Web 2D Graphics: State-of-the-Art

© David Duce, Ivan Herman, Bob Hopgood 2001

Contents

- 1. Introduction
  - 1.1 Images on the Web
  - 1.2 Supported Image Formats
  - 1.3 Images are not Computer Graphics
- 2. Early Vector Graphics on the Web
  - 2.1 CGM
  - 2.2 CGM on the Web
  - 2.3 WebCGM Profile
  - 2.4 WebCGM Viewers
- 3. SVG: an Introduction
  - 3.1 Arrival of XML
  - 3.2 Submissions to W3C
  - 3.3 SVG: an XML Application
  - 3.4 An introduction to SVG
  - 3.5 Coordinate Systems
  - 3.6 Path Expressions
  - 3.7 Other Drawing Elements
- 4. Rendering the SVG Drawing
  - 4.1 Visual Aspects
  - 4.2 Text
  - 4.3 Styling
- 5. Filter Effects
- 6. Animation
  - 6.1 Introduction
  - 6.2 What is Animated
  - 6.3 How the Animation Takes Place
  - 6.4 When the Animation Take Place
- 7. Scripting and the DOM
- 8. Current State and the Future
  - 8.1 Implementations
  - 8.2 Metadata
  - 8.3 Extensions to SVG
- A. Filter Primitives in SVG
- References

© David Duce, Ivan Herman, Bob Hopgood 2001

1. Introduction

- 1.1 Images on the Web
- 1.2 Supported Image Formats
- 1.3 Images are not Computer Graphics

1.1 Images on the Web

The early browsers for the Web were predominantly aimed at retrieval of textual information. Tim Berners-Lee's original browser for the NeXT computer did allow images to be viewed but they popped up in a separate window and were not an integral part of the Web page. In January 1993, the Mosaic browser was released by NCSA. The browser was simple to download and, by the Autumn of 1993, was available for X workstations, PCs and the Mac. From 50 Web servers at the start of 1993, Web traffic had risen to 1% of internet traffic by October and 2.5% by the end of the year. About a million downloads of the Mosaic browser took place that year. In February of 1993, Mark Andreessen proposed the <IMG> element as an extension to Mosaic's HTML to provide a way of adding images to Web pages. In 1994, Dave Raggett developed an X-browser that allowed text to flow around images and tables and from then on images were an accepted part of the Web page. Web pages became glossier and the enormous growth of the Web started [1] [2]. Organisations could customise their home pages with the company logo. Maps, albeit images, could be added to show how to reach the organisation. Its products could be displayed on the Web. Eventually, the Web would become a major commercial outlet.

1.2 Supported Image Formats

The only image format supported by all the early browsers was the GIF format developed by CompuServe. The original GIF format only supported 256 colours. By 1995, the possibility of JPEG also being supported was growing and the lossy compression available with JPEG meant that real world images could be half or a third of the size of the same image stored in GIF format. Also, JPEG was a full 24-bit format allowing the possibility of 16 million colours [3].

The ability to add images of various types (maps, drawings, photographs etc) to Web pages enhanced the capabilities and made them more exciting. The downside was that the inclusion of images slowed the download time of the Web page by an order of magnitude. Browsers provided the option of turning off the images thus negating their use for core information. Browsers opened multiple channels to improve the download speed but, at the same time, congested the Internet for others even more.

In December 1994, CompuServe and Unisys announced that developers would need to pay a license fee to use the GIF format as the technique used to compress the image data, called LZW (after Lempel-Ziv-Welch), was patented by Unisys. In consequence (although in the end license fees were not charged to end users), a new image format, PNG (Portable Network Graphics) was developed that does not have the patent problems associated with GIF. It provides an efficient lossless format for greyscale, true colour and palette-based images [4].

1.3 Images are not Computer Graphics

But images are not computer graphics. The Oxford Dictionary of Computing has the following definition of computer graphics: **the creation, manipulation of, analysis of, and interaction with pictorial representations of objects and data using computers**. A digital image on the other hand is usually a 2-dimensional regular grid of pixels. The ability to interact with it is limited. Being just an array of pixels, most of the information that existed in the original object is lost. All that remains is what the eye can see. Figure 1.1 shows the difference between zooming in on the plane if the original is an image compared with what is seen if the drawing is defined as lines and areas.

The aim of this paper is to look at 2-dimensional computer graphics on the Web and to give some insight into why the Web has come so far without computer graphics being an integral part (given the importance of computer graphics in many applications).

© David Duce, Ivan Herman, Bob Hopgood 2001

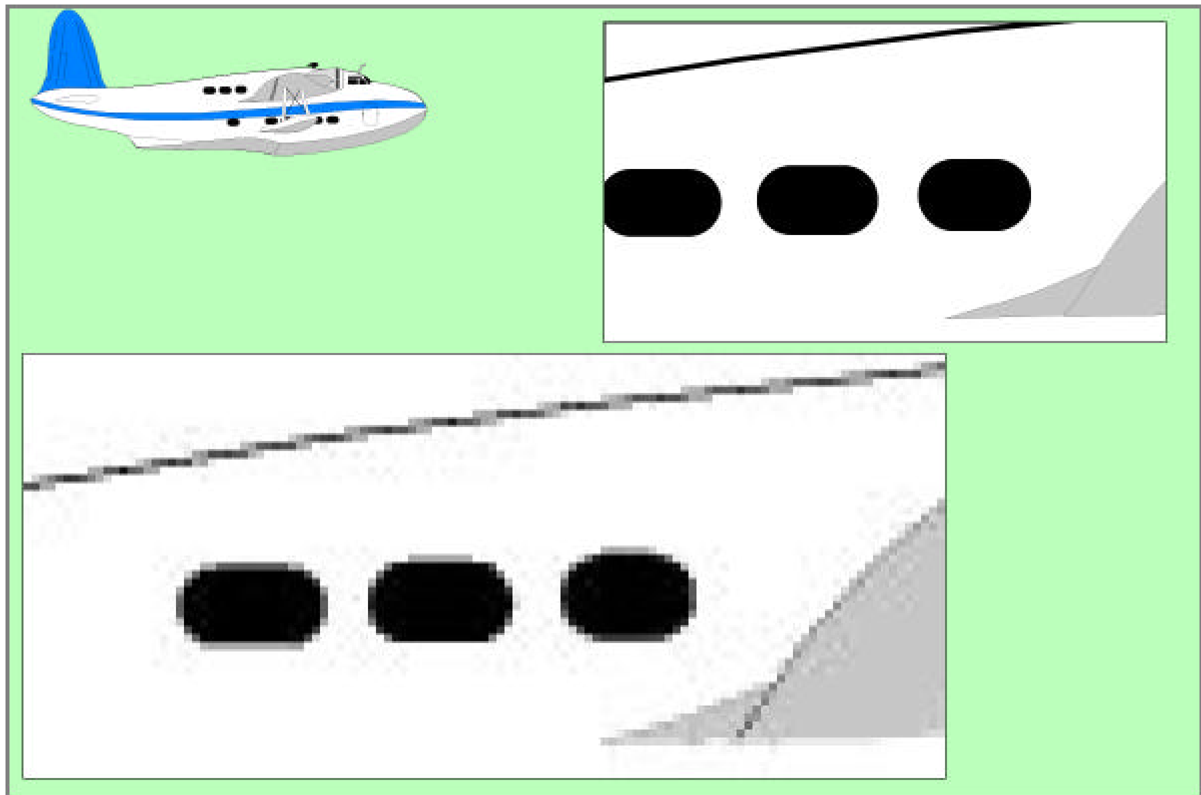

Figure 1.1: Images versus Vector Graphics

The image formats all share many disadvantages that are serious obstacles to the development and adoption of new technologies on the Web. Some of the major problems are listed below.

Bandwidth

Images are large. Improvements in network bandwidth have helped to hide this. Also image compression techniques have improved. Even so, images are a major bottleneck to accessing Web sites. This creates significant problems when designers want to follow their own style in creating new Web pages.

Flexibility

Images inherently have a fixed resolution. In consequence, an application destined to run on a range of PCs, PDAs and mobile phones is unable to adapt to the constraints of the device. Colour, resolution, aspect ratio and bandwidth often differ significantly between devices.

Hyperlinking

Hyperlinking is a fundamental requirement on the Web. However, to link to different places, dependent on where the user clicks on an image, is not simple. Early on, image maps were added to HTML. This allowed the coordinates of where the user clicked to be returned to the server where a program was run to determine which page to link to. Server side image maps are not efficient adding another round trip from client to server. The map is separate from the HTML page and is dependent on the server for translation. Different servers used different map file formats so that pages often could only be read by certain browsers. Client side image maps were added in HTML 3.0 and these allowed rectangular, elliptical or polygonal areas to be defined. Clicking on an area causes the link defined for that area to be taken. Creating image maps is cumbersome and is not related to the real objects being viewed but their image on the display.

Animation and Interaction

Many applications profit from the use of animation and interaction (cartography, CAD, remote teaching, etc). Image formats only provide crude animation limited to the sequential playback of a sequence of images combined into a single file. Interaction is limited to the use of image maps.

© David Duce, Ivan Herman, Bob Hopgood 2001

Separation of Style from Content

The same drawing in terms of meaning can be represented in many different ways dependent on the capabilities of the device. Dotted line on a mono display might be rendered as a different colour on a colour display. Images do not have the ability to make such changes.

Integration

In the early days of the Web, an HTML page was transmitted across the Internet using the HTTP protocol and there was a 1-1 relationship between documents and downloads. Today, the Web is much more complex. Separating style and content meant that a style sheet might be transmitted as well as the Web page. The move to XML [10] allows appropriate markup for different information in the Web page. No longer is it necessary to force the HTML elements defined for textual documents to be used for other purposes. Mathematical markup [20], multimedia [19], and chemical markup, for example, each use their own XML application. Any computer graphics on the Web should be integrated with this model of the Web.

The rest of this paper will concentrate on the way 2D computer graphics is being made a more integral part of the Web.

2. Early Vector Graphics on the Web

- 2.1 CGM
- 2.2 CGM on the Web
- 2.3 WebCGM Profile
- 2.4 WebCGM Viewers

2.1 CGM

The ISO Computer Graphics Metafile (CGM) Standard [5] is a format for describing vector graphic pictures compactly. It has proven to be a very good format for a whole range of demanding 2-dimensional graphics presentation applications [6]. CGM first became an ISO standard in 1987 and has been enhanced over the years by enriching the drawing primitive set and providing more structural information. As it became richer, the concept of CGM Profiles for specific application sectors evolved.

In 1997, an analysis was done by W3C to see if it would be possible to define a CGM Web Profile that could satisfy the requirements for computer graphics on the Web. It passed most of the necessary criteria. CGM was an open specification that had been widely implemented. CGM separated abstract syntax from the concrete representation allowing multiple encodings to be defined. The vector drawing facilities were more that what was required. The HTML <OBJECT> element could be used to add CGM diagrams to a Web page. However, styling in CGM was provided by the bundle table approach also used in the ISO standards, PHIGS and GKS. This was a different approach to the one adopted in Cascading Style Sheets on the Web to separate style and content. A major drawback was that linkage between drawings in the manner of the Web was not provided.

2.2 CGM on the Web

The CGM community saw major advantages in using CGM rather than images on the Web:

- Vector graphics can be zoomed in and out while retaining the quality of the picture, unlike images.
- Vector graphics files are smaller and can be downloaded and viewed faster than images.
- Vector graphics can be interacted with in a meaningful way.
- Text in a CGM vector graphics drawing can be searched as easily as text in an HTML page.

