MANCHESTER 1824

The University of Manchester

Scalable Vector Graphics

Tobias Schiebeck

Open to Europe Intensive Programme – TRABHCI

29th March - 9th April 2011

Universidad Politécnica de Valencia

MANCHESTER 1824

The University of Manchester

Overview

- Introduction
- Viewports
- Coordinate System
- Shapes
- Document Structure
- Grouping and Referencing
- Paths
- Animation and Scripting
- Conclusion

MANCHESTER 1824

The University of Manchester

What is SVG?

- Scalable Vector Graphics (SVG) is a graphics file format and Web development language based on XML.
- SVG enables Web developers and designers to create dynamically generated, high-quality 2D graphics from real-time data with precise structural and visual control.

Using SVG

- A viewer (a browser plugin) is required in order to view SVG files. There are many viewers are available, but Adobe who are constantly developing on the SVG standard has a viewer which can be downloaded from:
  - http://www.adobe.com/svg/viewer/install/main.html
- The following platforms are supported:
  - Microsoft Windows
  - Macintosh OS 8.6-9.1
  - Macintosh OS 10.1 native
  - RedHat Linux 7.1
  - Solaris 8

SVG Features

- With this powerful new technology, SVG developers can create a new generation of Web applications based on data-driven, interactive, and personalized graphics.
  - Data Driven - Due to the fact that it is written in XML, SVG content can be linked to back-end business processes such as e-commerce systems, corporate databases, and other rich sources of real-time information.
  - Interactive - It is possible to create web-based applications, tools or sophisticated user interfaces with common Web scripting and programming languages such as JavaScript, Java, and Microsoft Visual Basic.

SVG Features…

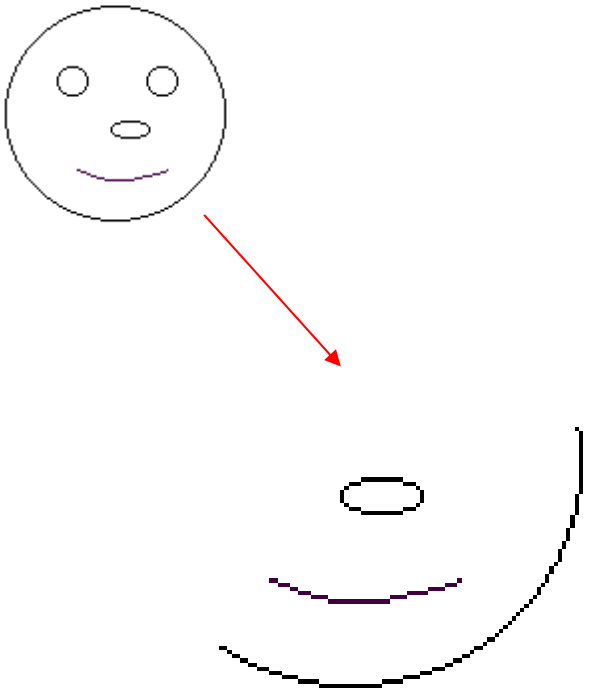
- Personalized Graphics - SVG content can be customized and tailored for many audiences, cultures, and demographic groups. Non-Roman and other unusual fonts and typefaces may be embedded in an SVG document. SVG can also be dynamically generated using information gathered from databases or real-time user interaction.

Raster Graphics vs Vector Graphics

- There are two types of graphics systems which must be discussed so as to understand the basic infrastructure of SVG:
  - Raster Graphics
  - Vector Graphics

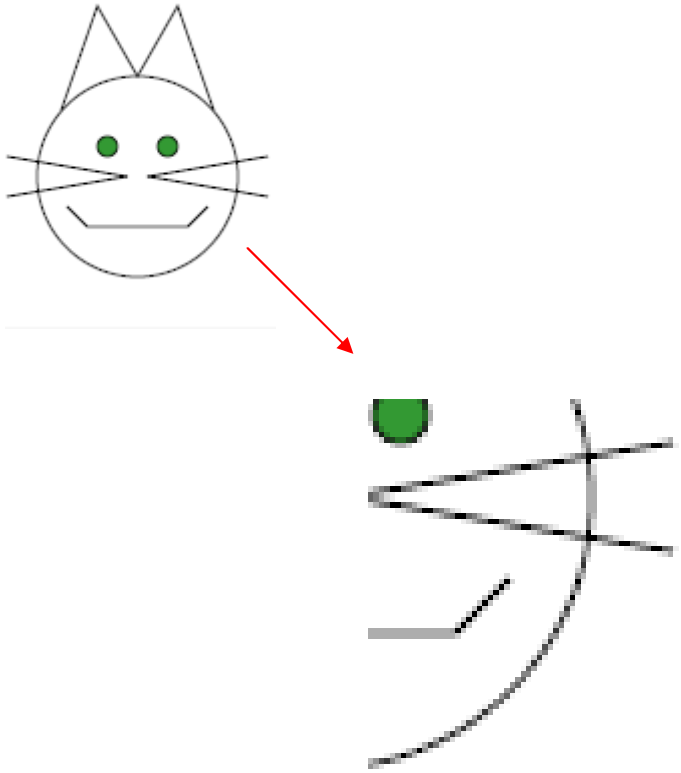
Raster Graphics

- In raster graphics an image is represented as a rectangular array of pixels. Each pixel is represented by its RGB colour values or as an index into a list of colours. Scaling a raster graphic results in loss of quality and correction techniques such as anti-aliasing must be used.



