Introduction to Scalable Vector Graphics

Presented by developerWorks, your source for great tutorials

`ibm.com/developerWorks`

Table of Contents

If you're viewing this document online, you can click any of the topics below to link directly to that section.

1. Introduction 2
2. What is SVG? 4
3. Basic shapes 10
4. Definitions and groups 16
5. Painting 21
6. Coordinates and transformations 32
7. Paths 38
8. Text 46
9. Animation and interactivity 51
10. Summary 55

Section 1. Introduction

Should I take this tutorial?

This tutorial assists developers who want to understand the concepts behind Scalable Vector Graphics (SVG) in order to build them, either as static documents, or as dynamically generated content.

XML experience is not required, but a familiarity with at least one tagging language (such as HTML) will be useful. For basic XML information, see the *Introduction to XML* tutorial. JavaScript is used for a single scripting example at the end of the tutorial, but is kept fairly simple.

What is this tutorial about?

Scalable Vector Graphics (SVG) makes it possible to specify, using text, graphical images that appear on the page. For example, where a traditional graphic would need to specify every pixel of a rectangle, an SVG simply states that the rectangle exists, and specifies its size, position, and other properties.

The advantages are many, including the ability to easily generate graphics (such as graphs and charts) from database information, and the ability to add animation and interactivity to graphics.

This tutorial demonstrates the concepts necessary for building SVG documents, such as basic shapes, paths, text, and painting models, and also animation and scripting.

Tools

The tutorial demonstrates the building of several SVG documents. To follow along, you will need an editor and a viewer:

- A plain text editor, such as Notepad, is all you need to create SVG files. Editors specifically geared toward SVG do exist, and are listed in the Resources on page 55 . This tutorial assumes you are using a text editor.
- An SVG viewer: The most up-to-date SVG viewer as of the time of this writing is Adobe's SVGViewer, version 3.0, available free from Adobe at *http://www.adobe.com/svg/viewer/install/main.html*. Other SVG viewers, such as IBM's SVGView, are listed in the Resources on page 55 .
- A browser: The Adobe viewer works with most current browsers, such as Netscape Navigator 4.x (available at *http://home.netscape.com/computing/download/index.html?cp=hophb2* -- Netscape 6 is not supported) and Microsoft Internet Explorer 5.x and above (available at

http://www.microsoft.com/downloads/release.asp?releaseid=32210&area=top&ordinal=1).

To see the examples in action, download the *examples* and uncompress them. Open a new browser window and use it to read the readme.html file.

About the author

Nicholas Chase has been involved in Web site development for companies such as Lucent Technologies, Sun Microsystems, Oracle, and the Tampa Bay Buccaneers. Nick has been a high school physics teacher, a low-level radioactive waste facility manager, an online science fiction magazine editor, a multimedia engineer, and an Oracle instructor. More recently, he was the Chief Technology Officer of Site Dynamics Interactive Communications in Clearwater, Fla., and is the author of three books on Web development, including *Java and XML From Scratch* (Que). He loves to hear from readers and can be reached at *nicholas@nicholaschase.com*.

Section 2. What is SVG?

Vector vs. rasterized graphics

For most of the history of the World Wide Web, graphics displayed by browsers were rasterized. In a rasterized image, such as a GIF or JPEG image, the file contains a color value for each and every pixel in the image. The browser reads these values and acts accordingly. It has knowledge only of the individual parts, and no concept of the whole.

Overall, this system has its strengths, such as the ability to faithfully recreate photographic images, but there are certain situations where it falls short. For example, while browsers can display an image at different sizes, this usually results in jagged edges where it has had to interpolate or guess at values for pixels that don't exist in the original image. Also, the binary nature of rasterized file formats make it difficult (though certainly not impossible) to dynamically create images based on database information, and animation is mostly limited to "flip book" type animations, with individual images displayed in rapid succession.

Vector graphics overcome some of these difficulties by specifying the *instructions* needed to determine the values for each pixel, instead of the values themselves. For example, rather than providing the pixel values for a circle one inch in diameter, a vector graphic instead tells the browser to create a circle one inch in diameter, and lets the browser (or plug-in) do the rest.

This eliminates many of the limitations of rasterized graphics; using vector graphics the browser simply knows that it has to produce a circle. If the image needs to be displayed at three times its normal size, the browser produces a circle of the proper size without having to perform the normal interpolations of a rasterized image. Similarly, the instructions the browser receives are more easily tied into external sources such as applications and databases; to animate the image, the browser simply receives instructions on how to manipulate properties such as the radius or color.

Vector images on the Web

The first vector images on the Web were probably Virtual Reality Markup Language (VRML) images. VRML sought to bring the ease of HTML to image creation, but despite some impressive examples, it was intended for 3D modeling and was so complex that it never really caught on.

Next came Macromedia's entry into the fray, Flash. Flash movies are created using Macromedia's Flash application, which allows fairly complex animations to be built, and tied to sound and interactivity. Because Flash files primarily contain instructions on how to create images, they are much smaller than traditional Web movies (for example, QuickTime movies) -- plus they can be scaled.

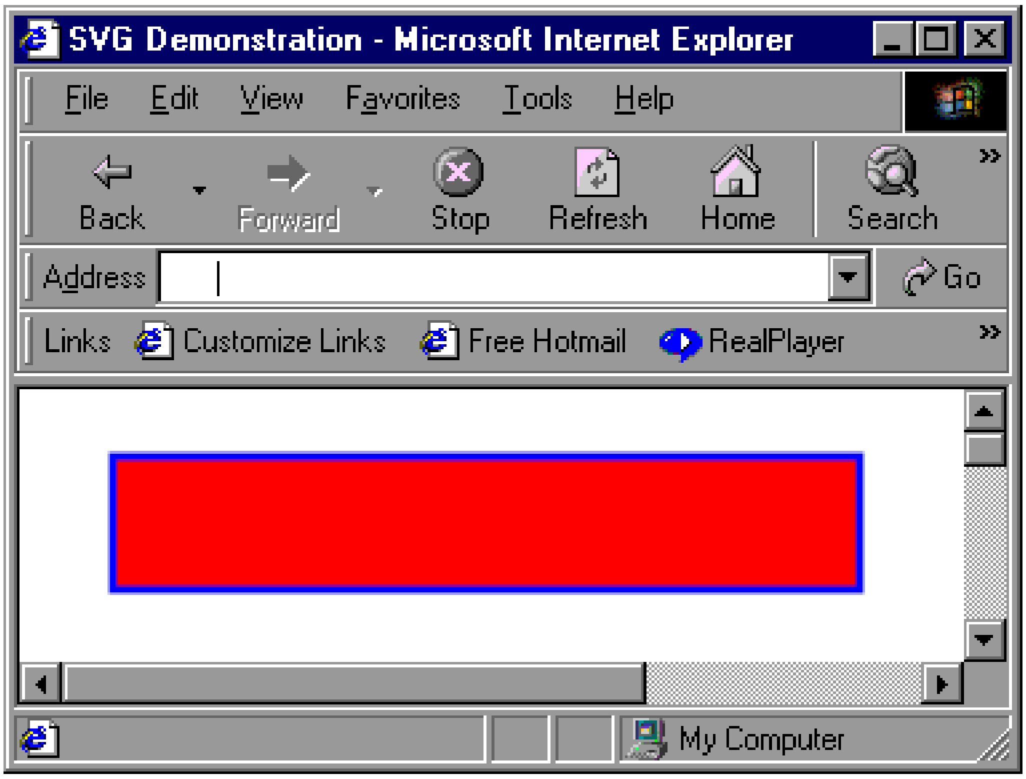
However, Flash files are still binary files, which makes it difficult (though not impossible) to create them dynamically. There are also limitations to the scripting that can be done from a browser.

Defining images using text

Scalable Vector Graphics solve many of these problems by defining images, animations, and interactivity using XML. These text-based instructions are read by the browser (or more specifically, by a plug-in to the browser), which then carries out the instructions. For example, a simple SVG image of a rectangle might look like the following:

```
<?xml version="1.0" standalone="no" ?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN"
 "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">

<svg width="300" height="100" xmlns="http://www.w3.org/2000/svg">

    <rect x="25" y="10" width="280" height="50"
          fill="red" stroke="blue" stroke-width="3"/>

</svg>
```

The document instructs the browser to create a rectangle, and provides property information such as position (x, y), size (`height`, `width`), colors (`fill`, `stroke`), and line width (`stroke-width`). (The overall document is discussed in The basic SVG document on page 10 .)


SVG Demonstration - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search
Address
Go
Links Customize Links Free Hotmail RealPlayer
My Computer

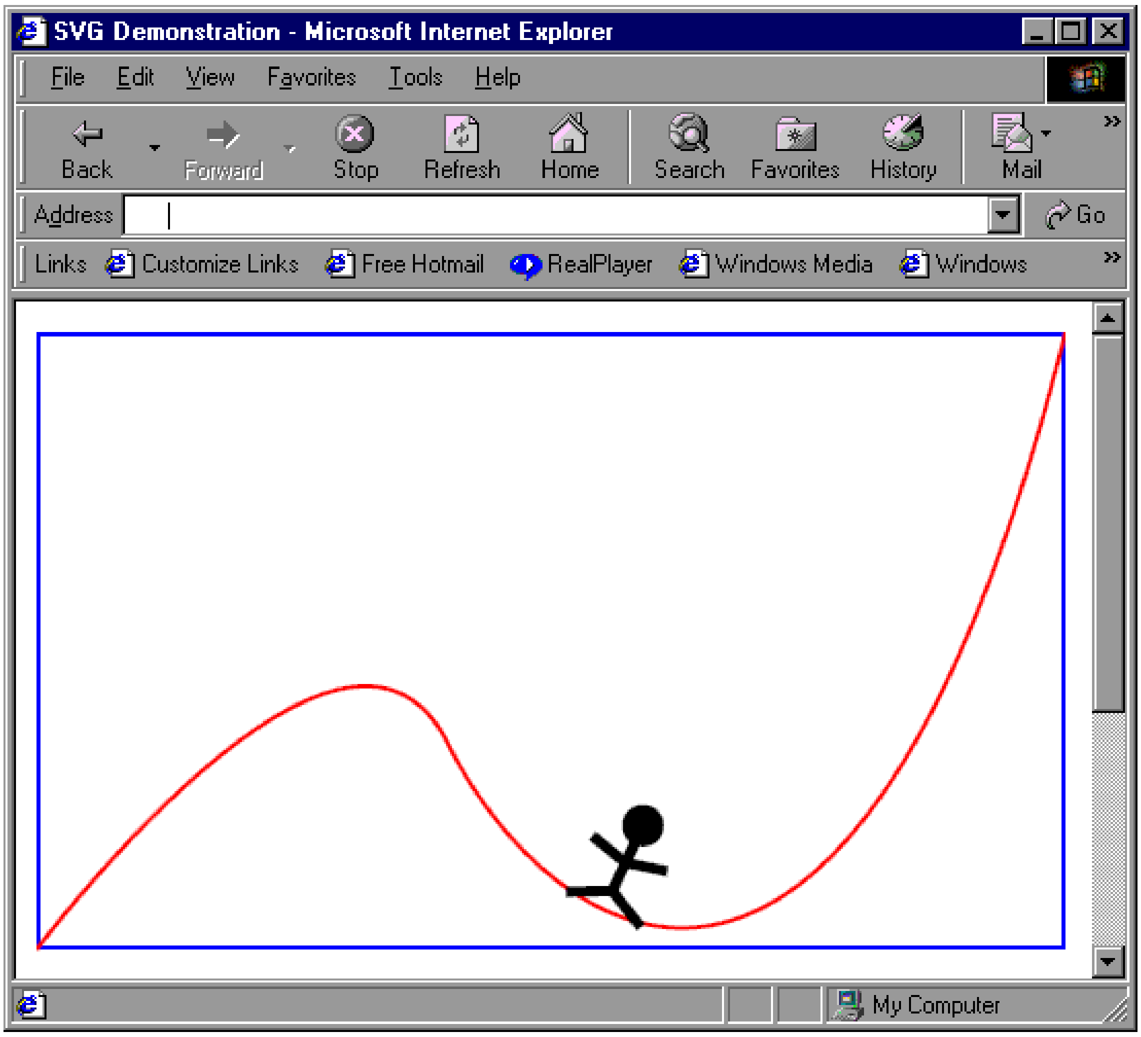
Animation and interactivity

Because of this structure, SVG is well suited to animation and interactivity. To change the size, position, or color of a graphic element, a script simply adjusts the relevant property.

In fact, SVG has properties specifically designed for event handling (much like HTML), and even elements specifically geared toward animation. For example, this document creates a stick figure that traverses a specific path over a period of 8 seconds, repeated indefinitely:

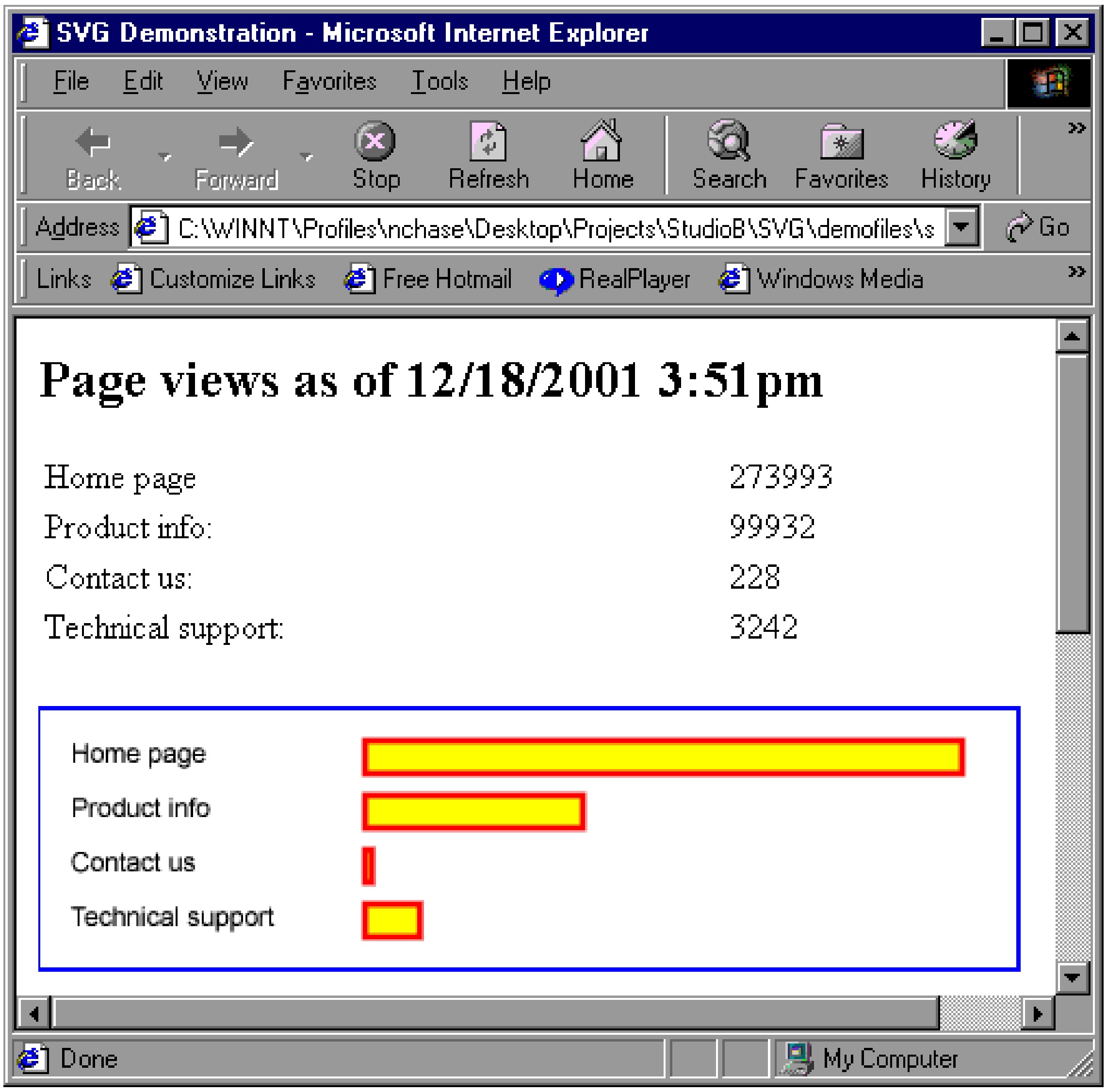
```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="500" height="300" xmlns="http://www.w3.org/2000/svg">

  <!-- Box around the image -->
  <rect x="1" y="1" width="498" height="298"
        fill="none" stroke="blue" stroke-width="2" />

  <!-- Visible path -->
  <path d="M0,300 S150,100 200,200 S400,400 500,0"
        fill="none" stroke="red" stroke-width="2"  />

  <!-- Group of elements to animate -->
  <g stroke-width="5" stroke="black">

    <!-- Stick figure pieces -->
    <circle cx="0" cy="-45" r="10" fill="black"/>
    <line x1="-20" y1="-30" x2="0" y2="-25"/>
    <line x1="20" y1="-30" x2="0" y2="-25"/>
    <line x1="-20" y1="0" x2="0" y2="-10"/>
    <line x1="20" y1="0" x2="0" y2="-10"/>
    <line x1="0" y1="-10" x2="0" y2="-45"/>

    <!-- Animation controls -->
    <animateMotion path="M0,300 S150,100 200,200 S400,400 500,0"
                   dur="8s" repeatCount="indefinite"
                   rotate="auto" />
  </g>
</svg>
```

Because this is all based in instructions, even a non-artist (such as the author of this tutorial) can create basic graphics and animations, if not actual art.