© David Duce, Ivan Herman, Bob Hopgood 2001

In consequence, CGM suppliers provided CGM plug-ins to access CGM vector graphics on the Web using the existing encodings. A CGM MIME type was agreed in 1995. The only problem was that the CGMs produced by one vendor could not be read by viewers produced by another as different Profiles were implemented and the hyperlinking mechanisms introduced differed from one supplier to another.

A joint activity between the World Wide Web Consortium (W3C) and the CGM Open Consortium [24] (launched in May 1998) was initiated to define a common Web Profile for CGM that would be accepted both by ISO and W3C. This resulted in the WebCGM Profile, completed in January 1999 [7].

2.3 WebCGM Profile

WebCGM was based on the ATA CGM Profile for graphics interchange (GREXCHANGE). The Graphics Working Group (ATA 2100) of ATA, the Air Transport Association, had defined this CGM Profile for the aerospace industry. It was also working towards an intelligent graphics exchange profile (IGEXCHANGE) that associated semantic information to aid query, searching and navigation.

The structure of a WebCGM file is shown in Figure 2.1. Picture size and scaling and properties such as line width and background colour are defined in the Picture Descriptor. This is equivalent to styling provided for the <body> element in an HTML page.

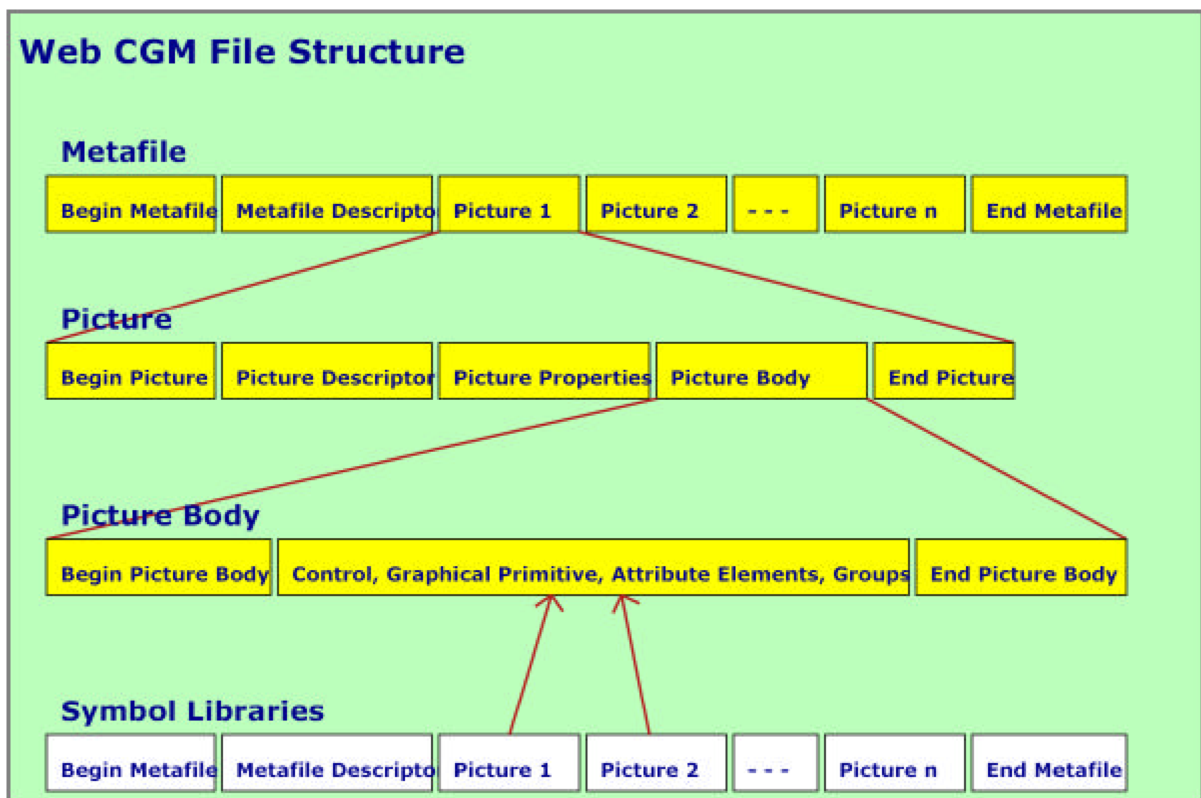

Figure 2.1: CGM Architecture

Each picture contains CGM graphic elements. There are 4 groupings of graphical elements provided:

- **grobject**: a graphical object with a unique id and possibly linkURI and tooltip attributes. It is used to identify sources and destinations of hyperlinks. It is also possible to define the region of the group for picking and the initial view when the group is linked to. For example, more than the group may need to be visible to indicate the context.
- **layer**: this has a name and a list of objects. It allows a picture to be divided into a set of graphical layers that can be used to switch display to parts of an illustration.
- **para**: defines a paragraph as the grouping of several text drawing elements. The elements may be scattered across the drawing but for searching purposes are similar to an HTML paragraph.

© David Duce, Ivan Herman, Bob Hopgood 2001

- **sub-para**: a sub-paragraph used to identify fragments of text (for example as hotspots within a paragraph).

These provide the basis for searching and linking within and between CGM pictures. An object may be the target of a link. Browsers are expected to move the object into view and scale it to fit into the viewport. If the object has a **ViewContext** attribute the rectangle defining the view context must be within the viewport.

Links from WebCGM objects are defined by **linkURI** elements that are modelled on the XLink facilities [8]. Objects may have multiple links. Links can be bi-directional. Linkage can be from places outside the CGM and links from the CGM can be to any destination defined by a URL. Following a link can display the new picture in a separate window, load the picture into the current frame, load it over the parent of the current frame or replace the current picture.

WebCGM is a reasonably full profile of CGM containing a rich set of graphics elements:

- Polylines, disjoint polylines, polygons, polygon sets.
- Rectangles, circles, ellipses, circular and elliptical arcs, pie slices.
- Text: both the Restricted Text primitive of CGM (which defines its extent box) and the Append Text element (continuation of a text string with a change of attributes).
- Closed Figure and Compound Line: allows complex paths to be defined as a sequence of other primitives.
- Polysymbol: placement of a sequence of symbols defined in the Symbol Library (another valid WebCGM metafile).
- Smooth curves: the smooth piece-wise cubic Bezier defined by CGM's Polybezier element.
- Cell Array and Tile Array allow PNG, and JPEG images to be integrated with the vector drawing.

Most of the line and fill attributes of CGM are included but only as INDIVIDUAL attributes. The bundled attribute functionality of CGM is omitted. Thus, WebCGM diagrams consider properties such as linestyle, color, fill types etc as content rather than styling.

The full set of CGM colour models is provided including sRGB and sRGB-alpha. International text is defined by selecting either Unicode UTF-8 or UTF-16.

2.4 WebCGM Viewers

Probably the most widely used Viewer is the Micrografx free ActiveCGM plug-in. SDI has also released a CGM Plug-in while Tech Illustrator has a TI/WebCGM Hotspot Plug-in module to author hotspots for exporting to CGMs. There is good industrial support for WebCGM and it is widely used in the CAD and aerospace industries. A major interoperabilty demonstration took place at XML Open in Granada in May 1999.

A good source of further information on CGM is CGM Open [24], an organisation dedicated to open and interoperable standards for the exchange of graphical information. The W3C Web site is also a valuable source of news and reference information.

© David Duce, Ivan Herman, Bob Hopgood 2001

3. SVG: an Introduction

- 3.1 Arrival of XML
- 3.2 Submissions to W3C
- 3.3 SVG: an XML Application
- 3.4 An introduction to SVG
- 3.5 Coordinate Systems
- 3.6 Path Expressions
- 3.7 Other Drawing Elements

3.1 Arrival of XML

By 1997, it was becoming clear that the Extensible Markup Language (XML) [10] would make a profound difference to the way that the Web would develop. The first edition became a W3C recommendation in February 1998 but by then its impact was already being felt. XML is a meta-language for defining markup languages. An XML application is a document format for storing structured information in an unambiguous and appropriate manner.

An XML application consists of a set of elements, rather like HTML, and the elements can have attributes associated with them. For example:

```
<text size="20" font="Palatino">Abracadabra</text>
```

The **text** element has a start and end tag written as **<text>** and **</text>** and the content of the element is the string Abracadabra. The text element has two attributes, **size** and **font**, that might define the height of the text and the font to use. These are defined as part of the start tag. Being an XML application, several rules have to be obeyed:

- There can only be one outer element that encloses the complete document, the root element.
- Every start tag must have a correctly nested end tag.
- The form of the start and end tag must be identical. If the start tag is upper case so must be the end tag.
- Attributes must be enclosed in quotes (either single or double).
- Elements that have no content can be written <text /> which is equivalent to <text></text>.

These rules are sufficient to define the structure of an XML well formed document.

3.2 Submissions to W3C

The question now arose: could XML markup be used to express vector graphics? In 1998, there were four Submissions to W3C proposing an XML-based vector graphics markup language for the Web. In order, these were:

- **Web Schematics** [15]: similar to the troff pic language, it defined objects with anchor points that could be composed into pictures.
- **Precision Graphics Markup Language (PGML)** [16]: a lower level language that could be described as an XML-based version of PostScript.
- **Vector Markup Language (VML)** [17]: just as PGML has a relationship to Postscript, VML had a similar relationship to PowerPoint.
- **DrawML** [18]: a constraint-based higher level language that allowed the drawing to adjust to the content. Changing the text in a box would increase the size of the box and adjust the box surrounding that box etc.

In retrospect, it is surprising that the CGM community did not put forward a proposal for a CGM Profile defined using XML notation.

© David Duce, Ivan Herman, Bob Hopgood 2001

These submissions resulted in a Working Group being formed to define a single language for vector drawings on the Web called Scalable Vector Graphics (SVG) [14]. This has reached Candidate Recommendation status within W3C and should become a full Recommendation in 2001. The Candidate Recommendation stage within the W3C process is to allow trial implementations to test the quality of the specification. The strong interest in SVG meant that there were implementations of the Candidate Recommendation early in 2001 even though the Candidate Recommendation was not issued until November 2000.

Of the four Submissions, PGML had the most impact on the functionality of SVG. Even so, SVG has evolved into a standard significantly different from all of the initial Submissions.

3.3 SVG: an Application of XML

SVG is an application of XML. This has the benefit that the overall syntactic structure of SVG is known and parsers exist to handle it. It also means that SVG can benefit from the other activities within W3C concerned with the XML Family of standards. In many cases, this is a strong plus but occasionally the constraints imposed by the other standards will mean that the functionality provided within SVG may be less elegant or have different characteristics from the form it would have taken if it had not been part of the XML Family. However the advantages far outweigh the disadvantages. Some examples of the influences on SVG are:

- **Cascading Style Sheets (CSS)** [9]: CSS is used to separate style from content initially in an HTML document. A CSS style sheet consists of a set of commands that specify the styling to be associated with a specific element. As CSS is not restricted to HTML elements, it can also be used to style an XML application.
- **Namespaces in XML** [12]: with many XML applications emerging, it is more likely that several will be used together in which case it is necessary to identify which elements belong to which application. XML achieves this by defining a prefix that identifies the namespace. For example, <svg:text> defines the start of the SVG text element. The prefix may be anything the user wants it to be as long as the appropriate namespace declaration identifies the application.
- **XML Linking Language (XLink)** [8]: as all XML applications are likely to require hyperlinking, a separate Recommendation, XLink, defines a flexible hyperlinking mechanism. Rather than define its own, SVG is able to use the XLink hyperlinking functionality via the XLink namespace.
- **XSL Transformations (XSLT)** [11]: XSLT defines a transformation functionality to be applied to XML documents. CSS effectively performs a single pass through an HTML document transforming the elements by defining their styling. XSLT provides similar functionality in terms of styling but also allows complex transformations of the XML documents. For SVG, higher-level functionality can be realised by defining an XSLT transformation down into SVG. In consequence, SVG need not define such functionality within the core version of SVG.
- **Synchronised Multimedia Integration Language (SMIL)** [19]: the SVG Working Group included animation functionality within its design objectives. SMIL was also considering similar functionality for multi-media presentations. The two Working Groups have, therefore, produced a single suite of animation functionality that can be used by both SMIL and SVG.
- **Document Object Model (DOM)** [21]: The DOM provides a standard method of interacting with an XML application. In consequence, SVG can use this functionality as the basis for interaction between a user and an SVG drawing.

