VISUALIZING THE WEB

# RENDERING HTML5 ILLUSTRATION

Matthew David

Focal Press

# RENDERING HTML5 ILLUSTRATION

Matthew David

ELSEVIER

AMSTERDAM • BOSTON • HEIDELBERG • LONDON • NEW YORK • OXFORD
PARIS • SAN DIEGO • SAN FRANCISCO • SINGAPORE • SYDNEY • TOKYO

Focal Press is an imprint of Elsevier

Focal Press

Focal Press is an imprint of Elsevier
30 Corporate Drive, Suite 400, Burlington, MA 01803, USA
Linacre House, Jordan Hill, Oxford OX2 8DP, UK

© 2010 Elsevier, Inc. All rights reserved.

No part of this publication may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or any information storage and retrieval system, without permission in writing from the publisher. Details on how to seek permission, further information about the Publisher's permissions policies and our arrangements with organizations such as the Copyright Clearance Center and the Copyright Licensing Agency, can be found at our website: www.elsevier.com/permissions.

This book and the individual contributions contained in it are protected under copyright by the Publisher (other than as may be noted herein).

**Notices**
Knowledge and best practice in this field are constantly changing. As new research and experience broaden our understanding, changes in research methods, professional practices, or medical treatment may become necessary.

Practitioners and researchers must always rely on their own experience and knowledge in evaluating and using any information, methods, compounds, or experiments described herein. In using such information or methods they should be mindful of their own safety and the safety of others, including parties for whom they have a professional responsibility.

To the fullest extent of the law, neither the Publisher nor the authors, contributors, or editors, assume any liability for any injury and/or damage to persons or property as a matter of products liability, negligence or otherwise, or from any use or operation of any methods, products, instructions, or ideas contained in the material herein.

ISBN: 978-0-240-81384-4

For information on all Focal Press publications
visit our website at www.elsevierdirect.com

Working together to grow
libraries in developing countries

www.elsevier.com | www.bookaid.org | www.sabre.org

ELSEVIER BOOK AID International Sabre Foundation

# CONTENTS

# RENDERING HTML5 ILLUSTRATION

Tags are used in HTML5 to place and organize content at a level that is descriptive. This does not mean that the page will look good. Presentation of content on the page is controlled using Cascading Style Sheets Level 3, or CSS3, in HTML5.

There are times, however, when you need to present graphics, too. Typically, HTML has only provided support for pixel-based images in JPEG and GIF image format. With HTML5, you can now create mathematically generated images. The new formats are scalable vector graphics (SVG) and CANVAS. The difference between the two is that SVG is an XML-based language that describes how an image should be displayed in two-dimensional (2D) constructs. The CANVAS tag also describes 2D images, but it does so using JavaScript. The CANVAS tag also allows you to easily integrate interactivity within it using JavaScript.

In this article you will learn the following:

- The new image formats available in HTML5.
- How to draw using SVG.
- How to draw with CANVAS.
- How to add interactivity to CANVAS using JavaScript.

The goal at the end of this article is that you will understand how you can use the image formats in HTML5.

## The Tale of Web Image Formats

The Web is not a friendly place for a designer. For many years you have been limited to the number of file formats you can use. There are two predominant file formats used on the Web for creating graphics: JPEG and GIF.

## Bitmap Images: Using JPEG, GIF, and PNG Images on the Web

Both JPEG and GIF image formats are raster images created from pixels of individual color. Both have positives and negatives. JPEG images are an open standard managed by the Joint

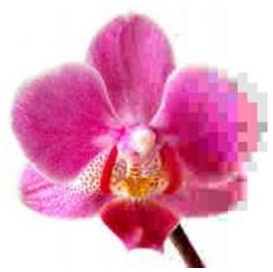


**Figure 3.1** This image is in JPEG format. The right side shows the pixel-by-pixel construction of the image.

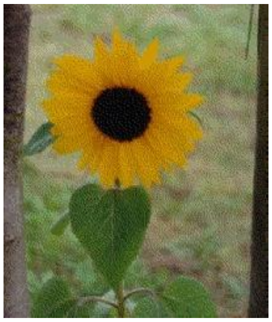


**Figure 3.2** The GIF image is using a Web-safe color palette of 256 colors. You can see by the grainy texture that the image is not photorealistic.

Photographers Expert Group. The JPEG file format allows you to create photorealistic images (Figure 3.1). A great place to go to view millions of JPEG images is Yahoo's Flickr. A JPEG image is identified with the extension of either JPEG or JPG.

The second file format used widely on the Internet is GIF, graphics interchange format. Unlike JPEG, which support millions of colors, the GIF file format only allows you to create images that support a color palette of 256 colors (Figure 3.2). On the face of it, the GIF format appears to be inferior to the JPEG format. However, the GIF format does have two features the JPEG format does not: setting transparency as a color and sequencing a series of images together to play back as a simple animation.

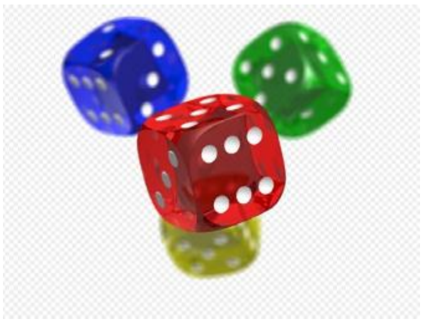
Both JPEG and GIF image formats, however, are now being superseded by a more sophisticated image format: PNG. Portable network graphics (PNG) are a raster-based file format that gives the best of both JPEG and GIF and a little more (Figure 3.3). A PNG image format will support 32-byte images for photorealistic presentation. Additionally, like GIF images, backgrounds in PNG images can be set to be transparent.

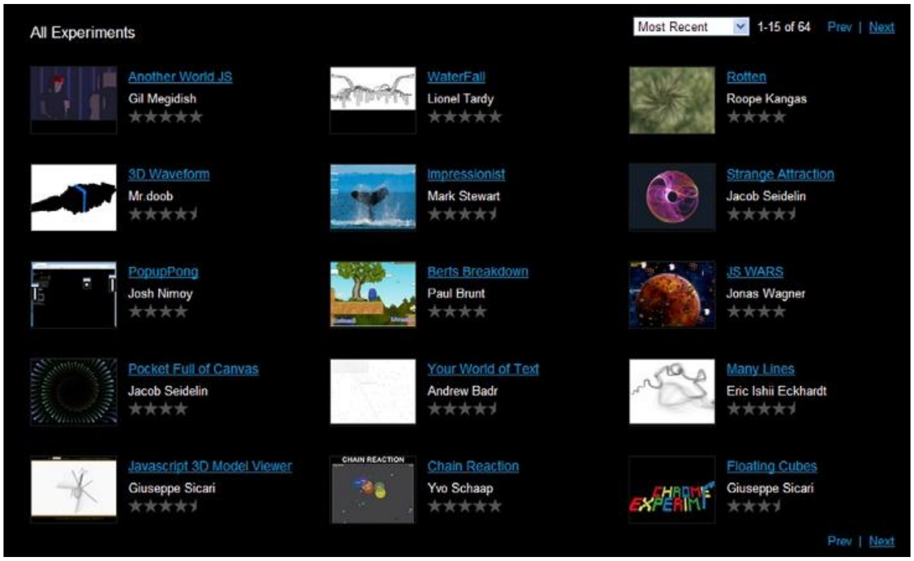
While PNG, GIF, and JPEG images are all great, it is difficult to programmatically change the graphical display of the images. For instance, you cannot create a bar chart using JPEG images that change as new data come in. HTML5 introduces two solutions that address this problem: SVG and CANVAS.

The CANVAS HTML5 element allows you to create bitmap images programmatically using JavaScript as the designer. Through this technique, complex animations and interactive solutions can be created. Google has established ChromeExperiments.com (*http://www.chromeexperiments.com/*) to demonstrate powerful CANVAS and JavaScript experiments (Figure 3.4).

The second technology, SVG (scalable vector graphics), is a vector-based technology that enables you to create images and animation using XML syntax similar to HTML. SVG started

**Figure 3.3** PNG graphics allow you to have the best of JPEG and GIF technologies in a single format.





Another World JS
Gil Megidish

WaterFall
Lionel Tardy

Rotten
Roope Kangas

3D Waveform
Mr.doob

Impressionist
Mark Stewart

Strange Attraction
Jacob Seidelin

PopupPong
Josh Nimoy

Berts Breakdown
Paul Brunt

JS WARS
Jonas Wagner

Pocket Full of Canvas
Jacob Seidelin

Your World of Text
Andrew Badr

Many Lines
Eric Ishii Eckhardt

Javascript 3D Model Viewer
Giuseppe Sicari

Chain Reaction
Yvo Schaap

Floating Cubes
Giuseppe Sicari

**Figure 3.4** ChromeExperiments.com showcases how far you can take technologies such as SVG and CANVAS.

as an Open Standard in 1999. The support for SVG started out patchy, but, with the release of FireFox, that all changed. FireFox 1.5 introduced support for SVG, with other competing browsers such as Chrome and Safari rapidly adopting the standard through support of the WebKit Web Browser project. SVG as an alternative vector graphics technology is being widely adopted. As an example, Wikipedia.org has over 250,000 SVG images on its site.

