XML-Technologies - 232055

SVG: Scalable Vector Graphics

Aurelien Beltrame – s0221066

Gábor Fehér – s0221090

Summary

Introduction

SVG means Scalable Vector Graphics. It is a free open-standard that defines a vector graphic format to create two-dimensional graphics by using XML. The W3C started to work on it more than ten years ago. The development of SVG specification is supported by many software companies (for example: Adobe, Apple, Corel, HP, IBM, Sun Microsystems, ...).

In this report we start by listing what is necessary in the SVG format, after we continue with a large part to explain the main possibilities of SVG format, with examples. Then we report if this format is a good answer to the demand of first part, and we speak about the usage in general. To conclude we propose a part about tools to use SVG.

Motivation

1. Static picture

Modular
The idea is to use a language that supports modularity to allow the reuse of parts of SVG files. It is also necessary to have the possibility to include SVG in web page in different way.

Scalable
The idea of using vector pictures is to make it possible to increase or decrease easily the size (height/width) of the picture without changing its memory use.

Compact
One kind of programs that W3C promotes to use with SVG are the web-browsers. This means it is necessary to have a small file sizes to make downloads fast. The use of XML text also allows using non-destructive text compression, where algorithms are known and performing.

Portable and XML-compatible
Because SVG is XML oriented, it is decided to make SVG totally compatible with XML. This allows the use of all existing XML tools.
Namespace management is also required, to use SVG in other XML documents.

Text in picture
The possibility to put text into an SVG picture is required. For sure, this demand means the text has to be given with the font, size, color and all other properties like a normal other shape in an SVG. It is also necessary that this text can be readable by crawler of search engine.

2. Dynamic picture

Animation
With the presence of other vector files like SWF (Flash) the possibility of animated pictures is required. For example it is necessary to be able to move some shapes, to transform a kind of shape to another kind of shape, ...

Event
One of the major usages of SVG is in web-browsers, so it is very interesting to have event detection and to allow modification of the image in answer to this event (for example a mouse click).

Scripting
Because one of the ideas of SVG is to build picture from other XML documents, or data from some databases, it is necessary to have the possibility to use to extract or to modify, the structure of an SVG file.

Explanation

1. Introduction of explanation

An SVG document is actually an XML document that contains vector-graphical elements. These elements can be rendered by a program to the screen. Its root element is named 'svg', and it basically contains graphic elements, but it also can contain for example metadata, scripts and animations. A typical SVG document can look like this:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
        "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">