Vector Graphics

- In vector graphics an image is described as a series of geometric shapes. Vector graphics have one feature that makes them invaluable in many applications - they can be scaled without loss of image quality.



Simple Example

```
<?xml version="1.0"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN"
"http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="140" height="170">
<title>Cat</title>
<desc>Stick Figure of a Cat</desc>
<circle cx="70" cy="95" r="50" style="stroke: black; fill: none"/>
<circle cx="55" cy="80" r="5" stroke="black" fill="#339933"/>
<circle cx="85" cy="80" r="5" stroke="black" fill="#339933"/>
<g id="whiskers">
  <line x1="75" y1="95" x2="135" y2="85" style="stroke: black;"/>
  <line x1="75" y1="95" x2="135" y2="105" style="stroke: black;"/>
</g>
<use xlink:href="#whiskers" transform="scale(-1 1) translate(-140 0)"/>
<!--ears-->
<polyline points="108 62, 90 10, 70 45, 50 10, 32 62"
  style="stroke: black; fill: none;" />
<!--mouth-->
<polyline points="35 110, 45 120, 95 120, 105 110"
  style="stroke: black; fill: none;" />
</svg>
```

General

- File Header

```
<?xml version="1.0"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN"
"http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
```

  - ?xml – file contains XML
  - !DOCTYPE svg – the document is an SVG document
  - Svg10.dtd – defines the document type definition. A DTD allows XML code to describe its content for verification whenever the document is parsed; as such, SVG requires its own DTD. The SVG DTD is used to validate the SVG document by checking items such as the structure, the elements, and their attributes (and corresponding values).

General…

- Comments

```
<!--
This is a comment
-->
```

Viewports

- Before you can define the content of an SVG file, you need to define the size of the SVG document – in other words the viewport.
- A viewport can be compared to an artists canvas.
- The SVG element defines the viewport e.g.,

```
<svg width="140" height="170">
```

- In the above example the SVG graphic will be in a viewport of 140 pixels by 170 pixels.

Viewports…

- If no measurement unit is specified with a numerical value (for attributes such as width and height), an SVG viewer will assume this value to be in user space values. For example you could specify:

```
<svg width="2cm" height="3cm">
```

- The above gives you a viewport of 2cm x 3cm.
- You can even mix measurement systems e.g., a viewport of 2cm x 36pt.

```
<svg width="2cm" height="36pt">
```

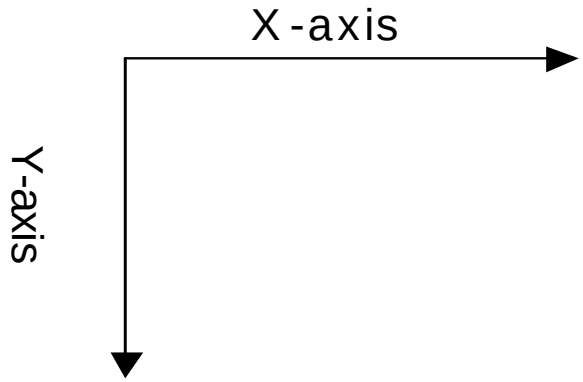
Viewports…

- The following table lists the popular measurement formats:

Unit Name	SVG Identifier	Unit Conversion to Pixels
Pixel	px	1
Point	pt	1.25
Pica	pc	15
Millimetres	mm	3.543307
Centimetres	cm	35.43307
Inches	in	90

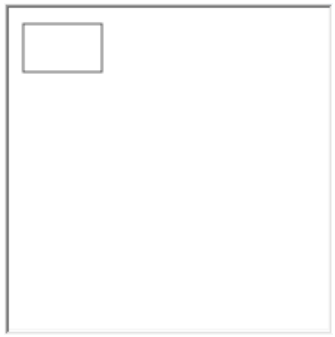
Coordinate System

- SVG is a 2D graphic display language, and thus all of its measurements reflect off an x,y axis.
- The most important difference that must be noted is that SVG's coordinate system is upside down. The x units increase positively as you move to the right of the y axis and increase negatively as you move to the left. Y units increase positively as you move down from the x axis and increase negatively as you move up.



Coordinate System…

- Unless otherwise specified the default coordinate start-point is 0,0.
- However, coordinates can be set as follows:



```
<svg width="200" height="200">
<rect x="10" y="10" width="50" height="30"
      style="stroke: black; fill: none;"/>
</svg>
```

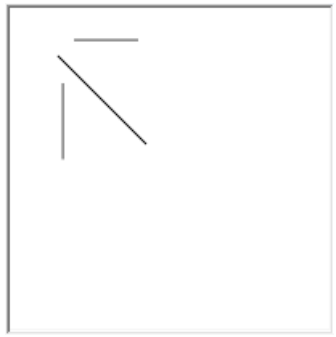
- The above example sets up the viewport to be 200 x 200 pixels. Then a rectangle is drawn whose upper left corner is at coordinate point (10,10) and whose width is 50, and height is 30 - again this is in pixels.