## Creating SVG Graphics

If you are comfortable working with HTML code then you will feel comfortable working with SVG. SVG is an XML-based drawing language that allows you to describe your drawing using standard XML elements. For instance, the following code is describing how to create a star shape. Figure 3.5 shows the resulting SVG drawing.

```
<?xml version=”1.0” standalone=”yes”?>
<svg version=”1.1”
viewBox=”0.0 0.0 720.0 540.0”
fill=”none” stroke=”none”
```

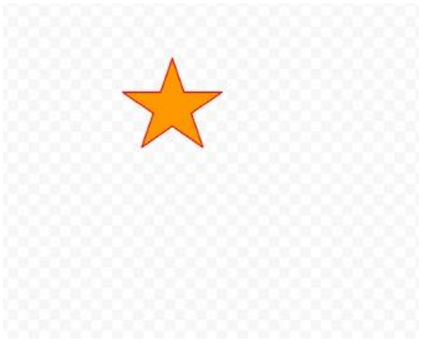


**Figure 3.5** A star drawn using SVG XML syntax.

```
stroke-linecap="square"
stroke-miterlimit="10"
xmlns="http://www.w3.org/2000/svg"
xmlns:xlink="http://www.w3.org/1999/xlink">
<path
d="M240 148L298 148L316 96L334 148L392 148L345
180L363 232L316 200L269 232L287 180Z"
fill-rule="nonzero"
fill="#ff9900"
stroke="#ff0000"
stroke-width="2.0"
stroke-linejoin="round"
stroke-linecap="butt">
</path></svg>
```

If you want to change the fill color inside of the star, you only need to change the value of the fill property to a new color. Say you would prefer a red star; simply change the fill value to red, as follows:

```
fill=red
```

Figure 3.6 shows that the star is now red.

Allowing the browser to control the color, shape, and visual elements of the SVG image allows you to write programs that dynamically control the SVG illustrations.

SVG comes with some fundamental benefits:

- Images scale easily.
- SVG is accessible.
- Search engines can easily read and understand SVG images.

These benefits make using SVG very compelling.

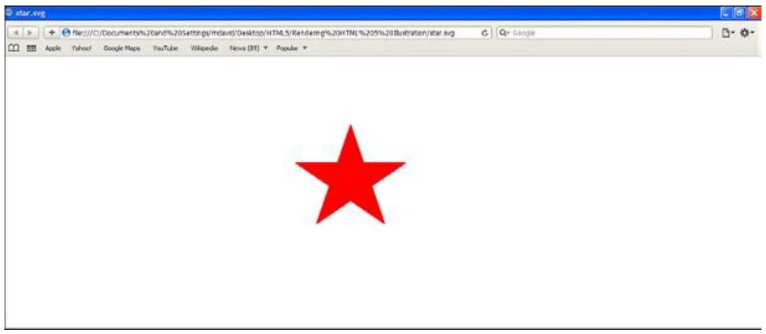


**Figure 3.6** Any element used to build an SVG drawing can be easily edited.

## The Fundamentals of Creating SVG Images and Adding Them to Your Web Pages

Unlike traditional drawing, SVG can be "drawn" all in code. You can use your favorite text editor to create any type of SVG illustration. The easiest way to manage SVG drawings is to save each illustration to a text file with the extension SVG. You can then treat your SVG drawings as if they are image files like JPEG or PNG files.

All SVG files will start with a line declaring the document is an XML file. The following line should be placed at the start of all your SVG documents:

```
<?xml version="1.0" standalone="yes"?>
```

Following the XML declaration is some information explaining the SVG document. The first line specifies which version of SVG you are using. The most commonly adopted version is 1.1:

```
<svg version="1.1"
```

The viewBox property identifies the size of the canvas you are working with. The viewBox is constructed of four properties that identify the X and Y coordinates of the viewBox and width and height.

```
viewBox="0.0 0.0 300.0 800.0"
```

You can specify drawing attributes that should be used for all objects in the image in the opening SVG properties. Here all objects in the illustration will, by default, have no fill or stroke, and a square line will be used to draw images with a stroke miter of 5.

```
fill="none"

stroke="none"

stroke-linecap="square"

stroke-miterlimit="5"
```

The final two attributes provide links to the SVG namespace standard.

```
xmlns="http://www.w3.org/2000/svg"

xmlns:xlink="http://www.w3.org/1999/xlink">
```

Drawing is managed through a number of elements, with PATH being the main one. The role of the PATH element is to draw out the specific coordinates of an image point by point. In the following example, a single line is drawn.

```
<path d="M8 16L776 0" fill-rule="nonzero"
stroke="blue" stroke-width="5" stroke-
linejoin="round" stroke-linecap="butt">
</path>
</svg>
```

A closing SVG element completes your code. Your drawing can now be saved and added to a web page.

When you have completed an SVG drawing there are several ways in which you can add SVG images to your web page. Unlike JPEG, GIF, and PNG files, you cannot use the IMG element to add an SVG drawing to your web page. You do, however, have three alternative methods.

The first is to use the OBJECT element:

```
<object data="star.svg" width="300" height="800"/>
```

The OBJECT tag has several attributes. The most important is the `data` attribute that references the location of the SVG file. The `width` and `height` attributes define the space used on the screen for the SVG drawing.

A second method you can use to add SVG images to your web page is through the use of the IFRAME element. Typically, you use the IFRAME element to load an external web page, but you can also load an SVG image directly into your web page. Here is an example:

```
<iframe src="star.svg" width="300" height="800"></
iframe>
```

These two methods for embedding SVG images into your web page are relatively easy to use and are not much more complicated to use than the IMG element.

The third method of adding SVG images to a web page is to insert the SVG XML directly into the HTML code itself. The following code is HTML saved as a web page. There is no need to use separate SVG files in this example.

```
<html>
  <head>
    <title>SVG embedded inline in XHTML</title>
  </head>
  <body>
    <h1>SVG embedded inline in XHTML</h1>
    <svg xmlns="http://www.w3.org/2000/svg"
width="300" height="800">
<path
d="M240 148L298 148L316 96L334 148L392 148L345
180L363 232L316 200L269 232L287 180Z"
fill-rule="nonzero"
```

```
fill="#ff9900"
stroke="#ff0000"
stroke-width="2.0"
stroke-linejoin="round"
stroke-linecap="butt">
</path> </svg>
  </body>
</html>
```

Adding the SVG coded directly to a web page only requires that you use SVG element tags inside of your HTML.

At the end of the day, it is really up to you as to how you want to add SVG images to your web pages.

## Understanding the Basics of Creating Shapes

As with HTML, SVG is built of elements. The difference between SVG and HTML is that the elements in SVG are used to construct images. The main elements you will use in building your drawings are:

- LINE—for defining lines
- POLYLINE—for defining shapes constructed of lines
- RECT—for defining rectangles
- CIRCLE—for defining circles
- ELLIPSE—for defining ellipses
- POLYGON—for defining polygons
- PATH—for defining arbitrary paths

The most basic drawing element for SVG is a line. To define a straight line you need to declare where in the viewBox property the line starts on the X and Y axes and where the line ends on the X and Y axes. This is referred to as X1, Y1 and X2, Y2. Here is an example of a straight line.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="100%" height="100%" version="1.1"
xmlns="http://www.w3.org/2000/svg">
<line x1="25" y1="150" x2="300" y2="150"
style="stroke:red;stroke-width:10"/>
```

In this example the line starts 25 pixels in from the left side of the browser window, the line is 300 pixels long, and the line is horizontal along the Y axis. Figure 3.7 is how it looks in your web browser with additional CSS styling to emphasize the line.

**Figure 3.7** A line is drawn in SVG using the LINE element.

You can easily modify the settings for the X and Y axes to change the position of your line. In the following SVG code, the line is changed to run vertically.

```
<?xml version=”1.0” standalone=”no”?>
<!DOCTYPE svg PUBLIC “-//W3C//DTD SVG 1.1//EN”
“http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd”>
<svg width=”100%” height=”100%” version=”1.1”
xmlns=”http://www.w3.org/2000/svg”>
<line x1=”300” y1=”310” x2=”300” y2=”10”
style=”stroke:red;stroke-width:10”/>
</svg>
```

Figure 3.8 shows the results and how the line is displayed.

The POLYLINE element extends the functionality of the LINE element to enable you to build drawings created with lines. The construction is created through valued pairs of X and Y coordinates using POLYLINE’s `point` attribute. Here is an example of creating a square shape using the POLYLINE element.