© David Duce, Ivan Herman, Bob Hopgood 2001

3.4 An Introduction to SVG

Figure 3.1 shows the result that an SVG-enabled browser or viewer would make of the SVG document defined below.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20001102//EN"
"http://www.w3.org/TR/2000/CR-SVG-20001102/DTD/svg-20001102.dtd">
<svg width="300" height="220">
<rect width="300" height="220" fill="white" stroke="black" />
<g transform="translate(10 10)">
<g stroke="none" fill="lime">
<path d="M 0 112
L 20 124 L 40 129 L 60 126 L 80 120 L 100 111 L 120 104
L 140 101 L 164 106 L 170 103 L 173 80 L 178 60 L 185 39
L 200 30 L 220 30 L 240 40 L 260 61 L 280 69 L 290 68
L 288 77 L 272 85 L 250 85 L 230 85 L 215 88 L 211 95
L 215 110 L 228 120 L 241 130 L 251 149 L 252 164 L 242 181
L 221 189 L 200 191 L 180 193 L 160 192 L 140 190 L 120 190
L 100 188 L 80 182 L 61 179 L 42 171 L 30 159 L 13 140Z"/>
</g>
</g>
</svg>
```

The initial XML declaration and the Document Type Declaration can be omitted and, future examples will do this.

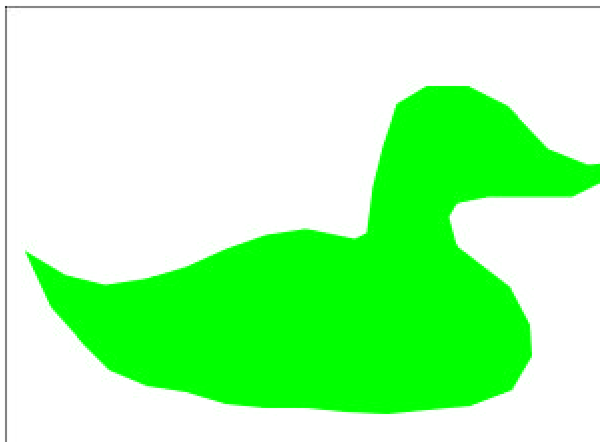

Figure 3.1: Simple SVG Drawing

This simple example reveals some of the basic characteristics of SVG:

SVG is an XML application

The root element is svg. All the elements are correctly nested. The attributes are enclosed in quotes and the **path** and **rect** elements do not have any content and so use the shorthand format.

© David Duce, Ivan Herman, Bob Hopgood 2001

SVG has a hierarchical structure

The g element in the example groups a set of elements. In this case there is just a single **path** element but normally there would be a sequence of drawing elements making up an object. Attributes can be defined on the g element that apply to the whole group. The hierarchical structure in SVG is similar to the scene graph approach used in systems like OpenInventor, PostScript and most graphics editors.

It is also possible to define special groups called symbols (similar to those in CGM) that can then be reused by a group through a <use> element:

```
<symbol>
<path id="duck" d="M 0.0...">
</symbol>
<g transform="translate(10 10)">
<g stroke="none" fill="lime">
<use xlink:href="#duck"/>
</g>
</g>
```

The reference to the symbol duck uses an XLink simple link to achieve the connection between the symbol resource and its use.

3.5 Coordinate Systems

All graphics elements are rendered conceptually on to an SVG infinite canvas. A viewport can be established that defines a finite rectangular region within the canvas. Rendering uses the painter's model; elements later in the document are rendered on top of the preceding ones.

In SVG, the y origin is at the top left corner of the viewport and y values increase down the canvas.

The viewport is specified by attributes of the **svg** element. Explicit width and height values can be given as in the example in 3.4 or by using the **viewBox** attribute. For example:

```
<svg viewBox="0 0 600 400">
```

The two approaches are subtly different. In the first case, no units have been specified so pixels are the assumed coordinate system. The viewport required is 600 pixels wide and 400 pixels high. The local user coordinate system for the duck is also set to be 600 by 400 with one pixel equal to one local user coordinate. In the second case, the local user coordinate is set to 600 wide and 400 high and this is to be mapped to fit in the viewport. A small viewport would have the mapping from user coordinate to pixels different from a large viewport. If the aspect ratio of the viewport is different from that of the viewBox then various options are provided as to how the user would like the mapping to take place.

In SVG, if no units are specified the assumption is that the coordinates are defined in the local coordinate system. However, in defining the viewport size and in specifying the drawing, the complete set of units defined in CSS are available to the user (em, ex, px, pt, pc, cm, mm, in, %).

If the drawing is to be displayed as part of a Web page, a complex negotiation then takes place between the SVG plug-in and the browser taking into account any constraints imposed by the user on inserting the drawing in the Web page or by the styling applied to the page as a whole. As a result of this negotiation, part of the image could be clipped, scaled or distorted, depending on how the constraints are resolved. The user can control the effect somewhat through a **preserveAspectRatio** attribute.

© David Duce, Ivan Herman, Bob Hopgood 2001

The viewport specification defines the initial user coordinate system. Transformations can be defined for each group or individual drawing element to effectively change the coordinate system being used. Scaling, rotation, and skew in both the x and y directions are provided.

The transform attribute can be a sequence of transformations:

```
<g transform="translate(10 10) scale(0.2) skewX(0.2)">- - -</g>
```

The transformations are applied from left-to-right. Transformations can be nested using groups; the previous example is equivalent to:

```
<g transform="translate(10 10)">
<g transform="scale(0.2)">
<g transform="skewX(0.2)">
- - -
</g>
</g>
</g>
```

Figure 3.2 shows the effect of transformations on a duck initially sitting in the top left corner.

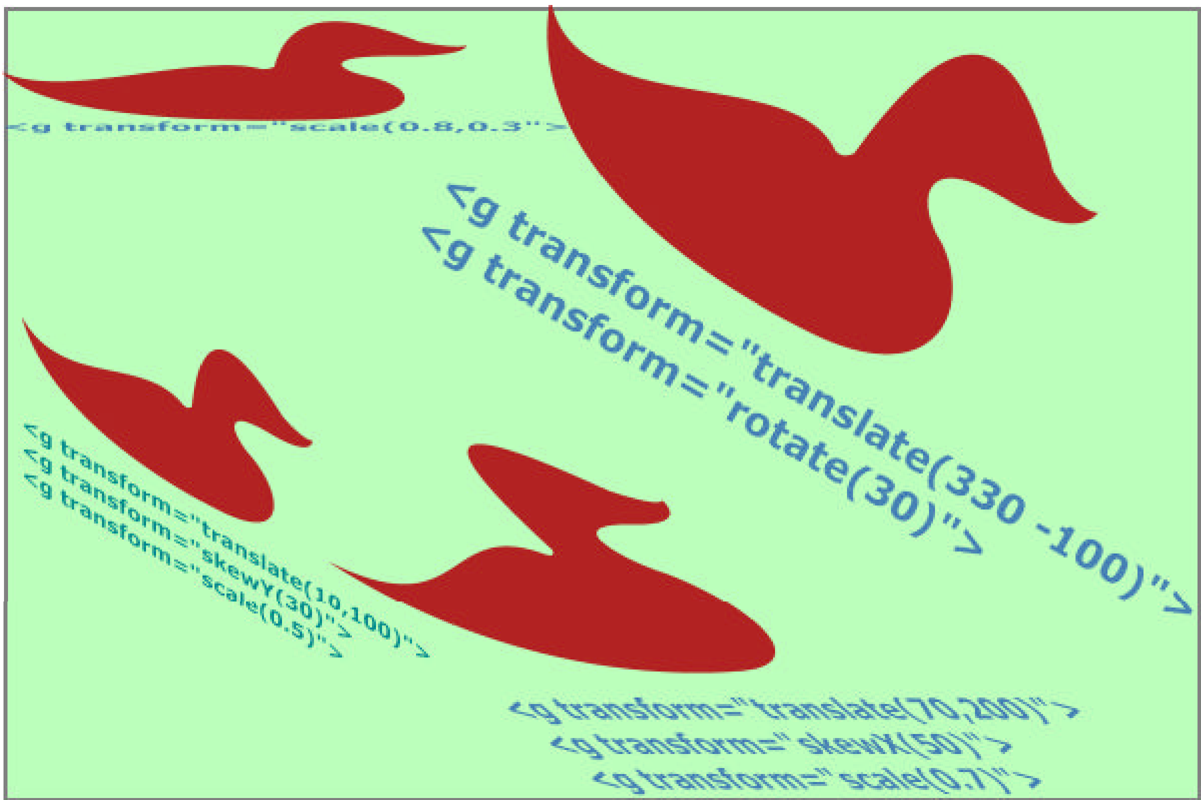

Figure 3.2: SVG transformations

3.6 Path Expressions

The **path** element is the main drawing element in SVG. Figure 3.1 defined the contour of a duck as a path with the **d** attribute defining the path expression.

A path expression is a sequence of commands consisting of an upper or lower-case letter followed by one or more numeric values. In the example, the first command is a letter M followed by a pair of coordinates defining a move to that coordinate position. This is followed by a sequence of L commands defining a straight line from the current position to the position specified in the L command. The final Z command closes the contour.

© David Duce, Ivan Herman, Bob Hopgood 2001

SVG is designed for a wide range of applications on the Web. The drawings may be complex and download times are important. In consequence, the conciseness of path expressions is of fundamental importance. It is for this reason that the path expressions themselves are not defined using a more verbose XML syntax. Every attempt is made to keep the number of characters in path expressions to the minimum. For this reason, paths are not restricted to polylines. Quadratic and cubic Bezier splines and elliptical arcs are also provided. The duck above could be defined as a set of cubic Bezier commands as:

```
<path d="M 0 312
C 40 360 120 280 160 306 C 160 306 165 310 170 303
C 180 200 220 220 260 261 C 260 261 280 273 290 268
C 288 280 272 285 250 285 C 195 283 210 310 230 320
C 260 340 265 385 200 391 C 150 395 30 395 0 312 Z"/>
```

Each Cubic Bezier command takes the starting position as the current position from the previous command and specifies the two control points and the end point. Figure 3.3 shows how the control points can define significantly different curves with the same start and end points.

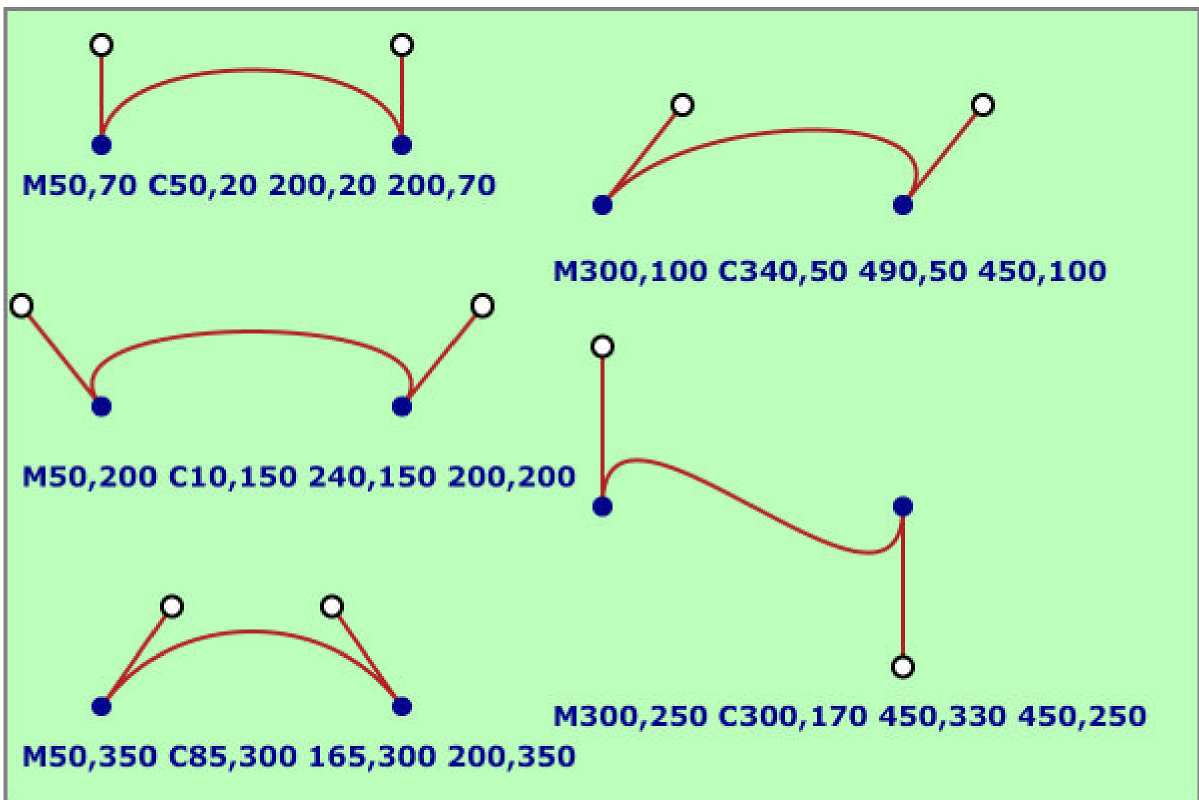

Figure 3.3: Cubic Bezier Curves

SVG includes a number of semantic and syntactic measures to reduce the size of path expressions even further:

Using relative coordinates

Using relative instead of absolute coordinates can reduce the number of characters per coordinate significantly. Each command has a lower case equivalent which defines the coordinate values as relative to the current position.

© David Duce, Ivan Herman, Bob Hopgood 2001

Smooth curves

Often adjacent curves need to be joined smoothly, that is by sharing a tangential direction at the joint. This is achieved by defining the first control point of the second curve as the reflection of the second control point of the first curve. By defining separate commands for smooth joining curves (T and S for the quadratic and cubic splines) a number of coordinate pairs can be omitted.

Syntactic simplifications

White space can be omitted when the result is unambiguous. For example, no space is required between the command letter and the coordinate value; negative coordinates do not need a separation from the previous one; if the next command is the same as the previous one, the command letter can be omitted.

The result of this kind of compression can be significant. For example, the compact version of the path describing the (smooth) duck is:

```
<path d="M 0 312c40 48 120-32 160-6c0 0 5 4 10-3c10-103 50-83 90-42 c0 0 20 12 30
7c-2 12-18 17-40 17c-55-2-40 25-20 35c30 20 35 65-30 71c-50 4-170 4-200-79z"/>
```

This is 160 characters compared to the original 443 characters. This reduction of over 60% can be significant.

SVG files can also be compressed using the gzip compression algorithms. The viewer decompresses the SVG picture on the fly. The SVG element definitions can be significantly compressed by this approach.

3.6 Other Drawing Elements

Path expressions are extremely versatile but can be unnecessarily powerful when defining simple shapes. As a consequence, SVG includes a number of basic shapes in its specification, such as rectangles (with optional rounded corners) circles, ellipses, single lines, simple polylines and polygons. These shapes are all equivalent to a particular path, and can be considered as simple shorthands.

SVG includes two drawing elements which cannot easily be derived from a path: images and text. Text will be described in more detail later. Images can be included in an SVG drawing by using an external jpg, gif, or png image, in much the same way as it is done in HTML. Images can be positioned anywhere on the canvas and can also be transformed like any other geometric shape.

© David Duce, Ivan Herman, Bob Hopgood 2001

4. Rendering the SVG Drawing

- 4.1 Visual Aspects
- 4.2 Text
- 4.3 Styling

4.1 Visual Aspects

The geometry of an SVG element is largely controlled by the specific attributes associated with the element. The geometry can be transformed either on an element basis or as a group. It is also necessary to define the visual aspects of the drawing elements (colour, line styles, polygon fills, text size, etc).

SVG defines most of the aspects that are common in computer graphics:

- colour of the interior and the border of shapes both as RGB values and as 149 different colour names.
- opacity of the border and interior of shapes from opaque to transparent.
- clipping (any path or closed drawing element can serve as a clipping path).
- line width, style, join and end characteristics.
- fill rules (even-odd or non-zero).
- painting borders and areas using gradients or patterns.

Figure 4.1 shows how linear and radial gradients can be used to make significant differences in the appearances of quite simple shapes.

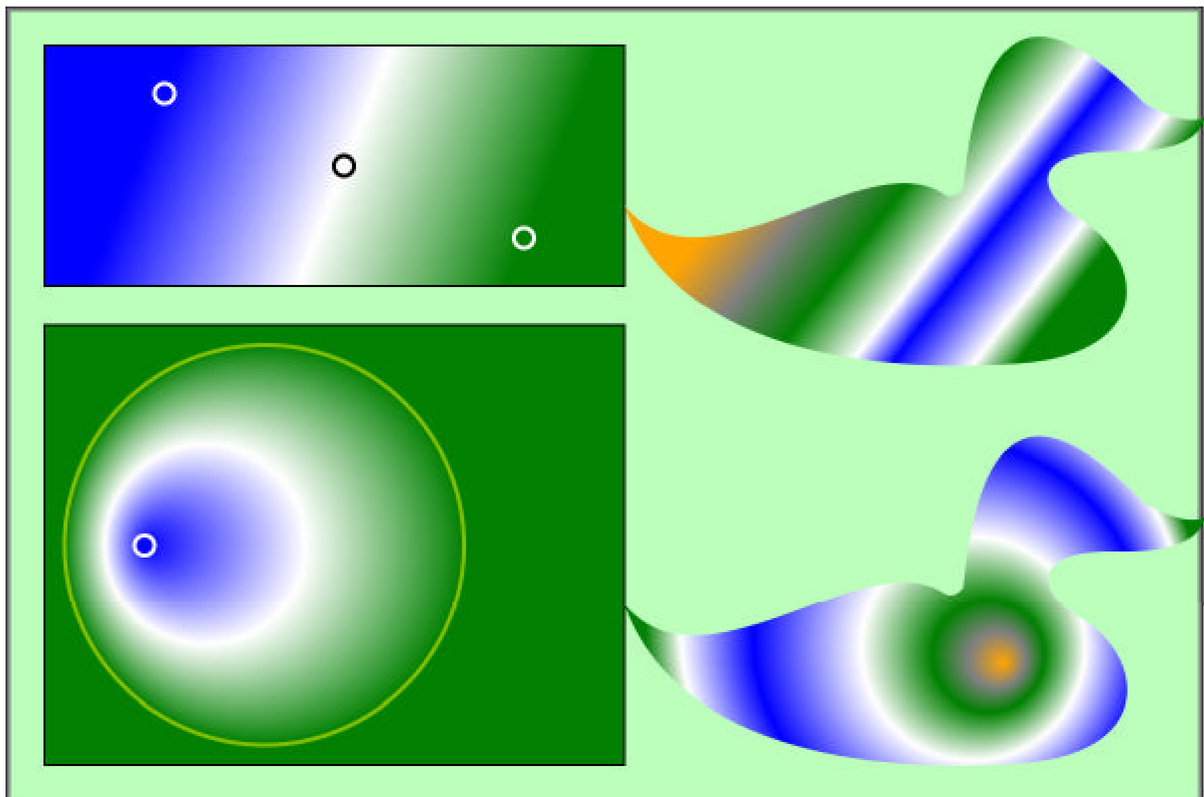

Figure 4.1: Gradient Fills in SVG

Gradients are defined by specifying internal stop points for the colour transitions (blue, white, green in the two left hand side diagrams). The user agent will fill the missing colour areas with a linear or radial interpolation of the colours. In the bottom left radial gradient definition, the centre point and radius defines the boundary for the transition to green. A focal point different from the centre point allows an asymmetric gradient to be defined.

© David Duce, Ivan Herman, Bob Hopgood 2001

The two ducks on the right have more intermediate stops defined.

4.2 Text

Although text is displayed using a separate set of output elements, text uses the same visual aspects. A simple text string would be drawn on the canvas by:

```
<text x="..." y="...">Simple Text</text>
```

The attributes x and y define the origin of the text string. Text has a rich set of visual attributes including text alignment, font selection, font size, text decoration, spacing, baseline control, etc. There are also attributes which ensure a proper inclusion of non-latin character sets, including right-to-left or top-to-bottom writing directions, character orientation, etc. These include and extend the styling aspects available in CSS.

By default, these attributes apply to the whole text string. The **tspan** element provides independent control of both the geometric positioning and rendering of a part of a text string. For example:

```
<text x="..." y="...">Simple <tspan font-style="italic">Italic</tspan> Text</text>
```

This word italic is rendered in italic. Figure 4.2 shows how the **tspan** element can also change the geometric positioning of the sub-string as well as the other visual aspects. This is a rendering of the string: "This is a single text string that has been distributed across the canvas".