<svg width="100%" height="100%" version="1.1" xmlns="http://www.w3.org/2000/svg">
   <!-- graphic elements, etc. -->
</svg>
```

An SVG document usually appears in a host document, for example in a HTML or in another SVG document. Its actual size is determined by the 'width', 'height' and other attributes of 'svg' and size constraints of the host.

The SVG standard has a modular structure which means that it is decomposed into such parts that can be implemented separately or in combination with each other or with other XML standards. This also makes it possible to define profiles that only implement a set of SVG features, with possibly some small extensions or restrictions. Such profile is for example SVG Tiny or SVG Basic, which are designed for mobile applications. In the following subsections, we will go through the basic functionality of SVG 1.1, introducing the basics of features of most of the modules.

2. Basic visual elements

a. Shapes

The main part of an SVG document is the list of visual elements. Their order is important, because it also defines the z-order: if two elements overlap, the one that is defined later will be the visible. The simplest way to define a visual element is the use of an SVG basic shape. The basic shapes are:

- rectangles
- circles
- ellipses
- lines

- polylines
- polygons

Actually, there is one element named path, which is the generalization of all these, in other words, using a path element any of the above and more can be represented. All these elements share the same attributes for specifying their visual style: like fill color, border color, patterns, opacity, etc. This will be discussed in more detail in the next section.

Rectangles, with optionally rounded corners:
Element: 'rect'.
Attributes: 'x', 'y', 'width', 'height', 'rx', 'ry'. 'x' and 'y' specify the top-left corner of the rectangle, they are optional and their default values are zero. Attributes 'width' and 'height' specify the size of the rectangle, while 'rx' and 'ry' specify the roundness of the corners. These two attributes are optional.
Example:
<rect x="20" y="30" rx="8" ry="8" width="100" height="40"
style="fill:red;stroke:black;stroke-width:2;"/>
The style element controls the visual style of the rectangle. In this particular case, it specifies the rect to be filled by red, and bordered by a black line. This style will be used for all the following elements, and more styling possibilities will be introduced later.

Circles:
Element: 'circle'.
Attributes: cx, cx, r. cx and cy define the center of the circle, they are optional and their default value is 0. r specifies the radius.
Example:
<circle cx="200" cy="50" r="30" style="fill:red;stroke:black;stroke-width:2;"/>

Ellipses, with axes aligned to the main axes of the coordinate system:
Element: 'ellipse'
Attributes: cx, cx, rx, ry. cx and cy define the center of the ellipse, their default values are zero. rx and ry define the length of the x-axis and y-axis radius of the ellipse.
Example:
<ellipse cx="330" cy="50" rx="50" ry="20" style="fill:red;stroke:black;stroke-width:2;"/>

Lines define simple line segments with two endpoints. It's important to note that they have no interior part which can be filled, they only have a boundary part.
Element: 'line'
Attributes: x1, y1, x2, y2. x1 and y1 is the starting point of the line, x2 and y2 is the ending point. Each attribute is optional, and defaults to zero.
Example:
<line x1="20" y1="120" x2="120" y2="200" style="fill:red;stroke:black;stroke-width:2;"/>

Polylines define a connected sequence of line segments. They are usually used for open shapes, but despite this they have an interior which can be filled with a color.
Element: 'polyline'
Attributes: 'points': a list of coordinate pairs that make up the endpoints of the segments. The first and second, second and third, ..., next to last and last coordinate pairs define the segments. It is an error if the list contains an odd number of values.
<polyline points="160 120 240 120 240 200 160 200 160 180 220 180"
style="fill:red;stroke:black;stroke-width:2;"/>

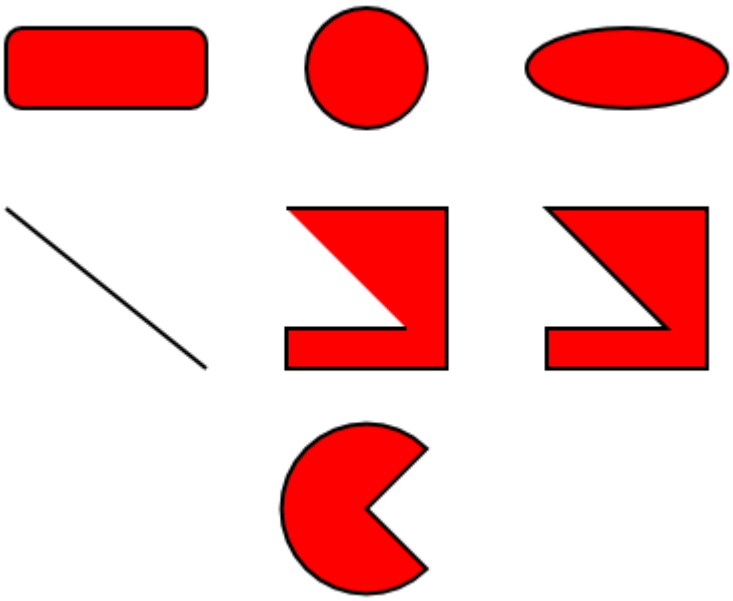
Polygons define a connected and closed sequence of line segments:
Element: 'polygon'
Attribute: points. The difference from polylines is that now the last and first coordinate pairs also define a line segment.
Example:
<polygon points="290 120 370 120 370 200 290 200 290 180 350 180"
style="fill:red;stroke:black;stroke-width:2;"/>

Paths can define a wide range of geometrical shapes.
Element: 'path'
Attribute: d, d contains the path data, in other words, the description of the geometry of the path. It is in a language that is defined particularly for this purpose. It follows the concept of current point model, which is analogous to drawing with pen on a paper: the pen is at point called the current point, and we have commands to move this pen and draw with it. Commands can be used to:

- draw lines
- draw curves (cubic and quadratic Bézier-curves)
- draw arcs

Example:
<path d="M200,270 l30,-30 a42.43,42.43 45 1,0 0,60 z" style="fill:red;stroke:black;stroke-width:2;" />
This example creates a 270° arc of a circle and drows two lines from the center to the arc endpoints.

The following image is a rendering of the above SVG elements. From left to right, from top to down: rectangle, circle, ellipse, line, polyline, polygon, path. Note that the difference between a polyline and polygon is the "missing" line segment.



There are a wide range of possibilities to manipulate these shapes: different styles or geometric transformations can be applied to them. Another example is to use shape to clip another one, so in other words, they can be intersected. These will be explained in more detail later.

b. Texts

An SVG document can also contain texts. An important characteristic of them is that they are rendered the same way as the previously mentioned shapes. This means that all the manipulations that can be done to shapes can also be done to texts. In addition to this, SVG texts also support typesetting text for different fonts and bi-directional writing. Since fonts by the same name can slightly differ on different systems, authors have the possibility to attach the font definitions to SVG files. This is done the same way as in HTML/CSS.

All the text in an SVG image is stored in 'text' elements.
Name: 'text'
Attributes: 'x', 'y', 'dx', 'dy', 'rotate', 'textLength', 'lengthAdjust'. Now we only highlight the use of 'x', 'y' and 'rotate'. When 'x' and 'y' are two single coordinate value, then they specify the position of the first character. In this case, the positions of subsequent characters are calculated according to the rules specified in the font. In simple words, the characters follow each other with souitable-sized spaces between them, like in a text. Either x or y or both can also be a sequence of n coordinate values. In this case, they specify the coordinates of the first n characters, thus overriding the default calculations of the fonts. Attribute dx and dy have similar semantics to x and y, but they define relative distances between characters. Attribute rotate specifies a rotation for each character. It can also contain n values, in this case, the first n characters are rotated. The text to show is enclosed between the tags of the 'text' element.
Examples:

```
<text x="10" y="50" style="fill:red;stroke:black;stroke-width:1;font-size: 40px;">hello, world</text>
<text x="10" y="110 130 150 170" style="fill:red;stroke:black;stroke-width:1;font-size: 40px;">
        hello, world
</text>
<text x="10" y="210" rotate="20" style="fill:red;stroke:black;stroke-width:1;font-size: 40px;">
        hello, world
</text>
```

Alhough the above mentioned attributes provide flexible ways to manipulate text, there are other, more convenient ways: text can be defined to follow the line of a path:

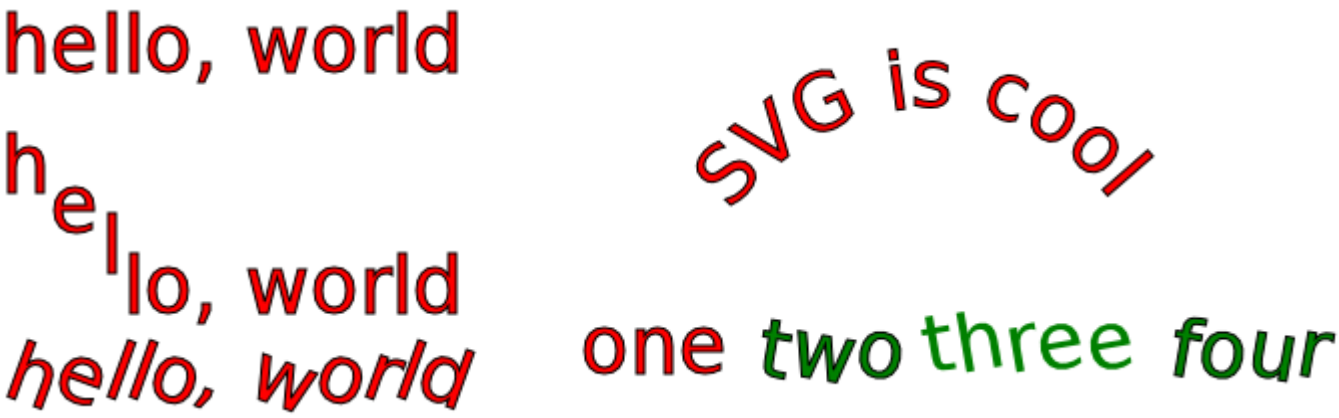
```
<defs>
  <path id="MyPath" d="M 300 120 C 355 50 445 50 495 120" />
</defs>
<text style="fill:red;stroke:black;stroke-width:1;font-size: 40px;">
 <textPath xlink:href="#MyPath">SVG is cool</textPath>
</text>
```

In this example, we defined a path, but not to display it, but to use it in the following 'text' element. Note that the text to display is contained in a subelement named 'textPath'. In general, subelements of 'text' can be used to alter the text properties defined by the 'text' element or defined globally. These can be for example rotation, stroke or fill colour:

```
<text x="300" y="200" style="fill:red;stroke:black;stroke-width:1;font-size: 40px;">
 one
```

```
 <tspan rotate="10" style="fill:green;">
  two
   <tspan rotate="-10" style="stroke:none;">three</tspan>
  four
 </tspan>
</text>
```

The following picture shows how the examples of SVG text are rendered by the Opera Web Browser. Note that for the second "hello, world" text, the y coordinates of the first 4 characters are controlled by the 'y' attribute, but the x coordinates are calculated automatically.



There are some more possibilities for defining graphical elements in SVG documents. External images can be included, and several graphical elements can be grouped to form one element. These possibilities will be discussed in the next section.

2. Advanced structures

SVG documents make use of the XLink standard to reference several kinds of elements. We already saw an example for this for the text "SVG is cool": the path that the text was aligned to, was not defined in the element of the text, but before that.
Elements can be identified by their 'id' tags, in the form of #id. More precisely, this is only enough to identify elements from the same document, because elements from another documents are also possible to be referenced. Element referencing is used for several purposes, not only for text paths. A simple example is the 'use' tag:

```
<defs>
  <circle id="c1" cx="400" cy="400" r="50" />
</defs>
<use xlink:href="#c1" />
```

Elements that are defined enclosed in 'defs' tags are not rendered. The 'use' element can be used to specify where to render them. It is also possible to use an element more than one times, or to override certain properties at the place of use:

```
<use xlink:href="#c1" x="120" style="fill:green;" />
```

An interesting possibility is to include elements that are defined in other documents:

```
<ellipse cx="230" cy="250" rx="110" ry="100"
        style="fill:url('http://www.w3schools.com/svg/radial2.svg#grey_blue')"/>
```

By the use of 'image' elements, it is also possible to include external images. These can be SVG images, or even raster based images like PNG or JPEG.