```
<?xml version=”1.0” standalone=”no”?>
<!DOCTYPE svg PUBLIC “-//W3C//DTD SVG 1.1//EN”
“http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd”>
<svg width=”100%” height=”100%” version=”1.1”
xmlns=”http://www.w3.org/2000/svg”>
<polyline points=”5,5 5,150 150,150 150,5 5,5”
style=”fill:white;stroke:red;stroke-width:2”/>
</svg>
```

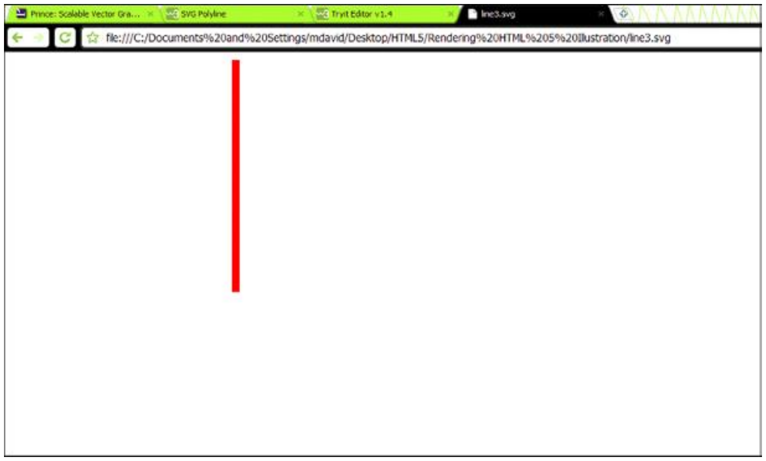

Prince: Scalable Vector Gra...
SVG Polyline
Tryit Editor v1.4
line3.svg
file:///C:/Documents%20and%20Settings/mdavid/Desktop/HTML5/Rendering%20HTML%205%20Illustration/line3.svg

**Figure 3.8** Changing the XML element values changes the display of the line.

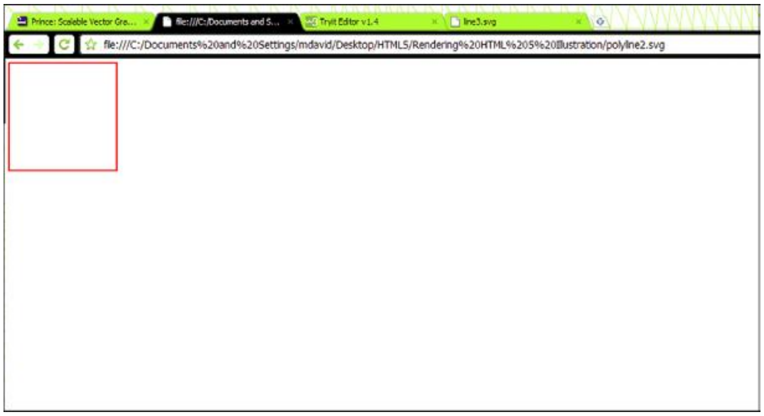

Prince: Scalable Vector Gra...
file:///C:/Documents and S...
Tryit Editor v1.4
line3.svg
file:///C:/Documents%20and%20Settings/mdavid/Desktop/HTML5/Rendering%20HTML%205%20Illustration/polyline2.svg

**Figure 3.9** The POLYLINE element can be used to draw images with straight lines. In this case a square is drawn.

Figure 3.9 shows the final drawing.

You can create more complex shapes with the POLYLINE element. In this example a set of stairs is created. All you have to remember is that the first value in the value pair is the X axis and the second value is the Y axis. Figure 3.10 shows the results.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="100%" height="100%" version="1.1"
xmlns="http://www.w3.org/2000/svg">
<polyline points="5,0 5,40 40,40 40,80 80,80 80,120"
style="fill:white;stroke:red;stroke-width:2"/>
</svg>
```

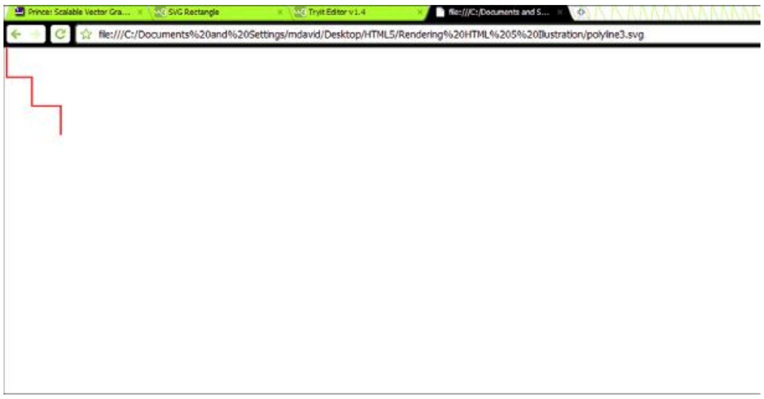

Prince: Scalable Vector Gra...
SVG Rectangle
Tryit Editor v1.4
file:///C:/Documents and S...
file:///C:/Documents%20and%20Settings/mdavid/Desktop/HTML5/Rendering%20HTML%205%20Illustration/polyline3.svg

**Figure 3.10** A drawing created of straight lines can be created using the POLYLINE element.

Rectangle shapes can be created in SVG using the RECT element. The RECT element has two attributes, `width` and `height`. The following SVG adds a rectangle of width 400 pixels and height 400 pixels. Additional styling using CSS has been added to the drawing so you can see it. The results are shown in Figure 3.11.

```
<?xml version=”1.0” standalone=”no”?>
<!DOCTYPE svg PUBLIC “-//W3C//DTD SVG 1.1//EN”
“http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd”>
<svg width=”100%” height=”100%” version=”1.1”
xmlns=”http://www.w3.org/2000/svg”>
<rect width=”400” height=”400”
style=”fill:red;stroke-width:5;
stroke:yellow”/>
</svg>
```

Creating circles is similar to creating rectangles in SVG. The difference is you use the CIRCLE element. At its most basic, the CIRCLE element only requires that you define the radius of the circle using an `R` attribute. The following SVG code draws a circle with a radius of 150 pixels.

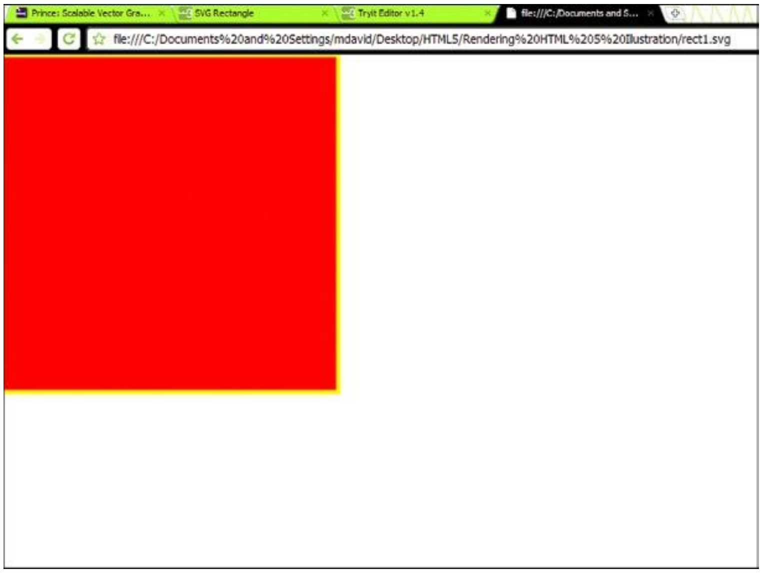

Prince: Scalable Vector Gra...
SVG Rectangle
Tryit Editor v1.4
file:///C:/Documents and S...
file:///C:/Documents%20and%20Settings/mdavid/Desktop/HTML5/Rendering%20HTML%205%20Illustration/rect1.svg

**Figure 3.11** The RECT element allows you to easily create rectangle shapes such as this square.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="100%" height="100%" version="1.1"
xmlns="http://www.w3.org/2000/svg">
<circle r="150" stroke="yellow"
stroke-width="5" fill="red"/>
</svg>
```

Figure 3.12 shows the results of the CIRCLE element in more detail.

As you can see in the figure, defining only the radius forces most of the circle to drop off the top-left corner of the browser window. To correct this you can use two additional, optional attributes, CX and CY, to define the X and Y axes positions of the circle on the screen.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="100%" height="100%" version="1.1"
xmlns="http://www.w3.org/2000/svg">
<circle cx="160" cy="160" r="150" stroke="yellow"
stroke-width="5" fill="red"/>
</svg>
```

Figure 3.13 shows the use of these attributes.

