could only use proprietary standards.

2. Who develops SVG?

SVG is developed by a consortium with members of the W3C, research institutions and companies in the domain of graphics, multimedia and networking. Companies involved in the standardization process include Adobe, Apple, Autodesk, Bitflash, Canon, Corel, HP, IBM, Kodak, Macromedia, Microsoft, Netscape, Quark, Sun, Visio. SVG also enjoys strong support in the Open−source community. As an open consortium the W3C and SVG working group is open to suggestions from institutions and individuals.

3. Usage Scenarios for SVG

As an open standard, that supports high−quality vector and raster−graphics, animation and interactivity, SVG qualifies for all websites with the aim of dynamic content delivery and high−standard web−graphics. Possible usage scenarios include:

- Webdesign in general
- Technical illustrations
- Scientific Data visualization (chemistry, physics, exploratory visualization, etc.)
- Webmapping and Online GIS−services, navigation and routing
- Location based services (SVG Mobile)
- Online−Catalogs and E−Commerce sites
- Online learning systems
- Multimedia and entertainment
- User−Interfaces for websites in general
- SVG as an open Graphics exchange format (between graphics software and different

platforms)

- SVG viewer as a rendering component of offline applications (ActiveX, COM, etc.)

4. SVG's architecture and document−structure

As every XML file, SVG also consists of a header, a root−element (svg−element) and several child−elements with attributes. SVG makes extensive use of attributes. Entities may be defined and used later in the file (e.g. for CSS−definitions) and for reusable graphics. Another way to re−use SVG−elements is to set references to object−ids with the <use xlink:href="">−element. SVG developers also can use the grouping features and switch statements (f.e. having different processing instructions depending on client capabilities and language). Within the »<defs>«−section one can centrally define re−usable objects f.e. have symbols, gradient, fills, etc. SVG−code within the »<defs>«−section is not rendered, but can be instantiated later on. Elements in an SVG document have an implicit drawing order, with the first elements in the SVG document »painted« first. Subsequent elements are painted on top of previously painted elements, taking into account opacity settings.

```
<?xml version="1.0" standalone="no"?>
   <!DOCTYPE svg PUBLIC "−//W3C//DTD SVG 20010904//EN"
   "http://www.w3.org/TR/2001/REC−SVG−20010904/DTD/svg10.dtd">
    <svg width="5cm" height="4cm" xmlns="http://www.w3.org/2000/svg">
       <desc>Four separate rectangles</desc>
       <rect x="0.5cm" y="0.5cm" width="2cm" height="1cm"/>
       <rect x="0.5cm" y="2cm" width="1cm" height="1.5cm"/>
       <rect x="3cm" y="0.5cm" width="1.5cm" height="2cm"/>
       <rect x="3.5cm" y="3cm" width="1cm" height="0.5cm"/>
    </svg>
```

The above code−snippet shows a well−formed and valid SVG−document, describing four rectangles. The code is easy to read, edit and interpret.

5. SVG's graphical capabilities

SVG allows to define high−level 2D−graphics objects: basically we have vector−based geometry, raster−images and text. SVG−viewers usually render the graphics in high−quality, using antialiasing−techniques. SVG−viewer internally build a hierarchical DOM−tree in order to quickly access and manipulate elements and their attributes. Elements may be added, deleted and re−ordererd within the tree. All graphics produced by any graphics and/or DTP software may be saved and displayed with SVG. SVG knows the following shape types:

- Rectangles
- Circles
- Ellipses
- Lines
- Polylines
- Polygons
- Symbols
- Path−elements

The code below shows some basic−elements. You may copy it to your favourite text−editor, change some parameters and view it in a SVG−viewer. Source: [14, WINTER, 2001].