```
<image xlink:href="http://www.w3.org/Graphics/SVG/logo/tmpSVGlogo.png"
        x="10" y="100" width="51" height="50" />
```

Elements can also be enclosed in groups. Groups are actually 'g' elements, which can have the same structure as 'defs' elements, but they are rendered on the screen by default. Groups make it possible to apply the same style, geometric transformations or other operations to a set of elements in a convenient way. Groups are especially useful for storing meta-information, like descriptions and titles. There are two elements defined for this 'title' and 'desc'. The following example shows a possible use:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg SYSTEM "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="4in" height="3in" version="1.1"
        xmlns="http://www.w3.org/2000/svg"
        xmlns:mynmspc="...">
<g>
        <title>
                Map of South America
        </title>
        <g>
                <title>Colombia</title>
                <desc>
                        <mynmspc:population>44,928,970</mynmspc:population>
                        <mynmspc:GDP-PPP>$399.4 billion</mynmspc:GDP-PPP>
                </desc>
                <!-- visual elements of Colombia come here-->
        </g>
        <g>
                <!-- next country comes here -->
        </g>
</g>
</svg>
```

These group elements can also be nested in each other. This example also showed that custom namespaces can be imported to SVG documents, and this way such SVG documents can be defined that are very rich of information. Note that this meta-infromation will not be-rendered by user agents. Titles may appear as mouseover texts and descriptions can be used by specific tools. Document level metadata can also be specified in a 'metadata' element which is a child of the outermost 'svg' element. A similar element to 'g' is 'symbol'. It also contains a group of elements like 'g', but the main difference is that symbols are not rendered by default, and their purpose is to use them by multiple numbers of 'use' statements.

3. Styling

For the shapes and texts in the previous section, always the same style was applied, except for the text "one two three four". In SVG there are flexible ways to apply graphical styles for one or more elements. They are quite similar to how CSS in HTML works. For example a simple to apply a certain graphical style for an element is to use its 'style' attribute. We already saw this for the text "one two three four". In the first part of this section, the different styling possibilities will be demonstrated using the 'style' attribute. In the second part, it will be demonstrated what other places can style information be placed.

SVG style sheets can have attributes that are common with HTML/CSS, but also can have such attributes that are specific to SVG. In the following list, we highlight some of the more interesting ones. The format of specifying styles is the same as for HTML: style="name1: value1; name2: value2; ...". For example:

```
<rect x="20" y="30" rx="8" ry="8" width="100" height="40
        style="fill: salmon; stroke: black; stroke-width: 3;" />
```

stroke: Specifies the boundary colour of the shape. It has the same possibilities as the following 'fill' attribute.
stroke-width: Specifies the width of the boundary of the shape, thus the width of the area the stroking applies to.
fill: Specifies the interior fill of the shape. It can be in any form that HTML/CSS allows, like: red, #ff0000, rgb(255, 0, 0), rgb(100%, 0%, 0%). Note that SVG supports an extended set of colour names, compared to CSS names. Filling is not only possible by a solid colour, but also with patterns or gradients. In this case, a pattern or gradient has to be defined before, and it can be referenced at the same place as the colour. The following example defines a gradient between the colours "blueviolet" and "chartreuse", the colour will change along the x axis. It will start with "blueviolet", transition to "chartreause" will start at 30% of the width of the element and it will finish at 70%, and continue from here with "chartreuse":

```
<defs>
  <linearGradient id = "g1" x1 = "30%" x2 = "70%">
    <stop stop-color = "blueviolet" offset = "0%"/>
    <stop stop-color = "chartreuse" offset = "100%"/>
  </linearGradient>
</defs>
<circle cx="170" cy="50" r="30" style="fill: url(#g1);" />
```

The gradient is defined by the 'linearGradient' element and referenced by its id width 'url' keyword for 'fill'. Not only linear, but radial gradients can also be defined. A pattern is defined and referenced in a similar way. A pattern is defined by graphic elements that are repeated - or tiled - over the filled surface. The possibilities for 'fill' property can also be used for other properties that define colours.
fill-opacity: A numeric value between 0.0 and 1.0. The closer this value is to zero, the more transparent the fill of the shape is. This can be interesting for example when more shapes overlap. In this case, as we mentioned earlier, the later-defined one would hide the overlapped parts of the first one. With the use of this attribute, this situation can be controlled better.
We highlight here some further possibilities to control styling of elements:

- specifying font properties for text
    - orientation and reading direction of text
- specifying the mouse cursor type to appear over an element
- for strokes
    - specify markers at end of line segments (like arrows)
    - specify dashing

Style sheets can not only be placed in 'style' attributes:

- Some of the style properties can also be defined as attributes, for example:
  <rect x="20" y="30" rx="8" ry="8" width="100" height="40"
  fill="salmon" stroke="black" stroke-width="3" />

- Global style-sheets with classes and selectors can be defined like for HTML. They can be either in the SVG document, or in external files. For example the repeated use of style attribute in the first example could have been avoided by putting the following fragment at the beginning of the SVG document:

```
<style>
svg {
        fill:red;stroke:black;stroke-width:2;
        font-size: 40px;
}
</style>
```

4. Coordinate system

The coordinate system in which the elements are defined is transformed to the view coordinate system

Before a graphic element is rendered on the screen of the user, it goes through a series of coordinate transformations. For the examples above, we assumed the default transformations are in action. When a transformation is applied to an element, then the element and all of its child elements are transformed. Transformations can also be nested. In this case, they are applied from the outside to inside, so first the transformations of the root 'svg' element are applied to all elements, then the transformations of container elements to contained elements, and then finally the specific transformations defined for shape or other graphical elements. There are two attributes that control these transformations: 'viewBox' and 'transform'. Attribute 'transform' contains a list of transformations, which are applied subsequently. This list can include rotations, scaling, translations, skewing and finally, general affine transformations defined by a 3x3 matrices.

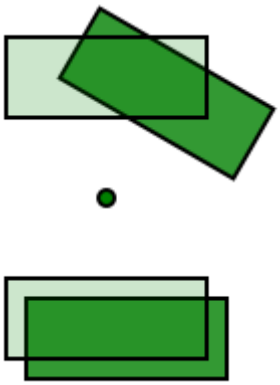
In the following example, four rectangles and a circle are shown. The first two rectangles are grouped, and the group is rotated around the circle. However, after this the second rectangle is rotated back, and translated by a (10, 10) vector. The last two rectangles mark the original untransformed places of the first two. They are styled to be more transparent.

<circle cx="150" cy="100" r="4" />
<g ***transform="rotate(30, 150, 100)"****>*

```
    <rect x="100" y="20" width="100" height="40" style="fill-opacity: 0.8;"/>
    <rect x="100" y="140" width="100" height="40"
          transform="rotate(-30, 150, 100) translate(10, 10)" style="fill-opacity: 0.8;"/>
</g>

<rect x="100" y="20" width="100" height="40" style="fill-opacity: 0.2;" />
<rect x="100" y="140" width="100" height="40" style="fill-opacity: 0.2;" />
```

It can be checked on this example, that the transformations are actually applied in the order as they appear in the document. The following image shows how it is rendered. (The green fill and the black strike styles were set by a global style sheet.)



The area where the SVG document is rendered is said to be the viewport. By specifying a rectangle in the 'viewBox' attribute of 'svg', we apply such a transformation to the whole document, which stretches that particular area of the image to fill the viewport. It is also possible to define nested viewboxes in a document, like for example the 'symbol' elements define their own viewboxes when they are instantiated by 'use' elements. The 'viewBox' attribute can be used in this case and in any other cases when new viewports are created. There are also ways to keep aspect ratios of images in viewboxes.

In an SVG document, coordinate values can have unit identifiers, like in CSS. They can be on of em, ex, px, pt, pc, cm, mm, in or percentages. In case no unit is specified, the default one is pixel, 'px' in short. Pixels are also called user units. If there are no length-modifying transformations specified, one 'px' equals to one pixel of the parent environment. The SVG standard shows ways to better support containing maps in SVG documents: metadata about the geographical coordinate system of the map, and transformations of this system can be attached to the document by the use of Resource Description Framework.

5. Manipulating graphical elements: clipping, masking and filter effects

SVG supports clipping and masking, which means that one path can be used to define which parts of an element to show. The remaining parts of that element are clipped or masked. In case of clipping, the edges of the clipped area are sharp, in other words, the original object is preserved inside the clipping, and made transparent outside of that. A mask can also do this, but in addition to this, it also makes it possible to make some areas of the object semi-transparent by certain degrees.

For the rendering of semi-transparent areas (either defined by masks or the fill-opacity style attribute) on the top of already rendered objects, SVG uses a simple alpha compositing algorithm.

The following example shows how to define a clip path and apply it to an image:

```
<defs>
 <clipPath id="MyClip">
   <path d="M130 160 L370 150 330 250 140 230 z" />
 </clipPath>
</defs>
<image xlink:href="http://home.student.utwente.nl/g.feher/train.jpg"
        x="100" y="100" width="275" height="207" clip-path="url(#MyClip)" />
```

On the following figure, the left image was produced by this SVG code:

The right one was produced by the following mask...