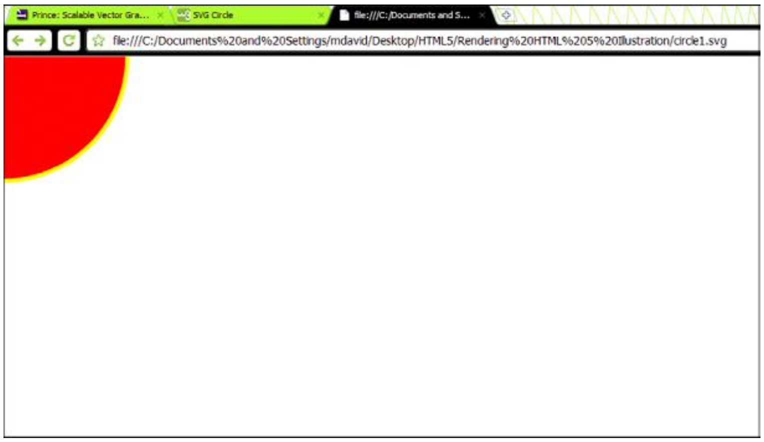

Prince: Scalable Vector Gra...
SVG Circle
file:///C:/Documents and S...
file:///C:/Documents%20and%20Settings/mdavid/Desktop/HTML5/Rendering%20HTML%205%20Illustration/circle1.svg

**Figure 3.12** The CIRCLE element allows you to draw circles on the screen.

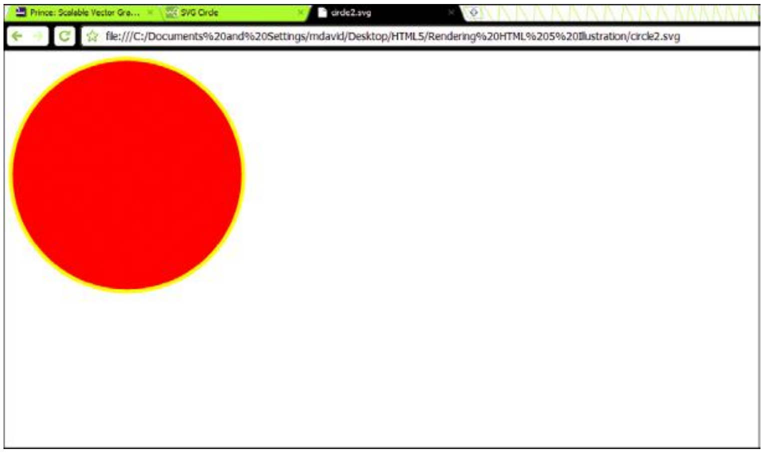

Prince: Scalable Vector Gra...
SVG Circle
circle2.svg
file:///C:/Documents%20and%20Settings/mdavid/Desktop/HTML5/Rendering%20HTML%205%20Illustration/circle2.svg

**Figure 3.13** Using the `CX` and `CY` attributes enables you to control where on the screen the CIRCLE element is placed.

The ELLIPSE element extends the functionality of the CIRCLE element by allowing you to control radius along the X and Y axes using the `RX` and `RY` attributes. You will see in the following code that the ELLIPSE element also leverages the CIRCLE element's `CX` and `CY` attributes to position the ellipse in the web browser. Figure 3.14 shows the results

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="100%" height="100%" version="1.1"
xmlns="http://www.w3.org/2000/svg">
<ellipse cx="300" cy="150" rx="250" ry="120"
```

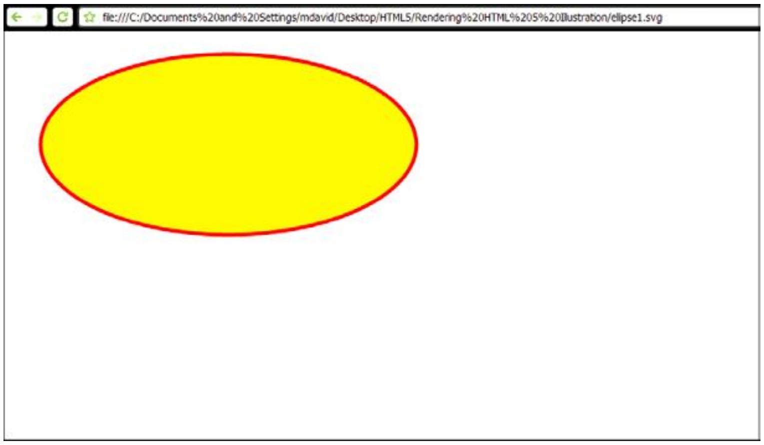

file:///C:/Documents%20and%20Settings/mdavid/Desktop/HTML5/Rendering%20HTML%205%20Illustration/ellipse1.svg

**Figure 3.14** An ellipse can be created using the ELLIPSE element.

```
style="fill:yellow;
stroke:red;stroke-width:5"/>
</svg>
```

A POLYGON shape is similar to the POLYLINE element. Using X and Y value pairs you can draw whole polygon shapes. The following is an example of a triangle. Figure 3.15 shows the results.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="100%" height="100%" version="1.1"
xmlns="http://www.w3.org/2000/svg">
<polygon points="220,100 290,220 150,220"
style="fill:yellow;
stroke:red;stroke-width:5"/>
</svg>
```

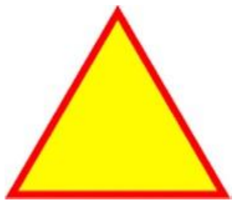


**Figure 3.15** A triangle is created using the POLYGON element.

The most complex drawing element you will create using SVG is the PATH element. Each drawing you create with the PATH element is built using a series of special codes that explain where the line is supposed to move to on the screen. Those codes are:

- M = move to
- L = line to
- H = horizontal line to
- V = vertical line to
- C = curve to
- S = smooth curve to
- Q = quadratic Belzier curve to
- T = smooth quadratic Belzier curve to
- A = elliptical arc to
- Z = close path to

The following code creates a smiley face illustration using the PATH element and the codes above to create the drawing. Figure 3.16 shows the results from the code.

```
<?xml version="1.0" standalone="yes"?>
<svg version="1.1" viewBox="0.0 0.0 1152.0 864.0"
fill="none" stroke="none" stroke-linecap="square"
stroke-miterlimit="10" xmlns="http://www.
w3.org/2000/svg" xmlns:xlink="http://www.
w3.org/1999/xlink">
<path d="M56 108L56 108C56 66 92 32 136 32C180
32 216 66 216 108C216 150 180 184 136 184C92
184 56 150 56 108Z" fill-rule="nonzero"
fill="#ffff00"></path>
```

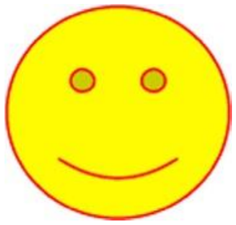


**Figure 3.16** The POLYLINE element allows you to create complex images such as this smiley face.

```
<path d="M102 85C102 81 106 77 110 77C115 77
119 81 119 85C119 90 115 93 110 93C106 93 102
90 102 85M153 85C153 81 157 77 162 77C166 77
170 81 170 85C170 90 166 93 162 93C157 93 153
90 153 85" fill-rule="nonzero" fill="#cccc00"
stroke="#ff0000" stroke-width="2.0" stroke-
linejoin="round" stroke-linecap="butt"></path>
<path d="M93 141Q136 169 179 141" fill-rule=
"nonzero" stroke="#ff0000" stroke-width="2.0"
stroke-linejoin="round" stroke-linecap=
"butt"></path>
<path d="M56 108L56 108C56 66 92 32 136 32C180 32
216 66 216 108C216 150 180 184 136 184C92 184 56
150 56 108Z" fill-rule="nonzero" stroke="#ff0000"
stroke-width="2.0" stroke-linejoin="round"
stroke-linecap="butt"></path>
</svg>
```

As you can see, it is quite complex to create PATH-defined illustrations. For this reason, it is recommended that you use an SVG drawing tool to create PATH-based illustrations (more on that later).

## Adding CSS-Based Color

SVG is a technology that allows you to create drawings. To add color to those drawings, however, you leverage Cascading Style Sheets. There is no need to use a different technology for applying color, as CSS and SVG are partners in HTML5. Both have strengths that can be enhanced with each other.

To provide an example, let's look back at the ellipse drawing created earlier (see Figure 3.14).

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="100%" height="100%" version="1.1"
xmlns="http://www.w3.org/2000/svg">
<ellipse cx="300" cy="150" rx="250" ry="120"
style="fill:yellow;
stroke:red;stroke-width:5"/>
</svg>
```

After the ellipse image is drawn there is a `style` attribute. The `style` attribute in SVG allows you to add a CSS style to the image. In HTML you have a `style` attribute that behaves exactly the same.

Modifying the `style` attribute will visually change the presentation of the ellipse. The following example changes the fill to blue and the stroke color to gray. Figure 3.17 shows the results.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="100%" height="100%" version="1.1"
xmlns="http://www.w3.org/2000/svg">
<ellipse cx="300" cy="150" rx="250" ry="120"
style="fill:blue;
stroke:gray;stroke-width:5"/>
</svg>
```

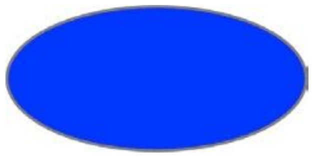


**Figure 3.17** CSS is used to set the visual appearance of an SVG drawing.

Both the Fill and Stroke properties control the color of the inside of an image. In this example a CSS color name is used, but you can use any of the color formats you use to control CSS, including the following:

- Color name—you can have names for colors such as brown, black, red, or cyan.
- Full hexadecimal—a hexadecimal value comprised of six alpha-numeric values.
- Short hexadecimal—a hexadecimal value comprised of three alpha-numeric values.
- RGB—a combination of red, green, and blue values.
- RGBA—a combination of red, green, and blue values with a transparency value (alpha).
- HSL—a combination of hue, saturation, and lightness.
- HSLA—a combination of hue, saturation, and lightness with a transparency value (alpha).