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
   "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg xml:space="preserve" width="225px" height="300px" viewBox="0 0 300 400">
 <defs>
  <style type="text/css">
   <![CDATA[
    .str {stroke:black;stroke-width:1}
    .fnt {font-weight:normal;font-size:20;font-family:'Arial, sans-serif'}
    .red {fill:red}
    .blu {fill:blue}
    .yel {fill:yellow}
    .gre {fill:green}
    .frn {fill-rule:nonzero}
   ]]>
  </style>
 </defs>
 <g id="mainlayer">
  <text class="fnt" x="44" y="88">rect</text>
  <rect class="blu str" x="150" y="15" width="100" height="50" rx="12" ry="18" />
  <text class="fnt" x="140" y="88">rect (rounded)</text>
  <text class="fnt" x="36" y="180">circle</text>
  <text class="fnt" x="36" y="263">line</text>
  <text class="fnt" x="170" y="180">ellipse</text>
  <text class="fnt" x="140" y="363">path:<tspan x="140" y="383">simple +
    bezier</tspan></text>
  <text class="fnt" x="16" y="363">polygon</text>
  <rect class="red str" x="15" y="15" width="100" height="50" />
  <text class="fnt" x="156" y="255">polyline</text>
  <circle class="yel str" cx="62" cy="135" r="20" />
  <ellipse class="gre str" cx="200" cy="135" rx="50" ry="20" />
  <g style="fill:none; stroke:green">
    <line x1="15" y1="240" x2="30" y2="200" style="stroke-width:2" />
    <line x1="30" y1="240" x2="45" y2="200" style="stroke-width:4" />
    <line x1="45" y1="240" x2="60" y2="200" style="stroke-width:8" />
    <line x1="60" y1="240" x2="75" y2="200" style="stroke-width:10" />
    <line x1="75" y1="240" x2="90" y2="200" style="stroke-width:12" />
  </g>
  <polyline style="fill:none; stroke:red; stroke-width:2" points="160,200 180,230
     200,210 234,220"/>
  <polygon class="yel" transform="scale(.3),translate(-188,830)"
     style="stroke-width:3;stroke:black;" points="350,75 379,161 569,111 397,215
     423,301 350,250 277,301 303,215 231,161 321,161" />
  <path class="str" style="fill:none;stroke-width:3" d="M150 280l19,10 -22,33
     40,3c12,43 44,-83 83,20" />
 </g>
</svg>
```

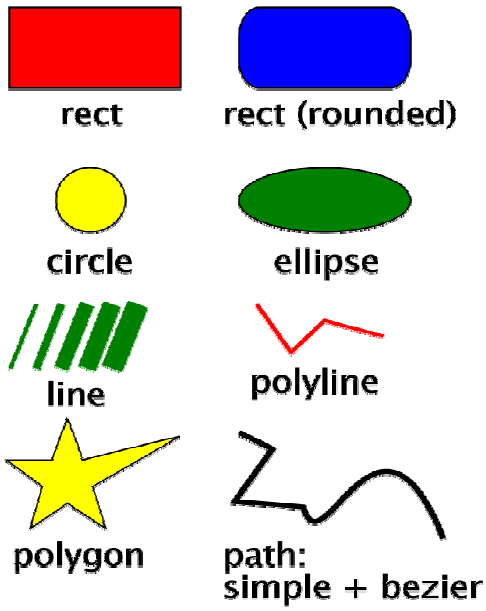

Figure 1: Basic Shapes.

The symbol−element is especially useful to symbolize point−objects. Symbols are defined in the header of the file and may contain any SVG−code (other basic shapes). The path−object is the most powerful object. It may contain line−segments, quadratic and cubic bezier curves and arc−segments. SVG path−elements may contain holes and may comprise of several disjunct path−elements (f.e. to represent a country with separate islands as a whole object). Marker symbols may be used to be placed along vertices on a polyline and to create arrows on end−points.

As to »stroke«−properties SVG supports »line−width«, »color«, »opacity«, »dashing«, »line−caps« (butt, round and square) and »line joins« (miter, round, bevel). For filling objects SVG introduces the concept of having a »paint−server«, that describes methods on how the objects should be filled. Current fill types include »solid fills« (colors), »gradients« (linear and radial), »patterns« (both vector and raster, tiled or not) and »custom paints«. One can also define the »fill−rule« (nonzero−winding, evenodd) and »opacity values«. Gradients and patterns are also implemented as »paint−servers«. Gradients may have several »stop−values« and different »spreadMethods«.

SVG allows the use of different units, as defined in the CSS−specifications. So far, we can use em, ex, px, mm, pt, pc, cm, in and percentages. SVG allows the definition of two separate coordinate−systems: The device−coordinate system (or viewport coordinate system) as defined in the »width« and »height«−attribute of the root−element, and the »user coordinate system« as defined by the »viewBox«−attribute of the root−element. The origin is − like in CSS − in the upper left corner, with the positive x−axis to the right and the positive y−axis to the bottom of the canvas. Viewport and user coordinate system may differ in order to use different scale−factors and translations without having to do much calculation. This is in particular useful when displaying maps or scaled technical illustrations. As for transformations, SVG allows scaling, translation, rotation, skewing and matrix−operations. Each object or group may have a separate local coordinate−system, according to the transformation parameters. Transformations may be nested several times. Almost every SVG geometry−element may be clipped, masked or may serve as a clipping−path, incl. Text−elements.

Concerning styling, SVG is completely transparent to CSS and XSL styling schemes. Styles may be centrally defined in the header of an SVG−file, in external stylesheet−files (in this case it can share styles with HTML/XHTML/XML) and directly as an attribute of a geometry− or text−element. It is also common to define styles as entities within the doctype−section of the SVG−header. Styles are a very powerful instrument to get an easy to manage, common »look and feel« throughout a whole web−project.

6. SVGs text capabilities

Text within SVG is a graphics object like all the other basic shapes − which means, that all fill, stroke− and opacity−settings may be applied. Text may be clipped and may serve as clipping path, may be transformed, animated, etc. SVG has full support for internationalization (unicode), bidirectional and right−to−left text. The SVG switch−element allows to check the browsers language and offers support for different languages (similar to a switch/case−statement in programming).

The SVG text specification supports all text−properties and features you might know from DTP software (different styles, weights, variants, sizes, stretching, kerning, etc.). However, SVG does not know multiline paragraphs (next spec. will most probably support). Currently

you have to either use several lines or embed a foreign−XML object (f.e. XHTML) to use text−blocks. SVG allows you to embed single glyphs or font−sets to be sure the user gets the font you specified as an author, even if it is not installed on the clients system. Single characters and words may have different baselines and individual glyphs may be rotated. A very useful feature is the ability to align text to path−elements. This feature is frequently used in cartography to letter line features.

7. SVG and filter effects

A specific, nevertheless very useful, feature of SVG is the use of filter effects. Every object, not only raster images, may be filtered. Filters are implemented within the rendering pipeline, after the objects are rasterized, but before the graphic is displayed on the output−device. Currently available filters include »lightning−effects«, »blurring«, »blending«, »compositing«, »flooding«, »merging«, »morphology«, »turbulence«, etc. Filters may be combined in any order and may effect only subregions of an object.

The code and figure below shows two path−elements and a text−element with combined filters applied: Gaussian Blur, Specular Lighting, PointLight, compositing and merging. Source: SVG−specification at [4, FERRAIOLO et.al, 2001].