A mask can be defined several ways, for example by using the RGBA data of an image, path, text or shape. In the following example, a gradient similar to that we defined in styling section is used to create a rounded rectangle with gradient fill colour. This rectangle is not rendered, but it is applied to a picture as a mask. Note that masks and clips can not only be applied to images, but on any other graphical elements as well.

```
<defs>
 <linearGradient id = "g1" x1 = "10%" x2 = "90%">
   <stop stop-color = "white" offset = "0%"/>
   <stop stop-color = "black" offset = "100%"/>
 </linearGradient>
 <mask id="mask1" x="410" y="110" width="255" height="187" maskUnits="userSpaceOnUse">
   <rect x="410" y="130" width="255" height="147" rx="8" ry="8"
        style="fill:url(#g1); stroke:none;" />
 </mask>
</defs>
<image xlink:href="http://home.student.utwente.nl/g.feher/train.jpg"
        x="400" y="100" width="275" height="207" mask="url(#mask1)" />
```

The visual appearance of elements can be altered by several different filters defined for SVG. They include among others gaussian blur, lighting effects, blending, convolution, color matrix, etc. Filters can be applied to single elements or to groups of elements for example with the use of 'g' element. More than one filter can be applied to a single element. A filter can be defined by the 'filter' element, and it can be applied to a graphic element by its 'filter' attribute. The following example shows how to do it:

In this case, a Gaussian Blur filter is defined, and it is applied to a group of a text and polygon element.

```
<defs>
  <filter id="MyFilter" filterUnits="userSpaceOnUse">
    <feGaussianBlur stdDeviation="1" />
  </filter>
</defs>


<g filter="url(#MyFilter)">
        <polygon points="290 120 370 120 370 200 290 200 290 180 350 180"
                style="fill:red;stroke:black;stroke-width:2;"/>
        <text x="296" y="220">SVG filters</text>
</g>
```

In the general case, more filters can be enumerated in the 'filter' element and then they are applied sequentially on the subject of the filter. In an even more general case, the input and output of these filters can be assigned to names, thus their application order can be controlled, and results of different filters can be combined in customized ways.

6. Interactivity

The SVG standard provides four mechanisms to interact with a user:

- Script executions, starts or ends of animations can be assigned to incoming user actions, like for example mouse clicks.
- Similar hyperlinks that exist in HTML can be added to an SVG document.
- In certain cases the user can zoom or pan on the image that he/she sees.
- The cursor of the pointing device may change when it is over certain elements.

a. User actions

Response to user actions is possible by adding event handlers to SVG elements. These handlers specify what to do when a certain event occurs. Events can not only be user actions but there are such ones that are triggered by scripts. Possible events are:

- (mouse)pointer related events, like click, move, etc.
- DOM-related events, for example when element nodes are inserted or modified
- SVG document-related events, for example at load, at user zoom/pan, or at certain errors
- animation-related events, when an animation starts, stops or repeats

Note that keyboard events are not supported. Some of these events can be set for an element by setting their corresponding attributes on those elements. However some of the events has no such corresponding attributes, but all of them can be set through the DOM interface.

b. Scripts

Scripting in SVG is possible in different languages. The default one is EcmaScript, which by the way also forms the basis of JavaScript used on the web. Scripts can access the elements of the SVG document through the DOM - Document Object Model interface. The following example shows how to create a simple event handler function and assign it to the click event of a polygon. The event handler function uses DOM to change the fill attribute of the object that called it, from red to green or from green to red.

```
<script type="text/ecmascript"> <![CDATA[
   function move_obj(evt) {
      //change color of caller object through DOM
      var obj = evt.target;   //acquire caller object
      var fill = obj.getAttribute("fill");  //acquire fill attribute
      //set new fill attribute:
      if (fill == "red") obj.setAttribute("fill", "green");
      else obj.setAttribute("fill", "red");
   }
]]> </script>
<polygon points="290 120 370 120 370 200 290 200 290 180 350 180" fill="red"
         style="stroke:black;stroke-width:2;" onclick="move_obj(evt)" />
```

Scripts are defined in 'script' elements and it is a good idea to put them into a CDATA sections to avoid XML parsing of them.