Shapes

- As in most graphics languages SVG supports basic elements such as:
  - Lines
  - Rectangles
  - Rounded Rectangles
  - Circles
  - Ellipses
  - Polygons
  - Polyline

Lines

- A line element has only four attributes:
  - x1
  - x2
  - y1
  - y2



- x1 and y1 represent the initial coordinates of the line, whereas x2 and y2 represent the end coordinates.
- A few examples:

```
<svg width="200" height="200" viewBox="0 0 200 200">
      <!-- horizontal line -->
      <line x1="40" y1="20" x2="80" y2="20" style="stroke: black;" />
      <!-- vertical line -->
      <line x1="0.7cm" y1="1cm" x2="0.7cm" y2="2cm" style="stroke: black;" />
      <!-- diagonal line -->
      <line x1="30" y1="30" x2="85" y2="85" style="stroke: black;" />
</svg>
```

Lines…

- A style property in the line element is needed, otherwise the line will be rendered invisible.
- Stroke specifies the line colour, and stroke width the thickness of the line:

```
<line x1="40" y1="20" x2="80" y2="20"
style="stroke-width: 10; stroke: black;" />
```

- The colour of a line can be set as well using the stroke command - note that we have specified the colour in previous examples:

```
<line x1="40" y1="20" x2="80" y2="20"
style="stroke-width: 10; stroke: red;" />
```

Lines…

- A colour can be set in a variety of ways:
  1. Using the colour name i.e.: aqua, black, blue, fuchsia, gray, green, lime etc.
  2. By specifying the six-digit hexadecimal form i.e., #rrggbb where rr is the red component, gg is the green component and bb is the blue component. The range of course is 0-ff.
  3. You can also specify the colour as a three-digit hexadecimal i.e., #rgb. The range in this case is 0-f.
  4. Lastly, you can use the rgb form, where each value is in the range 0-255 or specified as a percentage.
- Some examples of the above:

```
<line x1=”40” y1=”20” x2=”80” y2=”20” style=”stroke-width: 10; stroke: #9f9;” />
<line x1=”40” y1=”20” x2=”80” y2=”20” style=”stroke-width: 10; stroke: #99ff99;” />
<line x1=”40” y1=”20” x2=”80” y2=”20” style=”stroke-width: 10; stroke: rgb(255, 128, 64);” />
<line x1=”40” y1=”20” x2=”80” y2=”20” style =”stroke-width: 10; stroke: rgb(60%, 20%, 60%);” />
```

Lines…

- Stroke opacity (the opposite of transparency) can also be specified, where 0 is completely transparent and 1 is completely opaque e.g.,

```
<line x1="40" y1="20" x2="80" y2="20" style="stroke-opacity: 0.2; stroke-width: 10; stroke: red;" />
```

- You can also have dashed and dotted lines using the stroke-dasharray command. The following example does a nine-pixel dash with a five-pixel gap:

```
<line x1="40" y1="20" x2="80" y2="20" style="stroke-dasharray: 9, 5; stroke-width: 10; stroke: red;" />
```

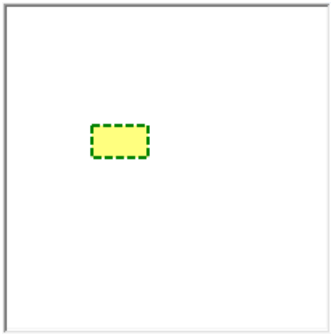
Rectangles

- Rectangles are created using the <rect> element.
- With rectangles you can define:
  - X
  - Y
  - Width
  - Height
- Using the style attribute again you can define:
  - fill colour
  - stroke colour
  - fill opacity
  - stroke opacity
  - stroke width
  - stroke-dasharray

Rectangles…

- An example of such is:

```
<rect x=”50” y=”70” width=”35” height=”20”
style=“fill: yellow; fill-opacity: 0.5;
       stroke: green; stroke-width: 2;
       stroke-dasharray: 5 2;” />
```



If you do not specify x or y - then a starting point of 0,0 is assumed. If you specify a width or height of zero - the rectangle is not displayed.

Rounded Rectangles

- The attributes for a rectangle with rounded corners are the same as for a rectangle, except you must specify the corner curvature using:
  - x-radius
  - y-radius

- The max number you may specify for rx (the x-radius) is one half the width of the rectangle; and the max ry (the y-radius) is one half of the height of the rectangle.
- If you only specify rx then it is assumed that rx and ry is equal and vice versa. An example:

```
<rect x="40" y="60" width="20" height="40" rx="5" ry="10"
style="fill: none; stroke: green;" />
```

Circles

- A circle is drawn using the <circle> element.
- The elements of which are:
  - centre x-coordinate (cx)
  - centre y-coordinate (cy)
  - radius (r)

- The default is to fill the circle with black and draw no outline e.g.:

```
<circle cx="30" cy="30" r="20" style="stroke: black;
fill: none;" />
```

Ellipses

- To create an ellipse you use the word “ellipse” and you will need to specify an x-radius (rx) and y-radius (ry) in addition to the centre x- and y-coordinates. e.g.:

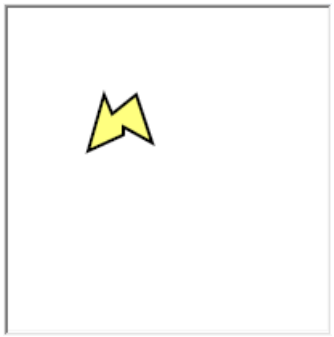
```
<ellipse cx=”30” cy=”80” rx=”10” ry=”20” style=”stroke: black; fill: none;” />
```

- If the radius is zero no circle/ellipse will be displayed.