In addition to using CSS colors you can use any of the following measurements:

- cm—centimeter
- in. —inch
- mm—millimeter
- pc—pica (1 pica = 12 points)
- pt—point (1 point = 1/72 inch)
- px—pixels

Through leveraging CSS you can change the stroke of the ellipse using short hexadecimal and the measurement in CM.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="100%" height="100%" version="1.1"
```

```
xmlns="http://www.w3.org/2000/svg">
<ellipse cx="300" cy="150" rx="250" ry="120"
style="fill:blue;
stroke:#999;stroke-width:1cm"/>
</svg>
```

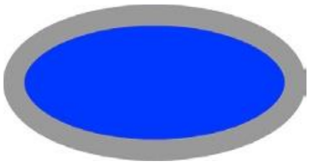


**Figure 3.18** Standard CSS colors and measurements can be used to control the presentation of a drawing.

Figure 3.18 shows how CSS colors and measurements can be used.

To add a linear of radial gradient to an SVG drawing you need to use specific SVG gradient elements.

## Applying Gradients to SVG Images

SVG employs a great technique that allows you to reuse a gradient definition over one or more images in your SVG illustration. This is done using either the LINEARGRADIENT or RADIALGRADIENT element types. Both gradients allow you to define the horizontal and vertical colors and direction of the gradient.

Let's look first at linear gradients. The LINEARGRADIENT element is constructed by five different attributes that define over a linear direction how the gradient will behave. The first attribute you need to provide information for is the `ID` attribute, which allows you to give your gradient a name you can use to reference from your drawing.

For a linear gradient you can draw your gradient moving over an X–Y axis direction. To determine the direction of the gradient you have to specify the start and end X and start and end Y axes points. The following illustrates a left–right gradient:

```
<linearGradient x1="0%" y1="0%" x2="100%" y2="0%">
```

To create a vertical gradient you change the Y and X axes to:

```
<linearGradient x1="0%" y1="100%" x2="0%" y2="0%">
```

You can see the difference between the two numbers is changing the X or Y axis to 100%. See Figure 3.19.

Changing the X and Y axes percentages will change how the gradient is drawn. Adding color to the gradient is the next step. To do this, you create a list of two or more colors using the STOP element. For instance, to create a simple yellow-to-red gradient color change you will add two STOP elements as shown in the following.

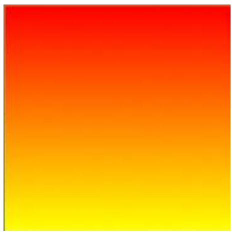


**Figure 3.19** Linear gradients can be applied in SVG images using the LINEARGRADIENT element.

```
<stop offset="0%" style="stop-color:yellow;stop-
opacity:1"/>
<stop offset="100%" style="stop-color:red;stop-
opacity:1"/>
```

The offset attribute dictates where in the drawing the gradient starts. The example above draws a smooth gradient color change over the space of the image. The style attribute allows you to list any CSS-specific color. You can add the two colors to the LINEARGRADIENT in the following example.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="100%" height="100%" version="1.1"
xmlns="http://www.w3.org/2000/svg">
<defs>
<linearGradient id="yellow_red" x1="0%" y1="100%"
x2="0%" y2="0%">
<stop offset="0%" style="stop-color:yellow;stop-
opacity:1"/>
<stop offset="100%" style="stop-color:red;stop-
opacity:1"/>
</linearGradient>
</defs>
<rect width="400" height="400"
style="fill:url(#yellow_red);
stroke:yellow"/>
</svg>
```

You can see that the rectangle image uses a URL string to find the style called #yellow_red. The yellow_red color style is the name of the gradient. See Figure 3.20.

Radial gradients are similar to linear gradients. The difference is that you define essentially two circles—an outer and inner circle—with the radial gradient. As with linear gradients, the RADIALGRADIENT element requires a valid ID name to identify the gradient. Following that, you have five attributes to define the inner and outer circle and radius. Following is an example where the CX and CY attributes are the outer circle, the R is the radius, and the FX and FY attributes are the inner circle.

```
<radialGradient id="yellow_red" cx="50%" cy="50%"
r="50%"
fx="50%" fy="50%">
```

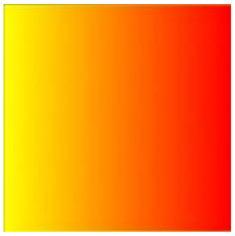


**Figure 3.20** Linear gradients can be drawn horizontally.

The colors for the gradient are defined using a STOP list. The following code shows the radial gradient applied to a rectangle.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="100%" height="100%" version="1.1"
xmlns="http://www.w3.org/2000/svg">
<defs>
<radialGradient id="yellow_red" cx="50%" cy="50%"
r="50%"
fx="50%" fy="50%">
<stop offset="0%" style="stop-color:yellow;stop-
opacity:1"/>
<stop offset="100%" style="stop-color:red;stop-
opacity:1"/>
</radialGradient>
</defs>
<rect width="400" height="400"
style="fill:url(#yellow_red);
stroke:yellow"/>
</svg>
```

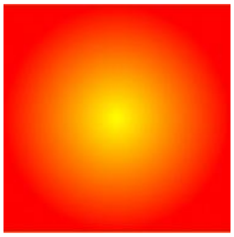


**Figure 3.21** A radial gradient applied to a rectangle shape.

Figure 3.21 shows the results.

Both the linear and radial gradients can have more than two colors. The following code (see Figure 3.22) has four colors.

```
<stop offset="0%" style="stop-color:yellow;stop-
opacity:1"/>
<stop offset="25%" style="stop-color:red;stop-
opacity:1"/>
<stop offset="50%" style="stop-color:blue;stop-
opacity:1"/>
<stop offset="100%" style="stop-color:black;stop-
opacity:1"/>
```

In addition, you can link multiple images to a single gradient. The following SVG code links a circle and rectangle to the same gradient.

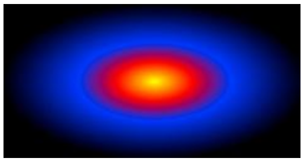


**Figure 3.22** Multiple colors can be created to change the gradient.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="100%" height="100%" version="1.1"
xmlns="http://www.w3.org/2000/svg">
```

```
<defs>
<radialGradient id=”yellow_red” cx=”50%” cy=”50%”
r=”50%” fx=”50%” fy=”50%”>
<stop offset=”0%” style=”stop-color:yellow;stop-
opacity:1”/>
<stop offset=”25%” style=”stop-color:red;stop-
opacity:1”/>
<stop offset=”50%” style=”stop-color:blue;stop-
opacity:1”/>
<stop offset=”100%” style=”stop-color:black;stop-
opacity:1”/>
</radialGradient>
</defs>
<rect width=”500” height=”250”
style=”fill:url(#yellow_red);
stroke:yellow”/>
<circle cx=”250” cy=”250” r=”180” stroke=”black”
stroke-width=”2” fill=”url(#yellow_red)” />
</svg>
```

The results are shown in Figure 3.23.

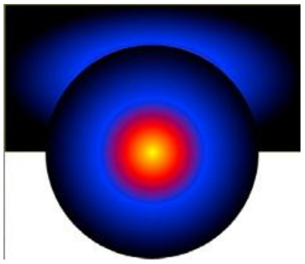


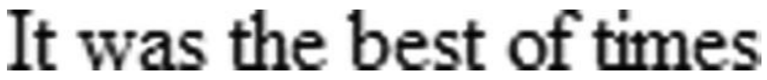
**Figure 3.23** Gradient color definitions can be shared among shapes.

## Adding Text to Your SVG Drawings

Text can be added to your SVG drawings using the TEXT element. At its most basic, all you need to do is add the TEXT element to your SVG document, as shown in the following code and Figure 3.24.

```
<svg xmlns=”http://www.w3.org/2000/svg”
    xmlns:xlink=”http://www.w3.org/1999/xlink”>
    <text x=”100” y=”40”>It was the best of times</
text>
</svg>
```

The `X` and `Y` attributes specify where on the screen the text will appear. Formatting of the text is controlled using CSS in the `style` attribute. Text can have the following styles applied to it:



It was the best of times

**Figure 3.24** Text can be easily inserted into an SVG drawing use the TEXT element.

- Font-family—the name of the font
- Font-size—the size of the font
- Kerning—the space between letters
- Stroke—the outside color of a font
- Fill—the inside color of a font

Following is an example SVG code showing text formatting.

```
<svg xmlns="http://www.w3.org/2000/svg"
     xmlns:xlink="http://www.w3.org/1999/xlink">
     <text x="100" y="40"
           style="font-family: Arial;
                  font-size  : 24pt;
                  stroke     : red;
                  fill       : yellow;
                  kerning    : 3; "
       >It was the best of times</text>
</svg>
```

Figure 3.25 shows the results of the code.