c. Animation

SVG supports animation based on the SMIL language. SMIL is a general purpose animation language in XML, and SVG integrated it to be able to animate SVG elements. The standard defines for each SVG elements, which of their attribute values can be animated. This means that the SMIL language parts can define certain attributes of certain elements to change over time. In the following simple scenario the 'animate' and 'animateTransform' elements are used to specify how the size of an ellipse and a rotation of a group of elements changes over time:

```
   <g transform="translate(230, 150)">
      <g>
         <ellipse cx="0" cy="0" rx="50" ry="20" style="fill:red;stroke:black;stroke-width:2;">
            <animate attributeName="rx" attributeType="XML" begin="0s" dur="2s" fill="freeze"
                  from="50" to="100" />
            <animate attributeName="ry" attributeType="XML" begin="1s" dur="3s" fill="freeze"
                  from="20" to="40" />
         </ellipse>
         <text x="-15" y="5">hello</text>

         <animateTransform attributeName="transform" attributeType="XML"
               type="rotate" from="0" to="45" begin="2s" dur="4s" fill="freeze" />
      </g>
   </g>
```

As we can see, the animation-related elements specify which property should be animated, in which time window, and between which values. Zero time is the time when the document is loaded. With 'animate' elements, we can interpolate certain single attributes over a given time. Since the inner structure of a 'transform' attribute is more complex, the 'animateTransform' element is provided to animate transformation properties. There are other similar elements provided for animating specific types of attributes. All these provide a declarative way to describe animations. It is also possible to programmatically specify animations with scripts and DOM interface.

d. Hyperlinks

SVG defines an 'a' element analogous the 'a' element of HTML. Linking is based on the XLink standard in SVG. The 'xlink:href'attribute of 'a' specifies the target of the link. It can be any web resource that has an URI, including elements of the current document. For example, we can easily make a link of one of the previously used examples:

```
<svg ... xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
  <a xlink:href="http://www.utwente.nl">
    <polygon points="290 120 370 120 370 200 290 200 290 180 350 180"
        style="fill:red;stroke:black;stroke-width:2;"/>
  </a>
</svg>
```

The SVG root element is shown to emphasize that the namespace for XLink should be imported.

Evaluation

1. Solves the problem?

About the static picture, SVG format offers a lot of possibilities. We can obtain a freely-scalable picture, totally xml-compliant, with some text and eventually external links.

The compactness of this format is enough to be used on the web.

About the dynamic picture, the format allows the creation of any animation.
The format takes only care of mouse events and document events (size modification, DOM events), there is no integration of keyboards events.

About (meta)data, SVG format is able to use text and even XML as description or metadata. It allows a crawler of search engine to have precise information of what this SVG should represent.

2. Usage in practice

The SVG format is good in regards of what it is required (see previous part "Motivation"). In spite of all its advantages and possibilities this file format is not really used on the web. In this part, we give some information and comments to try to understand this fact.

Firstly, SVG is a vector format, and lots of people do not need vector pictures. Because people do not need vector format to exchange some digital photos, or to build web sites with textures in backgrounds or any other possibility, the format was never in the spotlight.

Secondly, and probably one of the main reason is the compatibility with Microsoft Internet Explorer. Or it is probably better to say the incompatibility, because Microsoft never took the time to develop a native compatibility with this format under Internet Explorer.
To try to democratize this format, some people built plug-in to permit the usage of SVG in IE. (One of the team was Adobe SVG plug-in, before Adobe bought Macromedia).
But in the same time, Flash was developed by a good and bigger team than SVG-plug-in teams and Macromedia Flash become a really famous and fashionable software. And the Flash plug-in was more performing than SVG plug-in. And because Flash format was proprietary, there is no problem with different implementations (because SVG plug-ins do not implement all SVG specifications).

So, lot of web programmers and designers had a choice: use SVG or SWF (flash). Probably because of better performance and marketing of Flash, the Macromedia Format became the most used format. It is also necessary to remind that XML was not really in "fashion" many years ago. The fact that Flash is a proprietary system can explain why some web-designers use it: because it is not really easy to do "reverse-engineering" of SWF file by comparison with a SVG file (right click, view source and it is possible to see everything). So web designers/programmers can keep "safe" intellectual properties.

Nevertheless, SVG format is used in specific cases.
It is use for static pictures in the case of "corporate pictures". By corporate, we mean some logo for companies or organizations. Because SVG is vector format and totally open source, it erase all problem of compatibility.

In the same kind of ideas, SVG can be used for any document with vector picture where the scalability is required, for example, a page of advertising which need to be print on a small flyer or in a big poster.
However, SVG suffer of concurrencies from Adobe Illustrator format, which is also in use by designers for these kind of graphical works.

Because SVG is a totally open xml-format, SVG files can be generated by any programs. So, it is very simple to generate an SVG file from statistics, or any data that is possible to compute on a server. That is why SVG is in use to generate pictures by a server from any graphical data...
There are two different types of utilization. It can be to show statistical data on graphical representation. The server generate with a database, the statistics which are needed, and send to the user a SVG or a web page with a SVG.
The second kind of server usage is in a Geographical Information System.
A Geographical Information System is a system which regroups a huge number of data, generally a geographic map with lot of information like level/depth of earth/water, road, forest, building, etc... A unique system can have a lot of data on the same area, for example road, water pipes and electric wires, if it is shown in the same times a normal human cannot use it. That is why the GIS has to generate a map with only the characteristics which are required, for example a map with road and forest to help firemen during summer. Or a map with middle depth water for scientists who want to explore a lake.