Polygons

- The Polygon element allows you to specify points that describe a geometric shape.
- The points consist of a series of x- and y-coordinate pairs.
- Please note that you do not have to go back to the starting point as a shape will automatically be closed. e.g.,

```
<polygon
    points="60 60, 65 72, 80 60, 90 90,
            72 80, 72 85, 50 95"
    style="stroke: black;
           fill: yellow;
           fill-opacity: 0.5;
           stroke-width: 2;" />
```



Polygons…

- There are two fill rules for polygons, as you can appreciate it is rather hard to fill an object made up of intersecting lines. In the fill-rule you can choose a value of either *nonzero* or *evenodd*.
  - The nonzero rule determines whether a point is inside or outside a polygon by drawing a line from the point in question to infinity, it then counts how many times the line crosses the polygon's lines, adding one if the polygon line is going right to left, and subtracting one if it is going left to right. If the total comes out to zero the point is outside of the polygon and inside the polygon if it is not equal to zero.

Polygons…

- The evenodd rule also draws a line from the point to infinity, but it simply counts how many times that line crosses the polygons lines. If the total number of crossings is odd then the point is inside, if its even the point is outside.

Polyline

- The polyline element is used to create shapes where you don't want the lines to join up. It has the same attributes as polygon e.g.,

```
<polyline points="5 20, 20 20, 25 10,
                  35 30, 45 10, 55 30,
          65 10, 75 30, 80 20, 95 20"
style="stroke: black; fill: none; stroke-width: 3;" />
```

- When drawing a line or polyline you can also specify the shape of the endpoints by setting stroke-linecap to either butt, round, or square e.g.,

```
<polyline points="5 20, 20 20, 25 10,
                  35 30, 45 10, 55 30,
          65 10, 75 30, 80 20, 95 20"
   style="stroke-linecap: round;
          stroke: black; fill: none; stroke-width: 3;" />
```

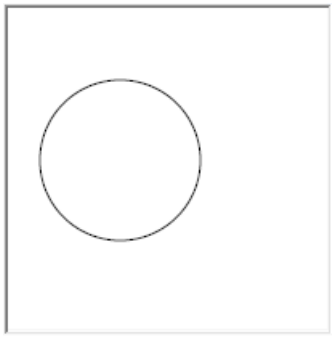
Document Structure

- XML encourages you to separate structure and presentation of your document.
- SVG allows you to specify presentational aspects of a graphic in four ways:
  - Inline styles
  - Internal stylesheets
  - External stylesheets
  - Presentation attributes

Inline Styles

- Inline styles are set by the style element within an object (what we have been using in the previous examples) e.g.:

```
<circle cx="70" cy="95" r="50"
        style="stroke: black; fill: none"/>
```



Internal Stylesheets

- An example of an internal stylesheet is:

```
<defs>
<style type="text/css><![CDATA[
    circle {
           fill: #ffc;
           stroke: blue;
           stroke-width: 2;
           stroke-dasharray: 5 3
           }
    ]]></style>
</defs>
```

- Then whenever the user specifies circle the above style would be used. However, you can override the internal style by using an inline style.

External Stylesheets

- An example of an external stylesheet (ext_style.css):

```
* {fill:none; stroke: black; }
                        /* default for all elements */

rect { stroke-dasharray: 7 3; }
circle.yellow { fill: yellow; }
.thick {stroke-width: 5; }
.semiblue {fill: blue; fill-opacity: 0.5;}
```

- An example of the external stylesheet being used:

```
<?xml-stylesheet href="ext_style.css" type="text/css"?>
```

- Inline styles will almost always render more quickly than styles in an internal or external stylesheet; stylesheets and classes add rendering time due to look up and parsing.

Presentation Attributes

- Presentation attributes are at the very bottom of the priority list.
- Any style specification coming from an inline, internal, or external stylesheet will override a presentation attribute.
- In the following example, the circle will take its style from the external stylesheet specified, therefore, the circle will be filled red - not green.

```
<svg width=”200” height=”200”>
   <defs>
       <style type=”text/css”>
       <![CDATA[ circle{fill:red;} ]]></style>
   </defs>
   <circle cx=”20” cy=”20” r=”15” fill=”green”/>
</svg>
```

- It is emphasized that using style attributes or stylesheets should always be your first choice.
- Stylesheets let you apply a complex series of fill and stroke characteristics to all occurrences of certain elements within a document without having to duplicate the information.

Grouping and Referencing Objects

- The <g> element is also known as the group element.
- Attributes which may be used with the <g> element are:
  - `id attribute - gives the group a unique name`
  - `<title>`
  - `<desc>`
- Both the <title> and <desc> attributes allow the group to be identified for text-based XML applications to aid in accessibility for visually-impaired users.
- Any style you specify in the starting <g> tag will apply to all the child elements in the group (cutting down duplication).

Grouping and Referencing Objects…

- An example of the <g> element:

```
<g id="house" style="fill: none; stroke: black;">
   <desc> House with door </desc>
   <rect x="6" y="50" width="60" height="60"/>
   <polyline points="6 50, 36 9, 66 50"/>
   <polyline points="36 110, 36 80, 50 80, 50 110"/>
</g>
```

- The <use> element allows you to display objects that you have defined in a group.
- An example:

```
<use xlink:href="#house" x="80" y="80"/>
```

Grouping and Referencing Objects…

- With the <group> and <use> elements there are a few drawbacks:
  - You need to know the positions of the original objects when deciding where to place the copies.
  - The fill and stroke colour were defined by the original and you can't override in using <use>.
  - All objects are drawn on the screen (not stored until you want to see them).
- The <defs> element is used around the group objects, and this means that nothing is drawn to the screen.
- The user can then define where he/she wants the objects to appear on screen.