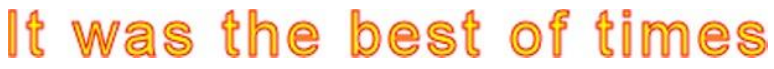


**Figure 3.25** Formatted SVG text.

Text is treated as simply another image type in SVG. This allows you to add some additional visual effects. As an example, you can use a gradient as the FILL style for your text. The following SVG code exaggerates the size of the text to show a gradient fill (see Figure 3.26).

```
<svg xmlns="http://www.w3.org/2000/svg"
     xmlns:xlink="http://www.w3.org/1999/xlink">
<radialGradient id="yellow_red" cx="50%" cy="50%"
r="50%" fx="50%" fy="50%">
<stop offset="0%" style="stop-color:yellow;stop-
opacity:1"/>
<stop offset="25%" style="stop-color:red;stop-
opacity:1"/>
<stop offset="50%" style="stop-color:blue;stop-
opacity:1"/>
<stop offset="100%" style="stop-color:black;stop-
opacity:1"/>
</radialGradient>
```

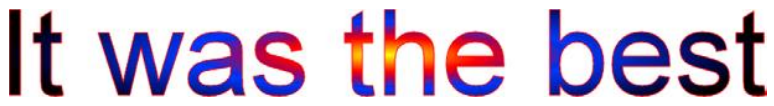


**Figure 3.26** Both linear and radial gradients can be used to style text.

```
  <text x="100" y="140"
        style="font-family: Arial;
               font-size  : 96pt;
               stroke     : red;
               fill       : url(#yellow_red);
               "
  >It was the best</text>
</svg>
```

SVG supports a method where you can embed a font into the document. Embedding a font in SVG is, however, tricky. The challenge is that to embed a font you must specify the exact shape of each font glyph you use. A glyph is a shape matched to a key on your keyboard. Figure 3.27 is a glyph of the letter "A."

**Figure 3.27** A glyph of the letter "A."

SVG's GLYPH element draws the outline of the font and ties it to a character. This can get complex very quickly. The following is an example of what you will need to duplicate just the letter "a" as a reusable glyph in SVG.

```
<glyph unicode="a" glyph-name="a" horiz-adv-x="577"
d="M595 -324H-36V898H595V-324ZM117 27Q123 25 130
20T146 29Q154 41 159 59T166 86Q169 96 167 103T172
113Q181 115 185 106T202 97Q213 97 227 102T273
108Q306 108 320 105T347 109Q359 115 370 123T387

127Q388 126 393 118T403 101T412 82T417 67Q420 57
426 41T451 23Q471 19 477 25T486 47Q491 60 478
71T460 96Q457 102 448 126T426 178T404 235T387
279Q382 296 372 321T351 371T332 416T321 443Q318
448 317 460T313 485T307 510T297 528Q291 533 284
528T271

516T261 500T254 485Q253 473 257 464T261 448Q261
442 255 434T245 414Q241 405 240 397T238 381Q238
373 220 338T189 265Q177 230 176 217T172 194Q171
186 166 169T151 138Q147 128 138 112T120 78T110
44T117 27ZM366 235Q375 218 377 201T376 169Q375 163

368 162T351 163T329 168T310 171Q280 171 269
176Q265 180 256 176T228 165Q208 158 201 159T187
164Q183 165 186 172T195 189T207 208T215 228Q217
237 224 257T240 299T255 337T261 355Q261 359 256
362T245 369Q242 370 249 377T262 392Q269 402 271
408T282

411Q286 409 298 385T324 330T351 271T366 235ZM243
222L250 238Q254 246 257 253T261 264Q267 272 268
284Q269 292 272 302Q267 298 262 285T253 256Q247
238 243 222ZM294 186Q292 193 288 201T277 207Q272
205 265 205H257L294 186ZM201 78L340 83Q328 86 306
```

```
86T260 83Q231 82 201 78ZM194 70Q190 60 188 53Q181
36 176 30Q172 27 169 25T165 18Q162 15 162 12Q175
23 181 31Q196 52 194 70ZM491 126Q494 122 489
136T475 174T452 232T424 303Q390 388 344 496Q383
394 414 314Q429 278 442 245T465 187T481 144T491
126Z"/>
```

To use a full alphabet you will need to create the lowercase and uppercase for each character on the keyboard. Your files for a simple font will get very large very quickly.

## Adding Interactivity and Javascript to Your SVG Drawings

You can use JavaScript to add interactivity to your SVG illustrations. You do this using the SCRIPT element in your SVG document. The following example adds a JavaScript that changes the color of a rectangle shape each time you click on it.

```
<?xml version="1.0" encoding="UTF-8"
standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN"
"http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/
svg10.dtd">
<svg xmlns="http://www.w3.org/2000/svg"
xmlns:xlink="http://www.w3.org/1999/xlink"
width="800" height="800">
  <script type="text/ecmascript">
  <![CDATA[
    function randomColor(evt) {
    var red = Math.round(Math.random() * 255);
    var green = Math.round(Math.random() * 255);
    var blue = Math.round(Math.random() * 255);
    evt.target.setAttributeNS(null,"fill","rgb("+
red +","+ green+","+blue+")");
    }
  ]]>
  </script>
    <rect id="myBlueRect" width="600"height="600"
x="40" y="20" fill="orange" onClick="randomColor
(evt)"/>
</svg>
```

You can see that the JavaScript is wrapped in a CDATA element. This allows the script to be correctly interpreted by the JavaScript engine running in the web browser. The `onClick` event attribute links the name of the JavaScript function with the rectangle.

Through using JavaScript you can do a lot with SVG. Some great sites that push the interactive limits of JavaScript and SVG integration are:

- *http://raphaeljs.com/*
- *http://svgkit.sourceforge.net/*
- *http://www.liquidx.net/plotkit/*
- *http://www.lutanho.net/svgvml3d/index.html*
- *http://code.google.com/p/svgweb/*

Each of these sites gives you libraries of JavaScript code that allow you to complete complex, interactive SVG presentations.

## Leveraging SVG Drawing Tools

If you have gotten this far then you have realized that drawing with SVG is complex. There are, unfortunately, very few illustration tools you can use to create SVG drawings. Fortunately, the few tools that are on the market just happen to be very good.

Adobe's Illustrator has supported, since CSS2, the ability to export any illustration in SVG format. This is great news, as you can take complex drawings and import them directly to SVG.

While Illustrator will export to SVG, the Open Source project InkSpace will save and edit SVG files directly. InkSpace is not as easy to use as Illustrator, but it is free, and it is certainly easier to create SVG illustrations with InkSpace than by scratch.

Sketsa is a Java-based SVG drawing tool. The tool itself is quite basic, but, again, it is better than nothing.

Finally, if you use Google's Docs to create documents online then you will be interested to know that the "Insert Image" feature uses SVG to create the images. Additionally, if you use a text, then Google uses the complex Glyph editor to embed the fonts for you.

The good news is that there are tools you can use to create SVG illustrations. The bad news is that there are few tools you can use to visually apply interactivity to your SVG drawings. With SVG becoming more popular for sites such as Google Maps and Wikipedia and now as a first-class citizen in HTML5, we should expect SVG authoring tools to become more common.

## Adding the CANVAS Element to Your Web Page

There is a royal battle happening in the Web-o-sphere between technologies that enable you to create cool, interactive animations online. The current "king" is Adobe's Flash, with Microsoft's SilverLight coming in guns blazing. The "black horse" contender is the emerging HTML5 standard. Baked into HTML5 is a new element called CANVAS. Not sure what CANVAS is? Do you own a Mac? Most of the widgets you run on your dashboard are built with HTML5's CANVAS element.

The CANVAS element gives you the ability to build Adobe Flash–like applications without having to use Flash. It is in the early stages of development, but some of the things you can already accomplish are very impressive. If you are running Google's Chrome, FireFox, or Safari, then you will want to check out *http://www.chromeexperiments.com/*, a site that pushes the capabilities of what can be done in your browser (Figure 3.28). In particular, look at the CANVAS experiments. Is it me, or do they look very Flash-like?

In many ways, CANVAS looks and feels very similar to SVG. The very valid question is: Why two technologies that are the same? There is a fundamental difference between CANVAS and SVG. SVG is a drawing technology that creates vector images. CANVAS, on the other hand, dynamically creates bitmap images. You can think of CANVAS as a programmable version of JPEG/PNG images.

Unlike SVG, a technology that has been maturing for a decade as a separate standard and only recently became included as part of HTML5, CANVAS was created as part of HTML5. The CANVAS technology was originally created by Apple to help in the creation of desktop widgets for the Mac OSX operating system. The technology was quickly included into WebKit, the technology Apple leverages to power its Safari web browser, and CANVAS has been adopted by Mozilla's FireFox, Opera's Browser, and Google's Chrome. CANVAS is a powerful drawing tool that aligns with competing technologies such as Adobe's Flash and Microsoft's SilverLight.