```
<g font-family="Verdana" font-size="18" fill="darkblue" font-weight="bold">
<text x="30" y="30">
<tspan x="30" dy="30" font-size="16" >This </tspan>
<tspan x="330" dy="30" fill="red">is </tspan>
<tspan x="530" dy="-30" font-weight="normal">a </tspan>
<tspan x="130" dy="30" font-family="Courier" font-size="28" >single </tspan>
<tspan x="330" dy="-60" fill="green">text </tspan>
<tspan x="30" dy="60" font-style="italic">string </tspan>
<tspan x="430" dy="30" font-size="18">that </tspan>
<tspan x="330" dy="30" font-size="20">has </tspan>
<tspan x="230" dy="30" font-size="24">been </tspan>
<tspan x="130" dy="30" font-size="28">distributed </tspan>
<tspan dx="30" dy="30">across </tspan>
<tspan dx="130" dy="30">the </tspan>
<tspan dx="-230" dy="30">canvas</tspan>
</text>
</g>
```

© David Duce, Ivan Herman, Bob Hopgood 2001

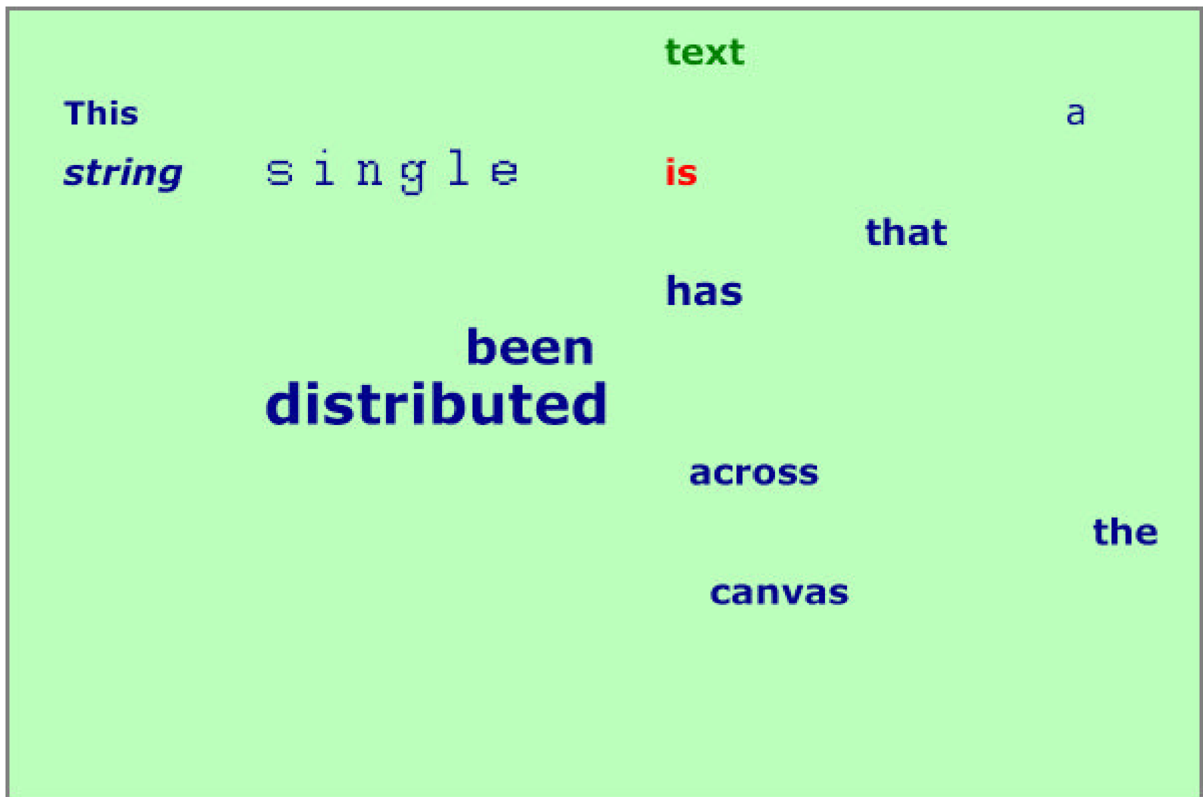

Figure 4.2: Use of tspan

By default, a character string is drawn on a line, whose orientation and length depends on the writing direction and the text alignment properties. However, it is also possible to put a string along a curve; to be more precise, along an arbitrary path. Figure 4.3 shows an example.

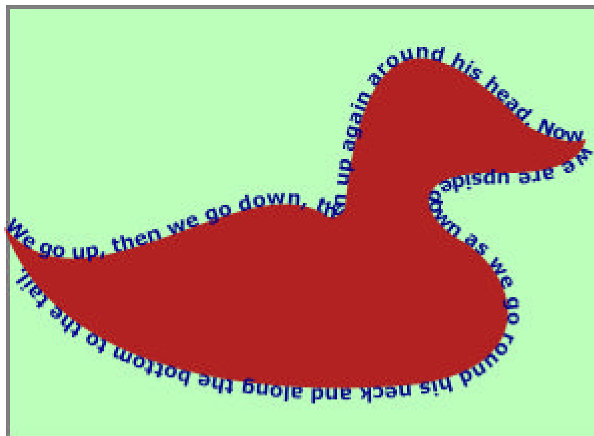

Figure 4.3: Text on a path

© David Duce, Ivan Herman, Bob Hopgood 2001

```
<g stroke="none" fill="lime" id="duck">
<path d="M 0.0 112 L 20 124 L 40 129
C 40 360 120 280 16 30620 124
C 160 306 165 310 170 303
. . .
C 150 395 30 395 0 312 Z"/>
</g>
<text>
<textPath xlink:href="#duck">
We go up, then we go down, then up again around his head.
Now we are upside down as we go round his neck and
along the bottom to the tail.
</textPath>
</text>
```

The text enclosed in the <textPath>...</textPath> pair is displayed along the duck curve.

Text has a rich set of visual aspects in SVG. However, SVG does not have the concept of a paragraph, or a block of text. Hence it does not have the concept of a line height or of justifying text within a specific area by properly positioning lines, and/or adjusting spacing between words. In SVG, these have to be calculated by the application generating the SVG file and repositioning portions of text using the <tspan> element within the text. Text in SVG is reminiscent of the facilities offered by PostScript.

4.3 Styling

The separation of style and content has been an issue in text processing and computer graphics for many years. In the Unix typesetting system, troff, for example, the raw text of a document could be "marked up" to indicate headings, paragraphs, enumerated lists, tables etc. The precise way in which these documents elements were to be presented was described through a macro language. Typically a set of macros (the "ms" macro set was one popular set provided with the troff system) would be constructed to impart a particular appearance or "house style" to a collection of documents. The LaTeX document production system took a similar approach, essentially extending the TeX typesetting system with a particular markup command language. The task of constructing a document using LaTex was reduced (as Lamport puts it) to a "logical design" task. The LaTeX system provided typographic design, through particular style files, and the document's author provided the logical design. A whole class of documents (for example the papers in a journal) can thus be given a uniform appearance; an appearance furthermore, that is controlled by a design expert. Word processing systems such as MSWord and Framemaker provide stylesheets that can be used in a similar way.

Conceptual separation is, however, rarely so clean, and both text and word processing systems also provide functionality that enable the style rules in a sense to be broken, for example, functionality to change to a bold or italic font at a particular point in a document. Thus one author might use bold text directly, where the intention is to emphasise a word, another might use italic for this purpose, even though an "emphasis" style is provided. This might be laziness on the part of the document author, though sometimes bold and italic are used directly because bold and italic are intrinsic aspects of the presentation of the text, for example, a trademark might be set in a particular font and weight of text.

A similar separation between style and content in Web documents has been achieved by Cascading Style Sheets [9]. The intention is that the page author should think in terms of the content and structure (the logical design) of the Web document, using HTML elements such as h1, h2, ul, etc. and set up style sheets to control the visual appearance (the visual design) of these elements when rendering in a Web browser or printed. Such style sheets can be embedded directly into Web pages or can be linked to the pages through an appropriate URL. This basic approach provides a mechanism to control the consistency of the visual rendering of a collection of Web documents. Advantages of this approach include:

© David Duce, Ivan Herman, Bob Hopgood 2001

- **Easy maintenance**: changing the colour of all **h1** elements can be done by changing just the style sheet, instead of scanning through the whole document.
- **House style**: can be defined by a collection of separate style sheet files.
- **Clarity**: pages using style sheets are usually structurally cleaner and hence easier to maintain.
- **Adaptation to the end-user**: style sheets may include special statements for audio browsers; browsers may allow the end user to use personal style sheets to adapt to personal disabilities or operating environment.
- **Design control**: style sheets may be prepared by professional designers, thus improving the overall visual quality and representation of the Web pages.

The use of style sheets is not limited to HTML, style sheets may also be used with XML pages in a similar way.

There are some parallels in the development of style control in computer graphics. In early graphics systems it was commonplace to control the visual appearance of graphical output primitives by attributes, for example to control properties such as linestyle, line width, colour, text font, etc. Attributes were typically set modally, for example by a set_colour function, the value remaining in force until a new value was set. If a particular attribute value was not supported by a particular device, it was permissible to simulate the effect of that value using other values for that attribute, other attributes and other primitives. A particular dashed linestyle could, for example, be simulated by a sequence of individual lines. The essence of this approach was that the application provided a precise specification of the required appearance and the system did its best to achieve the specified effect.

The earliest versions of the Graphical Kernel System (GKS) proposed an alternative approach [22]. The central idea was the notion of a "pen". Output primitives were drawn with a particular pen number. Properties such as colour, line thickness, and linetype, were attributes of the pen that could be controlled by a set pen representation function. Pen representations could be set differently for different types of workstation, which gave a convenient mechanism for tailoring the appearance of the output to the capabilities of the workstation.

These ideas evolved during the development of GKS and it became clear that there was room for both mechanisms, recognising that visual appearance can be used for different purposes. Sometimes the precise appearance of a primitive is important. In architecture, for example, different patterns denote different types of building material in a precise way; indiscriminate substitution could result in a house of sand rather than of stone! At other times, patterns are used purely to achieve differentiation between different types of object, the precise pattern used is unimportant, what matters is that pattern A should be visually distinguishable from pattern B. The first edition of GKS [22] distinguished between workstation independent properties (termed "aspects" in GKS) of primitives which have the same value on all workstations on which the primitive is displayed, and workstation dependent aspects which could have different values on different workstations. Values of aspects were controlled by attributes. For workstation independent aspects, there was one attribute per aspect. Workstation dependent aspects could be specified by **bundled** specification or **individual** specification. Bundled specification used a lookup table approach. A single attribute for each primitive, the primitive index, controlled the values of all the workstation dependent aspects of the primitive. The primitive index was an index into a primitive bundle table, which specified the values of all the workstation dependent aspects for that type of primitive. Each workstation had its own set of bundle tables and each type of primitive had its own bundle table. Thus appearance could be tailored to the capabilities of each workstation. A set of Aspect Source Flags (ASFs) controlled the mode of specification of each aspect independently. Some aspects could be specified individually, whilst others were specified by a bundle. This control was provided at the individual primitive level. Each GKS implementation had a default setting for the ASFs, either all bundled or all individual, in an implementation-dependent manner.

At the time they were made, some of these design decisions were the result of political compromise between the nations involved in the standardisation process, but it eventually became clear that they reflected a real attempt to recognise and separate concerns.

© David Duce, Ivan Herman, Bob Hopgood 2001

In the 2nd Edition of GKS, GKS:94, these ideas were taken a little further. A central notion in GKS:94 was the concept of a picture in normalised device coordinate space of which different views could be taken on workstations. Each workstation could select different parts of the NDC picture for display. There was a notion of NDC picture in GKS:85 but it was a weaker, less well-defined notion.

GKS:94 provided similar mechanisms for controlling the appearance of primitives to GKS:85, though the language in which they were described was changed. For the present purposes, it is enough to note that source, NDC and logical attributes were bound to a primitive when it was created. Each workstation had bundle tables for each class of primitive. At a workstation, the values of the logical attributes (which controlled the appearance of the primitive on the workstation's display surface) were the values bound in the NDC picture for attributes for which the ASF was INDIVIDUAL, and were replaced by values from the workstation bundle table indexed by the bundle index bound to the primitive for attributes for which the ASF was BUNDLED.

Selection of parts of the picture for display was achieved by namesets and selection criteria. A nameset attribute consisting of a set of user-defined names could be associated with each primitive. A primitive was selected for display if its nameset attribute satisfied a selection criterion, constructed from the boolean operators and the operators contains(nameset), isin (nameset) and equals(nameset). Shorthands SELECTALL and REJECTALL were also included. This approach was very powerful, and a significant advance on the earlier approach to selection (based on namesets and filters constructed from an inclusion set and exclusion set) in the PHIGS standard [23]. In GKS:85 it was not possible to change the value of a primitive's attributes after the primitive was created. In GKS:94 such editing operations were permitted, again using the nameset and selection criterion mechanism to select the set of primitives to which an editing operation would be applied.

SVG is defined as an XML language and makes use of the styling functionality provided by Cascading Style Sheets for XML documents. However, as hinted at above, styling for graphics is potentially more complex than for text (or at least more complex than the styling model for Web documents). Is colour in graphics, for example, style or content? If colour is used on a map to differentiate different countries, it is probably style. What is important is that the colour of one country should be distinguishable from that of another. Styling can be very valuable in this situation: the best choice of colour might depend on the context in which the map is used, specifying colour through the style mechanism makes it straightforward to change colours from one context to another.

However colour is not always style. The colour chosen in a logo, or in an artistic image, or in the precise representation of real world objects is inherently a part of the content of the picture. In GKS:94 terms, colour here is an attribute completely defined in the NDC picture, to be rendered exactly (so far as is possible) in every view of the picture. Changing colour through a style sheet mechanism in such a picture in a Web document would be fundamentally wrong.

These cases: colour as style and colour as an intrinsic property of a primitive, are recognised in SVG and two different mechanisms for setting visual attributes are provided. One method is to set rendering attributes directly. For example:

```
<rect fill="yellow" stroke="black" x="20" y="30" width="300" height="200"/>
<rect fill="none" stroke="red" x="20" y="330" width="300" height="200"/>
```

Describes two rectangles, the first is yellow with a black border; the second is hollow with a red border.

© David Duce, Ivan Herman, Bob Hopgood 2001

The second method using styling is illustrated by:

```
<svg viewbox= "0 0 500 300" >
<style type="text/css"> <![CDATA[
rect {stroke:black; fill:yellow}
rect.different {stroke:red; fill:none}
]]>
</style>
...
<rect x="20" y="30" width="300" height="200"/>
<rect class="different" x="20" y="330" width="300" height="200"/>
</svg>
```

This achieves the same visual result as the first approach. The "style" element encloses a style sheet expressed in the CSS syntax. (The CDATA annotation is used in order to escape the style language from the XML syntax checker.) Two styles are defined, the first for rectangles in general (filled in yellow with a black border) and a second for rectangles belonging to the class "different" defining rectangles with a red border and hollow interior.

The same effect could be achieved by defining an external sheet in the file mystyle.css as:

```
rect {stroke:black; fill:yellow}
rect.different {stroke:red; fill:none}
```

and attaching it to the SVG document by:

```
<?xml-stylesheet type="text/css" href="mystyle.css" ?>
<svg viewbox= "0 0 500 300" >
<rect x="20" y="30" width="300" height="200"/>
<rect class="different" x="20" y="330" width="300" height="200"/>
</svg>
```

Style can also be associated directly with an element through the style attribute. The example above could also be written:

```
<rect style="stroke:black;fill:yellow" x="20" y="30" width="300" height="200"/>
<rect style="stroke:red; fill:none" x="20" y="330" width="300" height="200"/>
```

It is instructive to compare the SVG mechanisms with the mechanisms in GKS:94.

The GKS:94 nameset attribute has parallels in the SVG class attribute. Both can be used to select primitives/elements for some purpose. The nameset attribute allows a *set* of names to be associated with a primitive. In the examples usually seen in print, the class attribute consists of a single name, though the specification does allow multiple names (separated by spaces). Thus, in terms of the specification of namesets and class attributes, GKS:94 and SVG have equivalent capability.

However the ways in which selection occurs in the two specifications are interestingly different. The NDC picture in GKS was essentially a flat sequence of primitives. The semblance of structure could be injected by appropriate choice of nameset values (for example, {car, wheel, front-nearside}) but the NDC picture remained a sequence of primitives.

SVG on the other hand allows pictures to have arbitrary hierarchical structure. CSS provides powerful mechanisms for controlling appearance, both on the basis of the values of attributes (usually the class attribute, but other attributes could be used also) and the actual structure of the SVG element tree.

© David Duce, Ivan Herman, Bob Hopgood 2001

A simple example of the use of the class attribute was given above. It is also possible to write:

```
rect [class ~="different"] {stroke:red; fill:none}
```

which would also select rectangles whose class attribute contains the value different in a set of space separated class attribute values. The "." notation introduced earlier in fact corresponds to the "~=" construct. A more complex example is shown below.

```
<svg width="400" height="250" >
<style type="text/css"> <![CDATA[
rect {stroke:black; fill:white}
rect.different {stroke:red; stroke-width:4; fill:none}
rect.different.again {stroke:none; fill:white}
rect.different.again.encore {stroke:blue; stroke-width:8; fill:none}
]]>
</style>
<rect x="20" y="20" width="100" height="100"/>
<rect class="different" x="20" y="140" width="100" height="100"/>
<rect class="different again" x="140" y="20" width="100" height="100"/>
<rect class="different again encore" x="140" y="140" width="100" height="100" />
</svg>
```

Figure 4.4 shows the result.

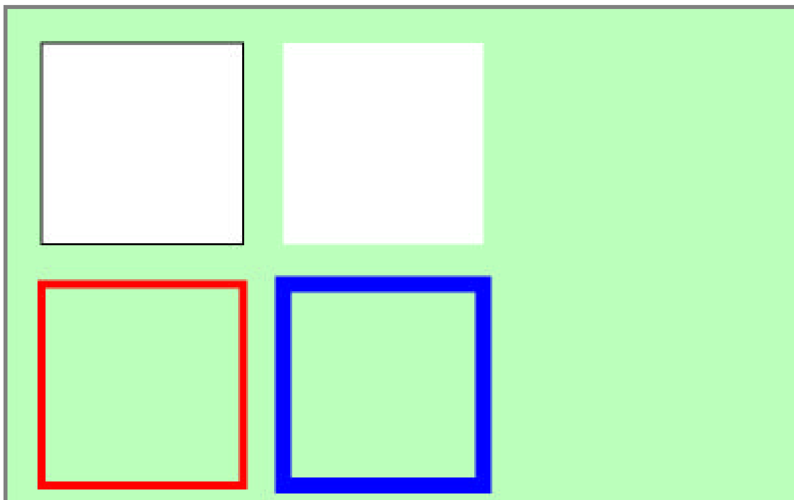

Figure 4.4: Class selection

This example shows that SVG and CSS have some of the flavour of the GKS:94 selection criteria, though the ways in which selection criteria could be constructed in GKS:94 were more powerful than the mechanisms in SVG/CSS.

The CSS functionality for matching the structure of the document tree is quite powerful, though slanted more towards the structures found in text documents than the more general structures found in graphics. A full discussion of this functionality is beyond the scope of this paper, but the example below illustrates the general idea.

© David Duce, Ivan Herman, Bob Hopgood 2001

```
<svg width="600" height="400" >
<style type="text/css"> <![CDATA[
rect {stroke:black; fill:yellow}
g > rect:first-child {stroke:red; fill:none}
]]>
</style>
<g>
<rect x="20" y="20" width="100" height="100"/>
<rect x="140" y="20" width="100" height="100"/>
</g>
</svg>
```

The selector '>' matches any **rect** element that is the first child of a **g** element.

One of the important concepts in CSS is the notion of **cascade**. Three different types of style sheet can be associated with a document: author, user and user agent. The author of a document can supply style information, so can the user and so can the user agent (usually a browser). In general, style sheets from these three sources may overlap in the styling they specify for a particular element (indeed there may be overlap from within a single style sheet - there is an example of this in Figure 4.4) and so the notion of cascade is introduced to define the effect. In essence, weights are assigned to each style rule and when several rules apply, the one with the greatest weight takes precedence. The details are quite involved and go beyond the scope of this paper. The interested reader is referred to the CSS2 specification [9]. One of the consequences though of this general mechanism is that presentation attributes attached to SVG elements have lower priority than other CSS style rules specified in author style sheets or style attributes. SVG and CSS do not have the equivalent of the GKS ASFs, so there is no general way to ensure that the analogues of individual attribute specification (presentation attributes) will actually apply in all contexts in which SVG is used. From a graphics perspective this might be considered unfortunate, but it is the price paid for embedding graphics in a context where different priorities normally pertain.

© David Duce, Ivan Herman, Bob Hopgood 2001

5 Filter Effects

One important use for graphics on the Web is to include various types of artwork on the Web pages. These can be company logos, sophisticated advertisement images, computer arts, etc. In general, such images are produced by complex image processing tools resulting in images rich in shades and colours. SVG includes a number of tools, collectively called **filter primitives**, which can be used to achieve similar effects. The filters are obviously executed on the client side and that usually means that even very complex visual images can be described through relatively small files, rather than transmitting large pixel images. Also, the combination of filter effects with the more traditional graphics described in the earlier sections opens up rich possibilities for artwork creation.

As noted earlier, an SVG agent produces an image, conceptually, into a canvas. If no filter processing occurs, this image is then transmitted to the screen directly, using the painters' model. Filtering in SVG can occur between these two steps: the user can define filters which will operate on the output of the image generation on the canvas and it is the output of this filter which will, eventually, appear on the screen itself.

The filtering operation follows a dataflow model used in other image processing environments such as Khoros [30] or the pbm toolkit widely used in Unix. A filter is a combination of filter primitives; each primitive can have one or more inputs and an output. The inputs to a primitive are either the source image (ie, produced by previous SVG operations) or the output of another primitive; the output of the last primitive is displayed on the screen. The RGB values of the source image or the alpha (opacity) values can be separated for the input to a primitive, for example a filter primitive might operate on the opacity values only.

Figure 5.1 shows the result of applying some filter operations to the duck.

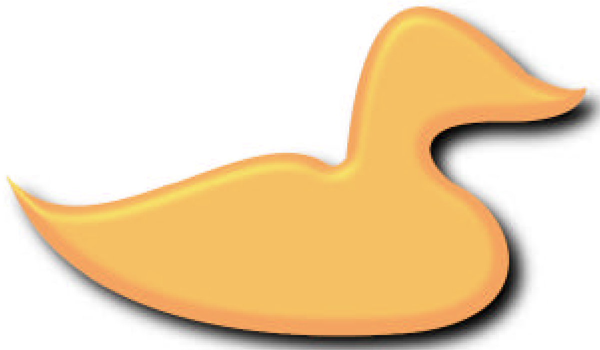

Figure 5.1: Filtered Duck

© David Duce, Ivan Herman, Bob Hopgood 2001

Figure 5.2 shows the dataflow network that creates the image.

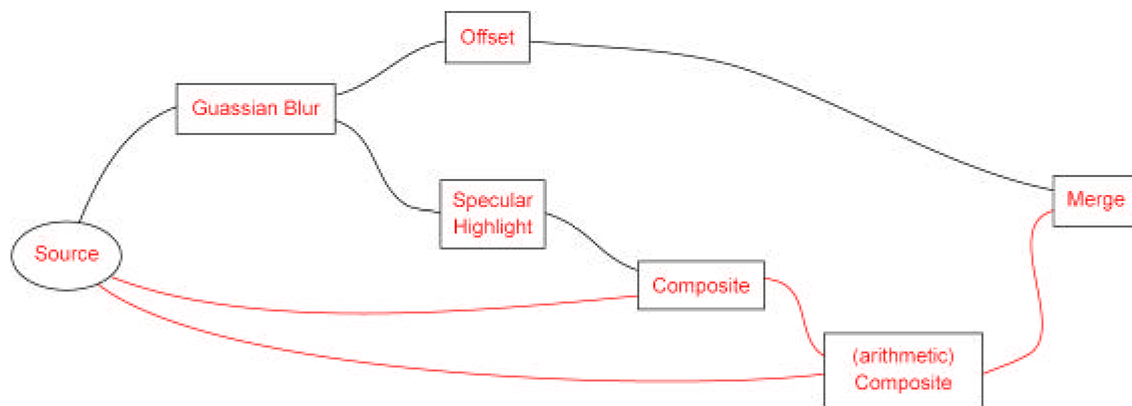

Figure 5.2: Data Flow of Filter Operations

An informal description of each filter primitive is:

1. A Gaussian Blur filter receives the opacity value of the source image and blurs it
2. An offset filter creates a slight offset in both the x and y direction of the blurred opacity value (this will be the dropped shadow of the final image)
3. The specular highlight will create a simulated highlight image on the blurred opacity image
4. The highlight image will be combined with the original opacity image to "cut off" the highlight so that the resulting image is no bigger than the original graphics
5. The original image is combined with the highlight to produce the original graphics with highlight (but without the shadow)
6. The shadow is combined with the previous step to produce the final image

When viewing Figure 5.2 on the Web, clicking on the processing nodes shows the result of that particular operation. The approximate SVG equivalent (trimming some of the details) is:

```
<defs>
<g transform="translate(10,-190)" id="theDuck" stroke="none" fill="sandybrown">
<path d="M 0 312 ... z"/>
</g>
<filter id="MyFilter">
<feGaussianBlur in="SourceAlpha" stdDeviation="4" result="blur"/>
<feOffset in="blur" dx="4" dy="4" result="offsetBlur"/>
<feSpecularLighting in="blur" surfaceScale="5" specularConstant="1"
specularExponent="10" lightColor="green" 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>
<use xlink:href="#theDuck" filter="#MyFilter"/>
```

© David Duce, Ivan Herman, Bob Hopgood 2001

Given the complex visual effects of the result, the size of the SVG source is not particularly large compared to the original image of the duck.

The specification of each filter is quite complicated and relies on an intimate knowledge of various image processing techniques. However, it can be expected that authoring tools that can export SVG will hide the details from the casual user. Appendix A lists the filter primitives available in SVG with a short description of their functionalities.

© David Duce, Ivan Herman, Bob Hopgood 2001

6. Animation

- 6.1 Introduction
- 6.2 What is Animated
- 6.3 How the Animation Takes Place
- 6.4 When the Animation Take Place

6.1 Introduction

The facilities of SVG described until now are comparable to a "classical" 2D graphics environment, such as GKS or PostScript. However, SVG also includes some so-called animation objects, which give a very different flavour to the system.

Animation in terms of SVG means the possibility to dynamically change of most of the attributes (both in XML in CSS) of SVG elements. For example, one might dynamically change the position, the geometry, various style elements (colour, linestyle, etc). The simple example below gives a flavour of the approach taken:

```
<rect x="20" y="10" width="120" height="200">
<animate attributeName="width" from="120" to="200" begin="0s" dur="8s"
fill="freeze"/>
</rect>
```

This simple animation will change the width of the enclosing rectangle from the (original) value 120 to the final value 200. The change will take place in 8 seconds (starting at moment the image is loaded) and the result will be "frozen", ie, the width of the rectangle will remain 200 beyond the 8 seconds.

It is worth noting that the animation concepts of SVG have not been defined by the SVG Working Group in isolation; instead, the functionality defined by another W3C document, called SMIL2.0, has been incorporated. SMIL2.0 is concerned about multimedia synchronisation and presentation; it concentrates on specifying how various media presentation (audio, still or moving images, video, etc) can be presented. (In terms of SMIL2.0 an SVG image is but another type of media.) SMIL2.0 is a coordinating language in the sense that the various media objects are considered as black boxes for SMIL2.0, and the specification concentrates on the (two dimensional) layout, on the timing and synchronisation among media objects, on transition control, etc. SMIL2.0 also defines animation facilities (eg, moving objects around on the screen, changing their attributes); in terms of SMIL2.0, animation objects are yet another type of media objects, subject to the same timing and synchronisation constraints as other types of media objects. SVG took over the exact specification of the animation objects including their related timing; it is a nice example of interoperability among W3C recommendations.

When defining animation objects the author has to decide the following aspects:

1. **What** is animated?
2. **How** should the animation take place?
3. **When** should the animation take place and for how long?

Each of these issues is now considered separately.

© David Duce, Ivan Herman, Bob Hopgood 2001

6.2 What is Animated

As noted above, almost all attributes and CSS properties can be animated. There are five different types of animation objects in SVG:

animate
to animate general attributes, as in the example above

set
a shorthand notation for cases when attributes are 'discrete', eg, visibility

animateTransform
an animation object tailored to the transform attribute

animateMotion
to specify movement of an object along a specified path

animateColor
an animation object tailored to colour changes

The semantics of the different animation objects, in terms of timing, animation control, etc, are identical; the differences reside solely in the way the target attributes are described.

We have already seen an example for an animate object changing the width of a rectangle. The following example changes the opacity of a circle, in which the opacity is defined as a (CSS) style property:

```
<circle cx="50" cy="50" r="20" style="fill:red; opacity:1">
<animate attributeType="CSS" attributeName="opacity" from="1" to="0" dur="2s"/>
</circle>
```

Note the **attributeType** attribute, which differentiates whether the target is XML (this is the default) or CSS.

The **set**element can be used as follows:

```
<circle cx="50" cy="50" r="20" style="fill:red; opacity:1">
<set attributeName="visibility" from="visible" to="invisible"/>
</circle>
```

There is no **dur** attribute: the effect of **set** is instantaneous. Note also that the values for **from** and **to** are not necessarily numerical, as shown in the example.

An example for the **animateTransform** element is:

```
<circle cx="50" cy="50" r="20">
<animateTransform attributeName="transform" type="translate" from="0,0" to="40,20"
dur="3s"/>
</circle>
```

which will move the centre of the circle from its initial position to the point (90,70) in 3 seconds.

The animateMotion object can be used to move an object along a general path, rather than specifying a series of transformation objects. For example:

```
<path d="M0 0 v -2.5 h10 v-5 l5 7.5 l-5 7.5 v-5 h -10 v-2.5">
<animateMotion dur="6s"
path="M 100 150 c 0 -40 120 -80 120 -40
c 0 40 120 80 120 40 c 0 -60 -120 -100 -120 -40
c 0 60 -120 100 -120 40"
rotate="auto"/>
</path>
```

© David Duce, Ivan Herman, Bob Hopgood 2001

In this case the surrounding object (an arrow) will be moved along the path that is specified within the **animateMotion** element. The **rotate** attribute defines the orientation of the arrow as it proceeds along the path: the arrow can point along the path (as in the example), it can stay at a constant rotation from its initial position, or it can point away from the direction of the motion.

Finally, the colour of the object can be changed through

```
<circle cx="50" cy="50" r="20">
<animateColor attributeName="fill" from="red" to="crimson" dur="3s"/>
</circle>
```

(Animation takes place in the standard RGB space.)

Note that in all our examples so far only one animation object has been defined for an element. This was only to keep the examples simple; one can include several animation objects that have simultaneous effects on the object.

6.3 How the Animation Takes Place

By default animation is linear. This means that a linear function is calculated between the **from** and the **to** values within the specific time duration. It is, however, possible to have finer control over the animation function through:

- specifying intermediate key time values
- replacing the linear animation functions by (cubic) splines

The first possibility is shown in the following example:

```
<rect x="20" y="10" width="120" height="200">
<animate attributeName="width" begin="0s" fill="freeze"
values="120; 180; 190; 200" keyTimes="0; 2; 4; 8"/>
</rect>
```

This also specifies that the rectangle width should change in 8 seconds from 120 to 200, but in this case intermediate values that must be reached at 2 and 4 seconds (a very fast change at the beginning, but slowing down at the end) are also specified.

To achieve a somewhat similar (but much smoother) effect one can also define a spline function for the time change:

```
<rect x="20" y="10" width="120" height="200">
<animate attributeName="width" begin="0s" fill="freeze"
values="120; 200" keySplines="0 0.75 0.25 1"/>
</rect>
```

The spline used (defined in the [0,1] interval) is shown in Figure 6.1. The change of values is relatively fast at the beginning of the 8 second animation and slows down towards the end.

© David Duce, Ivan Herman, Bob Hopgood 2001

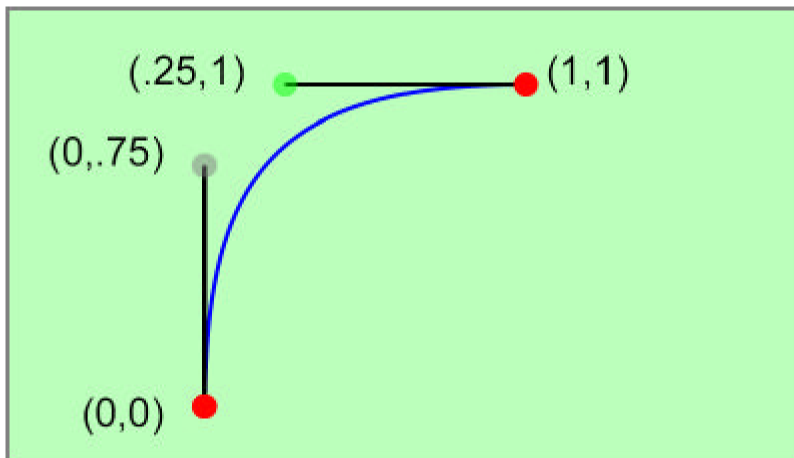

Figure 6.1: Spline Curve

Finally, the two control methods can be combined: a separate key spline could be defined for each key time interval.

6.4 When the Animation takes Place

In its simplest form, animation objects define a time range in which they operate. The attributes **begin**, **dur**, **end**, **repeatCount** and **repeatDuration** are the most important ones in this respect, and they all have a clear intuitive meaning. All these attributes can refer to time values in different formats (in practice, seconds and miliseconds are probably the most useful, but minutes or even hours can be used, too). Semantic conflicts can occur (eg, what happens if both duration and end times are defined and the values do not match?) and the SMIL2.0 document gives a very detailed account of these. However, in practice, authors use either one or the other. It is important to note that all the time count values are **relative to the load time of an object**, ie, `begin="3s"` means 3 seconds after the object has been loaded by the viewer.

Describing the animation solely on the basis of time is tedious and it also lacks the possibility for user interaction. However, the full specification of the **begin** (and **end**) attributes allows for **event based interaction**, too. These can be used for different purposes. Animation objects can be "chained", as in the following example:

```
<animate id="a1" begin="0s" ..../>
<animate id="a2" begin"a1.end" .... />
<animate id="a3" begin="a2.begin+8s" ... />
```

The second animation object would begin when the first one ends, while the third animation begins 8 seconds after the beginning of the second. Animation objects can be bound to user interactions:

```
<circle id="a1" ... />
...
<animate begin="a1.click" ..../>
```

Here the animation begins when the user clicks on the circle. While "click" is probably the most useful event type in this respect, one can also use "focusin", "focusout" (eg, when a text is selected) and "activate

The combination of the timing and the interaction facilities to control animation is quite powerful; it is possible to build up complex animated, and possibly interactive objects without using scripting (which is the "traditional" way of programming interaction on, eg, an HTML page). However, the animation objects also have their limitations: it is not possible to include complex calculations as part of attributes, nor can animation objects change over time. For this reason scripting is also included in the SVG specification, offering an alternative way of producing dynamic SVG images.

© David Duce, Ivan Herman, Bob Hopgood 2001

7. Scripting and the DOM

When a Web document (be it HTML, XHTML, or some other XML language) is loaded into a browser, the document is parsed and internal structures to represent the document are created. The Document Object Model (DOM) is an API that can be used to manipulate these internal structures, as if the document were represented as a hierarchically structured collection of objects. The DOM does not mandate that the browser implement a document representation as a hierarchy of objects, though it might. Using a scripting language, the DOM permits the presentation of a Web document to be modified dynamically within a browser. This interface also permits content to be created dynamically within the browser. It should be noted that the copy of the Web document held on the server is not modified, it is purely the representation within the browser that is modified.

Modification to a document via the DOM is usually triggered by a browser event, the mouse being moved over a particular region of the document, a mouse click over a particular region, etc. The example below illustrates the main ideas.