Grouping and Referencing Objects…

- The SVG specification recommends that you use <def> in all groups that you wish to re-use.
- The <symbol> element is another way of grouping elements. Unlike <g>, a <symbol> element is never displayed, so you do not have to enclose it in <defs>. However, it is customary to do so, since a <symbol> really is something you are defining for later use.
- The <image> element can include an entire SVG or raster file in the SVG document. Currently JPEG or PNG raster files can be used.
- An example of an <image> element:

```
<image xlink:href="kwanghwamun.jpg"
       x="72" y="922" width="160" height="120"/>
```

Transforming the Coordinate System - Translate

- A translate transformation can be done using the <use> element and stating the x and y attributes:

```
<use xlink:href="#square" x="50" y="50"/>
```

- Alternatively, the transform attribute can be used as follows:

```
<use xlink:href="#square" transform="translate(50,50)"/>
```

- The difference between specifying the x and y attributes and using the transform attribute translate is that translate picks up the entire grid and moves it to the new location (50,50).

Transforming the Coordinate System – Translate…

- As far as the square is concerned, it is still being drawn at (0,0).
- Definition: *A translation transformation never changes a graphic objects grid coordinates; rather, it changes the position of the grid on the canvas.*

Transforming the Coordinate System - Scaling

- Scaling can be done uniformly or non-uniformly - an example of both is:
  - **Uniform scale**
    - *transform="scale(value)"*
    - *e.g.,*
      - `<use xlink:href="#square" transform="scale(2)"/>`
  - **Non-Uniform scale**
    - *transform="scale(x-value,y-value)"*
    - *e.g.,*
      - `<use xlink:href="#square" transform="scale(3,1.5)"/>`

Transforming the Coordinate System – Scaling…

- Definition: *A scaling transformation never changes a graphic objects grid coordinates or its stroke width; rather, it changes the size of the coordinate system (grid) with respect to the canvas.*
- Scaling around a point rather than the origin is not possible. However, it is possible to do the following:

```
translate(-centerX*(factor-1),-centerY*(factor-1))scale(factor)
```

Transforming the Coordinate System - Rotate

- It is also possible to rotate the coordinate system by a specified angle.
- The centre of rotation is presumed to be (0,0).
- Note that the entire coordinate system can be rotated.
- An example:

```
<rect x=”70” y=”30” width=”20” height=”20” transform=”rotate(45)”
style=”fill: black;” />
```

Transforming the Coordinate System – Rotate…

- To rotate a single object rather than the whole coordinate system use:

```
translate (centerX, centerY)
rotate(angle)
translate(-centerX,-centerY)
```

- Therefore, SVG provides another version of rotate:

```
rotate(angle, centerX, centerY)
```

- This has the effect of creating a temporary coordinate system with the origin at the specified x and y points, doing the rotation, and then re-establishing the original coordinates.

Transforming the Coordinate System - Skew

- Two other transformations allow you to skew one of the axes. The general form of which is:
  - skewX(angle)
  - skewY(angle)
- SkewX “pushes” all the x-coordinates by the specified angle, leaving the y-coordinates unchanged.
- SkewY skews the y-coordinates, leaving the x-coordinates unchanged.

Transforming the Coordinate System – How Transformations are Applied

- Note that SVG applies transformations to the coordinate system before it evaluates any of the shape's coordinates. Therefore, the effect of applying a transformation to a shape is the same as if the shape were enclosed in a transformed group.
- It is possible to do more than one transformation on a graphics object. e.g.,

```
<rect x="10" y="10" height="20" width="15"
    transform="translate(30,20) scale(2)"
    style="fill:gray;" />
```

Transforming the Coordinate System – How Transformations are Applied…

- The order in which you do a sequence of transformation affects the results. In general transformation A followed by transformation B will not give the same result as transformation B followed by transformation A.

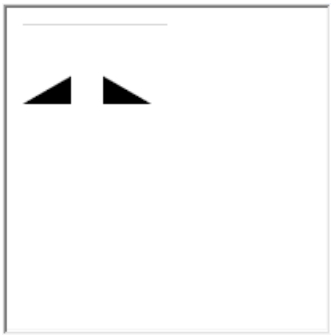
Paths

- All of the basic shapes previously described are really shorthand forms for the more general <path> element.
- It is advised by SVG to use the shortforms.
- The path element is more general; it draws the outline of any arbitrary shape by specifying a series of connected lines, arcs and curves.
- Additionally, these paths (as well as the shorthand basic shapes) may be used to define the outline of a clipping area or a transparency mask.
- The <path> element contains the d attribute which describes the following:
  - data attribute
  - describes the outline
  - consists of one letter commands such as M for moveto, l for lineto, followed by the coordinate information

Paths…

- Moveto
  - Every path must begin with a moveto command.
  - The command letter is a capital M followed by an x- and y-coordinate, separated by commas or whitespace.
  - This command sets the current location of the “pen” that’s drawing the outline.
- Lineto
  - Lineto is represented by a capital L, followed again by x- and y-coordinates, again separated by commas or whitespace.
  - Lineto draws a line to the point specified.
- An example:

```
<g styles="stroke: black; fill: none;">
    <path d="M 10 10 L 100 10"/>
    <path d="M 40 60, L 10 60, L 40 42.68,
             M 60 60, L 90 60, L 60 42.68“ />
</g>
```



Paths…

- A path takes up less memory when read into a document object model structure by an XML parser.
- If you want to use a <path> to draw a rectangle, you can do it in two ways:
  1. Draw all 4 sides
  2. Draw 3 lines and use the closepath command
- Closepath
  - Closepath is denoted by a capital Z.
  - Its function is to draw a line back to the beginning point of the current subpath e.g.,