SVG Demonstration - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Mail
Address
Go
Links Customize Links Free Hotmail RealPlayer Windows Media Windows
My Computer

Dynamically created graphics

The text-based nature of SVG also allows for easy creation of graphics "on the fly," because generating them is simply a matter of outputting the proper values to the page. For example, just as a Java servlet, ASP page, or CGI script can output values from a database, it can output a chart or graph. The image below shows an HTML page with both the plain data in an HTML table, and an SVG version:


SVG Demonstration - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History
Address C:\WINNT\Profiles\nchase\Desktop\Projects\StudioB\SVG\demofiles\s
Go
Links Customize Links Free Hotmail RealPlayer Windows Media
Done
My Computer

Page views as of 12/18/2001 3:51pm

Home page	273993
Product info:	99932
Contact us:	228
Technical support:	3242

Home page
Product info
Contact us
Technical support

The server-side application that creates the SVG is the same as that which created the HTML table, and uses the same data (scaled by a factor of 1000 to make it fit on the page):

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="500" height="300" xmlns="http://www.w3.org/2000/svg">

  <!-- Box around the image -->
  <rect x="0" y="25" width="450" height="120"
        fill="none" stroke="blue" stroke-width="2" />

  <!-- Headers -->
  <text x="15" y="50">Home page</text>
  <text x="15" y="75">Product info</text>
  <text x="15" y="100">Contact us</text>
  <text x="15" y="125">Technical support</text>

  <!-- Chart elements -->
  <g fill="yellow" stroke="red" stroke-width="3">
```

```
      <rect x="150" y="40" height="15" width="273.993"/>
      <rect x="150" y="65" height="15" width="99.932"/>
      <rect x="150" y="90" height="15" width="3.228"/>
      <rect x="150" y="115" height="15" width="25.242"/>
  </g>

</svg>
```

This is just one example of SVG's capabilities. SVG makes creating dynamic images as simple as creating dynamic HTML. Also, because SVG is based in XML, it can be easily created using Extensible Stylesheet Language (XSL) transformations.

Section 3. Basic shapes

The basic SVG document

An SVG document is, at its core, an XML document. That means that SVG documents have certain basic attributes:

- All tags must have a start and end tag, or must be noted as an empty tag. Empty tags are closed with a backslash, as in `<rect />`.
- Tags must be nested properly. If a tag is opened within another tag, it must be closed within that same tag. For example, <g><text>Hello there!</text></g> is correct, but <g><text>Hello there!</g></text> is not.
- The document must have a single root. Just as a single <html></html> element contains all content for an HTML page, a single <svg></svg> element contains all content for an SVG document.
- The document should start with the XML declaration, `<?xml version="1.0"?>`.
- The document should contain a DOCTYPE declaration, which points to a list of allowed elements. The DOCTYPE declaration for an SVG 1.0 document is:

```
<?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="200" height="200" xmlns="http://www.w3.org/2000/svg">

  <desc>All SVG documents should have a description</desc>

  <defs>
      <!-- Items can be defined for later use -->
  </defs>

  <g>
      <circle cx="100" cy="100" r="75" fill="green"/>
  </g>
</svg>
```

Including SVG on an HTML page

SVG documents are useful in and of themselves, but at this point in their development they are most useful when added to a Web page. Adding them to a Web page also makes it easier to display them in a browser; depending on how the user's system has file name associations set up, some browsers will refuse to open a *.svg file, but will have no problems displaying the SVG image as part of a Web page.

Adding the SVG image to an HTML page is straightforward: Simply add an `<object></object>` element with the appropriate attributes:

```
<html>
  <head><title>SVG Demonstration</title></head>
  <body>
  <h2 style="text-align: center">SVG Demonstration</h2>

  <p>A page may have other code besides the SVG image.</p>

  <object type="image/svg+xml" data="MySVG.svg"
                             width="300" height="200">
     <img src="NonSVG.gif" alt="Static version of SVG image" />
  </object>

  <p>Using objects allows the browser to decide what to display.</p>

  </body>
</html>
```

Pay particular attention to the height and width attributes on the `<object></object>` tag. If they are not specified, some browsers will not display the image properly. Also, the browser takes these values into account when performing certain calculations (most notably Scaling with viewBox on page 35), so if they are not specified correctly (such as simply using large values to display whatever may be present) they may interfere with proper display of the image.

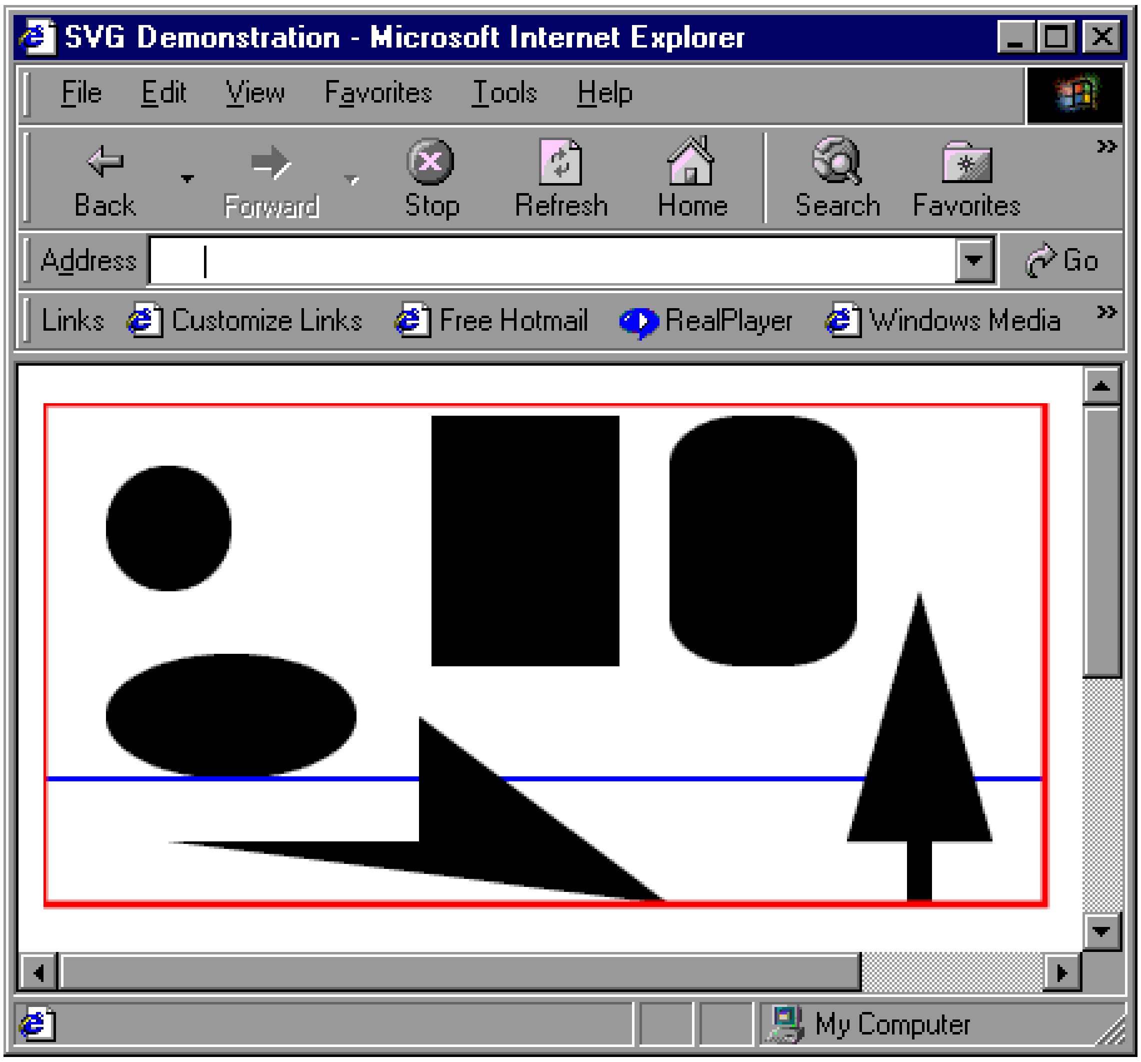
Basic SVG shapes

SVG defines six basic shapes that, along with paths (discussed in What is a path? on page 38), may be combined to form any possible image. Each of these shapes carries properties that specify its position and size. Their colors and outlines are determined by their `fill` and `stroke` properties, respectively. These shapes are:

- `circle:` Displays a perfect circle of the specified radius, with the center at the specified point.
- `ellipse:` Displays an ellipse with the center at the specified point and the major and minor radii as specified.
- `rect:` Displays rectangles (including squares) with the upper-left corner at the point specified, and the height and width as specified. Rectangles can also be drawn with rounded corners by specifying the x and y radii for the corner circles.
- `line:` Displays a line between two coordinates.
- `polyline:` Displays a series of lines with vertices at the specified points.
- `polygon:` Similar to `polyline`, but adds a line from the last point back to the first, creating a closed shape.

The following example demonstrates these shapes:

```
<?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="400" height="200" xmlns="http://www.w3.org/2000/svg">

  <desc>Basic shapes</desc>

  <g>
    <circle cx="50" cy="50" r="25" />
    <ellipse cx="75" cy="125" rx="50" ry="25" />

    <rect x="155" y="5" width="75" height="100"/>
    <rect x="250" y="5" width="75" height="100" rx="30" ry="20" />

    <line x1="0" y1="150" x2="400" y2="150"
                     stroke-width="2" stroke="blue"/>

    <polyline points="50,175 150,175 150,125 250,200" />
    <polygon points="350,75 379,175 355,175 355,200 345,200
                                  345,175 321,175" />

    <rect x="0" y="0" width="400" height="200"
            fill="none" stroke="red" stroke-width="3" />
  </g>
</svg>
```

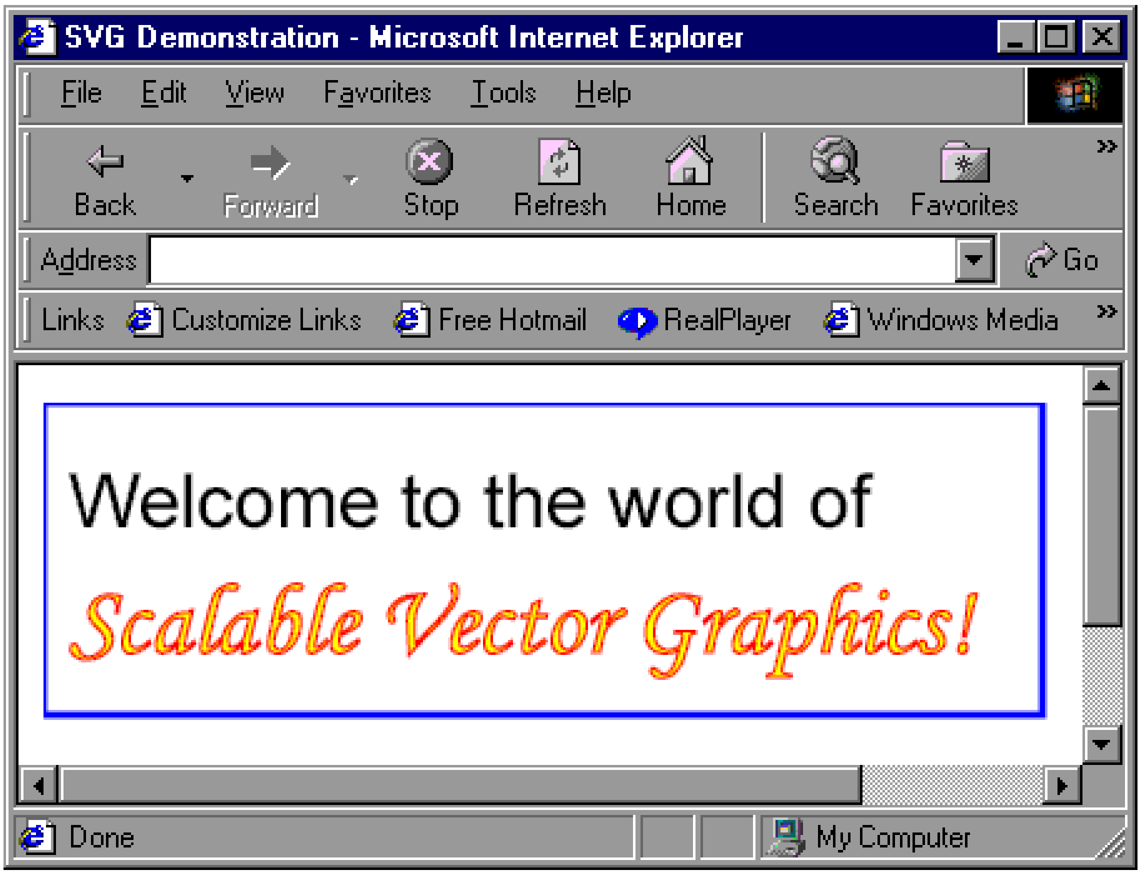

SVG Demonstration - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites
Address
Go
Links Customize Links Free Hotmail RealPlayer Windows Media
My Computer

Adding text

In addition to shapes, SVG images may also contain text. SVG gives designers and developers a great deal of control over text, allowing for significant graphic effects without resorting to an image that loses the actual textual information, as often happens with a `*.gif` or `*.jpg` image, or even a Flash movie.

SVG's text and font capabilities are discussed in the Text section starting with Adding text on page 46 , but for now it's important to understand that all of the effects available via Cascading Style Sheets in a traditional HTML page are also available in text elements within an SVG image. For 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="400" height="125" xmlns="http://www.w3.org/2000/svg">

  <desc>Basic text</desc>

  <g>
    <rect x="0" y="0" width="400" height="125" fill="none"
            stroke="blue" stroke-width="3"/>

    <text x="10" y="50" font-size="30">Welcome to the world of</text>
    <text x="10" y="100" font-size="40"
       font-family="Monotype Corsiva"
       fill="yellow" stroke="red">Scalable Vector Graphics!</text>
  </g>
</svg>
```


SVG Demonstration - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites
Address Go
Links Customize Links Free Hotmail RealPlayer Windows Media
Done My Computer

Welcome to the world of

Scalable Vector Graphics!

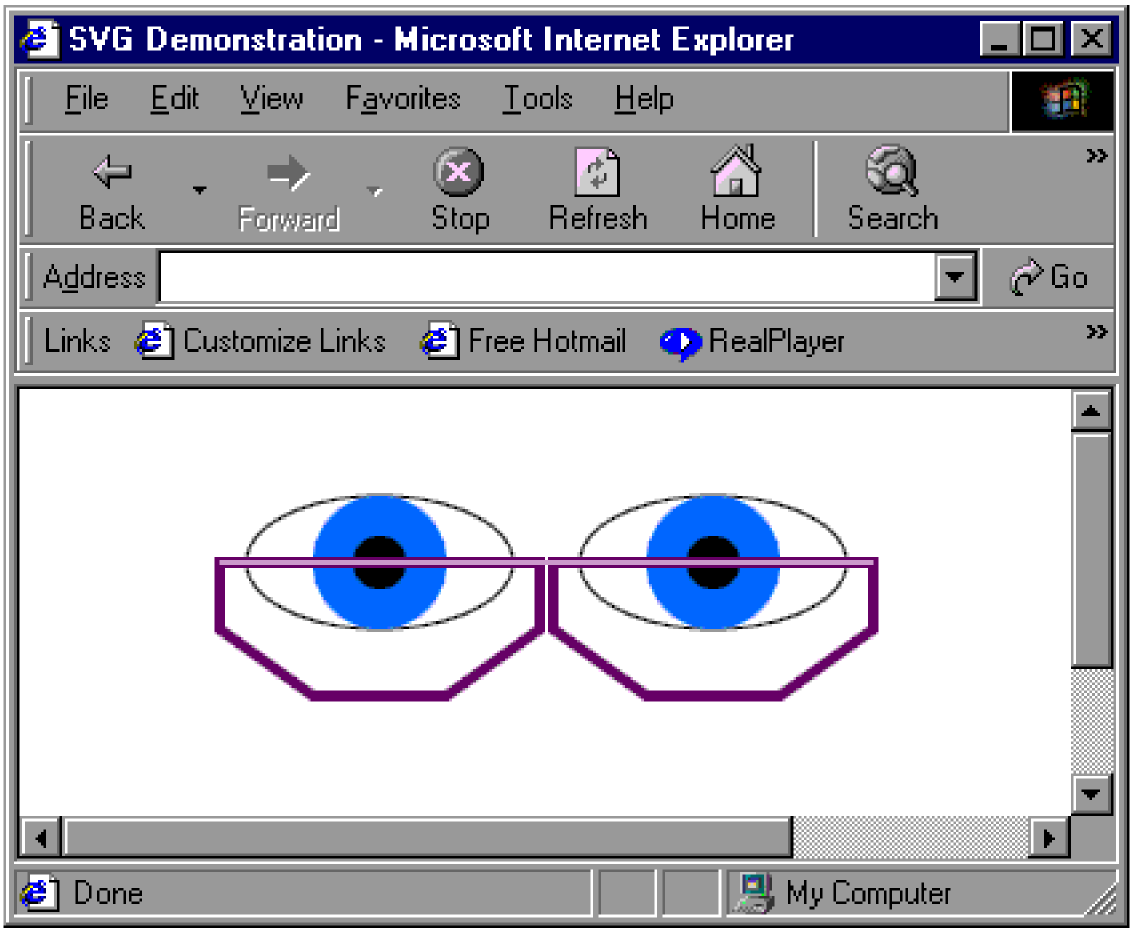
Rendering order

When compositing a number of different elements, as is the case with an SVG image, it's important to keep in mind the order in which items are laid down on the page, because this affects which ones appear "on top." On an HTML page, this layering effect is controlled using the `z-index` property, but with an SVG image, items are laid down in strict order. Each successive layer is placed "on top of" those that have already been laid down.

If an element has been specified as having no fill (using `fill="none"`), the items below it show through, as seen here:

```
<?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="400" height="200" xmlns="http://www.w3.org/2000/svg">

  <desc>Overlapping shapes</desc>

  <g>
    <ellipse cx="125" cy="50" rx="50" ry="25"
                           fill="none" stroke="black" />
    <circle cx="125" cy="50" r="25" fill="dodgerblue" />
    <circle cx="125" cy="50" r="10" fill="black" />

    <ellipse cx="250" cy="50" rx="50" ry="25"
                           fill="none" stroke="black" />
    <circle cx="250" cy="50" r="25" fill="dodgerblue" />
    <circle cx="250" cy="50" r="10" fill="black" />

    <polygon points="65,50 185,50 185,75, 150,100
                                          100,100 65,75"
             fill="none" stroke="purple" stroke-width="4"/>
    <polygon points="190,50 310,50 310,75, 275,100
                                          225,100 190,75"
             fill="none" stroke="purple" stroke-width="4"/>

    <line x1="65" y1="50" x2="310" y2="50"
                          stroke="plum" stroke-width="2"/>

  </g>
</svg>
```

Notice that each element overlaps those that came before it.


SVG Demonstration - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search
Address
Go
Links Customize Links Free Hotmail RealPlayer
Done
My Computer

Section 4. Definitions and groups

Defining reusable parts

Often in the building of an SVG image, pieces are either re-used or inconvenient to define within the body of an image. In such cases, it is often convenient to create these sections within the definition section of the document (as part of the `<defs></defs>` element) by assigning them an identifier that can then be called within the body of the image.