With the possibilities of dynamic SVG, it is also possible to code some function and to use them when the system build the SVG for a user. To come back to the firemen example, it is possible to imagine a map with road, forest, water and with an animated model of last fire. Or in the example of scientific exploration, it is possible to display a lot of information for each point of the map when the user clicks on it.

3. Current state

The current version of the W3C complete specification is 1.1 in date from 14th January 2003. (The last edition is 30th April 2009)

W3C is working on the next version (1.2): the last "Tiny" specification is dated from 22th December 2008, and the last "Full" specification draft is dated from 13rd April 2005.

Tools support

There is a lot of software which use SVG format. We select a few softwares for each category.

1. Web browser

In spite the lack of SVG support by Internet Explorer, there are some plug-in for IE and the other "usual" web browsers can render SVG.
Some people regularly test all this web browsers in respect to the specification and show all results. For this part we follow the results available on codedread.com.

First of all, there is no web browser which implements correctly the whole specification. If we look more in details, Safari, Chrome and Firefox have some troubles with Declarative Animation system for their last public build, but it is working in progress in their beta builds. Opera does not have this problem. The author of this web page finds that all these web browsers can perform at least 60 % of SVG specification. But if we do not take dynamic specifications this score goes to more than 80 % without problems.

In the next two years all used web browser can probably implement more than 90 % of SVG specification (for Safari, Chrome and Firefox, if they implement Declarative Animation, they will increase their score with 20%), and last Opera build already has a score of 94% of validation.
The same kinds of troubles are also visible with SVG plug-in for IE.

2. Drawing

The most famous software is probably Adobe Illustrator but it is not the only one which can use SVG. In proprietary software there is also CorelDraw which can use it. In open source software there are Inkscape and Zara Xtrem LX. Of course, there exist more than these five software.

For comparaison between Illustrator and Inkscape, Illustrator is more powerful than Inkscape (more possibility, better management of huge SVG files) but Illustrator take also more resources to be used (especially in RAM memory).

3. Text and/or XML editor

A SVG file is an XML file so it is possible to use XML editor to write directly the source of your SVG picture. With the possibility of theses program (syntax highlighting by colors, automatic close of markup, ...) it is not so long to write an SVG document.
For example we can quote XML Spy, Notepad++, ...

4. Library

In the same case of drawing or text editor software we do not write many pages, but we select two main pieces of code which can help to build software with SVG management: Batik and Cairo.

Batik is a Java Toolkit used to generate and manipulate SVG files. It is used by some programs to render picture in graphical user interface. The main tools of these toolkits are a generator of SVG, a modificator (with the DOM), an SVG visualizer and a module to transform SVG to raster formats (JPEG for example).

Cairo is an Application Programming Interface (API) with the essential function is to visualize and transform a SVG into other formats. It can be used to transform SVG into Postscript (PS) or into Portable Document Format (PDF). Cairo is available in lot of programming language (original core in C but it exists many bindings).

Conclusion

In our essay, we showed how the SVG format provides support for rendering static and dynamic vector graphics. Although the feature set is impressive, web authors should always be aware that implementations in browsers are not yet complete. We saw that because of this, it is not quite widespread over the web, even W3C's SVG standard document shows raster images by default.

However, the defined functionality of SVG hints that it has the potential to be more widespread on the web: it can be used to create things ranging from scalable images to interactive websites with rich graphics. Interestingly, we found that SVG became popular in other areas then the web: it is used as storage format for some graphical authoring applications and as exchange format between graphical applications. This is not a surprise however, since XML, on which it was built on, is well suitable for data exchange.

Sources

Official W3C specification 1.1: http://www.w3.org/TR/SVG/

Draft W3C specification 1.2: http://www.w3.org/TR/SVG12/

W3C specification of 1.2 Tiny version : http://www.w3.org/TR/SVGTiny12/

W3C information: http://www.w3.org/Graphics/SVG/

W3C XLink specification: http://www.w3.org/TR/xlink/

Wikipedia English page: http://en.wikipedia.org/wiki/Svg

Wikibooks English: http://en.wikibooks.org/wiki/XML_-_Managing_Data_Exchange/SVG

W3Schools Tutorial: http://www.w3schools.com/svg/

SVG Basics Tutorials: http://www.svgbasics.com/

Comparison between Web browsers: http://www.codedread.com/svg-support.php

Batik Java Toolkit: http://xmlgraphics.apache.org/batik/

Cairo API: http://cairographics.org/

Interview with Ted Gould: http://www.joshuazeidner.com/2008/02/ted-gould-svg-inkscape-and-web.html