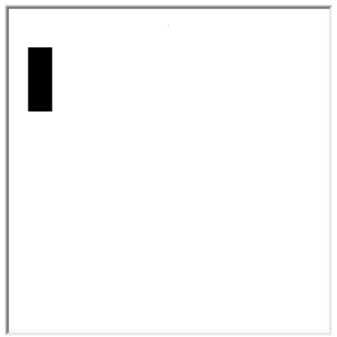
```
<path d="M 60 10, L 90 10, L 90 30, L 60 30, Z"/>
```

Paths…

- Relative moveto and lineto
  - If you use lowercase commands i.e., l and m, the coordinates are interpreted as being relative to the current pen position.
  - However, if you start a path with a lowercase m, it is considered the absolute as no previous pen exists.
  - Note that:
    - *All uppercase commands are absolute.*
    - *All lowercase commands are relative.*
  - However, this is different for closepath, whether Z or z is used, the effect is always the same.

Paths…

- Path Shortcuts
  - Horizontal line:
    - *H - followed by an absolute x-coordinate*
    - *h - followed by a relative x-coordinate*
  - Vertical line:
    - *V - followed by an absolute y-coordinate*
    - *v - followed by a relative y-coordinate*
  - An example:



```
<path d="M 12 24 h 15 v 15 v 25 h -15 z"/>
```

  - The above example draws a rectangle 15 units in width and 25 units in height, with the upper left corner coordinates at (12,24).

Paths…

- Elliptical Arc
  - An arc command begins with:
    - *A (absolute)*
    - *a (relative)*
  - and is followed by seven parameters:
  - x- and y-radius of the ellipse in which the points lie
  - x-axis rotation of the ellipse
  - large-arc-flag - 0 if < 180 degrees, 1 if > 180 degrees
  - sweep-flag, 0 if arc is drawn in the -ive angle direction, otherwise 1
  - The ending x- and y-coordinates of the ending point

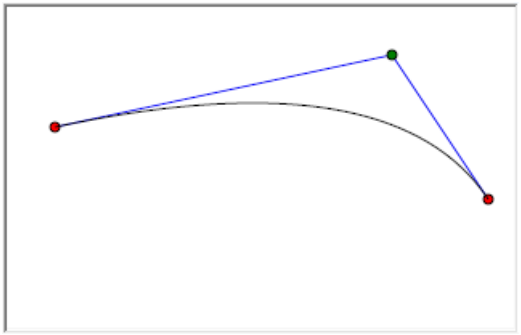
Paths…

- SVG uses what is known as *endpoint-and-sweep* to draw an ellipse. Other vector graphics allow you to draw an arc by defining a centre point, the x and y radius, and a starting angle - this is known as *center-and-angles*. A perl script is provided by SVG to convert center-and-angles format to endpoint-and-sweep. A second script exists to do the opposite.
- Bezier Curves
  - In SVG there is functions to draw:
    - *Quadratic Bezier Curves*
    - *Smooth Quadratic Bezier Curves*
    - *Cubic Bezier Curves*

Paths…

- Quadratic Bezier Curves
  - With quadratic bezier curves you specify:
    - *a beginning point*
    - *an ending points*
    - *a control point*
  - Q or q is used, followed by two sets of coordinates that specify a control point and an end point.
  - An example:

```
<path d="M30 75 Q 240 30, 300 120"
      style="stroke: black;
             fill: none;"/>
```



Paths…

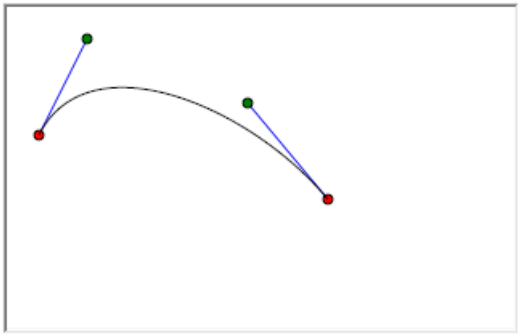
- Smooth Quadratic Bezier Curves
  - You may specify several sets of coordinates after a quadratic curve command to generate a polybezier curve.
  - To create a smooth quadratic curve:
    - *T or t is used*
    - *Followed by the next end point of the curve*
    - *The control point is calculated automatically*

Paths…

- Cubic bezier curves
  - C or c is used
  - Followed by three sets of coordinates that specify the control point for the start point, the control point for the end point and the end point
  - An example:

```
<path d="M20 80 C 50 20, 150 60, 200 120"
      style="stroke: black; fill: none;"/>
```

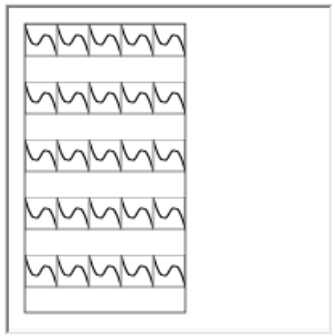
  - Again by specifying several sets of coords you can create a cubic polybezier.
  - If you want a smooth join between curves you can use S or s.