```
<?xml version="1.0"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
    "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="7.5cm" height="5cm" viewBox="0 0 200 120"
     xmlns="http://www.w3.org/2000/svg">
  <title>Example filters01.svg - introducing filter effects</title>
  <desc>An example which combines multiple filter primitives
        to produce a 3D lighting effect on a graphic consisting
        of the string "SVG" sitting on top of oval filled in red
        and surrounded by an oval outlined in red.</desc>
  <defs>
  <filter id="MyFilter" filterUnits="userSpaceOnUse" x="0" y="0" width="200"
       height="120">
      <feGaussianBlur in="SourceAlpha" stdDeviation="4" result="blur"/>
      <feOffset in="blur" dx="4" dy="4" result="offsetBlur"/>
      <feSpecularLighting in="blur" surfaceScale="5" specularConstant=".75"
                          specularExponent="20" lighting-color="#bbbbbb"
                          result="specOut">
        <fePointLight x="-5000" y="-10000" z="20000"/>
      </feSpecularLighting>
      <feComposite in="specOut" in2="SourceAlpha" operator="in"
         result="specOut"/>
      <feComposite in="SourceGraphic" in2="specOut" operator="arithmetic"
                   k1="0" k2="1" k3="1" k4="0" result="litPaint"/>
      <feMerge>
        <feMergeNode in="offsetBlur"/>
        <feMergeNode in="litPaint"/>
      </feMerge>
    </filter>
  </defs>
  <rect x="1" y="1" width="198" height="118" fill="#888888" stroke="blue" />
  <g filter="url(#MyFilter)" >
          <g>
      <path fill="none" stroke="#D90000" stroke-width="10"
            d="M50,90 C0,90 0,30 50,30 L150,30 C200,30 200,90 150,90 z" />
      <path fill="#D90000"
            d="M60,80 C30,80 30,40 60,40 L140,40 C170,40 170,80 140,80 z" />
      <g fill="#FFFFFF" stroke="black" font-size="45" font-family="Verdana" >
        <text x="52" y="76">SVG</text>
      </g>
    </g>
  </g>
</svg>
```

Figure 2: SVG filter effect

8. SVG and declarative animation

The SVG specification provides for almost any attribute to be animated. Syntax and specification is borrowed from SMIL animation. Color, geometry, transformation, location, are just a few of the elements and attributes that may be animated. Of particular interest is the animation of an object along a path−element, which allows to orient an object automatically according to the line−segments orientation. Animations may be combined and nested. SVG supports »discrete«, »linear«, »paced« and »spline« calculation modes, where »discrete« »jumps« from one state to the other, »paced« guarantees a continuous velocity and »spline« supports more control (f.e. acceleration) at the cost of increasing complexity. To control time, one can use the »duration« attribute. Quite powerful is the use of the »keyTimes« and »values« combination. With »keyTimes« one can define fractions of the whole animation duration and specify »values« of any attribute. This is similar to the »timeline« metaphor of Macromedias authoring tools. One can start and end an animation relative to other animation events. Alternative to the declarative animation, described within the SVG code, one can use Javascript and the DOM to change elements and their attributes by programming.

The following example shows the use of SVG animations and in particular the »<animateMotion>«−element which allows to move an object (in our case an airplane) along a path. This example includes three separate animations: animation along a path, a transform/scale animation for take−off and landing of the plane and an opacity animation to hide/show city−labels along the flight−path as the plane passes by.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20000303 Stylable//EN"
"http://www.w3.org/TR/2000/03/WD-SVG-20000303/DTD/svg-20000303-stylable.dtd" [
        <!ENTITY stKanton "stroke:#3984FF;stroke-width:150;fill-rule:nonzero;
          fill:#C5FFE8;">
        <!ENTITY flightRoute "stroke:orange;stroke-width:2000;fill:none;">
        <!ENTITY cityText "font-size:8000;font-family:'sans-serif';">
        <!ENTITY animDuration "10s">
]>
<svg xml:space="preserve" width="900" height="600" viewBox="480000 0 360000
240000">
<defs>
  <symbol id="symbolRect" overflow="visible">
    <rect x="-3000" y="-3000" width="6000" height="6000" style="fill:
      rgb(240,65,25); fill-opacity: 0.8; stroke: rgb(0,0,0); stroke-width:300" />
  </symbol>
  <symbol id="symbolCirc" overflow="visible">
    <circle cx="0" cy="0" r="3000" style="fill: rgb(12,166,107); fill-opacity:
      0.8; stroke: rgb(0,0,0); stroke-width:300" />
  </symbol>
  <symbol id="airplane" overflow="visible">
    <path style="stroke:blue;stroke-width:100;fill:lightgray;" d="M-4000,0
      a1000,300 0 0,1 1000,-300 H-1000 L1500,-3000 h400 L0,-300 h2000 L3000,-1500
      h500 L2500,-50 V100 L3500,1500 h-500 L2000,300 h-2000 L1900,3000 h-400 L-
      1000,300 H-3000 a1000,300 0 0,1 -1000,-300" />
```

```
  </symbol>
</defs>
<g id="Kantone">
<path id="Zuerich" style="&stKanton;" d="M673825,63480 l1077,-820 ... some
path_coordinates ... z"/>
<!-- more path elements cut off ..., holds geometry of swiss cantons -->
</g>

<g id="AnimationPaths">
        <path id="Zuerich_Geneva" style="&flightRoute;" d="M682500,53500
          C632500,53500 549500,80000 499500,181000" />
        <use id="AirplaneZurichGeneva" xlink:href="#airplane">
                <animateMotion id="animMotionZurGen" dur="&animDuration;"
                   repeatCount="indefinite" rotate="auto-reverse">
                      <mpath xlink:href="#Zuerich_Geneva"/>
                </animateMotion>
                <animateTransform id="animTransZurGen" attributeName="transform"
                  attributeType="XML" type="scale" keyTimes="0;0.2;0.8;1"
                  values="1.5;4;4;1.5" dur="&animDuration;" additive="replace"
                  fill="freeze" repeatCount="indefinite"/>
        </use>
</g>
<text id="labelZurich" x="687000" y="50000" style="&cityText;">Zurich
        <animate id="textAnimZur" attributeType="CSS" attributeName="opacity"
          keyTimes="0;0.15;0.3;1" values="1;1;0;0" dur="&animDuration;"
          repeatCount="indefinite" />
</text>
<text id="labelBern" x="608500" y="102500" style="&cityText;">Bern
        <animate id="textAnimBer" attributeType="CSS" attributeName="opacity"
          keyTimes="0;0.15;0.3;0.5;0.65;1" values="0;0;1;1;0;0"
          dur="&animDuration;" repeatCount="indefinite" />
</text>
<text id="labelGeneva" x="504000" y="187000" style="&cityText;">Geneva
        <animate id="textAnimGen" attributeType="CSS" attributeName="opacity"
          keyTimes="0;0.65;0.8;1" values="0;0;1;1" dur="&animDuration;"
          repeatCount="indefinite" />
</text>
<g id="staedteRect">
        <use id="Zurich" x="682500" y="53500" xlink:href="#symbolRect"/>
        <use id="Basel" x="612000" y="33500" xlink:href="#symbolRect"/>
        <use id="Genf" x="499500" y="181000" xlink:href="#symbolRect"/>
        <use id="Bern" x="604500" y="106000" xlink:href="#symbolRect"/>
</g>
<g id="staedteCirc">
        <use id="Luzern" x="665000" y="89000" xlink:href="#symbolCirc"/>
        <use id="St. Gallen" x="748000" y="46500" xlink:href="#symbolCirc"/>
        <use id="Chur" x="759500" y="109000" xlink:href="#symbolCirc"/>
        <use id="Lugano" x="718500" y="206500" xlink:href="#symbolCirc"/>
</g>
</svg>
```