For example, the image shown in the previous panel shows two eyes, each rimmed by a reading-class lens. Rather than creating the lens twice, the document can define one lens in the definitions section, and call it twice within the document as the next panel shows. Similarly, the eyes themselves can contain a gradient, which should also be defined for later reference. (Gradients are covered more fully in Gradients on page 25 .)

```
<?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="400" height="200" xmlns="http://www.w3.org/2000/svg">

  <desc>Referenced items</desc>
  <defs>

    <polygon id="lens" points="65,50 185,50 185,75, 150,100
                                  100,100 65,75"
             fill="none" stroke="purple" stroke-width="4"/>

     <radialGradient id="irisGradient">
        <stop offset="25%" stop-color="green" />
        <stop offset="100%" stop-color="dodgerblue" />
     </radialGradient>

  </defs>

  <g>
. . .
```

Using defined items as attributes

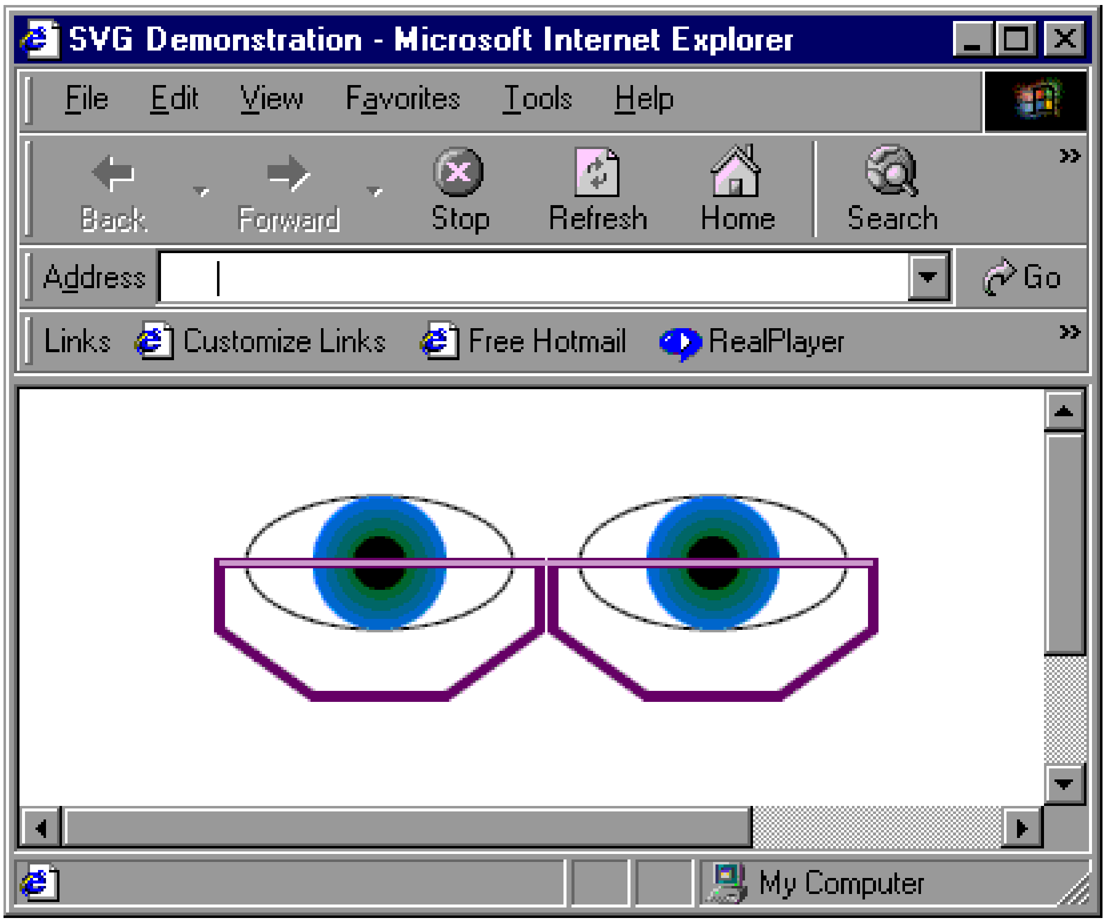
The actual use of a pre-defined item, such as the polygon and gradients defined in the previous panel, generally takes two forms.

In both cases, the defined item is referred to via its local URL, or URI. As with HTML pages, the `id` attribute creates a reference point within the document. This means that, for example, the URI `#irisGradient` refers to the section of the document identified as `irisGradient`, or the gradient definition. This means that it can be referenced from within the `fill` attribute of the circle element:

```
<circle cx="125" cy="50" r="25" fill="url(#irisGradient)"/>
```

Note the use of the `url()` function.

The final code is shown in the last panel of this section. Note that a gradient is now applied to the irises of the eyes:


SVG Demonstration - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search
Address
Go
Links Customize Links Free Hotmail RealPlayer
My Computer

Using defined items as elements

The second means for referencing pre-defined items is by linking them into the document with the `<use/>` element. For example:

```
<use xlink:href="#lens" />
```

places the polygon on the page using the coordinates supplied within the definition.

Two important items should be emphasized here. First, note the use of the `xlink` namespace. While most viewers will display this item properly without it, for conformance the `xlink` namespace should be defined on the `<svg></svg>` element, as shown on the next panel.

Second, note that when used in this way the `<use/>`; element becomes a container that can have its own coordinate system. Coordinate systems are covered in the

Coordinate systems and the initial viewport on page 32 section, but to look at a concrete example, the second lens was originally created with an initial coordinate of (190, 50) for an offset of 125 pixels from the first one. The element

```
<use xlink:href="#lens" x="125"/>
```

creates the second lens in its original position because its "container" is offset by 125 pixels.

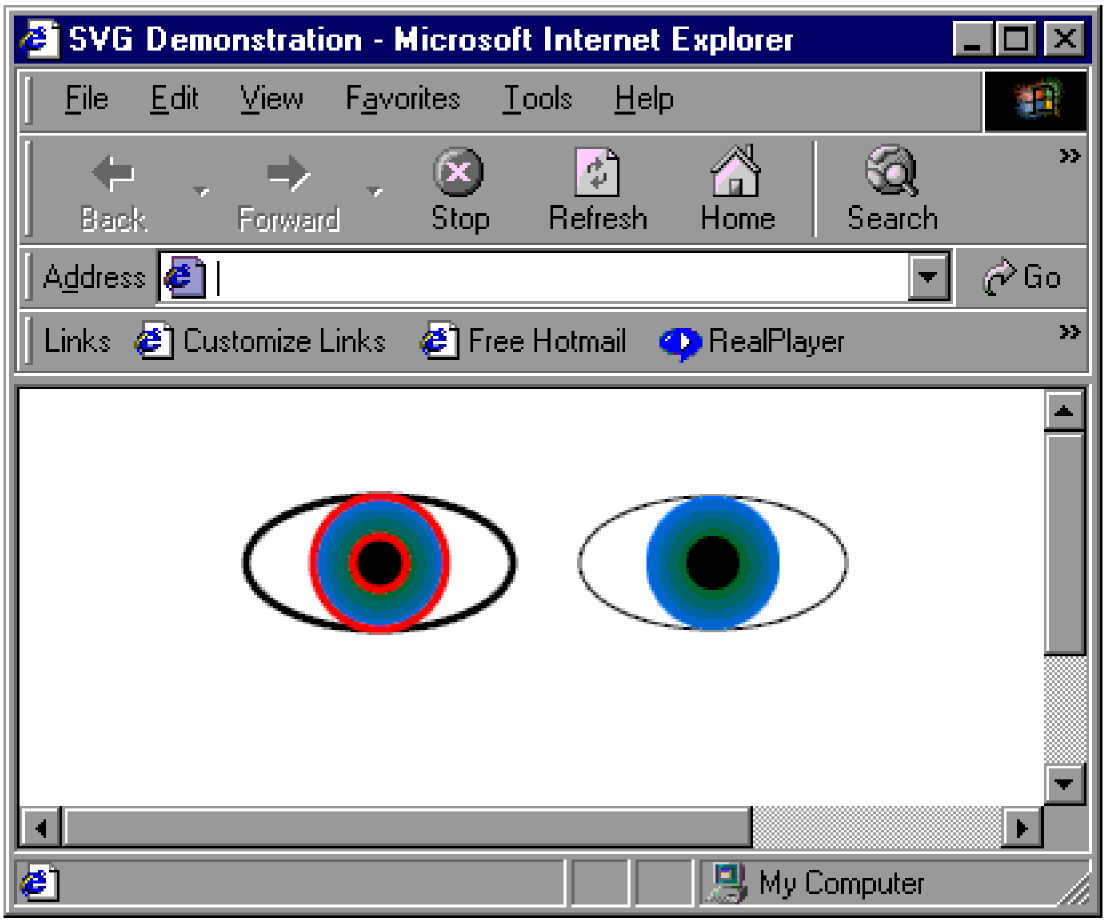
Grouping elements

Finally, it is not simply single elements that can be defined, as might be guessed from the `<radialGradient></radialGradient>` element discussed earlier.

For both readability and convenience, it is often a good idea to group elements together. For this purpose, SVG provides the <g></g> element, which creates a container for the elements within. This container can be used to identify the elements, or to provide a common attribute which will be overridden by a locally defined attribute. For example the code

```
. . .
    <g stroke="red" stroke-width="3">
      <ellipse cx="125" cy="50" rx="50" ry="25"
                           fill="none" stroke="black" />
      <circle cx="125" cy="50" r="25" fill="url(#irisGradient)" />
      <circle cx="125" cy="50" r="10" fill="black" />
    </g>
. . .
```

creates an eye where all strokes are 3 pixels wide (because none of the elements defines a stroke-width) but where all but the outside stroke are red (because the ellipse defines the stroke color).

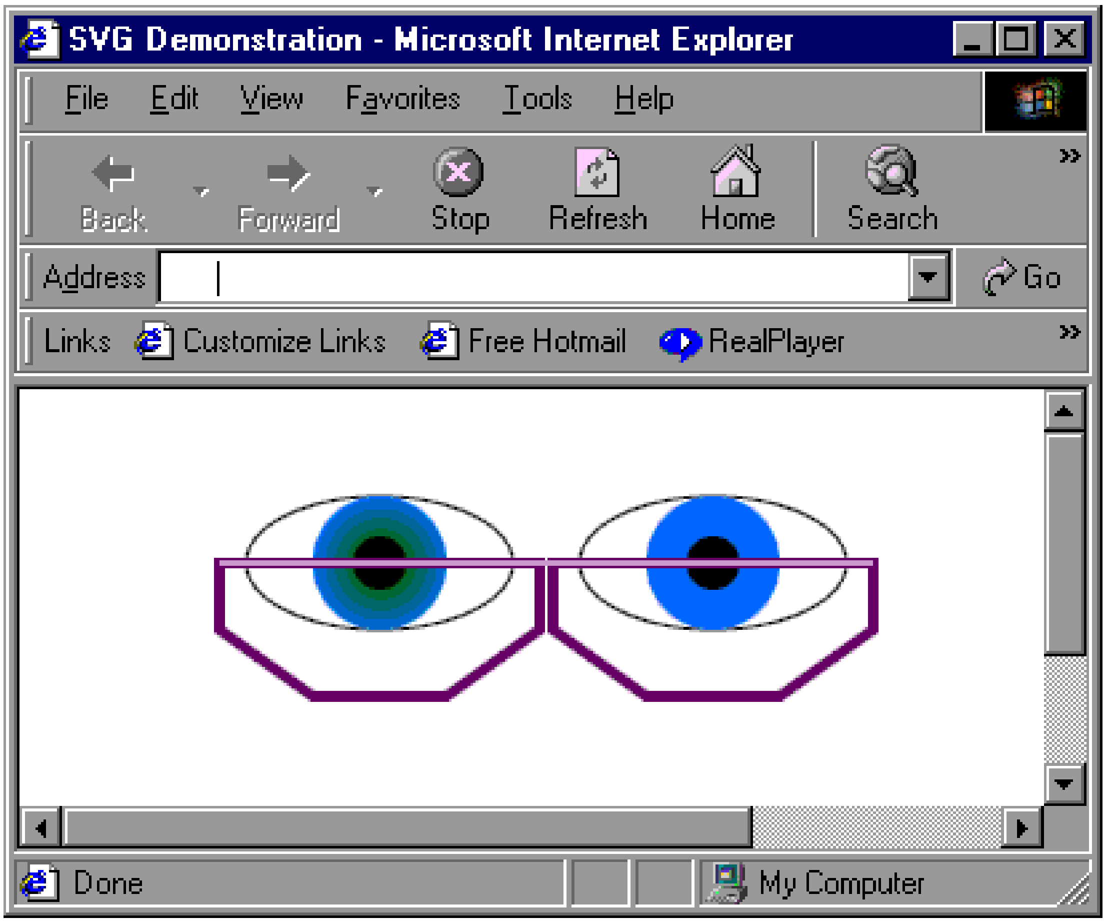

SVG Demonstration - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search
Address
Go
Links Customize Links Free Hotmail RealPlayer
My Computer

Putting it all together

The final document shows how each piece is added:

```
<?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="400" height="200" xmlns="http://www.w3.org/2000/svg"
                       xmlns:xlink="http://www.w3.org/1999/xlink">

  <desc>Reusing items</desc>
  <defs>

     <polygon id="lens" points="65,50 185,50 185,75, 150,100
                                     100,100 65,75"
             fill="none" stroke="purple" stroke-width="4"/>

     <radialGradient id="irisGradient">
        <stop offset="25%" stop-color="green" />
        <stop offset="100%" stop-color="dodgerblue" />
     </radialGradient>

     <g id="eye">
        <ellipse cy="50" rx="50" ry="25"
                             fill="none" stroke="black"/>
        <circle cy="50" r="25"/>
        <circle cy="50" r="10" fill="black"/>
     </g>

  </defs>

  <g>

    <use xlink:href="#eye" x="125" fill="url(#irisGradient)"/>
    <use xlink:href="#eye" x="250" fill="dodgerblue"/>

    <use xlink:href="#lens"/>
    <use xlink:href="#lens" x="125"/>

    <line x1="65" y1="50" x2="310" y2="50"
                          stroke="plum" stroke-width="2"/>

  </g>
</svg>
```

Note that reusable elements also allow for different attribute values for each use, as shown above with the fill on the irises.


SVG Demonstration - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search
Address
Go
Links Customize Links Free Hotmail RealPlayer
Done
My Computer

Section 5. Painting

Stroke and fill

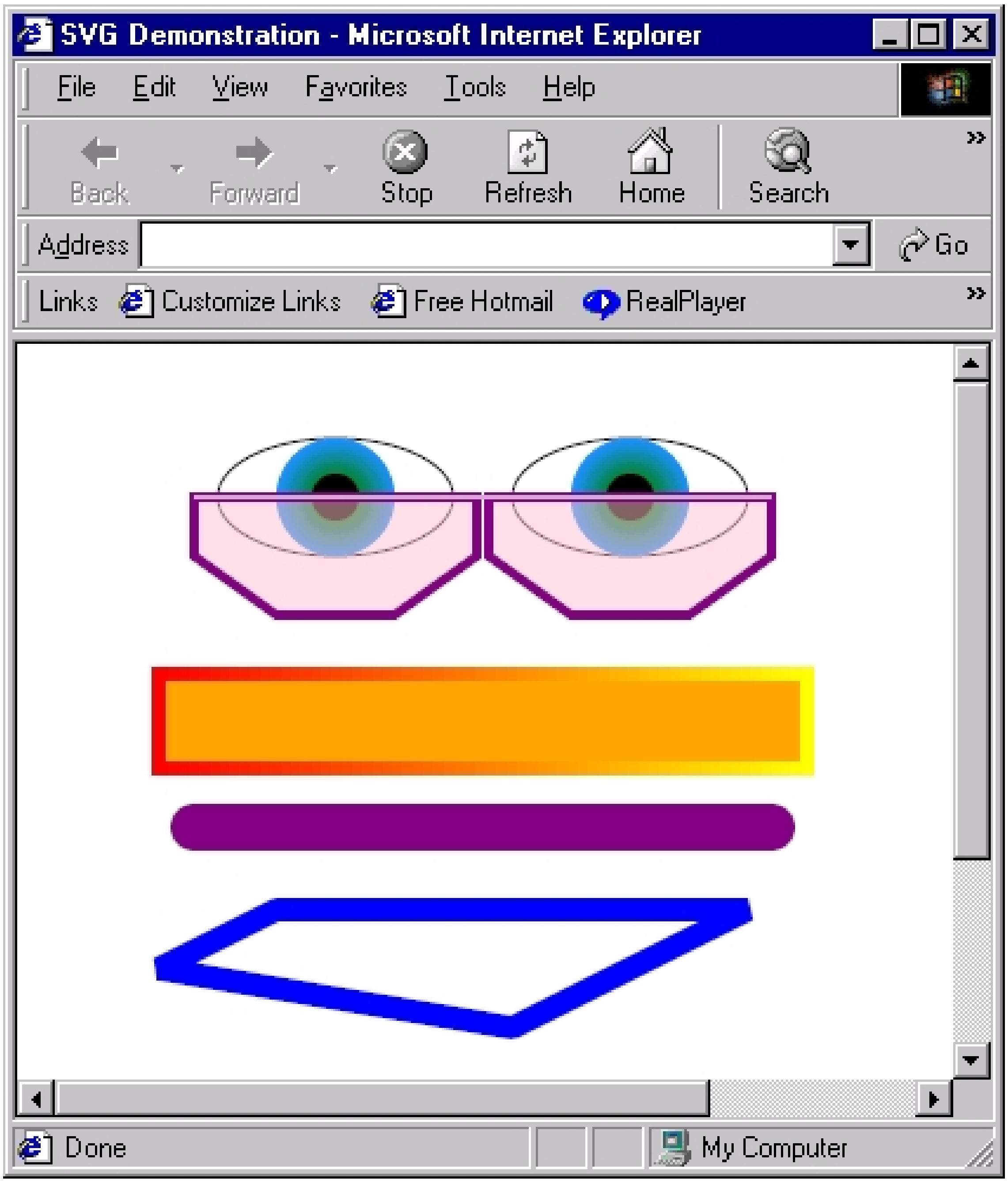
Throughout the tutorial so far, examples have shown the stroke, or line, around an object, and the fill -- the area within the object. These properties actually have sub-properties that can also be set to create various effects. These properties include:

- `fill`: This property specifies the paint used to fill the inside of an object. In most cases this will simply be a color, but it can also be a gradient or a pattern (which is covered in Patterns on page 27). The value is typically a keyword, color specification, or URI pointing to a predefined element.
- `fill-opacity`: This property specifies the transparency of an element. Values range from completely transparent (0) to completely opaque (1).
- `stroke`: This property specifies the appearance of the line that borders the outside of an element. Like fill, it refers to a paint, although it is normally specified as a simple color.
- `stroke-width`: This property specifies the width of the stroke line.
- `stroke-linecap`: This property, which may take the values butt (the default), round, and square, determines the shape of the end of the line.
- `stroke-linejoin`: This property determines the appearance of the corners of an object. Allowable values are miter (the default), round, and bevel, which "clips" the edges off of acute angles as seen in the example.
- `stroke-dasharray`: This property is a series of integers (such as 3,2,3,2,4,2,3,2,3) that allows control over the relative lengths of dashes in a dashed line.
- `stroke-opacity`: Similar to fill-opacity, this property determines the relative transparency of an element's stroke line.