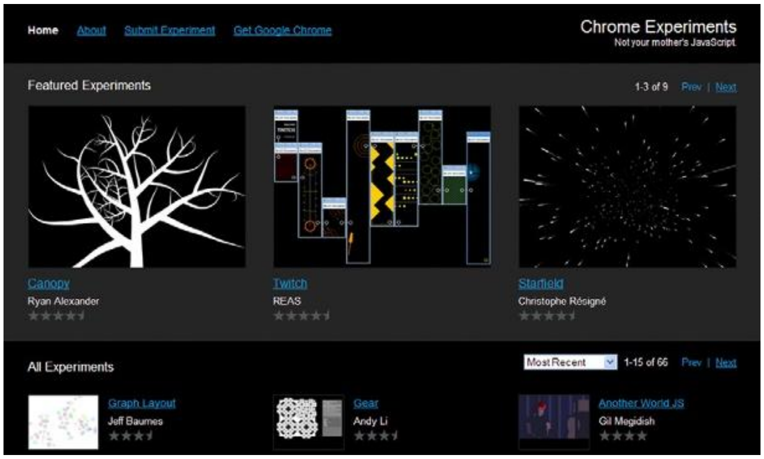


**Figure 3.28** Google's ChromeExperiments.com web site showcases some of the best CANVAS solutions on the Web.

The only browser that currently does not support CANVAS is Microsoft's Internet Explorer. You can, however, add CANVAS support to Internet Explorer through a plug-in called ExplorerCanvas, which can be downloaded at *http://excanvas.sourceforge.net/*.

At the end of the end day, SVG is a good solution, whereas CANVAS is an exciting emerging solution. A lot of technology from Google, Apple, Opera, and Mozilla is being invested into expanding the functionality of CANVAS.

## Starting with the Basics

There are two parts you need to create a visual element using CANVAS. The first is the CANVAS element itself used in your HTML. In many ways, the CANVAS element is very much the same as any other element used in HTML. Here is an example:

```
<canvas id="myCanvas" width="640" height="480">
</canvas>
```

The tag uses the new HTML5 element CANVAS as the opening and closing tag. The `width` and `height` attributes specify the size of the CANVAS space on the screen. It is the ID that is important. Here the ID is named "`myCanvas`".

Using JavaScript, you can now program the illustration that will appear in the CANVAS tag. The following example creates a black, outlined square that appears in your web page using JavaScript and Canvas.

```
<html>
  <head>
    <title>Basic Canvas Drawing</title>
    <script type="text/javascript">
      function draw(){
      var canvas = document.getElementById
('myCanvas');
      if (canvas.getContext){
         var ctx = canvas.getContext('2d');
        }
      }
    </script>
    <style type="text/css">
       canvas { border: 1px solid black; }
      </style>
    </head>
    <body onload="draw();">
       <canvas id="myCanvas" width="150"
height="150"></canvas>
    </body>
</html>
```

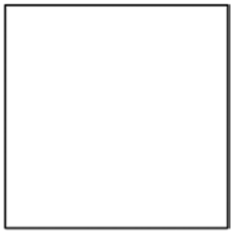


**Figure 3.29** The CANVAS element draws a simple rectangle.

Figure 3.29 shows the end result.

Stepping through the code you will see that the CANVAS element has not changed. What is modified is how the object in the CANVAS element is presented. Using JavaScript, you start a new function named “draw.” The draw function is constructed of a variable called "`myCanvas`". The "`myCanvas`" variable declares that the CANVAS element is a 2D object. The distinction of 2D is important, as it is expected that a three-dimensional (3D) definition will be added to the CANVAS element as part of the WebGL 3D program.

You use Cascading Style Sheets to define the color and border thickness for the drawing. In this instance, the drawing is black with a solid 1-pixel outline.

The "`onload`" event in the BODY element triggers when the CANVAS illustration is drawn.

## Controlling Shapes

The CANVAS element does not have the same rich collection of primitive drawing objects you find in SVG. The only primitive drawing object is a rectangle. This does not limit what you can draw, as CANVAS leverages an alternative, rich collection of path drawing functions that allow you to create complex paths, arcs, Bezier curves, and quadratic curves that you can use as the basis of your illustrations.

The rectangle shape is built of four basic parts:

- X—starting position of the rectangle along the X axis
- Y—starting position of the rectangle along the Y axis
- Width—width of the rectangle
- Height—height of the rectangle

The following is an example of a solid rectangle shape:

```
myRectangle.fillRect(15,15,100,100);
```

This description places the rectangle as starting 15 pixels in from the left side of the CANVAS element (the X axis), 15 pixels from the top of the CANVAS element (the Y axis), and with a width and height of 100 pixels each. You need to add the following HTML to view the rectangle.

```
<html>
    <head>
      <title>Basic Canvas Drawing</title>
<script>
      function draw(){
      var canvas = document.getElementById
      ('myCanvas');
      if (canvas.getContext){
```

```
    var myRectangle = canvas.getContext('2d');
      myRectangle.fillRect(15,15,100,100);
  }
}
</script>
    <style type="text/css">
      canvas { border: 1px solid black; }
    </style>
  </head>
  <body onload="draw();">
    <canvas id="myCanvas" width="150"
height="150"></canvas>
    </body>
</html>
```

The code describing the rectangle must be placed in the SCRIPT section of your HTML page. Below you will see that a variable called `myRectangle` is declared on line 6. Line 7 describes what the variable `myRectangle` will look like. The CANVAS element in the HTML body illustrates where the rectangle will be drawn.

There are three different types of rectangle primitive you can draw. The previous example demonstrates how to use the fillRect shape. You can also draw clearRect and strokeRect.

- clearRect draws a transparent rectangle on the screen.
- strokeRect draws only the outline of the rectangle on the screen.

Following is how you write the JavaScript describing how to draw the three different rectangle primitives.

```
  myRectangle.fillRect(15,15,100,100);

  myRectangle.clearRect(20,20,60,60);

  myRectangle.strokeRect(25,25,50,50);
```

All three rectangles can be combined with JavaScript and presented within your web page, as follows.

```
<html>
  <head>
    <title>Basic Canvas Drawing</title>
<script>
    function draw(){
  var canvas = document.getElementById('myCanvas');
  if (canvas.getContext){
    var myRectangle = canvas.getContext('2d');
    myRectangle.fillRect(15,15,100,100);
    myRectangle.clearRect(20,20,60,60);
    myRectangle.strokeRect(25,25,50,50);
```

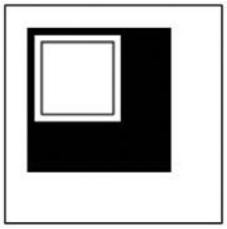


**Figure 3.30** There are three different types of rectangle primitive you can draw using the CANVAS element.

```
    }
  }
</script>
    <style type=”text/css”>
      canvas { border: 1px solid black; }
    </style>
  </head>
  <body onload=”draw();”>
    <canvas id=”myCanvas” width=”150”
height=”150”></canvas>
  </body>
</html>
```

See Figure 3.30 for how the rectangles appear. All other drawings are created from paths you must describe.

## Drawing Simple Shapes

Shapes are described in JavaScript and presented in the CANVAS element. The structure for describing a shape takes the following basic methods:

- beginPath
- moveTo
- closePath
- fill

The role of the beginPath method is to declare the start of the shape. Following the beginPath method is where you start drawing your shape. The moveTo method is used to describe that you have moved your virtual pen and are starting to draw a new shape. Following the moveTo method is where you describe the structure of the shape. The following code demonstrates how a triangle is started and drawn.

```
myShape.beginPath();

myShape.moveTo(750,500);

myShape.lineTo(1000,750);

myShape.lineTo(1000,250);
```

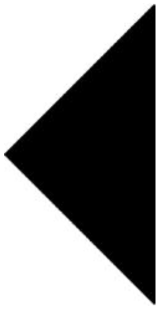


**Figure 3.31** A simple triangle is drawn using the CANVAS element.

The first line declares the start of the shape. The second line is the moveTo method stating that the drawing will start at 750 pixels from the left (X axis) and down 500 pixels (Y axis).

The triangle itself is a closed shape. By default you do not need to use the closePath method. You use the closePath method to close a shape when it is not clear where the closure for the shape is.

The final method is the fill method. Together, the code looks as follows, and Figure 3.31 shows the end result.

```
<html>
  <head>
    <title>Basic Canvas Drawing</title>
<script>
      function draw(){
    var canvas = document.getElementById
('myCanvas');
    if (canvas.getContext){
        var myShape = canvas.getContext('2d');
myShape.beginPath();
myShape.moveTo(750,500);
myShape.lineTo(1000,750);
myShape.lineTo(1000,250);
myShape.fill();
    }
}
</script>
    </head>
    <body onload="draw();">
      <canvas id="myCanvas" width="1500"
height="1500"></canvas>
  </body>
</html>
```

The lineTo method describes the shape. There are four tools you can use to describe your shape:

- Lines
- Arcs
- Bezier curves
- Quadratic curves

These four shape drawing tools allow you to create any type of shape.