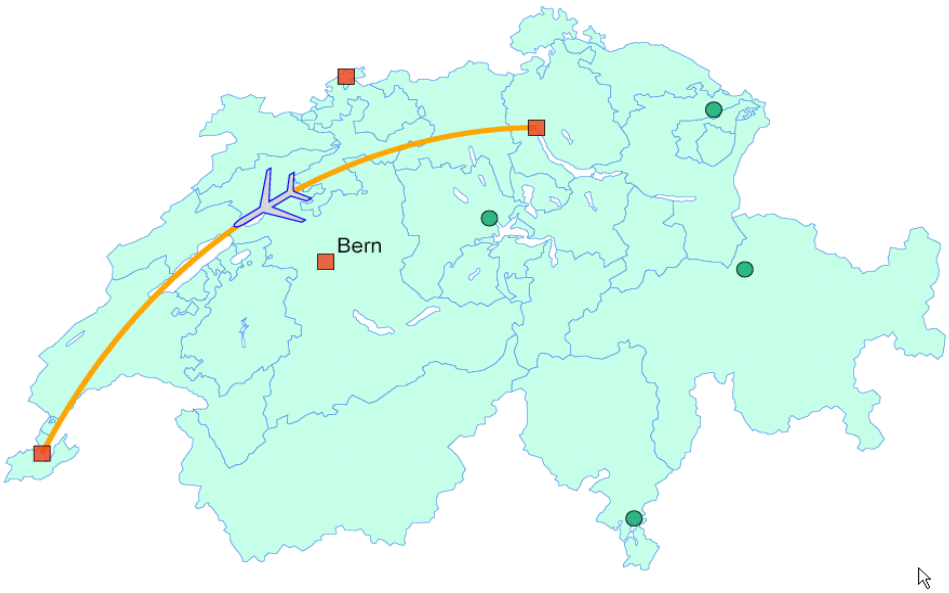

Figure 3: SVG animation. Source: [9, NEUMANN, 2001]

9. Interactive SVG

Interactivity is, no doubt, one of the strengthes that make SVG interesting for dynamic websites with graphical content. Interactivity can be achieved on both, the client and server−side − with different response−times of course. On the server, data may be selected and processed according to the clients settings (f.e. submitted through HTML−forms). Popular techniques for content−generation on the server include PHP, perl−scripts, JSP/Servlets and ASP technology. On the client, developers can use mouse−, keyboard− and state−events in order to react to the users input or a change in the DOM−scenegraph. Javascript helps to manipulate the complete SVG−DOM, including element creation, deletion and changing of attributes. Besides reacting to events, developers can change the shape and size of the mouse−cursor. Of course you may also use hyperlinks (XLINK/XPointer) in order to point to other objects and webpages.

One of the outstanding interactive SVG examples is the Adobe »SVG Draw« showcase application. It allows users to interactively draw on a SVG canvas, transform objects and manipulate their graphical attributes. Other interesting interactive examples include demos and tutorials at http://www.kevlindev.com/ and webmapping−demos at www.carto.net.

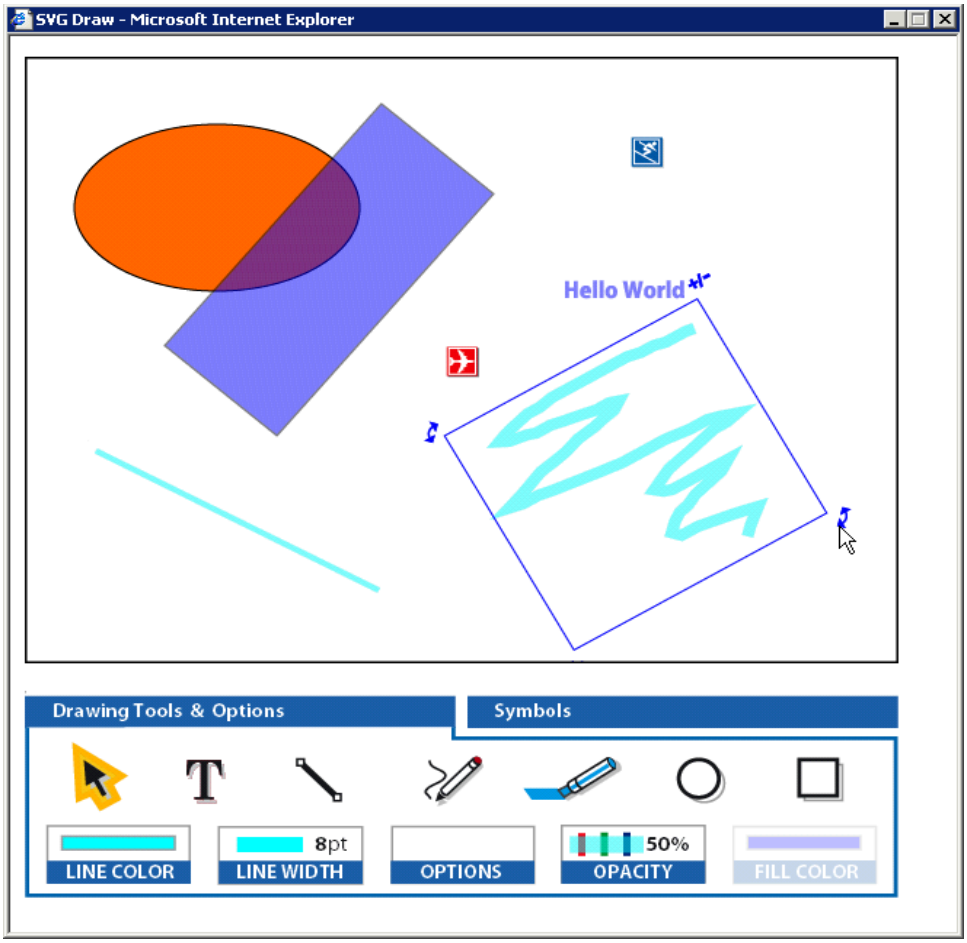

Figure 4: SVG Draw from Adobe shows the potential of interactive SVG on the client. Source: [1, ADOBE, 2001]