Some examples of these properties can be seen below:

```
<?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="400" height="200" xmlns="http://www.w3.org/2000/svg"
                      xmlns:xlink="http://www.w3.org/1999/xlink">

  <desc>Stroke and fill</desc>
  <defs>

      <linearGradient id="lineGradient">
         <stop offset="0%" stop-color="red" />
         <stop offset="100%" stop-color="yellow" />
      </linearGradient>

      <polygon id="lens" points="65,50 185,50 185,75, 150,100
                                         100,100 65,75"
              fill="pink" stroke="purple" stroke-width="4"
              fill-opacity=".5"/>

. . .
```

```
  </defs>

  <g>

. . .

    <line x1="65" y1="50" x2="310" y2="50"
                          stroke="plum" stroke-width="2"/>

    <!-- Box with gradient along the outside -->
    <rect x="50" y="125" width="275" height="40" fill="orange"
           stroke-width="6" stroke="url(#lineGradient)" />

    <!-- Purple line with rounded edges -->
    <line x1="65" y1="190" x2="310" y2="190"
                          stroke="purple" stroke-width="20"
                          stroke-linecap="round"/>

    <!-- Blue polygon with beveled corners -->
    <polygon points="50,250 100,225 300,225 200,275" stroke="blue"
          fill="none" stroke-width="10" stroke-linejoin="bevel" />

  </g>
</svg>
```

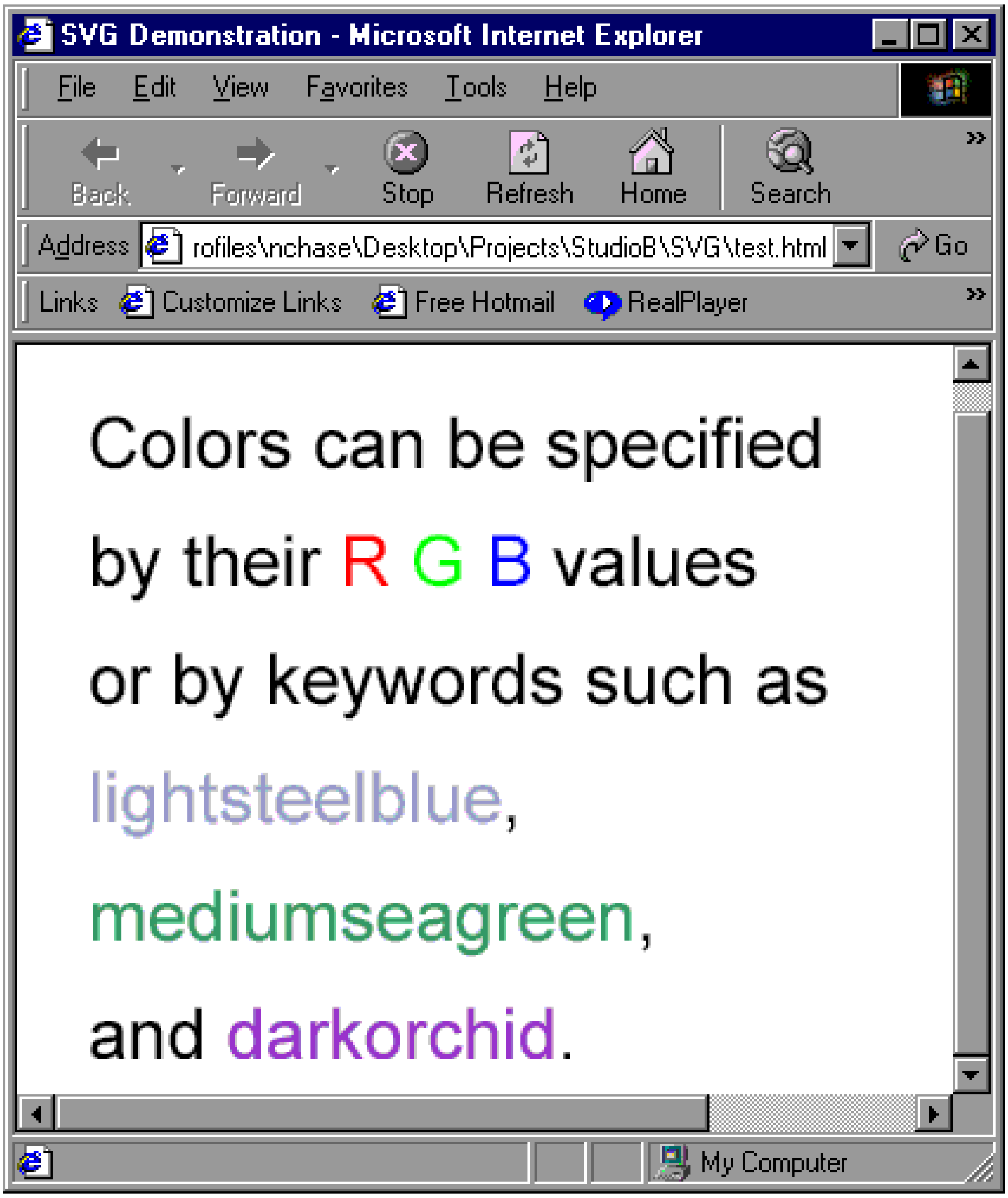

SVG Demonstration - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search
Address
Go
Links Customize Links Free Hotmail RealPlayer
Done
My Computer

Colors

Color is crucial to an SVG image. Individual colors may be specified directly using their RGB values, or indirectly using one of almost 150 color keywords that refer back to RGB values.

RGB values specify the relative intensity of a color's red, green, and blue components on a scale of 0 to 255. For 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="400" height="200" xmlns="http://www.w3.org/2000/svg">

  <desc>Colors</desc>
  <defs>
  </defs>

  <g>

     <text x="20" y="50" font-size="30">Colors can be specified</text>
     <text x="20" y="100" font-size="30">by their
        <tspan fill="rgb(255,0,0)">R</tspan>
        <tspan fill="rgb(0,255,0)">G</tspan>
        <tspan fill="rgb(0,0,255)">B</tspan>
     values</text>
     <text x="20" y="150" font-size="30">or by keywords such as</text>
     <text x="20" y="200" font-size="30">
        <tspan fill="lightsteelblue">lightsteelblue</tspan>,
     </text>
     <text x="20" y="250" font-size="30">
        <tspan fill="mediumseagreen">mediumseagreen</tspan>,
     </text>
     <text x="20" y="300" font-size="30">and
        <tspan fill="darkorchid">darkorchid</tspan>.
     </text>

  </g>
</svg>
```


Colors can be specified by their R G B values or by keywords such as lightsteelblue, mediumseagreen, and darkorchid.

SVG Demonstration - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search
Address rofiles\nchase\Desktop\Projects\StudioB\SVG\test.html Go
Links Customize Links Free Hotmail RealPlayer
My Computer

A complete list of color keywords can be found as part of the SVG recommendation at *http://www.w3.org/TR/SVG/types.html#ColorKeywords*.

Gradients

Gradients, as seen in previous examples, provide the ability to blend colors together. They come in two varieties. In each case, the code specifies color "stops," or points along the gradient vector at which the gradient reaches a certain color. For example, a gradient that specifies a red stop at 0%, a white stop at 50%, and a blue stop at 100%

will gradually change from red to white to blue, with white in the center of the gradient vector.

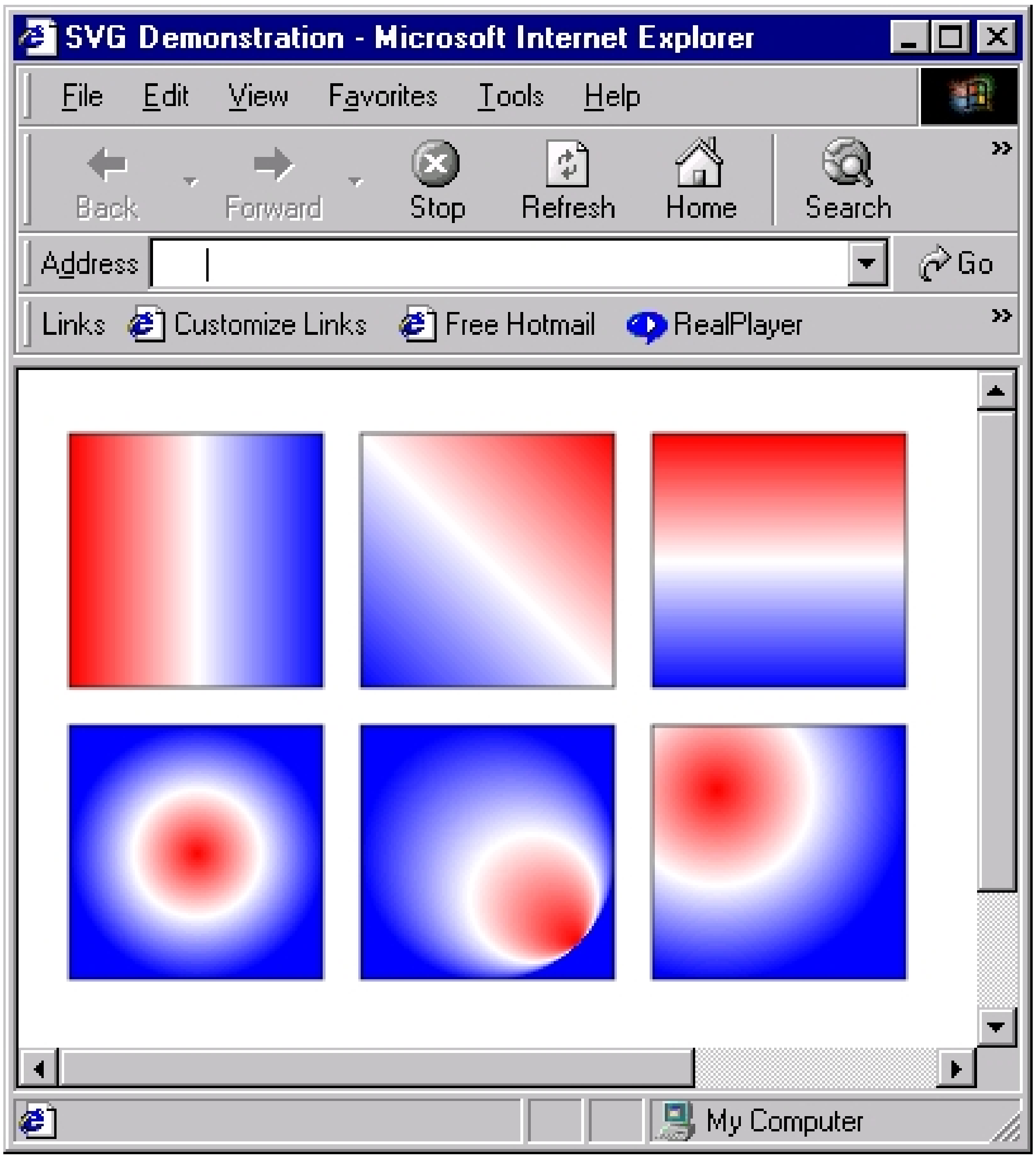
The gradient vector can be inferred, or it can be specified directly. In the case of a linear gradient, it is assumed to start at the left edge of the area to be filled and end at the right edge. This vector can be changed using the x1, y1, x2, and y2 attributes. The vector can also be transformed (according to Transformations on page 33) using the `gradientTransform` attribute.

In the case of radial gradients, which are based on a circle, the center and radius of the outer circle (at which the gradient vector ends) can be adjusted using the `cx`, `cy` and `r` attributes. The focal point (at which the gradient vector starts) can be adjusted using the `fx` and `fy` attributes.

Consider these examples of linear and radial gradients:

```
<?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="400" height="200" xmlns="http://www.w3.org/2000/svg">

  <desc>Colors</desc>
  <defs>
     <linearGradient id="linear1">
                <stop offset="0%" stop-color="red"/>
                <stop offset="50%" stop-color="white"/>
                <stop offset="100%" stop-color="blue"/>
     </linearGradient>
     <linearGradient id="linear2" x1="100%" y1="0%" x2="0%"
                                      y2="100%">
. . .
     </linearGradient>
     <linearGradient id="linear3" gradientTransform="rotate(90)">
. . .
     </linearGradient>
     <radialGradient id="radial1">
. . .
     </radialGradient>
     <radialGradient id="radial2" fx="225" fy="225">

. . .
     </radialGradient>
     <radialGradient id="radial3" cx="25%" cy="25%" r="75%">
. . .
     </radialGradient>

  </defs>

  <g>

     <!-- First row -->
     <rect x="10" y="10" height="100" width="100" stroke="black"
                                fill="url(#linear1)"/>
     <rect x="125" y="10" height="100" width="100" stroke="black"
                                fill="url(#linear2)"/>
     <rect x="240" y="10" height="100" width="100" stroke="black"
                                fill="url(#linear3)"/>

     <!-- Second row -->
     <rect x="10" y="125" height="100" width="100" stroke="black"
```

```
                                fill="url(#radial1)"/>
     <rect x="125" y="125" height="100" width="100" stroke="black"
                                fill="url(#radial2)"/>
     <rect x="240" y="125" height="100" width="100" stroke="black"
                                fill="url(#radial3)"/>

  </g>
</svg>
```


SVG Demonstration - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search
Address
Go
Links Customize Links Free Hotmail RealPlayer
My Computer

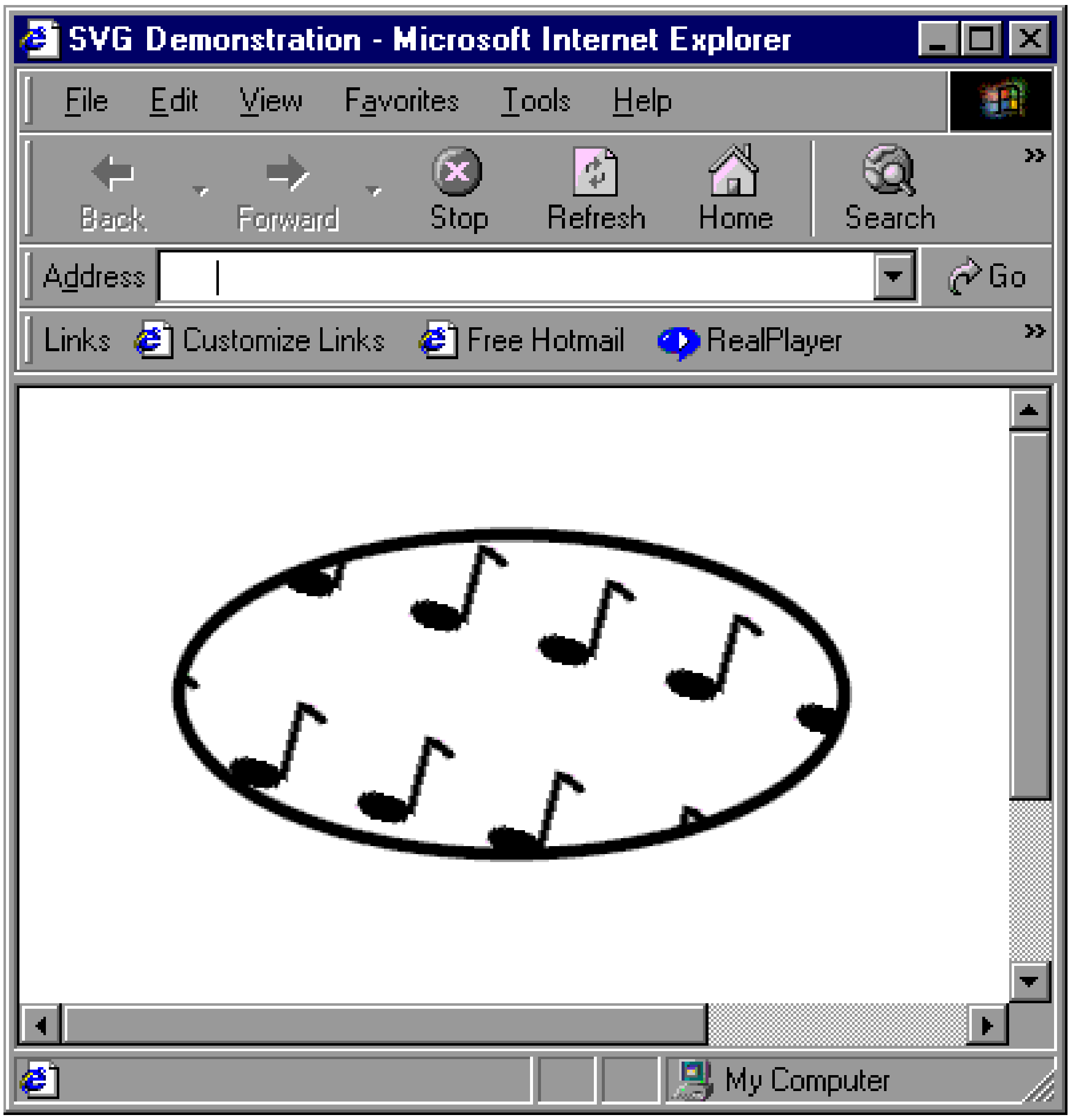
Patterns

Filling an object with a pattern is, in many ways, similar to filling it with a gradient. In both cases, the fill is defined and then called from within the fill attribute.

Defining a pattern is similar to defining any other object that appears as part of an SVG image. It has a position, a height and width, and, typically, one or more included objects. Whether the position is based on the overall document or on the object being filled is determined by the `patternUnits` attribute, which may be set to `objectBoundingBox` or `userSpaceOnUse`; these attributes set the coordinates based on the object and the document, respectively. Similar to gradients, patterns may be transformed (using the `patternTransform` attribute).

Consider the following 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="400" height="200" xmlns="http://www.w3.org/2000/svg">

  <desc>Patterns</desc>
  <defs>
      <pattern id="notes" x="0" y="0" width="50" height="75"
                    patternTransform="rotate(15)"
                    patternUnits="userSpaceOnUse">

         <ellipse cx="10" cy="30" rx="10" ry="5"/>
         <line x1="20" y1="30" x2="20" y2="0"
                   stroke-width="3" stroke="black"/>
         <line x1="20" y1="0" x2="30" y2="5"
                   stroke-width="3" stroke="black"/>

      </pattern>
  </defs>

  <g>
     <ellipse cx="175" cy="100" rx="125" ry="60"
          fill="url(#notes)" stroke="black" stroke-width="5"/>
  </g>
</svg>
```


SVG Demonstration - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search
Address Go
Links Customize Links Free Hotmail RealPlayer
My Computer

Filters

Perhaps one of the most powerful capabilities of SVG is the ability to add filter effects to an image. These effects duplicate many of the effects found in expensive graphics manipulation programs, such as lighting effects and Gaussian blurs. A complete discussion of these filters is well beyond the scope of this tutorial, but this panel discusses the basics.

Filtering an SVG image involves creating a series of filter primitives, each of which has its own purpose. For example, the offset filter moves the source image to the left or right, and up or down, as specified. The Gaussian blur primitive blurs the source image as requested.

The source image doesn't have to be the actual SVG image. For example, it could be the result of a previous primitive. The following code applies several filters to the pattern displayed in the previous panel.