Patterns

- Patterns can be done via tiling e.g.,

```
<defs>
    <pattern id=”tile” x=”0” width=”20%” height=”20%”
             patternunits=”ObjectBoundingBox”>
        <path d=”M 0 0 Q 5 20 10 10 T 20 20”
              style=”stroke: black; fill: none;”/>
        <path d=”M 0 0 h 20 v 20 h -20 z ”
              style=”stroke: gray; fill: none;”/>
    </pattern>
</defs>
<rect x="10" y="10"
      height="180" width="100“
      style="stroke: black;
             fill:url(#tile);" />
```

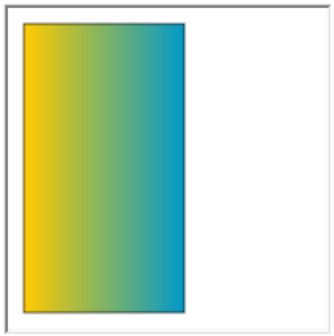


- You can also nest patterns.

Gradients

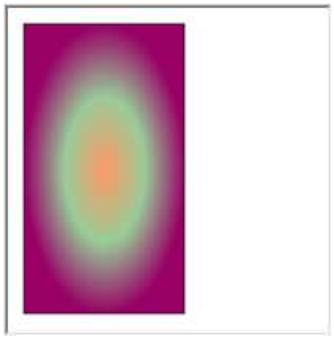
- A smooth colour transition fill from one shade to another.
- Gradients can be:
  - linear (where the colour transition occurs along a straight line)
  - radial (where the transition occurs along a circular path)
- Example of linear gradient:

```
<linearGradient id="two-hues">
    <stop offset="0%" style="stop-color: #ffcc00;"/>
    <stop offset="100%" style="stop-color: #0099cc;"/>
</linearGradient>
```



- Example of radial gradient:

```
<radialGradient id="threestops">
    <stop offset="0%" style="stop-color: #f96;"/>
    <stop offset="50%" style="stop-color: #9c9;"/>
    <stop offset="100%" style="stop-color: #906;"/>
</radialGradient>
```



Text

- The <text> element is used to do text.
- But before we can go into the attributes of <text> we must understand some of the SVG specification:
  - Character – a byte or bytes with a numeric value according to the unicode standard e.g., g=103
  - Glyph – is the visible representation of a character or characters (font etc). A single character can be composed of multiple glyphs i.e., é=233
  - Font – a collection of glyphs representing a certain set of characters. They normally have the following characteristics in common:
    - *Baseline*
    - *Ascent*
    - *descent*

Text…

- The <text> element has the following main elements:
  - x, y – defines the point where the baseline of the first character of the elements content is place.
  - Default colour is black, with no outline.
  - An example:

```
<text x="20" y="30">A Simple text example</text>
```

  - font-family e.g., serif, sans-serif, monospace
  - font-size – baseline-to-baseline distance of glyphs. Defined as normal i.e., 18pt
  - font-weight i.e., bold or normal
  - font-style i.e., italic and normal
  - text-decoration i.e., none, underline, overline, line-through
  - word-spacing – positive to increase, negative to tighten up
  - letter-spacing – same as word-spacing

Text...

- An example:

```
<text x="20" y="60" style="text-decoration: underline;">A
Simple text example with underlining</text>
```

- Text Alignment
  - The <text> element lets you specify the starting point, but you don't know its ending point.
  - This would make it difficult to centre, or right-align text.
  - In order to do this the text-anchor property may be used. You set it to a value of left, center, and right alignment e.g.,

```
<text x="100" y="30" style="text-anchor: start">Start
here</text>
```

Text...

- Tspan Element
  - This is analogous to the XHTML element.
  - <tspan> remembers the text position and can be used to change presentation properties such as font, size and colour e.g., if you want to user superscripts or subscripts you can use the dy attribute of <tspan> to offset the characters vertically (negative values are allowed). dx can be used to offset text horizontally. An example:

```
<tspan dy="8">hello</tspan>
```

  - You should always use <tspan>s within a <text> element to group related lines as it adds structure to your document.
  - However, for superscript and subscript it is better to use baseline-shift.

Text…

- Setting text length
  - Spacing and glyph size can be adjusted to change the text length.
  - To change space only only set lengthAdjust to spacing.
  - To change both space and glyph set lengthAdjust to spacingAndGlyphs.

Text...

- Vertical text
  - This can be done in two ways:
    - *Use a transformation to rotate the text 90 degrees*
    - *Change the value of the writing-mode style property to the value tb (meaning top-to-bottom)*
  - Going by two you sometimes want the letters to appear in a vertical column with no rotation. This is done by setting the glyph-orientation-vertical property with a value of zero. glyph-orientation-vertical's default value is 90 (90 degrees) e.g.,

```
<text x="90" y="30"
      style="writing-mode:tb;
      glyph-orientation-vertical: 0;">A simple
vertical text example</text>
```

Internationalization and Text

- Unicode and Bidirectionality
  - XML is based on the Unicode standard (http://www.unicode.org). This lets text display in any language that the underlying viewing software can display.
  - With Hebrew and Arabic the text is written left to right in which case in XML you have to used bidi-override e.g.,

```
<tspan style="direction: rtl; unicode-bidi: bidi-override; font-weight: bold;"> right-to-left </tspan>
```

  - This gives the effect:

tfel-ot-thgir

Internationalization and Text…

- The Switch Element
  - A switch element allows people viewing a document (which could be written in multiple languages) with Spanish system software to view the document in Spanish text, and Russians to see the document in Russian text.
  - The switch element searches through all its children until it finds one whose system language attribute has a value that matches the language the user has chosen in the viewer software's preferences. All other children will be bypassed e.g.,

```
<switch>
    <g systemLanguage="en-UK">
        <text x="20" y="30">A circle</text>
        <text x="10" y="100">without colour</text>
    </g>
</switch>
```

Clipping

- The viewport automatically becomes your clipping area; anything drawn outside those limits will not be displayed.
- <clipPath> element is used e.g., if you want to clip a rectangle i.e., a part of the cat:

```
<defs>
    <clipPath id="rectClip">
    <rect id = "rect1" x="15" y="15" width="40" height="45"
          style="stroke: gray; fill: none; "/>
</clipPath>
</def>
<!-- clip to rectangle -->
<use xlink:href="minicat.svg#cat"
     style="clip-path: url(#rectClip);"/>
<g transform="translate(100,0)">
<use xlink:href="#rect1"/>
<use xlink:href="minicat.svg#cat"/>
```

- The clipPath element can contain any number of basic shapes.