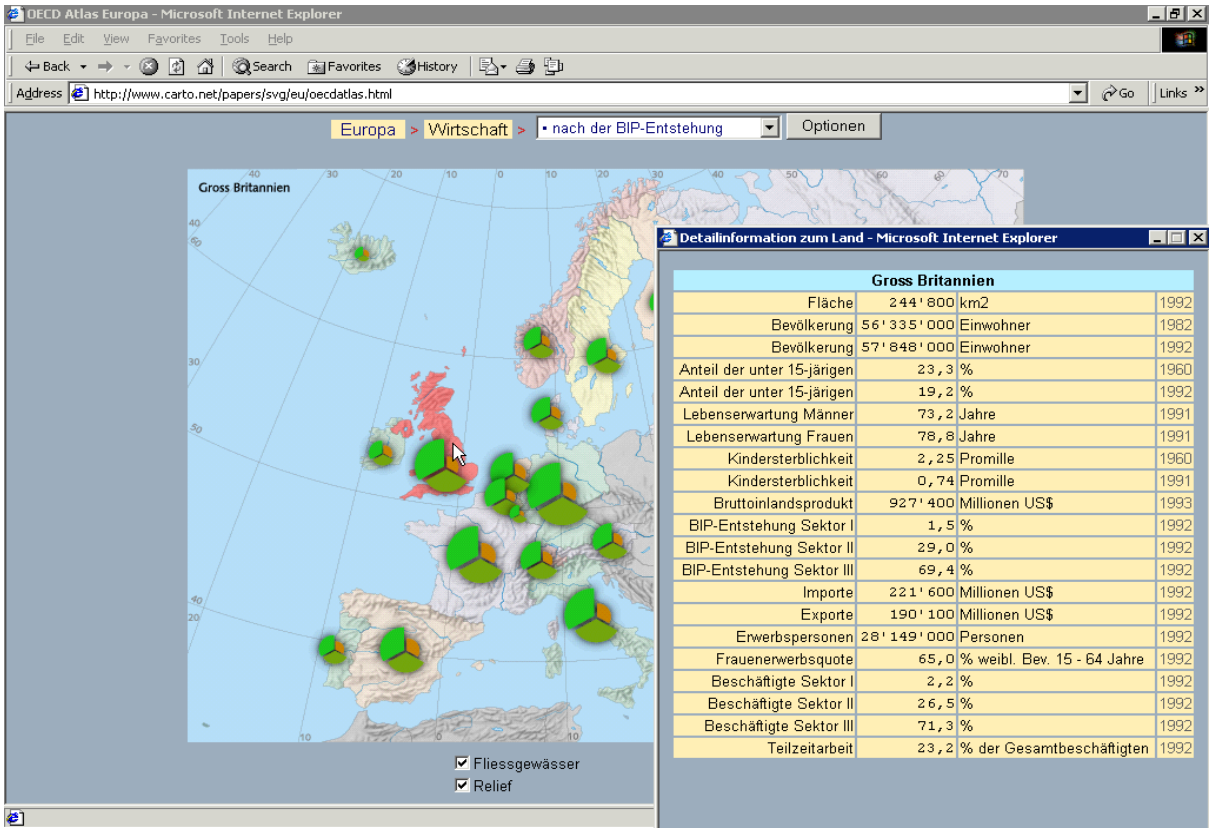

Figure 5: Interactive Map on Europe (Economic situation) − shows the use of SVG for mapping and presentation of geographic data. The diagrams are generated on the fly out of the statistical data. Source: [13, WINTER, 2000]

10. Extensibility and Metadata

SVG allows to embed foreign namespaces − either predefined by organizations or defined by yourself. You may also use your own attributes (as long as they are not colliding with the official ones) and step through them, using the DOM. However, using XML−code of foreign name−spaces does not guarantee that the SVG viewer can use and process that data. For including metadata one should follow the guidelines specified by W3C in the RDF (Resource Description Framework) specification. It should be included in the »<metadata>−tag«. VRML/X3D show an interesting technique to extend datatypes and objects: the »prototype« node. It allows to define new nodes that can hold geometry, variables, and logic (scripts). Prototypes could be instantiated while overloading specific parameters. So far, no similar functionality exists in SVG, though it would be a very useful feature.

11. SVG Workflows; how to generate SVG files

There are many ways to generate, convert or edit SVG−files; for implementations also see [7, LILLEY, 2001]:

- **Text− and XML−Editors**

Because SVG−files are readable text, they may be written or edited in any Text−Editor you have at hand. Specialized editors for programming may offer syntax−highlighting for SVG as well as advanced selection and replace functionality. XML−editors additionally support with checking wellformedness and validity, according to the SVG DTD. They may also assist with

choosing the valid attributes (similar to property lists in programming IDEs). While text−and XML−editors may not be well−suited to draw graphics and convert other graphic−files they may be of great value to edit, correct or optimize already converted or generated SVG−files from other applications.

- **Export from Graphics software**

At the time of writing this article, the following products/projects did have SVG−export− and /or import−filters: Adobe Illustrator, Corel Draw, Mayura Draw, Sphinx Open Editor (In−GmbH), Sketch (Open−Source) and Kontour (Open−Source, KOffice). Not all filters support high−level SVG−features, some of them reduce all objects to path−objects. The export of layer−names and object−ids is still not necessarily self−evident. Some export−filters could need some postprocessing for optimization ... More support from graphics software companies is desirable, but likely to come.

- **SVG Editors and Authoring Systems**

So far, there only a few specialized SVG authoring−systems available. The most advanced one currently available, is »WebDraw« from JASC (the company who did PaintShopPro). With WebDraw one can draw the basic shapes, path−elements and symbols. The SVG−developer/web designer may work in WYSIWYG−mode (canvas) or in source−code and has direct access to the Adobe SVG−viewer (as an embedded component) for previewing. Source−Code is evaluated to warn about syntax errors. It also features a hierarchical DOM−viewer (with icon−preview of the selected elements), a navigation panel and context−sensitive toolbars, according to the tool/SVG−element selected. Webdraw has in integrated filter−editor and a timeline−editor to generate animations. The product seems very promising and we are looking forward to the final version. It is also likely that Adobe and Corel will add more SVG−support to their existing products »Adobe LiveMotion«, »Adobe GoLive«, »Adobe Illustrator«, »Corel Draw« and »Corel Rave«.