```
<svg width="500" height = "500">
<script language="JavaScript"
type="text/javascript">
<![CDATA[
function onmouseover(evt) {
var elem = evt.target;
var style = elem.getStyle();
style.setProperty("fill-opacity", 0.5);
}
function onmouseout(evt) {
var elem = evt.target;
var style = elem.getStyle();
style.setProperty("fill-opacity", 1.0);
}
]]></script>
<rect x="20" y="20" width="250" height="250" style="fill:red; stroke:none"
onmouseover="onmouseover(evt)" onmouseout="onmouseout(evt)" />
<rect x="210" y="210" width="250" height="250" style="fill:green; stroke:none"
onmouseover="onmouseover(evt)" onmouseout="onmouseout(evt)" />
</svg>
```

The attributes **onmouseover** and **onmouseout** on the two rect specify the names of script functions to be invoked when the corresponding event occurs. The parameter passed to the function enables the corresponding element to be located in the document object tree.

The function onmouseover changes the **fill-opacity** property of the element to the value 0.5. This function is invoked when the mouse moves onto the region covered by the rectangle. The function onmouseout resets the **fill-opacity** property of the element to the value 1.0. This function is invoked when the mouse moves out of the region.

Comparing the functionality provided by CGM and SVG, disjoint polylines, pie slices, predefined dash styles are available with the WebCGM Profile but not available in SVG. SVG provides transformable symbols instantiated by the **use** element, bundled attribute styling using the **g** element, pattern and gradient fills. These facilities are available in CGM but not the WebCGM Profile. Almost all the other functionality is common between CGM and SVG. In consequence, there is some work underway attempting to provide a common Document Object Model [21] for interacting with documents in both WebCGM and SVG formats.

© David Duce, Ivan Herman, Bob Hopgood 2001

8. Current State and the Future

- 8.1 Implementations
- 8.2 Metadata
- 8.3 Extensions to SVG

8.1 Implementations

Current information concerning the state of browser plug-ins, stand-alone viewers, editors and converters is kept at the W3C SVG Working Group Home Page [25].

In April 2001, Adobe released the 2.0 version of their plug-in that works with most browsers on most platforms. Stand-alone viewers are available from IBM and CSIRO. Toolkits are available from Apache and CSIRO. Editors are available from JASC (WebDraw), Mayura (Mayura Draw). This is not a definitive list but just gives an indication of the level of support. Many existing graphics design packages have added an SVG export or import function.

To help implementers debug their applications, W3C maintains an SVG Test Suite [28] of over 100 basic functionality tests for SVG. Each test comes with a PNG image that shows what the test should produce in terms of output.

8.2 Metadata

Searching for content is a major function on the Web. SVG attempts to make the textual content of a picture easy to find by allowing international text to appear as a sequence of text characters irrespective of writing direction. A search engine that is aware of the **tspan** element can search for multi-line text and text that may be scattered around the diagram.

SVG also provides a **metadata** element whose contents should be elements from other XML namespaces. In particular, the metadata could be expressed in RDF [29]. An ontology such as the Dublin Core could be used to express information concerning the creator, the date it was produced etc. For CAD applications, the metadata might point into the application database to define the parts and materials to which the diagram refers.

SVG provides the mechanisms to allow rich metadata to be added to SVG diagrams. It will be interesting to see how this is used in practice.

8.3 Extensions to SVG

The W3C SVG Working Group does not believe that its job is done and early in 2001 they started collecting requirements for a new SVG 2.0.

SVG is seen as a basic platform which existing packages, such as Adobe Illustrator, can use as a Web delivery system. There has also been interest in extending SVG by defining richer languages that map down into SVG.

One example is CSVG [26] [27]. This is a constraint extension to SVG that defines a picture as a set of graphical objects. The properties of these objects including their position can be constrained by properties of other objects. For example, a rectangle can be specified to be above another rectangle, and a line can be constrained to connect the bottom of the top one to the top of the bottom one. To avoid over-constraining the system, each constraint can have its strength specified. Constraints are satisfied starting from the most important until a unique solution is found.

As XML applications become more common, there will be a need to define their inter-relationships more precisely. For example, MathML [20] is an XML application that renders mathematics correctly. The user may require mathematics to be included as text in an SVG diagram. SMIL [19] defines the layout and timing of a set of media objects. SVG is a specification that uses the SMIL animation facilities to create a media object. SVG allows media objects to be embedded within an SVG diagram. At some stage the relationships between these various specifications need to be more precisely defined.

SVG is a major advance in replacing images by vector graphics on the Web. However, it is just the first step and there is still a great deal of work to be done before the graphics environment on the Web reaches the maturity of existing computer graphics systems.

© David Duce, Ivan Herman, Bob Hopgood 2001

Appendix A:Filter Primitives in SVG

Here is the list of the various filter primitives in SVG.

Blend
Pixel-wide combination of two images using various blending modes (normal, darken, lighten, multiply, screen).

Color Matrix
A general transformation of RGB and alpha values (using a 5x4 matrix). Some predefined matrices for, eg, hue rotation or saturation are also available.

Component Transfer
A component level remapping of the R, G, B, and alpha values. It allows operations like brightness or contrast adjustment, colour thresholding, etc. The transfer functions can be linear, table driven or gamma (for gamma correction).

Composition
Combination of two input images pixel-wise in image space using one of the [31] compositing operations or a simple arithmetic combination.

Convolution
A convolution matrix can be specified to perform convolution on the source image (used in edge detection, sharpening, etc)

Diffuse and Specular Lighting
The filters use the alpha channel as a bump map, ie, an imaginary surface is created in 3D with the alpha values. Lighting (using the usual Phong model) is then used to simulate lighting effects. Light sources can be distant, point, or spot.

Displacement maps
The pixel values of one image are displaced in function of the pixel values of another image.

Flood
Creation of a rectangle filled with a specific colour and opacity; to be used with other filter primitives.

Gaussian Blur
The effect is clear; the standard deviation of the blur can be controlled by special attributes.

Image
Refers to an external image which can then be used by other primitives as an input

Merge
Merge a number of input images.

Morphology
Performs "fattening" or "thinning" of an artwork.

Offset
Displacement of an image in x and y directions.

Tile
Fill a rectangle repeating an input image in a tiled pattern

Turbulence
It creates an image using the Perlin turbulence function [32], it allows the synthesis of various artificial textures.

© David Duce, Ivan Herman, Bob Hopgood 2001

References

[1] 'How the Web was Born'. James Gillies , Robert Cailliau. Oxford University Press, 2000.

[2] 'Weaving the Web'. Tim Berners-lee with Mark Fischetti. HarperCollins, 1999.

[3] 'Spinning the Web'. Andrew Ford. Thomson, 1995.

[4] 'PNG, The Definitive Guide'. Greg Roelofs. O'Reilly, 1999.

[5] 'ISO/IEC 8632:1999: Information technology -- Computer graphics -- Metafile for the storage and transfer of picture description information

[6] 'The CGM Handbook'. Lofton Henderson, Anne Mumford. Academic Press, 1992.

[7] 'http://www.w3.org/TR/REC-WebCGM: WebCGM Profile' World Wide Web Consortium, 1999.

[8] 'http://www.w3.org/TR/xlink/: XML Linking Language (XLink) Version 1.0'. World Wide Web Consortium, 2000.

[9] 'http://www.w3.org/TR/REC-CSS2: Cascading Style Sheets, level 2 (CSS2) Specification'. World Wide Web Consortium, 1998.

[10] 'http://www.w3.org/TR/REC-xml : Extensible Markup Language (XML) 1.0 (Second Edition) '. World Wide Web Consortium, 2000.

[11] 'http://www.w3.org/TR/xslt: XSL Transformations (XSLT) Version 1.0' World Wide Web Consortium, 1999.

[12] 'http://www.w3.org/TR/REC-xml-names : Namespaces in XML'. World Wide Web Consortium, 1999.

[13] 'http://www.w3.org/TR/REC-png: PNG (Portable Network Graphics) Specification'. World Wide Web Consortium, 1996.

[14] 'http://www.w3.org/TR/SVG/ : Scalable Vector Graphics (SVG) 1.0 Specification'. World Wide Web Consortium, 2000.

[15] 'http://www.w3.org/Submission/1998/05/: Web Schematics'. Submission to W3C, March 1998.

[16] 'http://www.w3.org/Submission/1998/06/: Precision Graphics Markup Language (PGML)'. Submission to W3C, April 1998.

[17] 'http://www.w3.org/Submission/1998/08/: Vector Markup Language (VML) '. Submission to W3C, May 1998.

[18] 'http://www.w3.org/Submission/1998/20/: DrawML'. Submission to W3C, December 1998

[19] 'http://www.w3.org/TR/smil20: Synchronized Multimedia Integration Language (SMIL 2.0) Specification'. World Wide Web Consortium, 2001.

[20] 'http://www.w3.org/TR/MathML2: Mathematical Markup Language (MathML) Version 2.0'. World Wide Web Consortium, 2001.

[21] 'http://www.w3.org/TR/DOM-Level-2-Core : Document Object Model (DOM) Level 2 Core Specification'. World Wide Web Consortium, 2000.

[22] 'ISO Standards for Computer Graphics - The First Generation'. D.B. Arnold, D.A. Duce. Butterworths, 1990

[23] 'GKS-94: An Overview'. K.W. Brodlie, L.B. Damnjanovic, D.A. Duce, F.R.A. Hopgood. IEEE Computer Graphics and Applications, pp. 64-71, 1995.

[24] 'http://www.cgmopen.org'. CGM Open.

[25] 'http://www.w3.org/Graphics/SVG/'. W3C SVG Overview.

[26] 'http://www.cs.washington.edu/homes/gjb/CSVG/'. CSVG Home Page

[27] 'A Constraint Extension to Scalable Vector Graphics'. G. J. Badros, J.J Tirtowidjojo, K.Marriott, B Meyer, W. Portnoy, A. Borning. Tenth International World Wide Web Conference, Hong Kong, may 2001, pp 489-498.

[28] 'http://www.w3.org/Graphics/SVG/Test/'. SVG Test Suite.

© David Duce, Ivan Herman, Bob Hopgood 2001

[29] 'http://www.w3.org/TR/REC-rdf-syntax/: Resource Description Framework (RDF) Model and Syntax Specification', 1999.

[30] 'http://www.khoral.com/'. Khoros.

[31] 'Compositing Digital Images'. T. Porter and T. Duff. Computer Graphics, 1984. 18(3): p. 253-259.

[32] 'Texturing and Modeling, A Procedural Approach'. David Ebert, et al (Chapter by Ken Perlin). AP Professional, Cambridge, 1994.