Masking

- The <mask> element is used.
- You can specify a mask's dimension i.e., x, y, width and height.
- These dimensions are in terms of the masked objectBoundingBox (maskContentUnits=objectBoundingBox). If you want the dimensions to be in terms of user space coordinates, set MaskUnits to userSpaceOnUse (which is default).
- Any basic shapes can go between <mask> and </mask>. The coordinates on these elements are expressed in user coordinate space by default.
- An example:

```
<defs>
    <mask id = "redmask" x="0" y="0" width="1" height="1“
          maskContentUnits="objectBoundingBox">
    <rect x="0" y="0" width="1" height="1“
          style="fill:#0f0;"/>
</mask>
```

Filters

- When a filter is applied the graphic is not rendered directly. Instead SVG will render the graphics pixels into a temporary bitmap. The operations specified by the filter will be applied to the temporary area and the result will be rendered into the final graphic.
- SVG has a number of filters – some of which are listed below:
  - feGaussianBlur
  - fcColorMatrix – allows you to modify any colour or alpha values of a pixel
  - Gamma
  - feBlend – allows you to combine images
  - feDiffuseLighting
  - feSpecularLighting
  - fePointLight
  - feSpotLight
  - feTurbulence – allows you to create artificial textures

Animating

- The animation features of SVG are based on the World Wide Web Consortiums Synchronized Multimedia Integration Language Level 2 (SMIL2):

  http://www.w3.org/TR/smil20

- In this system you specify the starting and ending values of the attribute, colour, motion, or transformation that you wish to animate; the time at which the animation should being; and the duration of the animation.

Animation Example – Shrinking Rectangle

```
<?xml version="1.0"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN“
      "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="240" height="170">
    <title>Animate1</title>
    <desc>Animate Rectangle</desc>
    <rect x="10" y="10" width="200" height="20"
          stroke="black" fill="none">
        <animate
            attributeName="width“
            attributeType="XML“
            from="200" to="20“
            begin="0s" dur="5s“
            fill="freeze" />
    </rect>
</svg>
```

Animation…

- The animation is done within the <rect> element.
- Within the <animate> element attributeName specifies the variable which should be animated (or changed over time).
- Width is an XML attribute
- The beginning and end times as well as positions are given for the animation.
- The above example does the following:

Scripting

- The next step from linking is scripting.
- A program can be written in ECMA script (the European Computer Manufacturers Association standard version of what is commonly called JavaScript) to interact with an SVG graphic.
- Interaction occurs when a graphic object responds to events.
- Objects can respond to mouse events:
  - Mouse button click:
    - *Mousedown*
    - *Mouseup*
  - Moving the mouse:
    - *mouseover (the mouse pointer is within the object)*
    - *mouseout (the mouse pointer leaves object)*
    - *mousemove*

Scripting…

- Events can also be associated with an object's status:
  - load (object is fully parsed and is ready to render)
  - non-standardized events
    - *pressing keys – keydown and keyup*
- We will be using attributes to specify the functions which should handle the events. These event handler attributes begin with the word on. Thus, *onclick* is an attribute whose value specifies a function that handles a click event.
- To allow an object to respond to an event, you add an event Name attribute to the element in question.

Scripting…

- The value of the attribute will be an ECMA script statement, usually a function call. This function usually takes the reserved word evt as one of its parameters. evt had properties and methods that describe the event that has occurred. The three methods you will use most often are:
  - getTarget() – which returns a reference to the graphic object that is responding to this event
  - getClientX() – returns x-coordinate of the mouse when the event occurred
  - getClientY() – returns y-coordinate of the mouse when the event occurred

Scripting…

- A value of (0,0) indicates the upper left corner of the SVG viewport. These functions return the position in the viewer window regardless of any zoom or pan that the user may havedone.

Scripting and Animation

- Scripting and Animation can be used together, with the animation being triggered by an event.

Generating SVG

- SVG can be generated via many means such as:
  - A graphic tool which outputs SVG
  - A convert to SVG program written in Java, C++ etc
- But basically you are loosing absolute control over the SVG code by automatically generating it. Like all automatic conversion processes there is the slight draw back such as the SVG code is not created as per the standard.

Conclusion

- SVG has got a promising future. It can be compared to Macromedia Flash, but it is being predicted that it will outgrow Flash due to the fact that it is XML based, and XML is the future (look at the next stage in the VRML lifecyle – X3D).

Further Information

- Eisenbert, David J., SVG Essentials – Producing Scalable Vector Graphics with XML, O'Reilly.
- Laaker, Micah, Teach Yourself SVG in 24 Hours, SAMS.
- Duce, D. A., Herman, I., Hopgood, B., SVG Tutorial.

MANCHESTER 1824

The University of Manchester

Research Computing Services
University of Manchester

Contact Details

www.rcs.manchester.ac.uk
mary.mcderby@manchester.ac.uk
tobias.schiebeck@manchester.ac.uk