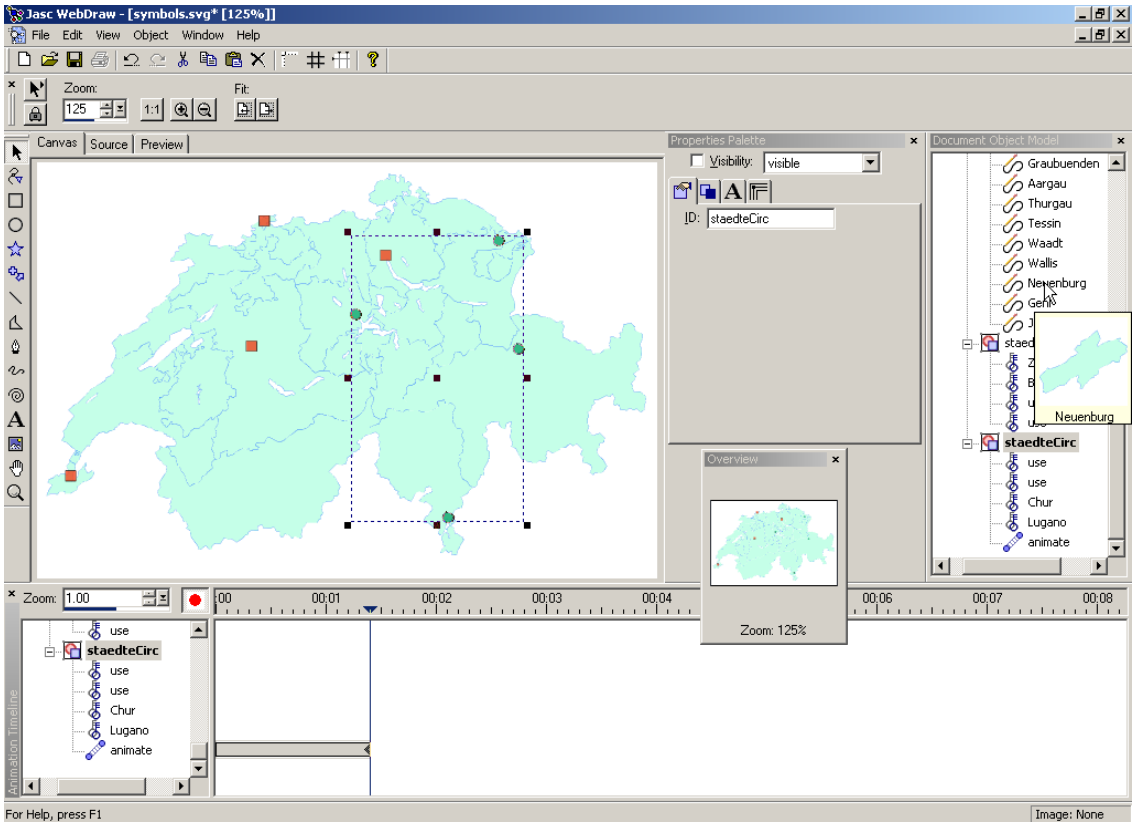

Figure 6: JASC WebDRAW, a specialized SVG authoring system with animation features.

- **SVG converters**

Specialized converters (both commercial and free) exist to convert other graphics formats to SVG, or vice−versa. No wonder, that open and documented formats are better supported than undocumented and/or proprietary formats. So far we know of support for converting swf, pdf, ps/eps and wmf. Additionally we have Java2D class support, so that every Java program can directly generate SVG. Using scripting languages (f.e. perl, python, etc.) − with their good support for string manipulation and pattern−matching − it is easy to write converters that directly convert other formats to SVG. The SVG−format is very well documented and you can use XML parsers and libraries to read and write SVG. If you write your own converter you only have to support the features you really need.

- **SVG printer−driver**

»Software Mechanics« created an SVG printer driver (SVGMaker) that works similar to Adobe Distiller and allows any application to print directly to an SVG−file. This is in principal a bright idea for adding SVG−support to applications that do not support it natively. However, the output usually cannot provide advanced structuring, because it only cares about correct printing, and cannot benefit from advanced features. So it might only be an alternative for static SVG graphics, where size does not matter that much.

- **Serverside SVG generation**

Serverside generation is a must for all dynamic sites that have to deliver up−to−date information − mostly database−driven − or need to select data out of larger data−sets. Compared to dynamic generation of swf (Flash) files it is much easier with SVG. Because it is a text−based format, one can use PHP, perl or python − scripting languages that all feature good support for manipulating text−strings and querying databases. Additionally one can use Java JSP/servlets or ASP technology. Like with converters, you can also write your own software to deliver dynamic SVG directly from the server. For GIS/cartography purposes the combination of the above techniques with spatial databases (f.e. PostgresGIS or Oracle Spatial) might be of interest. This solution offers spatial queries and simple GIS functionality (analysis), without having to pay expensive licenses to commercial GIS−vendors. Another interesting way to convert XML−data to SVG could be the use of XSL technology (templates and logic parts).

- **Client−Side SVG generation**

XSL can also be used to convert the data directly on the client, but only few of the current web−browsers do support XSL conversion and formatting yet. Javascript (or other client side scripting languages) may be used to dynamically generate or manipulate SVG elements. This can be done directly on the client, without having to interact with the server. The data used for SVG generation may be stored in XML−files or javascript−arrays. SVG−Viewers may also be used with Java−Applets and offer IDL bindings for other programming languages. Demonstrations for DOM manipulation and interactive generation/manipulation of SVG−elements may be viewed at [8, LINDSEY, 2001] and [11, NEUMANN/WINTER, 2001].

- **Optimizing SVG Graphics**

Quite often the converters or SVG−export filters do not produce optimal SVG−code. The following guidelines may help to minimize file−sizes and ease maintenance.