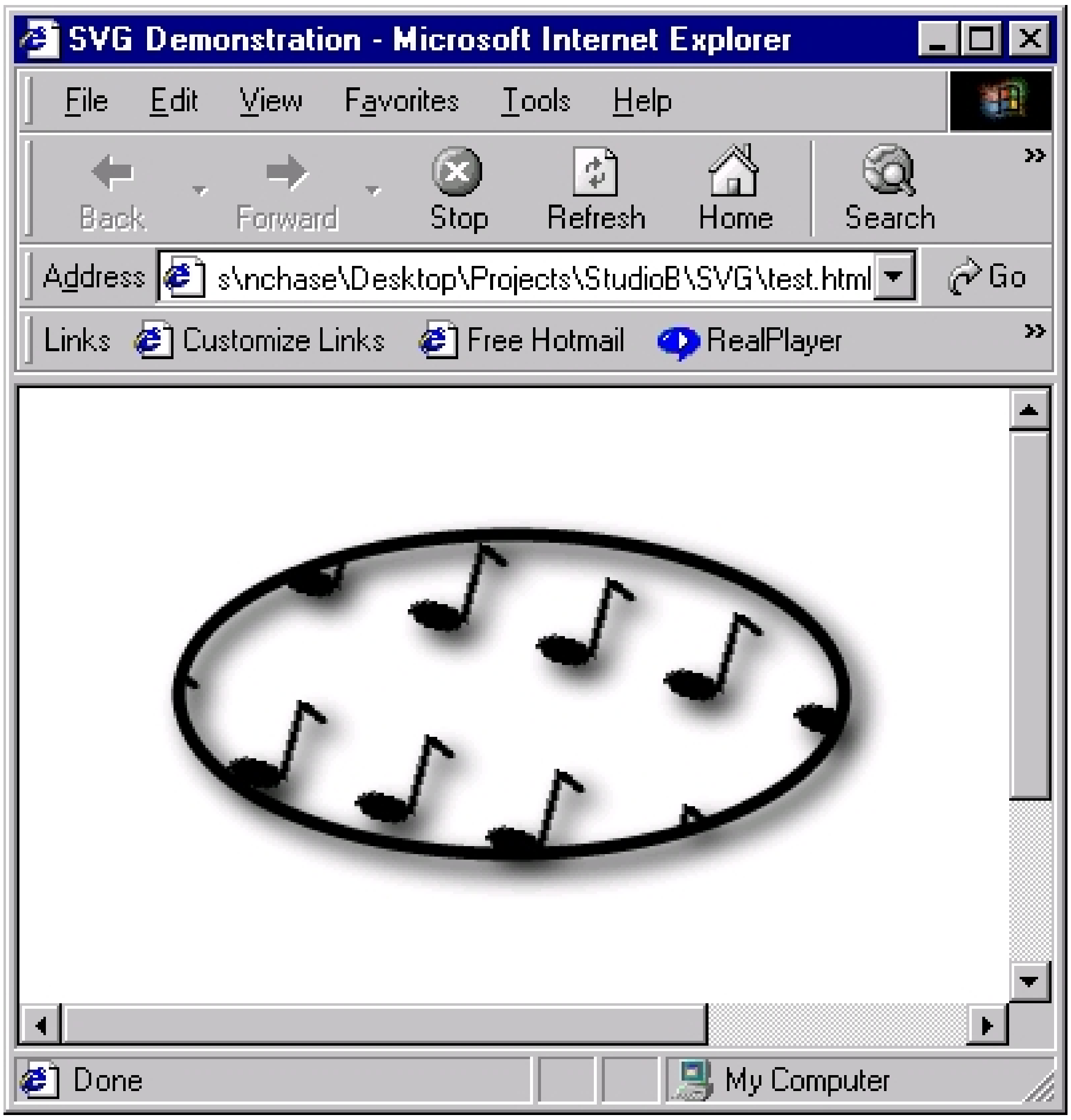
```
<?xml version="1.0"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="400" height="200" xmlns="http://www.w3.org/2000/svg"
                     xmlns:xlink="http://www.w3.org/1999/xlink">

  <desc>Filters</desc>
  <defs>

      <filter id="dropShadow" filterUnits="userSpaceOnUse"
                      x="0" y="0" width="400" height="200">
         <feOffset in="SourceAlpha" dx="5" dy="5" result="offset"/>
         <feGaussianBlur in="offset" stdDeviation="5" result="blur"/>
         <feMerge>
           <feMergeNode in="blur"/>
           <feMergeNode in="SourceGraphic"/>
         </feMerge>
      </filter>

. . .
  </defs>

  <g>
     <ellipse filter="url(#dropShadow)" cx="175" cy="100"
           rx="125" ry="60"
           fill="url(#notes)" stroke="black" stroke-width="5"/>
  </g>
</svg>
```

First, the offset filter takes as its source (using the `in` attribute) the alpha channel of the original ellipse and its pattern. The alpha channel consists of a black pixel for each non-white pixel in the image, and is specified using the `SourceAlpha` keyword. The offset primitive does its work and outputs the result to a buffer called, in this case, `offset`, as specified by the `result` attribute.

Next, the blur primitive takes over. It takes as its source the contents of the `offset` buffer, as specified by the `in` parameter. It outputs its result to a buffer called `blur`, as specified by the `result` attribute.

At this point, the filter consists only of the offset, blurred image. If the filter were left as-is, only this blur would appear on the page. The merge primitive takes the contents of the `blur` buffer and merges it with the original source graphic, as specified when the `in` attribute references the `SourceGraphic` keyword.

The result of all this is the original image with the drop-shadow:


SVG Demonstration - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search
Address s\nchase\Desktop\Projects\StudioB\SVG\test.html Go
Links Customize Links Free Hotmail RealPlayer
Done
My Computer

Section 6. Coordinates and transformations

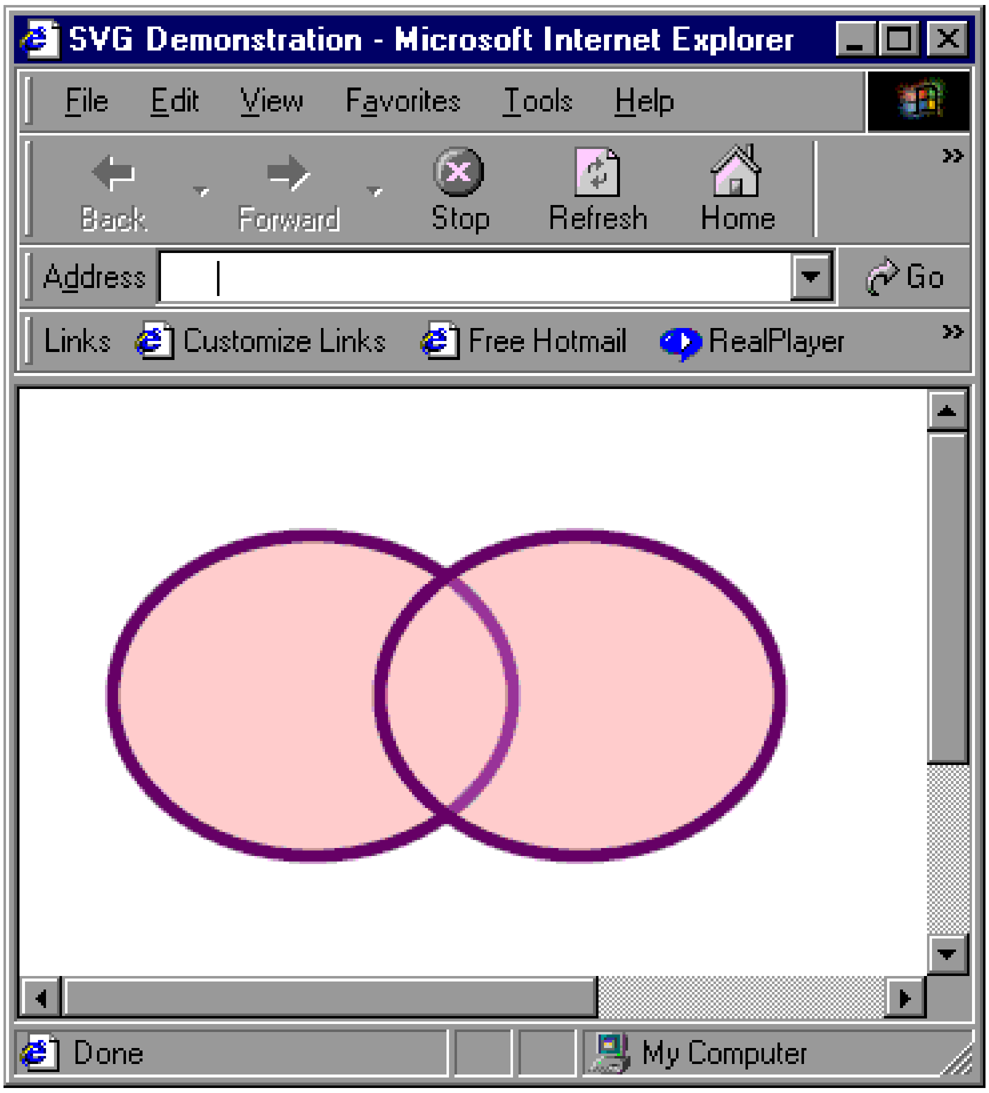
Coordinate systems and the initial viewport

Throughout this tutorial, elements have been positioned via coordinates. Now it is time to discuss the system into which those coordinates fit.

When a document is first accessed, the user agent (in most cases, the browser) determines the viewport for the image. The viewport is the portion of the document that is actually visible, and consists of a coordinate system that starts with the point 0,0 in the upper left-hand corner, with the positive x-axis running to the right, and the positive y-axis running downwards. One pixel in the coordinate system corresponds with one pixel in the viewport.

Several actions can create a new coordinate system. Transformations, which are covered next, create a new coordinate system within the transformed element, but a new coordinate system can be created directly by adding another `<svg></svg>` element to the document. Consider the following example, where the same element, with the same x and y attributes, is displayed in different locations because the second actually belongs to another coordinate system, offset from the first by 100 pixels:

```
<?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="400" height="200" xmlns="http://www.w3.org/2000/svg">

  <desc>Coordinates</desc>
  <g>
     <ellipse cx="100" cy="100" rx="75" ry="60"
           fill="pink" stroke="purple" stroke-width="5"
           fill-opacity=".5"/>

     <svg x="100" y="0">
        <ellipse cx="100" cy="100" rx="75" ry="60"
           fill="pink" stroke="purple" stroke-width="5"
           fill-opacity=".5"/>
     </svg>
  </g>
</svg>
```


SVG Demonstration - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home
Address
Go
Links Customize Links Free Hotmail RealPlayer
Done
My Computer

Transformations

Performing transformation alters the coordinate system in which an element lives, changing its appearance. Transformations can be used to alter the appearance of elements in several ways:

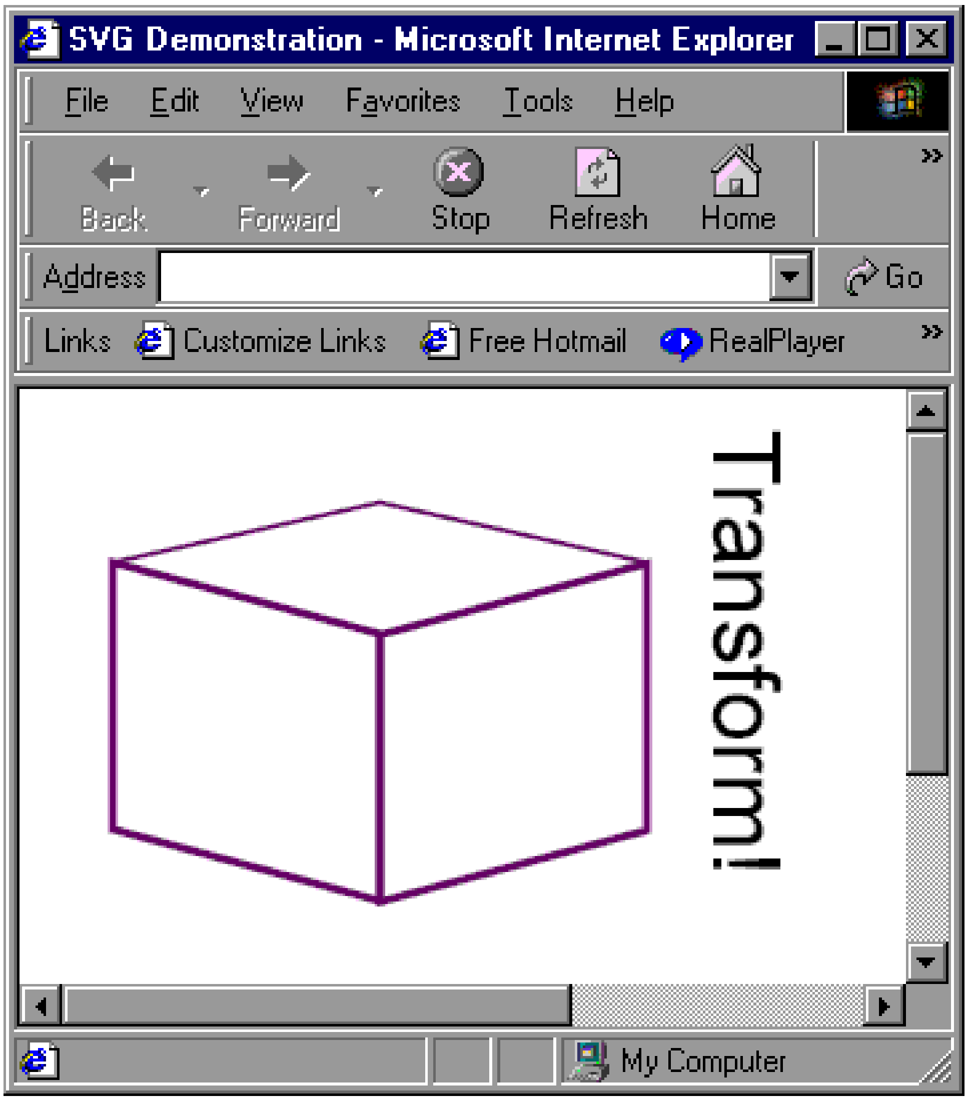
- `translate(x,y)`: This transformation offsets the element by the amounts specified.
- `scale(x, y)`: This transformation changes the size of the element. The amount of scaling in the x and y directions can be controlled separately, but if only one value is specified, it will be used for both.
- `rotate(n)`: This transformation rotates the element by the specified number of degrees.
- `skewX(n)`/ `skewY(n)` : These transformations skew the element by the specified number of pixels, with respect to the appropriate axis.

Transformations can also be specified using a matrix, but this is well beyond the scope of this tutorial.

Transformations are cumulative, and may be specified either as part of a single transform attribute or as part of nested elements as shown below:

```
<?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="400" height="200" xmlns="http://www.w3.org/2000/svg"
                      xmlns:xlink="http://www.w3.org/1999/xlink">

  <desc>Coordinates</desc>
  <defs>
     <rect id="refBox" x="0" y="0" height="100" width="100"
                stroke="purple" stroke-width="3" fill="none"/>
  </defs>
  <g>
     <!-- Top lines -->
     <g transform="scale(1, .5) translate(0, 50)">
         <path stroke="purple" stroke-width="3"
              d="M25 50 L 125 5 L 225 50" fill="none"/>
     </g>

     <!-- Left box -->
     <use xlink:href="#refBox"
            transform="translate(25, 50) skewY(15)"/>

     <!-- Right box -->
     <g transform="translate(25,25)">
        <g transform="skewY(-15)">
          <g transform="translate(100, 79)">
            <use xlink:href="#refBox"/>
         </g>
        </g>
      </g>

     <!-- Text along the side -->
     <g transform="rotate(90) translate(0, -250)">
        <text font-size="35">Transform!</text>
      </g>

  </g>
</svg>
```

Possibly the most important thing to note in this example is that it is the actual coordinate system that is being transformed. The object itself is not actually transformed, but the change in the coordinate system in which it lives makes it appear to change. Consider the "Transform!" text. It is being translated by a negative 250 pixels in the y direction, so it stands to reason that the text should disappear, being rendered above the top of the viewport. Before the translation takes place, however, the coordinate system is rotated 90 degrees, so a negative y value actually moves the text 250 pixels to the right.


SVG Demonstration - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home
Address
Go
Links Customize Links Free Hotmail RealPlayer
Transform!
My Computer

Scaling with viewBox

In the absence of any changes, the initial viewport specifies a size where the upper left-hand coordinate is `0,0`, and the lower right-hand coordinate is equal to the number of pixels between it and `0,0`. There are times, however, when the desired effect is instead for the image to scale to the available size, no matter what that size is. That's where the `viewBox` attribute comes in.

The `viewBox` attribute re-maps the viewport, specifying the new values to appear in the upper left-hand corner and the lower right-hand corners of the viewport. Remember: When placing an SVG graphic on a Web page, the dimensions of the `<object></object>` tag determine the size of the viewport.

For example, if the eyes and glasses had a `viewBox` attribute added, as in:

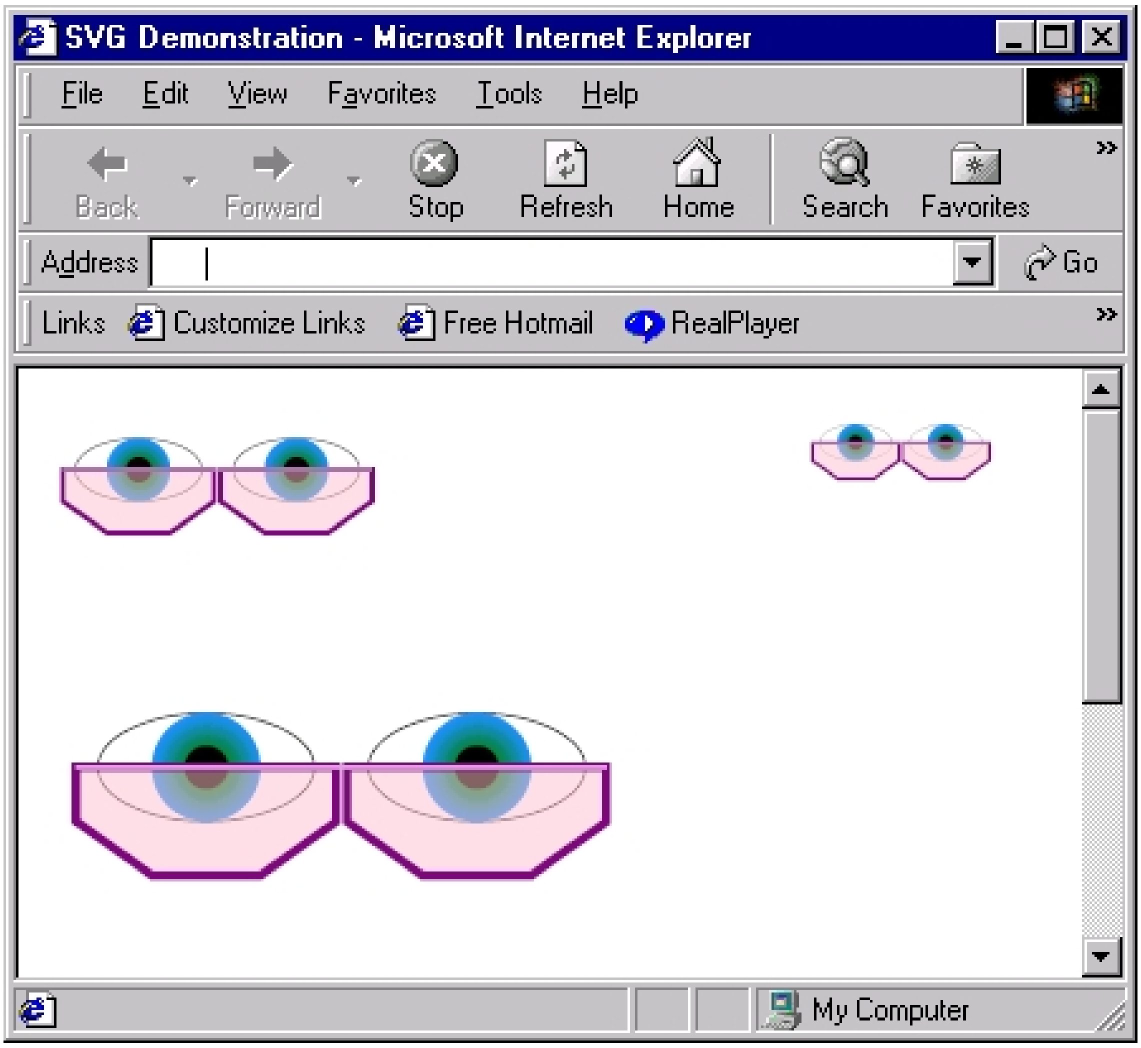
```
<?xml version="1.0" standalone="no"?>
```

```
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="300px" height="200px"
     viewBox="50 0 350 200" preserveAspectRatio="xMinYMin"
     xmlns="http://www.w3.org/2000/svg">
  <desc>ViewBox</desc>
  <defs>

     <linearGradient id="lineGradient">
        <stop offset="0%" stop-color="red" />
        <stop offset="100%" stop-color="yellow" />
     </linearGradient>
. . .
```

the page would display the image in whatever box was allotted to it, scaling appropriately. So a Web page of:

```
<html>
  <head><title>SVG Demonstration</title></head>
  <body>

  <object type="image/svg+xml" data="test.svg"
                              height="100" width="300">
     <img src="NonSVG.gif" alt="Static version of SVG image" />
  </object>

  <object type="image/svg+xml" data="test.svg"
                              height="100" width="100">
     <img src="NonSVG.gif" alt="Static version of SVG image" />
  </object>

  <object type="image/svg+xml" data="test.svg"
                              height="300" width="300">
     <img src="NonSVG.gif" alt="Static version of SVG image" />
  </object>

  </body>
</html>
```

displays the image three times, at various sizes:


SVG Demonstration - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites
Address
Go
Links Customize Links Free Hotmail RealPlayer
My Computer

The `preserveAspectRatio` attribute determines how the scaling is accomplished. A value of `none` will cause the image to stretch to fit the box, even if it's distorted. A value of `xMinYMin`, as shown above, aligns the minimum x and y values of the image with the minimum x and y values of the box. Other possible values are `xMinYMid`, `xMinYMax`, `xMidYMin`, `xMidYMid` (the default), `xMidYMax`, `xMaxYMin`, `xMaxYMid`, and `xMaxYMax`.