- Use Stylesheets (CSS or XSL)

- Use entity definitions
- Use symbols
- Use cross−references to geometry or attributes with xlink
- Use relative coordinates for larger geometry in path−elements
- Use bezier−curves instead of many short line−segments within the path−element
- Compress the path−geometry omitting unnecessary white−spaces
- Only use the precision (number of decimal places) you really need (esp. when dealing with spatial data)
- Compress SVG, CSS and js−files using gzip

- **SVG−Browser/Viewer implementations**

Because SVG is a quite young technology, there is rarely native SVG support in the major browsers. Adobe currently provides the most sophisticated viewer, implemented as a plugin for Internet−Explorer and Netscape, on both Macintosh and Windows−platform. The Mozilla team is working on a native browser implementation for Mozilla 1.0 (multiplatform) and the KDE/Konqueror team works for native support for this excellent Linux/Unix browser. The W3C added static SVG support for their Amaya project. Besides, there are some standalone implementations, mostly written in Java2: Batik ([2, APACHE.ORG, 2001] − the most popular standalone SVG−viewer, Sourcecode available), Csiro SVG viewer and IBM SVG view are only some of the many SVG/Java2 implementations. None of them already supports the whole SVG specification, Batik is the most advanced one − concerning static SVG display. Java2 and SVG are closely related to each other (both designed and implemented by Adobe and Sun as major contributors): the Java2D API has all the prerequisites necessary to implement SVG viewers.

12. Comparing SVG to other Vector−Graphics−Standards for the Web

Currently there are only two other alternatives (besides SVG) to deliver vector graphics to web−clients: Macromedia Flash (proprietary) and WebCGM (a W3C recommendation). After working for about two years on static and interactive SVG graphics and animation it is clear to us that SVG is a superset of the two above formats, concerning functionality and flexibility. Furthermore it is a W3C recommendation and is XML−based. XML is and will be the foundation of all future web−techniques. Learning XML once allows you to more quickly get into other XML−file formats and centrally use shared XML technology (such as XSL, DTD/Schema, XQL, etc.). This is a strong argument for people that complain about long learning processes. Using SVG you can use most of your XML−knowledge and concentrate on the essential problems and content delivery, rather than having to learn new syntaxes. Thus XML and SVG both can profit from each other and guarantee strong support, shorter learning phases and a fast development cycle.

For an in−depth comparison of Macromedias .SWF fileformat (probably the only direct competitor to SVG) and SVG you may have a look at [10, NEUMANN, 2001]. Please note that we are comparing the file formats and concepts but not authoring systems. We are therefore targeting technically interested developers that want to implement more complex projects that usually need special features and adaptations. The Macromedia Flash authoring system offers some of the functionality that we miss in the .SWF file−format itself − it is implementing higher−level objects at lower levels within the final .SWF file. Macromedia Flash distinguishes between the working file−format ».fla« (undocumented) and the less powerful but documented ».swf« format (see [12, OPENSWF.ORG, 2001]). Macromedia Flash did, no doubt, influence SVG in terms of animation features, but is now clearly

outplayed by the current SVG specification. Because Macromedia flash has a very good market penetration and is currently available to more platforms (incl. Linux) it is likely to stay in parallel to SVG for a while. At the time of writing this article, Flash also had better authoring systems support for the non−technical user.

WebCGM is a successor of the popular CGM (Computer Graphics Metafile) format. It is usually a binary format, but also allows clear text and character encodings. The CGM approach is far simpler than SVG, offering less interactivity and no animation features. It is therefore easier to implement, even more because significant CGM know−how is already available in Graphics− and CAD industry. WebCGM features graphical objects, layers and text−paragraphs/subparagraphs. Graphical base−objects are roughly the same than with SVG. Transparency is as well supported and raster images may be embedded. External attribute data may be attached using IDs. It also allows the use of external symbol−libraries and limited styling. Concerning interactivity, designers may use hyperlinking, predefined views, highlighted objects, and tool−tips. The WebCGM specification may be found at: [3, CRUIKSHANK et.al, 2001]. It seems to us that the use of WebCGM is still not very popular. There are no major advantages compared to SVG, except of simplicity for implementations. A more widespread use will depend on the availability of WebCGM implementations in the major−browsers and graphics software, providing WebCGM export−filters.

13. The Future of SVG

SVG is a hot topic for website developers with emphasis on rich graphical content and dynamic content generation. It is currently the most powerful and elegant approach to bring interactive and animated graphics to the web. The broad support from W3C, research institutions and companies should guarantee rapid development of tools, viewers and authoring−systems, as well as server−side generators. There is a large number of articles and developer tutorials available for web−developers to learn this new graphics−standard. The XML foundation fits well in future web−architectures. However, penetration of SVG plugins is still low and no SVG viewer already supports the entire specifications. Unfortunately, Adobes SVG viewer (the most popular and advanced viewer) is not available for the Linux/Unix platform. Hopefully, customer demand will urge them to implement one ... SVG’s future and success largely depends on high−quality implementations and a strong user−community. Two W3C working−groups are currently engaged in the further development of the SVG standard:

- **SVG Mobile Working Group, SVG 1.1.**

A new working group had been established to discuss specific requirements of SVG for mobile devices, such as PDAs and cellular phones. The group proposes to have two separate profiles, that represent a subset of the SVG 1.0 specification, in order to be able to view files on devices with limited memory, CPU power and bandwidth. The two profiles are SVGB (SVG Basic), for higher level and SVGT (SVG Tiny), for lower level devices. The currently discussed selection of features, elements and attributes may be read at [5, GRAHAM/CAPIN, 2001]. The group lists the following possible applications for Mobile SVG:

- Location−Based Services
- Mapping and Positioning
- Animated Picture Messaging
- Multimedia Messaging
- Entertainment
- Industrial Applications

- eCommerce
- User Interfaces

- **SVG 1.1/2.0 Working Group**

Although the SVG 1.0 specification only recently got a W3C recommendation, a new Working Group already discusses possible features/additions for the next minor and major specification upgrades (see [6, JACKSON, 2001]). The minor upgrade, SVG 1.1, will basically include features needed for the use of SVG with mobile devices (see above). It will be achieved by a modularized structure (similar to X3D profiles). Additionally to the SVGT and SVGB profile, a SVG printing profile could be worked out. SVG 2.0 will be a major upgrade. Because the results of the W3C working groups (requirements, drafts and specifications) are usually publicly discussed, feedback from the public as well as invited experts and SVG WG members is welcome.

One of the working group's main goals is to place SVG as a standard feature on desktops (web browsers, graphical applications, authoring tools, file interchange), mobile and small devices (browsers, user interfaces, automotive systems), printers and industrial applications. Although this may sound as an ambitious goal it may well be a big advantage for the software industry − having one standard that fulfills most requirements of common output and viewing devices. The WG will also push SVG as an graphics exchange format between graphics software applications. The W3C is also willing to split SVG up to several sub−profiles, but will provide comprehensive test suites for the different implementations to check up against.

It is not very useful to list all suggestions in detail. Improvements include requirements demanded by several specific domains, such as cartography, GIS, CAD and design. You may view the suggestions and make well grounded suggestions, if you are missing important features within the current specification and discussion topics. Take your chance if you can actively contribute to existing specifications!

14. SVG Resources

SVG Specification:
http://www.w3.org/TR/SVG/
http://www.w3.org/TR/SVGMobileReqs (Working Draft!)
http://www.w3.org/TR/SVG2Reqs (Working Draft!)

Selected SVG News−, Tutorial−, Links− and Demo−Sites:
http://www.w3.org/Graphics/SVG/Overview.htm8 (News and Links)
http://www.adobe.com/svg/community/external.html (Links)
http://www.adobe.com/svg/demos/main.html (Dynamic SVG examples)
http://www.adobe.com/svg/basics/intro.html (SVG developers tutorial)
http://www.kevlindev.com/ (SVG tutorials and examples, SVG and GUI)
http://www.carto.net/papers/svg/ (SVG tutorials, examples, articles)
http://www.pinkjuice.com/SVG/SVGlinks.htm (SVG links)
http://www.sun.com/software/xml/developers/svg/ (SVG at Sun)

Selected SVG Implementations:
http://www.adobe.com/svg/ (Home of Adobe SVG Viewer)
http://xml.apache.org/batik/ (Home of Batik, Java2 based SVG Viewer)
http://www.alphaworks.ibm.com/tech/svgview (IBM SVG Viewer)
http://sis.cmis.csiro.au/svg/ (CSIRO, SVG toolkit)

http://www.croczilla.com/svg/index.html (Mozilla SVG)
http://www.in−gmbh.de/en/Products/in−gmbh/sphinxopen/editor.htm (SVG Editor)
http://www.jasc.com/products/webdraw/ (Powerful SVG Authoring System)
http://broadway.cs.nott.ac.uk/projects/SVG/svgpl/ (SVG Perl Module)
http://sketch.sourceforge.net/ (Vector graphics software for Linux, Open Source, SVG exp.)
http://koffice.kde.org/kontour/ (Vect. Graphics softw. for Linux, Open Source, SVG im/exp.)
http://www.svgmaker.com/ (SVG printer driver, works like Distiller)
http://www.mayura.com/ (Graphics software with SVG export)

SVG mailinglists and discussion−groups:
http://groups.yahoo.com/group/svg−developers
http://www.carto.net/papers/svg/mail_e.html (mapping specific)
http://www.adobe.com/support/forums/main.html (see SVG Forum)
http://lists.w3.org/Archives/Public/www−svg/

15. References

[1] ADOBE Systems (2001): »Adobe SVG DRAW«, http://www.adobe.com/svg/demos/main.html

[2] APACHE.ORG (2001): »Batik SVG toolkit«, http://xml.apache.org/batik/index.html

[3] CRUIKSHANK David et.al. (2001) »WebCGM Profile«, http://www.w3.org/TR/REC−WebCGM/

[4] FERRAIOLO Jon, et.al. (2001): »Scalable Vector Graphics (SVG) 1.0 Specification«, http://www.w3.org/TR/SVG/

[5] GRAHAM Rick and Tolga CAPIN (2001): »SVG Mobile Requirements«, W3C Working Draft 3 August 2001, http://www.w3.org/TR/SVGMobileReqs

[6] JACKSON, Dean (2001): »SVG 1.1/2.0 Requirements«, W3C Working Draft 3 August 2001, http://www.w3.org/TR/SVG2Reqs

[7] LILLEY, Chris (2001): »SVG implementations«, http://www.w3.org/Graphics/SVG/SVG−Implementations.htm8

[8] LINDSEY, Kevin (2001): »KevLinDev − resources for SVG, Javascript, Perl, PHP, C«, http://www.kevlindev.com/

[9] NEUMANN, Andreas (2001): »Example for Animation along a path«, http://www.carto.net/papers/svg/path_animation_e.html

[10] NEUMANN, Andreas (2001): »Comparing .SVG and .SWF file−format«, http://www.carto.net/papers/svg/comparison_flash_svg.html

[11] NEUMANN, Andreas and Andréas M. WINTER (2001): »SVG webmapping examples«, http://www.carto.net/papers/svg/first_e.html

[12] OPENSWF.ORG (2001): »The Source for Flash File Format Information«, http://www.openswf.org/spec.html

[13] WINTER, Andréas M. (2000): »OECD Atlas Europa«, http://www.carto.net/papers/svg/eu/oecdatlas.html

[14] WINTER, Andréas M. (2001): »SVG basic shapes«, http://www.carto.net/papers/svg/shapes_e.html

About the authors:

Andreas Neumann
Institute of Cartography, ETH Zurich
ETH Hoenggerberg
CH−8093 Zurich
Email: neumann@karto.baug.ethz.ch
Web: http://www.karto.ethz.ch/neumann/

Andréas M. Winter
Freytag & Berndt
Brunnerstrasse 69
A−1231 Vienna
Email: andre.mw@gmx.net
Web: http://www.carto.net/

Both authors studied Geography/Cartography at Vienna University and started to work on SVG for interactive webmapping in 1999/2000. Their joint project »carto.net« (http://www.carto.net/) offers a forum for cartographers to present their work, with emphasis on webmapping projects. Carto.net offers tutorials, a mailinglist for discussion, documented examples, implementations and articles in order to ease learning SVG for cartographers. The examples, besides other interactive SVG websites, also show the potential of SVG for interactive webgraphics and webmapping purposes. Andreas Neumann currently starts working on his PHD at the cartographic institute as part of a project titled »Distributed Mapping on Demand«, while Andréas M. Winter is working at an Austrian mapping company located in Vienna.