Section 7. Paths

What is a path?

The predefined shapes that SVG provides are certainly useful, but there comes a time when they are not sufficient to get the job done. This is particularly true in two cases: first, when an image requires curves, which can't be created with a polygon or polyline, and second, when an animation or text needs to proceed along a particular shape on the page.

Enter paths. A path is a series of commands that are used to create a precisely defined shape as part of an image. This shape can be open (like a line) or closed (like a polygon), and can contain one or more lines, curves, and segments.

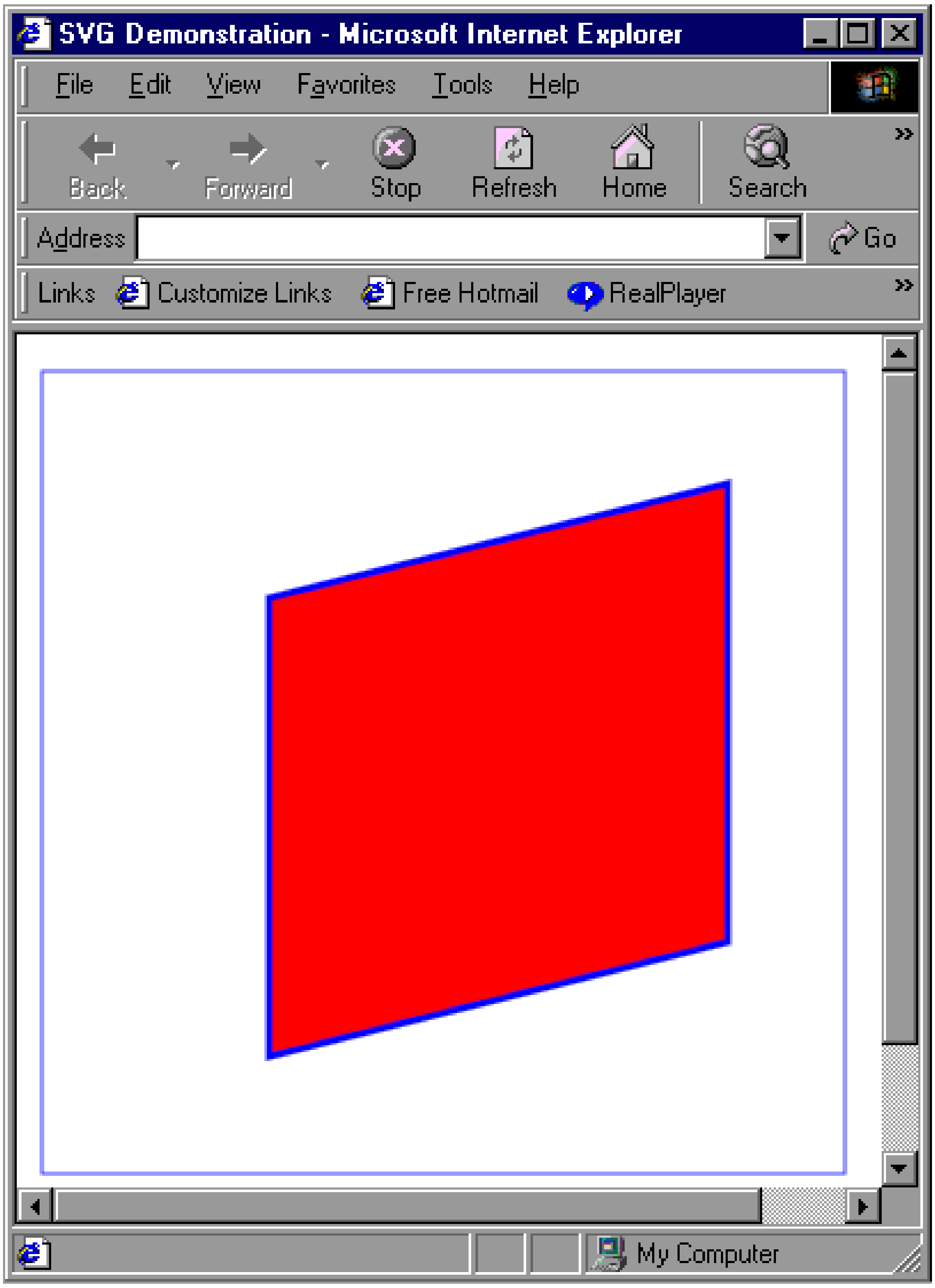
The most basic paths consist of a few line segments. For example:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="400" height="400"
     xmlns="http://www.w3.org/2000/svg">
  <desc>A simple path</desc>
  <rect x="1" y="1" width="350" height="350"
        fill="none" stroke="blue" />
  <path d="M 100 100 L 300 50 L 300 250 L 100 300 Z"
        fill="red" stroke="blue" stroke-width="3" />
</svg>
```

The above code generates a simple polygon according to the instructions provided. Those instructions are as follows:

1. `M 100 100` : Move to the point 100, 100.
2. `L 300 50` : Draw a line to the point 300, 50.
3. `L 300 250` : Draw a line to the point 300, 250.
4. `L 100 300` : Draw a line to the point 100, 300.
5. `Z` : Close the shape by drawing a line back to the original point. (Or more specifically, to the point specified by the last "move" command).

The end result is shown here:


SVG Demonstration - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search
Address
Go
Links Customize Links Free Hotmail RealPlayer
My Computer

Note that all commands shown here are uppercase, which specifies that the coordinates are absolute coordinates with respect to the overall coordinate system. Using lowercase commands specifies relative coordinates. So the command `l 50 50` creates a line from the current point to the point 50 pixels down and to the right, wherever that may be.

Other commands for simple lines include `H` (or `h`) for horizontal lines and `V` (or `v`) for vertical lines.

Curves

Path commands can create three types of curves:

- Elliptical curves are segments of an ellipse, and are known as arcs. The `A` (or `a`) command creates them by specifying the start point, end point, x and y radii, rotation, and direction, as shown below.
- Cubic Bezier curves are defined by a start point, an end point, and two control points that "pull" the curve towards them. The `C` (or `c`) command (specifying a start and end point) and `S` (or `s`) command (assuming that the curve picks up where the last command left off) create them.
- Quadratic Bezier curves are similar to their cubic cousins, but instead have only a single control point. The `Q` (or `q`) and `T` (or`t`) commands create them.

The example below shows some sample arcs with text removed for clarity. The arc command takes the form:

```
A radiusX, radiusY rotation large arc flag, sweep flag endX, endY
```

So an arc with radii of 50 and 25, and no rotation, which uses the larger part of the ellipse and the lower end of the sweep and ends 50 pixels to the right and 25 pixels down from the start point would use:

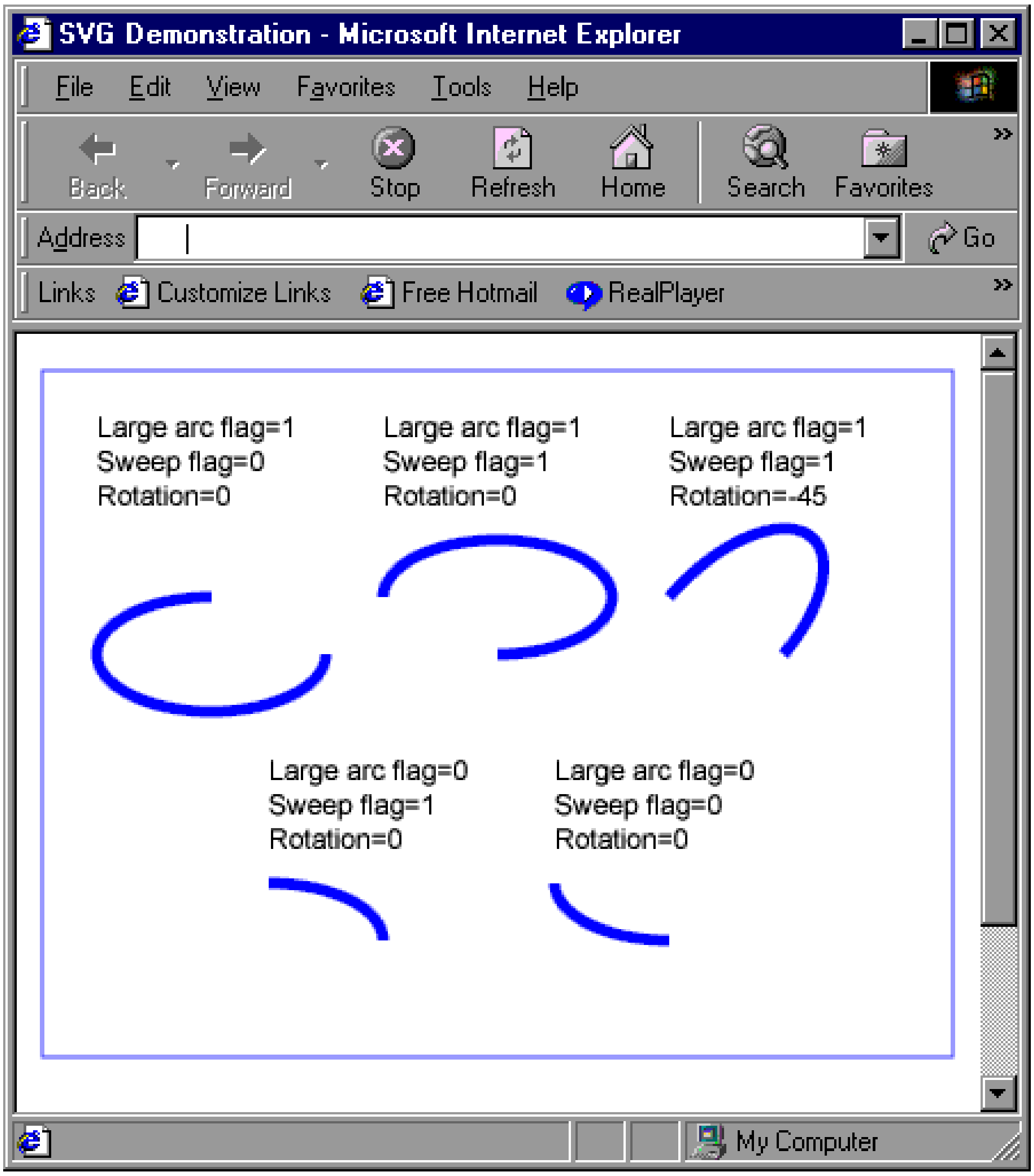
```
a50,25 0 1,0 50,25
```

Some variations are shown below:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="4cm" height="4cm" viewBox="0 0 400 400"
     xmlns="http://www.w3.org/2000/svg">
  <desc>Curved paths</desc>
  <rect x="1" y="1" width="398" height="300"
        fill="none" stroke="blue" />

  <!-- First row -->
  <text x="25" y="30">Large arc flag=1</text>
  <text x="25" y="45">Sweep flag=0</text>
  <text x="25" y="60">Rotation=0</text>
  <path d="M75,100 a50,25 0 1,0 50,25"
        stroke="blue" stroke-width="5" fill="none" />

  . . .
  <path d="M150,100 a50,25 0 1,1 50,25"
        stroke="blue" stroke-width="5" fill="none" />

  . . .
  <path d="M275,100 a50,25 -45 1,1 50,25"
        stroke="blue" stroke-width="5" fill="none" />

  <!-- Second row -->
  . . .
  <path d="M100,225 a50,25 0 0,1 50,25"
```

```
        stroke="blue" stroke-width="5" fill="none" />

  . . .
   <path d="M225,225 a50,25 0 0,0 50,25"
        stroke="blue" stroke-width="5" fill="none" />
</svg>
```

Note that all of the arcs have the same start and end points, but not the same shape.


SVG Demonstration - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites
Address
Go
Links Customize Links Free Hotmail RealPlayer
My Computer

Large arc flag=1
Sweep flag=0
Rotation=0

Large arc flag=1
Sweep flag=1
Rotation=0

Large arc flag=1
Sweep flag=1
Rotation=-45

Large arc flag=0
Sweep flag=1
Rotation=0

Large arc flag=0
Sweep flag=0
Rotation=0

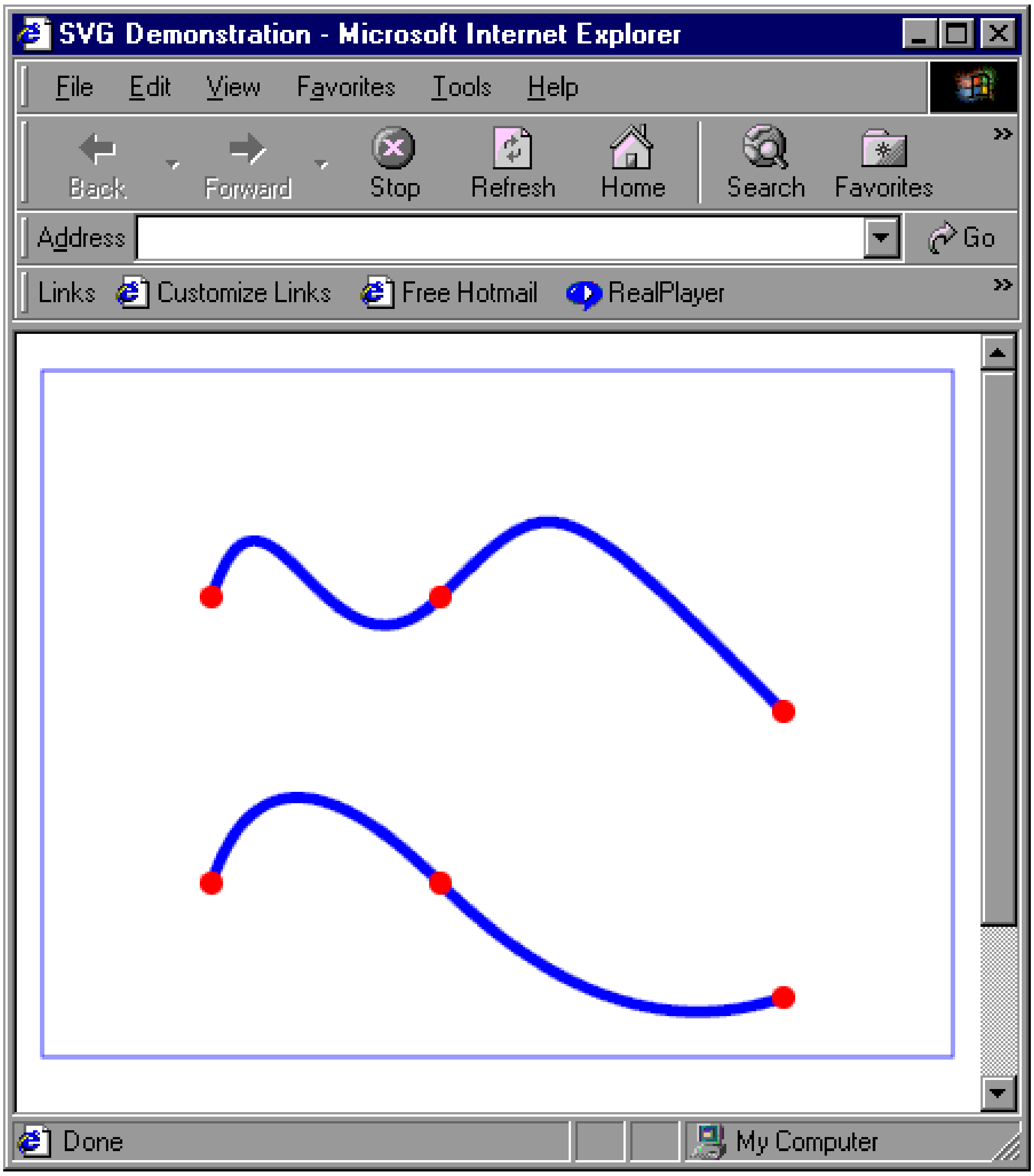
Curves, continued

The shape of a bezier curve is determined by the start and end points, as well as the location of the control points. The commands are formatted as follows:

```
C control1x, control1y, control2x, control2y, endx, endy
S control2x, control2y, endx, endy
Q controlx, controly, endx, endy
T endx, endy
```

For the S and T commands, the first control point is assumed to be a reflection of the second control point of the previous curve. For example:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="4cm" height="4cm" viewBox="0 0 400 400"
     xmlns="http://www.w3.org/2000/svg">
  <desc>Curved paths</desc>
  <rect x="1" y="1" width="398" height="300"
        fill="none" stroke="blue" />

  <!-- First row -->
  <path d="M75,100 c25,-75 50,50 100,0 s50,-50 150,50"
        stroke="blue" stroke-width="5" fill="none" />

  <circle cx="175" cy="100" r="5" fill="red" />
  <circle cx="75" cy="100" r="5" fill="red" />
  <circle cx="325" cy="150" r="5" fill="red" />

  <path d="M75,225 q25,-75 100,0 t150,50"
        stroke="blue" stroke-width="5" fill="none" />

  <circle cx="175" cy="225" r="5" fill="red" />
  <circle cx="75" cy="225" r="5" fill="red" />
  <circle cx="325" cy="275" r="5" fill="red" />

</svg>
```


SVG Demonstration - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites
Address
Go
Links Customize Links Free Hotmail RealPlayer
Done
My Computer

Markers

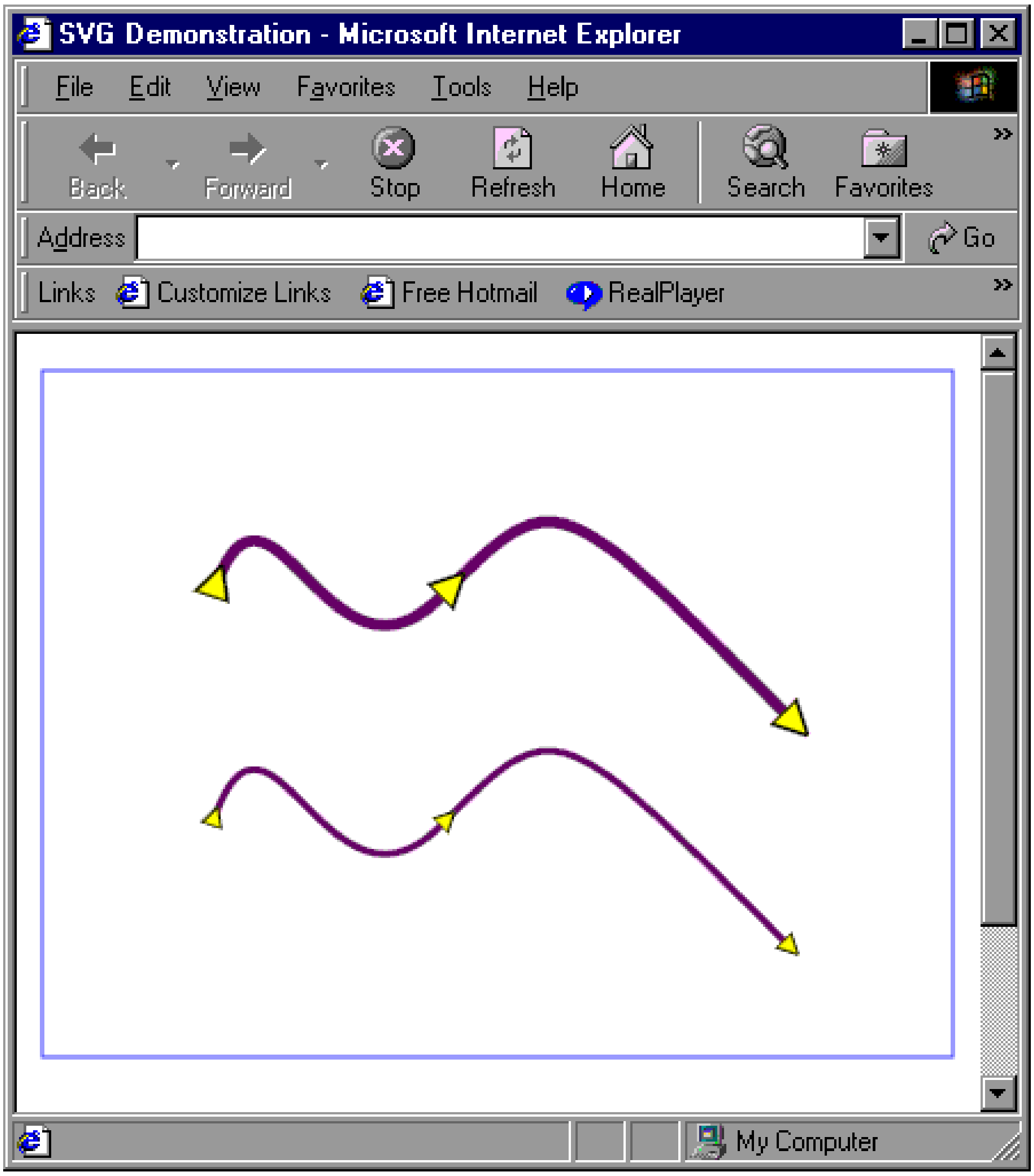
Markers are a natural complement to paths. They are elements that can be added to the start, end, and vertices of lines and paths. The most common use is to add arrows to the end of lines, but any object can be used.

The process is straightforward: Define the marker, then assign it to the relevant element using the `marker-start`, `marker-end`, and `marker-mid` attributes. For example:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="4cm" height="4cm" viewBox="0 0 400 400"
     xmlns="http://www.w3.org/2000/svg">
  <desc>Markers</desc>
  <defs>
    <marker id="arrow"
      viewBox="0 0 10 10" refX="0" refY="5"
      markerUnits="strokeWidth" markerWidth="3" markerHeight="10"
      orient="auto">

      <path d="M 0 0 L 10 5 L 0 10 z" fill="yellow" stroke="black"/>

    </marker>
  </defs>
  <rect x="1" y="1" width="398" height="300"
        fill="none" stroke="blue" />

  <!-- First row -->
  <path d="M75,100 c25,-75 50,50 100,0 s50,-50 150,50"
        stroke="purple" stroke-width="5" fill="none"
        marker-start="url(#arrow)"
         marker-mid="url(#arrow)"
         marker-end="url(#arrow)" />

  <!-- Second row -->
  <path d="M75,200 c25,-75 50,50 100,0 s50,-50 150,50"
        stroke="purple" stroke-width="3" fill="none"
        marker-start="url(#arrow)"
         marker-mid="url(#arrow)"
         marker-end="url(#arrow)" />
</svg>
```

The marker itself consists of a simple triangular path, acted upon by the marker attributes. The viewBox has been set so that the marker itself always fills the entire box, no matter what that box is. The box itself is affected by the size of the line to which the marker is applied, because of the value of `markerUnits`. The `markerUnits` attribute can also be set to `userSpaceOnUse`, which causes the marker to use the general coordinate system instead. The `refX` and `refY` attributes determine the point within the marker that is "attached" to the line being marked. Finally, the orientation for the marker is set to `auto`, causing it to be aligned with the Y-axis perpendicular to the tangent to the line. (In order to compensate for this orientation, the marker has been constructed to point along the X-axis.)

Notice that the marker size changes with the stroke size:


SVG Demonstration - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Forward
Stop
Refresh
Home
Search
Favorites
Address
Go
Links
Customize Links
Free Hotmail
RealPlayer
My Computer

Section 8. Text

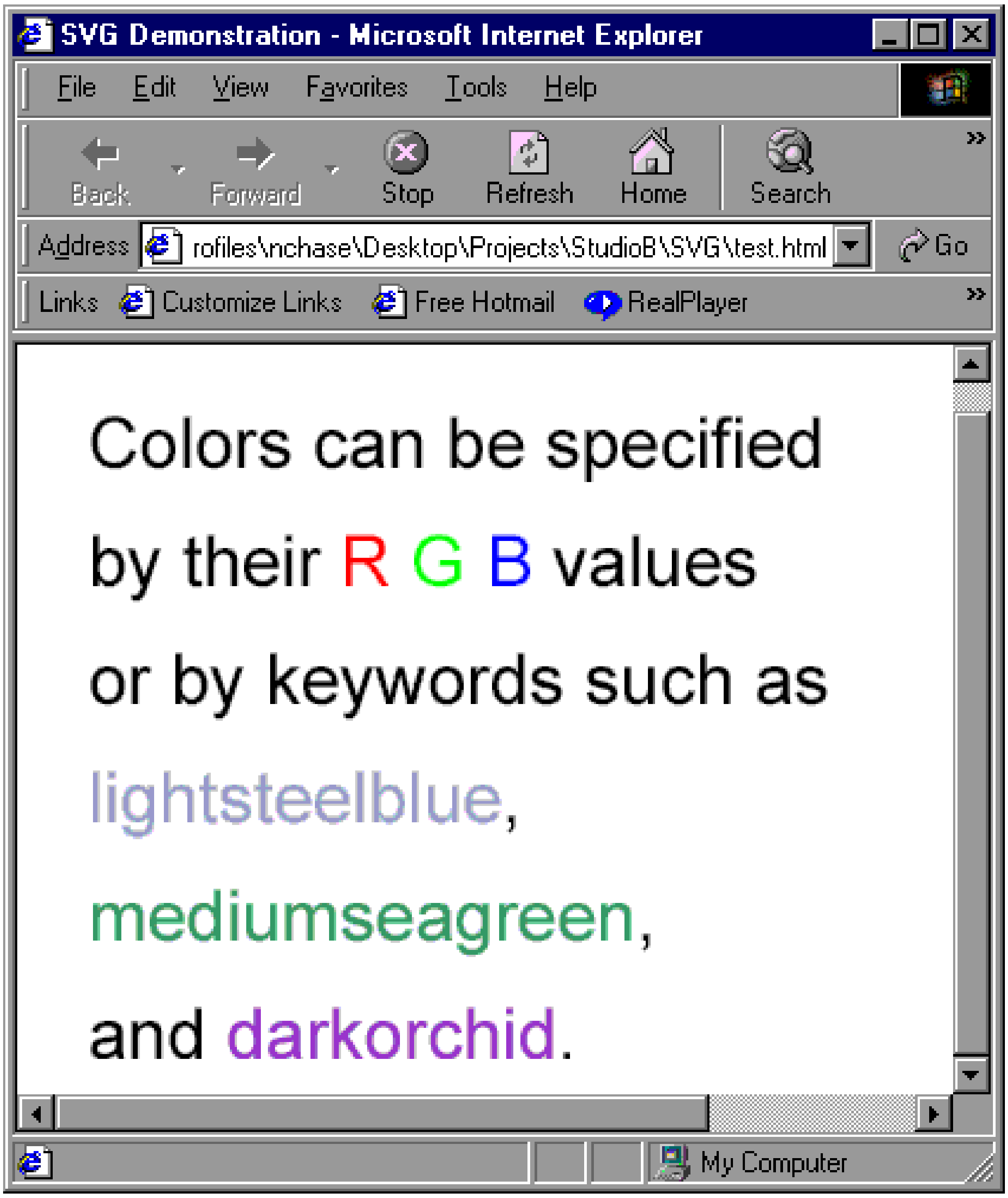
Adding text

One of the great strengths of SVG is its ability to control text to a degree unheard-of in a standard HTML page without having to resort to images or other plug-ins (which can create accessibility challenges). Any manipulation that can be performed on a shape or a path, such as painting or filters, can be performed on text.

The one down-side is that SVG does not perform line-wrapping. If text is longer than the allowed space, it is simply cut off. Creating multiple lines of text requires, in most cases, multiple text elements.

Text elements can be broken into pieces using the `tspan` element, allowing each section to be individually styled. Within `text` elements, white space is handled similarly to HTML; line feeds and carriage returns become spaces, and multiple spaces are collapsed to a single space, as seen in this earlier 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="400" height="200" xmlns="http://www.w3.org/2000/svg"
                      xmlns:xlink="http://www.w3.org/1999/xlink">

  <desc>Text</desc>
  <defs>
  </defs>

  <g>

     <text x="20" y="50" font-size="30">
        Colors can be specified
     </text>
     <text x="20" y="100" font-size="30">by their
        <tspan fill="rgb(255,0,0)">R</tspan>
        <tspan fill="rgb(0,255,0)">G</tspan>
        <tspan fill="rgb(0,0,255)">B</tspan>
     values</text>
     <text x="20" y="150" font-size="30">
        or by keywords such as
     </text>
     <text x="20" y="200" font-size="30">
        <tspan fill="lightsteelblue">lightsteelblue</tspan>,
     </text>
     <text x="20" y="250" font-size="30">
        <tspan fill="mediumseagreen">mediumseagreen</tspan>,
     </text>
     <text x="20" y="300" font-size="30">and
        <tspan fill="darkorchid">darkorchid</tspan>.
     </text>

  </g>
</svg>
```


SVG Demonstration - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search
Address rofiles\nchase\Desktop\Projects\StudioB\SVG\test.html Go
Links Customize Links Free Hotmail RealPlayer
My Computer

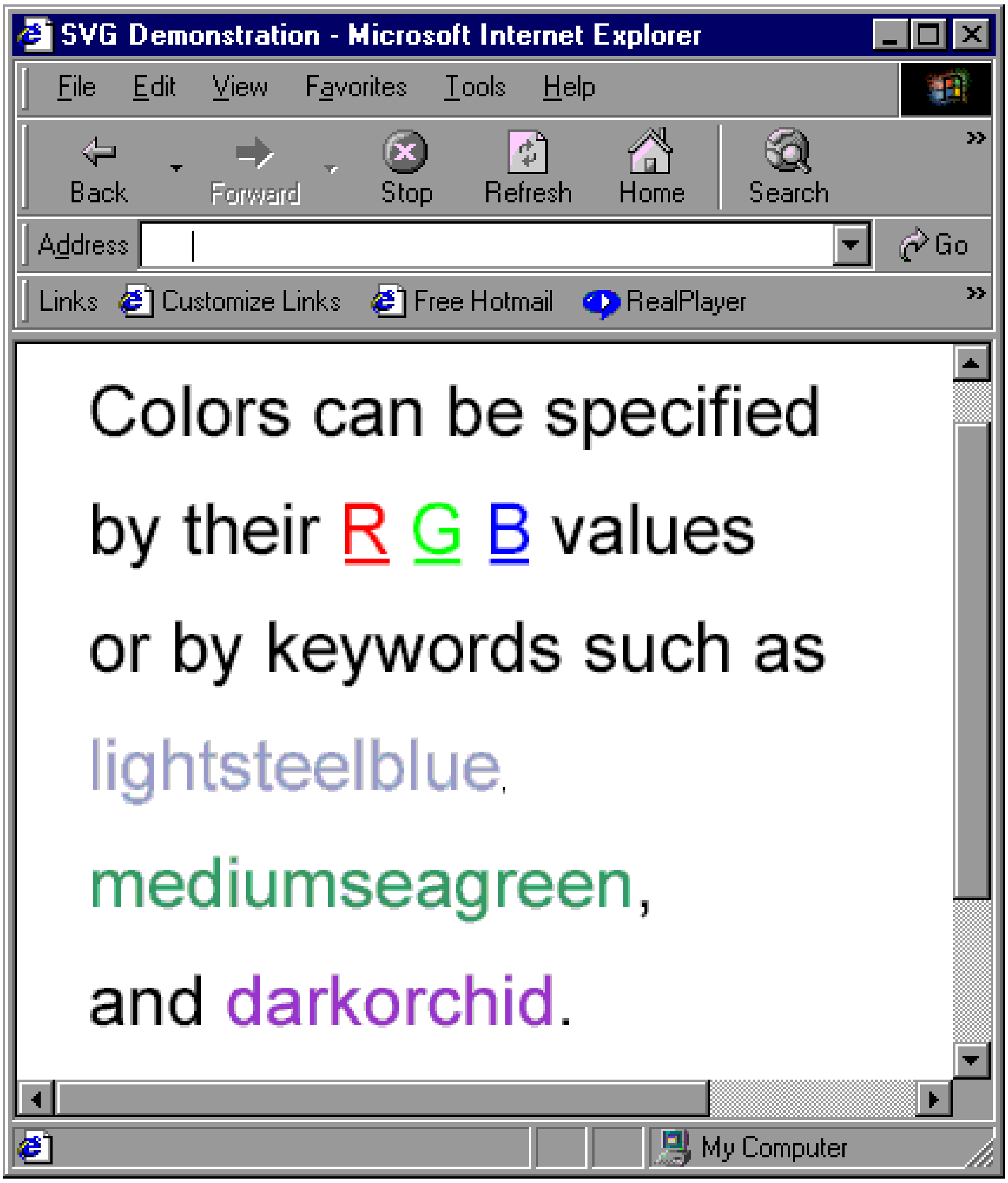
Colors can be specified by their R G B values or by keywords such as lightsteelblue, mediumseagreen, and darkorchid.

Using CSS properties

In actuality, all properties (for all elements, not just text) can be associated with an element using Cascading Style Sheets, and all CSS properties for text are available within an SVG image.

Elements may be styled using the style attribute directly, or using references to a style sheet. Style sheets should never be parsed (because they occasionally contain characters that cause problems) so they are included within an XML CDATA section.

```
<?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="400" height="200" xmlns="http://www.w3.org/2000/svg">

  <desc>Text</desc>
  <defs>
    <style type="text/css">
      <![CDATA[
      .abbreviation { text-decoration: underline; }
      ]]>
    </style>
  </defs>

  <g>

     <text x="20" y="50" font-size="30">Colors can be specified</text>
     <text x="20" y="100" font-size="30">by their
        <tspan fill="rgb(255,0,0)" class="abbreviation">R</tspan>
        <tspan fill="rgb(0,255,0)" class="abbreviation">G</tspan>
        <tspan fill="rgb(0,0,255)" class="abbreviation">B</tspan>
     values</text>
     <text x="20" y="150" font-size="30">or by keywords such as</text>
     <text x="20" y="200">
        <tspan style="fill: lightsteelblue; font-size:30">
           lightsteelblue
        </tspan>,
     </text>
. . .
  </g>
</svg>
```

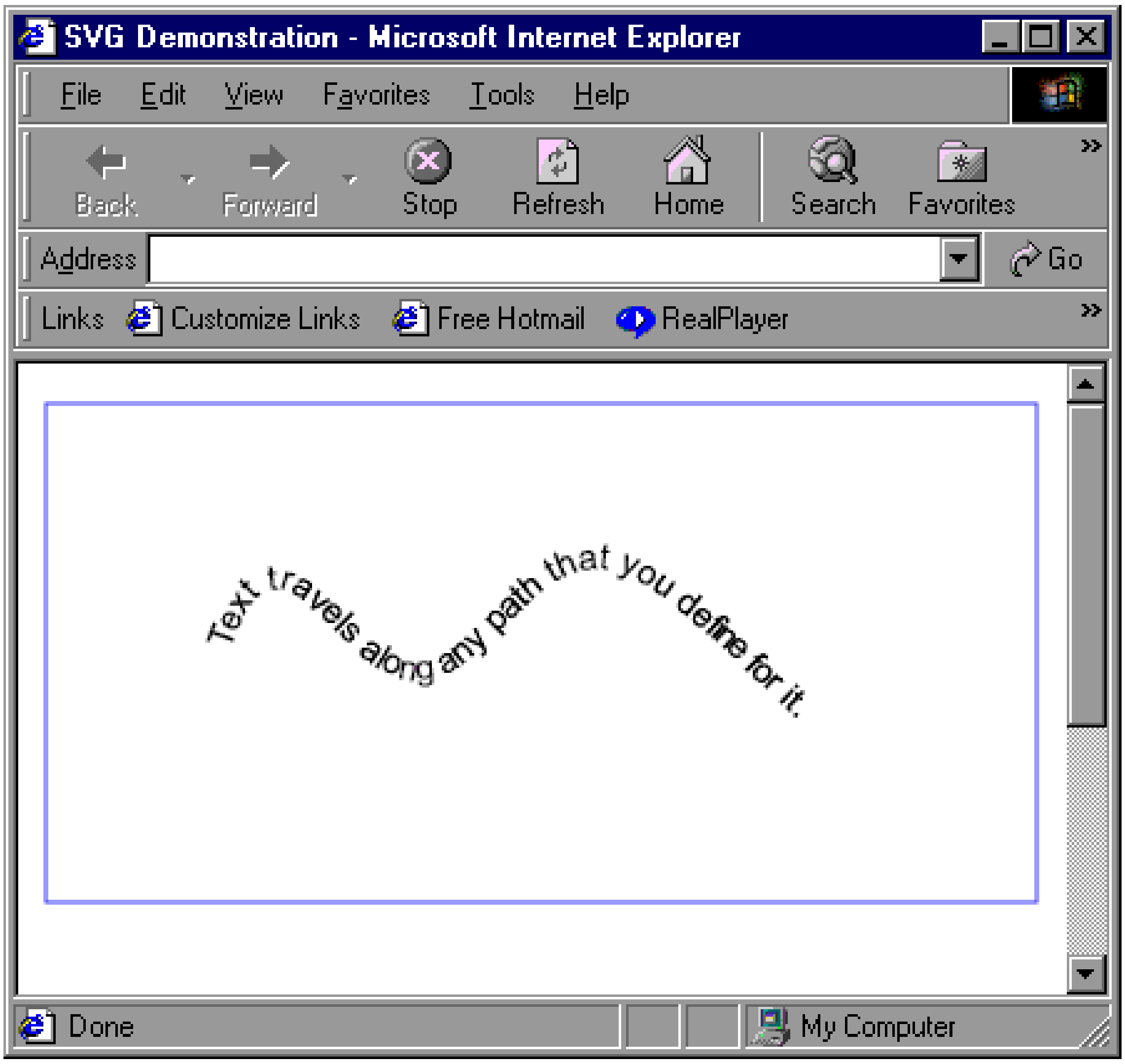

SVG Demonstration - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search
Address
Go
Links Customize Links Free Hotmail RealPlayer
My Computer

Colors can be specified
by their R G B values
or by keywords such as
lightsteelblue,
mediumseagreen,
and darkorchid.

Text on a path

One SVG capability that is impossible in straight HTML is aligning text along a path. To accomplish this, create a `textPath` element that links to pre-defined path information:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="400" height="300" xmlns="http://www.w3.org/2000/svg"
                      xmlns:xlink="http://www.w3.org/1999/xlink">
  <desc>Text on a path</desc>
  <defs>
     <path id="wavyPath"
         d="M75,100 c25,-75 50,50 100,0 s50,-50 150,50"/>
  </defs>
  <rect x="1" y="1" width="398" height="200"
        fill="none" stroke="blue" />

  <text x="50" y="50" font-size="14">
     <textPath xlink:href="#wavyPath">
      Text travels along any path that you define for it.
     </textPath>
  </text>

</svg>
```


SVG Demonstration - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites
Address Go
Links Customize Links Free Hotmail RealPlayer
Text travels along any path that you define for it.
Done My Computer

Section 9. Animation and interactivity

Controlling properties

Even before SVG came about, animation and interactivity were firmly entrenched on the Web. Although the implementations can get complicated, the concept is simple: Change the value for the property of an object, and the object itself appears to change. For example, add 50 pixels to the x coordinate, and the object moves 50 pixels to the right.

The concept is the same with SVG images, but implementation is much simpler due to the fact these capabilities have been built into the language from the start. SVG defines five elements devoted to animation:

`animate`: This element designates a specific property (via the `attributeName` attribute) whose value is changed from the value designated as the `from` attribute to the value designated as the `to` attribute over the amount of time specified in the `dur` attribute. The `repeatCount` attribute designates how many times the animation takes place. To make the animation run indefinitely, set the value of `repeatCount` to `indefinite`. The animation applies to the element that encloses it, so the code:

```
<rect x="50" y="50" width="100" height="100"
                         fill="none" stroke="purple">
   <animate attributeType="CSS" attributeName="stroke-width"
       from="1" to="50" dur="5s" repeatCount="indefinite" />
</rect>
```

creates a square with a `stroke-width` that gradually thickens to 50 pixels, then goes back to 1 and starts again.

`animateMotion`: This element provides an easy way to move an element through a specific path. The path data is the same as the d attribute for a path element, but is specified using the path element. It can also be linked to the `animateMotion` element using `xlink:href`. The start and end points are determined by the `from` and `to` attributes, and the object can be set to align itself perpendicular to the path by setting the value of `rotate` to `auto`. (The `rotate` attribute can also be set to `auto-reverse` to change this orientation by 180 degrees. Alternatively, a specific angle can be given.) As seen in Animation and interactivity on page 6 :

```
<animateMotion path="M0,300 S150,100 200,200 S400,400 500,0"
             dur="8s" repeatCount="indefinite" rotate="auto" />
```

`animateColor`: This element provides the means for changing the color of an element over a period of time. For example, to create a circle that changes from red to blue over a period of 8 seconds:

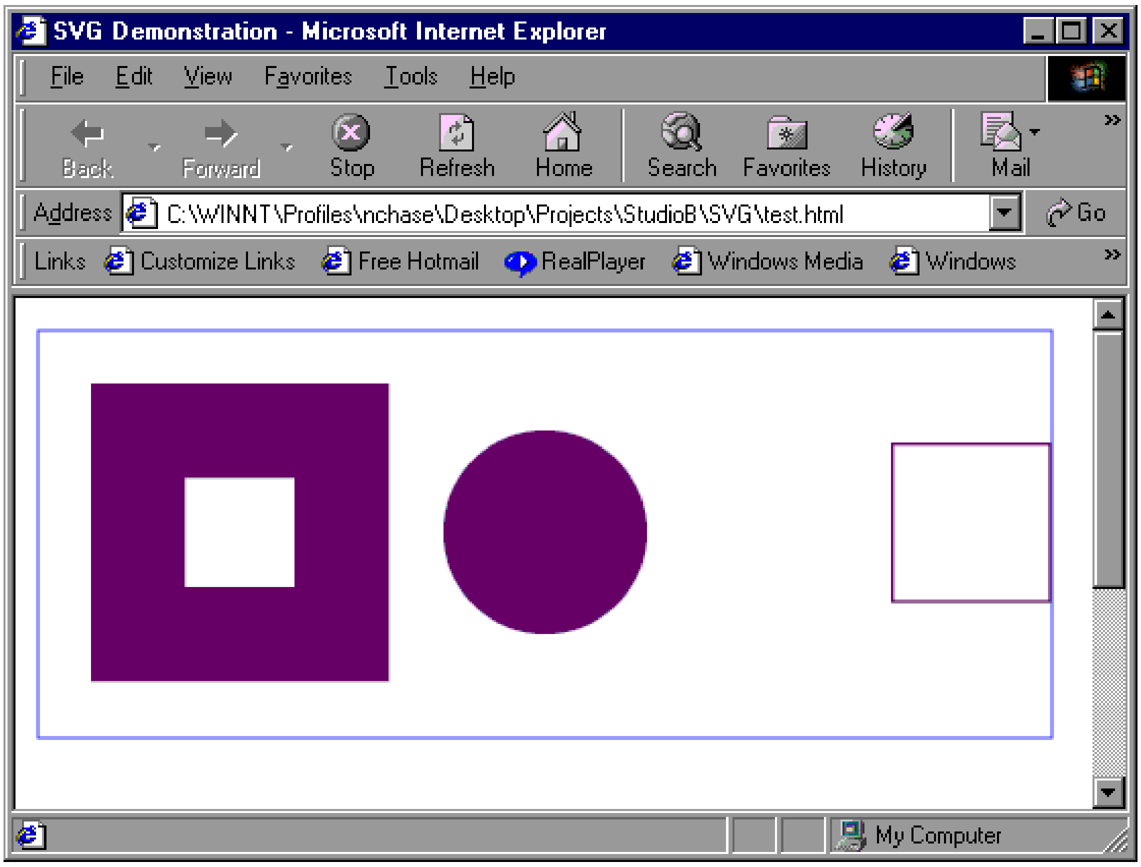
```
<circle cx="250" cy="100" r="50" fill="red">
   <animateColor attributeType="CSS" attributeName="fill"
       from="rgb(255,0,0)" to="rgb(0,0,255)" dur="8s"
       repeatCount="indefinite"/>
</circle>
```

`animateTransform`: This element performs transformations over a period of time. Remember, these transformations affect the overall coordinate system, so simply scaling a rectangle will also result in the rectangle's position changing. This example both scales the rectangle and gradually returns it to a similar position:

```
   <rect x="333" y="49" width="50" height="50" fill="none"
             stroke="purple">
      <animateTransform attributeName="transform" attributeType="XML"
           type="scale" from="1" to="3" additive="sum"
           begin="3s" dur="6s" fill="freeze" />
      <animateTransform attributeName="transform" attributeType="XML"
           type="translate" from="0,0" to="-222,-45" additive="sum"
           begin="3s" dur="6s" fill="freeze" />
  </rect>
```

`set`: This remaining element makes it easy to set a particular property on an element for a specific period of time. For example:

```
<circle cx="250" cy="100" r="50" fill="red">
    <set attributeName="r" to="100" begin="1s" dur="5s" fill="remove" />
</circle>
```


SVG Demonstration - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Mail
Address C:\WINNT\Profiles\nchase\Desktop\Projects\StudioB\SVG\test.html
Go
Links Customize Links Free Hotmail RealPlayer Windows Media Windows
My Computer

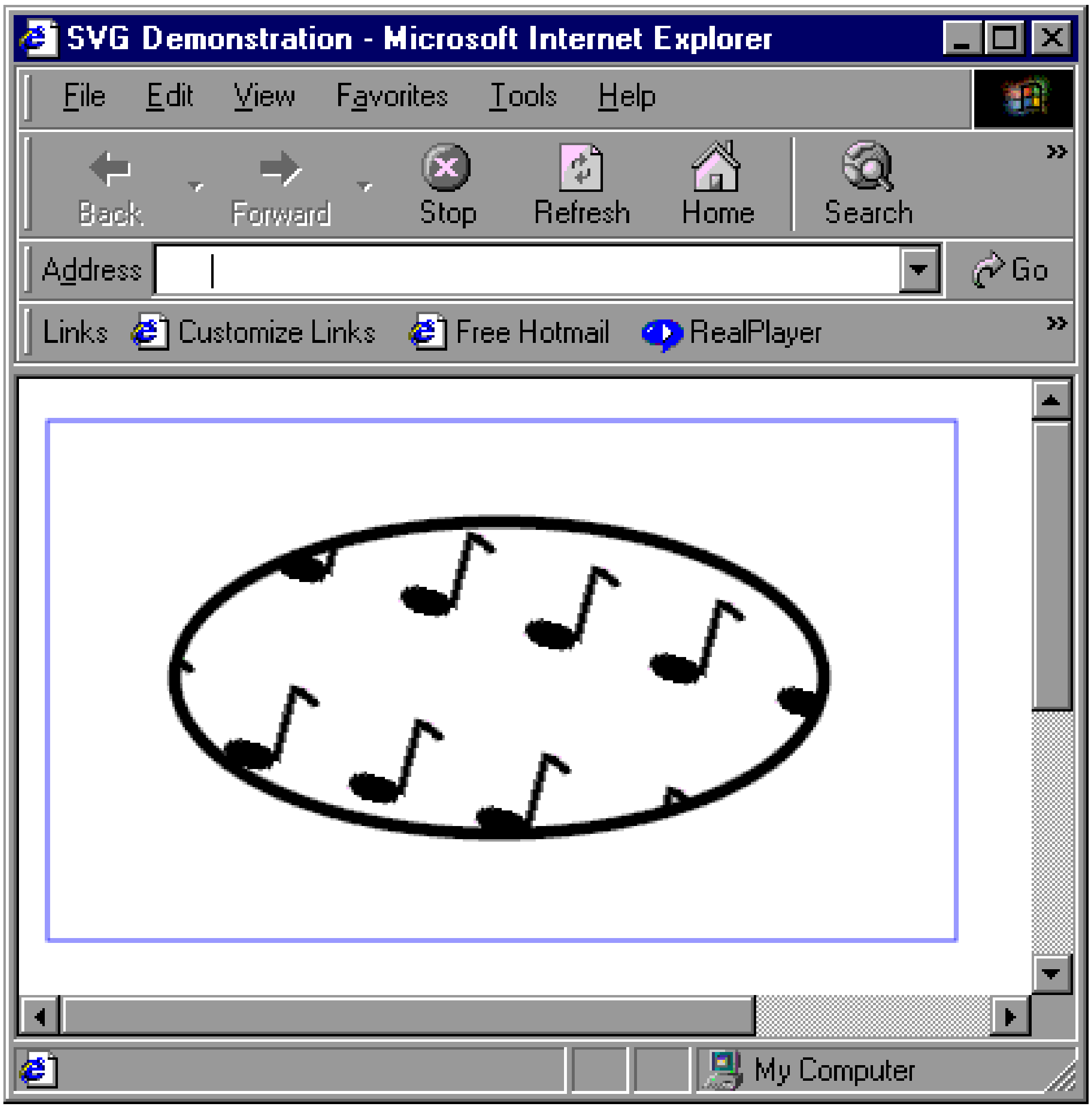
Scripting events

Like HTML pages, SVG images are set up to capture certain events, such as mouse-clicks and rollovers, and use them to fire scripts. In building simple SVG images, these events are captured via attributes. The most commonly used are `onclick`, `onactivate`, `onmousedown`, `onmouseup`, `onmouseover`, `onmousemove`, `onmouseout`, `onload`, `onresize`, `onunload`, and `onrepeat`.

When one of these events is triggered, the event object itself can be fed to the script which can, in turn, use it to determine what object has triggered the event (i.e., what object was clicked). The script can then manipulate the properties of that object, such as its attributes.

This example returns to the pattern example, but in this case the fill of the ellipse alternates from being white to using the pattern as the user clicks on it.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="500" height="300" xmlns="http://www.w3.org/2000/svg">
  <desc>Scripting the onclick event</desc>
  <defs>

    <script type="text/ecmascript">
    <![CDATA[
      function hideReveal(evt) {
        var imageTarget = evt.target;
        var theFill = imageTarget.getAttribute("fill");
        if (theFill == 'white')
          imageTarget.setAttribute("fill", "url(#notes)");
        else
          imageTarget.setAttribute("fill", "white");
      }
    ]]>
    </script>

    <pattern id="notes" x="0" y="0" width="50" height="75"
                   patternTransform="rotate(15)"
                   patternUnits="userSpaceOnUse">

         <ellipse cx="10" cy="30" rx="10" ry="5"/>
         <line x1="20" y1="30" x2="20" y2="0"
                   stroke-width="3" stroke="black"/>
         <line x1="20" y1="0" x2="30" y2="5"
                   stroke-width="3" stroke="black"/>

     </pattern>
   </defs>

  <!-- Outline the drawing area with a blue line -->
  <rect x="1" y="1" width="350" height="200" fill="none" stroke="blue"/>

  <ellipse onclick="hideReveal(evt)" cx="175" cy="100" rx="125" ry="60"
           fill="url(#notes)" stroke="black" stroke-width="5"/>


</svg>
```


SVG Demonstration - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search
Address
Go
Links Customize Links Free Hotmail RealPlayer
My Computer

Section 10. Summary

Scalable Vector Graphics summary

Scalable Vector Graphic (SVG) images are a means for using XML-based text information to create images. These images can consist of simple shapes, such as rectangles or circles, or more complex paths that can be mathematically specified. These images can then be embedded within a Web page, where an SVG viewer interprets them.

The objects that are part of an SVG image can be manipulated via filters, allowing for complex imaging effects, and providing a basis for both dynamically created graphics and animations.

This tutorial has provided an introduction to SVG, including:

- Creating the basic document
- Shapes
- Paths
- Text handling
- Styling
- Colors
- Patterns
- Animation and scripting (briefly)

Resources

Scalable Vector Graphics provide such enormous capabilities that it is impossible to cover them all in a single tutorial. For more information on SVG and related topics, see the following resources:

- For an understanding of how XML works, check out the developerWorks tutorial *Introduction to XML*.
- For complete listings of SVG capabilities and attributes, read the *Scalable Vector Graphics 1.0 Recommendation* from the World Wide Web Consortium.
- One of the advantages of SVG is the ability to create images on a variety of platforms, including handheld devices. For a look at the likely subsets of SVG to be used this way, see the W3C's look at versions of SVG for small handheld devices, in *Mobile SVG: SVGTiny and SVGBasic*.
- As part of his look at transforming XML using Extensible Stylesheet Language Transformations, Doug Tidwell discusses the generation of SVG images from other data in his tutorial, *Transforming XML into SVG*.
- New SVG tools are rapidly emerging. For an idea of the state of SVG tools as of November 2001, read Antoine Quint's article, *SVG: Where Are We Now?*

* For another look at the basics of SVG, read *An Introduction to Scalable Vector Graphics* by J. David Eisenberg.
* Kip Hampton takes a look at using CGI to generate SVG images in his article *Creating Scalable Vector Graphics with Perl*.
* Test drive the new *IBM WebSphere Studio development environments* that deliver dynamic e-business applications with ease.
* *Real-world SVG* by Jackson West discusses some of the less theoretical and more practical aspects of using SVG.
* Adobe has gathered an impressive array of tutorials and samples in their *SVG Zone*.
* Get a list of *recognized color keywords* from the recommendation and save yourself some time.

Downloads

* Download the IBM prototype SVGView from *http://www.alphaworks.ibm.com/tech/svgview*.
* Download the Adobe SVG Viewer (version 3.0) from *http://www.adobe.com/svg/viewer/install/main.html*.
* Download the Batik SVG viewer and toolkit from the Apache Project at *http://xml.apache.org/batik/index.html*.
* Download an SVG converter from *http://www.svgfactory.com*.
* Download Jasc's WebDraw from *http://www.jasc.com/products/webdraw/wdrawdl.asp*.

Feedback

Please send us your feedback on this tutorial. We look forward to hearing from you!

Colophon

This tutorial was written entirely in XML, using the developerWorks Toot-O-Matic tutorial generator. The open source Toot-O-Matic tool is an XSLT stylesheet and several XSLT extension functions that convert an XML file into a number of HTML pages, a zip file, JPEG heading graphics, and two PDF files. Our ability to generate multiple text and binary formats from a single source file illustrates the power and flexibility of XML. (It also saves our production team a great deal of time and effort.)

You can get the source code for the Toot-O-Matic at www6.software.ibm.com/dl/devworks/dw-tootomatic-p. The tutorial Building tutorials with the Toot-O-Matic demonstrates how to use the Toot-O-Matic to create your own tutorials. developerWorks also hosts a forum devoted to the Toot-O-Matic; it's available at www-105.ibm.com/developerworks/xml_df.nsf/AllViewTemplate?OpenForm&RestrictToCategory=11. We'd love to know what you think about the tool.