## Drawing Lines

The most simple path to describe is the line. Using the lineTo method you describe the starting and ending X and Y axes positions. For instance, the following code describes a basic rectangle.

```
myTriangle.beginPath();

myTriangle.moveTo(10, 10);

myTriangle.lineTo(500, 10);

myTriangle.lineTo(10, 500);

myTriangle.lineTo(10, 10);
```

The lineTo property describes the three lines used to create position:

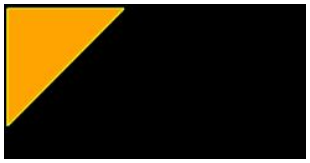


**Figure 3.32** The lineTo property allows you to draw lines in a CANVAS image.

```
myTriangle.lineTo(500, 10);
myTriangle.lineTo(10, 500);
myTriangle.lineTo(10, 10);
```

Figure 3.32 shows what the triangle will look like.

The following code shows how to present the triangle shape in your web browser.

```
<!DOCTYPE html> <html lang="en">
  <head> <meta charset="utf-8">
    <title>Drawing a Rectangle</title>
    <script type="text/javascript"><!--
window.addEventListener('load', function () {
  var elem = document.getElementById('myCanvas');
  if (!elem || !elem.getContext) {
    return;
  }
  var myTriangle = elem.getContext('2d');
  if (!myTriangle) {
    return;
  }
  myTriangle.fillStyle = "orange";
  myTriangle.strokeStyle = "yellow";
  myTriangle.lineWidth = 7;
  myTriangle.beginPath();
  myTriangle.moveTo(10, 10);
  myTriangle.lineTo(500, 10);
  myTriangle.lineTo(10, 500);
  myTriangle.lineTo(10, 10);
  myTriangle.fill();
  myTriangle.stroke();
  myTriangle.closePath();
}, false);
    // --></script>
  </head> <body style="background-color:#000;">
<canvas id="myCanvas" width="500" height="500">
</canvas>
  </body> </html>
```

Using the lineTo property allows you to draw simple, line-based shapes.

## Creating Arcs

When you want to draw a circle you use the Arc method. An arc is drawn with six different properties:

- X—the coordinates for the circle's center along the X axis.
- Y—the coordinates for the circle's center along the Y axis.

- Radius—the size of the circle.
- startAngle—the start point of the arc.
- endAngle—the end point of the arc.
- anticlockwise—a Boolean value that dictates the direction the circle is drawn.

The following code describes the structure of an arc:

```
context.arc(260,260,250,0,7,true);
```

Figure 3.33 shows the circle as it is drawn on the page.

The following code embeds the Arc method and instructions into a CANVAS drawing.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Canvas - Creating a Circle</title>
    <script type="text/javascript">
      window.onload = function() {
        var drawingCanvas = document.getElementById
('myCircle');
      if(drawingCanvas && drawingCanvas.
getContext) {
        var context = drawingCanvas.getContext('2d');
        context.strokeStyle = "yellow";
        context.fillStyle = "red";
        context.lineWidth = 20;
        context.beginPath();
        context.arc(260,260,250,0,Math.PI*2,true);
        context.closePath();
        context.stroke();
        context.fill();
      }
    }
    </script>
  </head>
  <body>
    <canvas id="myCircle" width="550" height="550">
    </canvas>
  </body>
</html>
```

Figure 3.34 illustrates how you can add color to your circles.

In addition to the Arc method you also can add Bezier and quadratic curves, both of which are mathematical calculations for creating an image. Bezier curves were developed by French mathematician Pierre Bezier in 1962. A Bezier curve is calculated from a parametric curve describing a parabola. Figure 3.35 shows the four points used to create a Bezier curve.

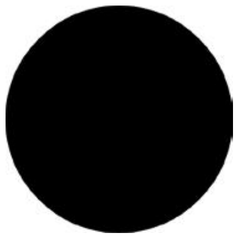


**Figure 3.33** The Arc method allows you to draw circles.

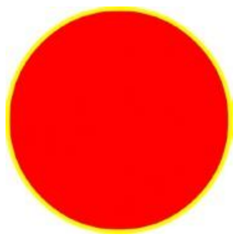


**Figure 3.34** The Arc method can be controlled visually with the same controls you use for other CANVAS drawing methods.

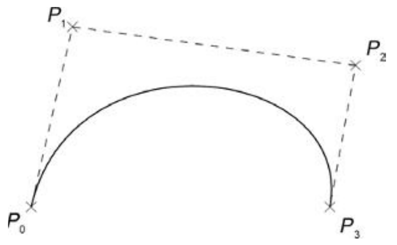


**Figure 3.35** A Bezier curve.

A quadratic curve is based on the Bezier curve. The difference is that the quadratic curve is constructed of three points of definition instead of just one. It can be difficult to draw complex, Bezier curve images in CANVAS for a single, simple reason: There are no visual drawing tools, such as Adobe Illustrator, that export images that can be read by the CANVAS element.

## Adding Color

So you now have basic shapes on the page. Big deal, right? Using JavaScript you can now begin to programmatically paint your objects. The following shows two semitransparent intersecting squares. You will see that the fill color is a CSS style.

```
<html>
  <head>
    <script type="application/x-javascript">
    function draw() {
      var canvas = document.getElementById
("myCanvas");
        if (canvas.getContext) {
        var ctx = canvas.getContext("2d");
        ctx.fillStyle = "rgb(0,0,500)";
        ctx.fillRect (10, 10, 150, 150);
        ctx.fillStyle = "rgba(0, 300, 0, 0.5)";
        ctx.fillRect (75, 75, 150, 150);
      }
    }
  </script>
</head>
<body onload="draw();">
    <canvas id="myCanvas" width="300"
    height="300"></canvas>
  </body>
</html>
```

The fillStyle method allows you to apply CSS style formatting. Leveraging CSS increases the amount of visual control you have on your drawings on the screen. As with SVG you can use any of the CSS color naming formats such as Hex and RGB.

Linear and radial gradients can also be applied to CANVAS images. As with SVG, the linear and radial gradients inherit how CSS implements gradients. The gradient construction is developed by first creating a shape, giving the gradient a name, defining the gradient, and then applying the gradient.

The first step is to create a shape. The following is a simple CANVAS rectangle:

```
myRectangle.fillRect(10,10,650,650);
```

The next step is to create a new variable that declares a new gradient. The following line creates a new gradient that is named verticalGradient. The name is arbitrary; what is not arbitrary is the description of the gradient type that follows the name.

```
var verticalGradient = myRectangle.createLinearGradient
(0,0,0,650);
```

Here you are associating the gradient with the CANVAS object myRectangle. At this point the gradient will not paint the image that comes later—the code at this point merely associates the gradient and the image. The property createLinearGradient dictates where the gradient will paint an object. The values in the parenthesis are the X and Y axes and height and width.

A gradient must have at least two colors. The following will paint a gradient color that starts red and then transitions 50% through the image to yellow.

```
verticalGradient.addColorStop(0, 'red');
verticalGradient.addColorStop(0.5, 'yellow');
```

The final step is to use the paintStyle property to paint the gradient into the rectangle:

```
myRectangle.fillStyle = verticalGradient;
```

The whole CANVAS script looks as follows.

```
<html>
  <head>
    <title>Linear Gradient</title>
    <script type="application/x-javascript">
      function draw() {
        var myRectangle = document.getElementById
('myCanvas').getContext('2d');
        var verticalGradient = myRectangle.
createLinearGradient(0,0,0,650);
        verticalGradient.addColorStop(0, 'red');
        verticalGradient.addColorStop(0.5, 'yellow');
        myRectangle.fillStyle = verticalGradient;
        myRectangle.fillRect(10,10,650,650);
      }
    </script>
  </head>
  <body onload="draw();">
      <canvas id="myCanvas" width="800"
height="800"></canvas>
  </body>
</html>
```

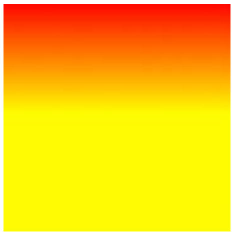


**Figure 3.36** Linear gradients can be applied to CANVAS images.

See Figure 3.36 for an example image.

As with the linear gradient, the radial gradient is painted onto an image and the gradient must be constructed of at least two colors. As with SVG, the radial gradients require starting and stopping radius definitions, size, and position. The following creates a radial gradient called `myRadialGradient` with three colors: red, yellow, and blue.

```
    var myRadialGradient = myCircle.
createRadialGradient(0,150,150,0,140,90);
     myRadialGradient.addColorStop(0, 'red');
     myRadialGradient.addColorStop(0.9, 'yellow');
     myRadialGradient.addColorStop(1, 'blue');
```

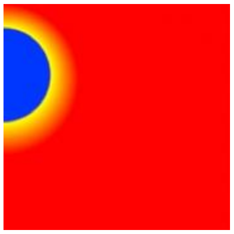


**Figure 3.37** A radial gradient applied to a CANVAS image.

You need to add the gradient to your CANVAS description, as follows. See also Figure 3.37.

```
<html>
  <head>
    <title>A canvas radialGradient example</title>
    <script type="application/x-javascript">
      function draw() {
        var myCircle = document.getElementById
('myCanvas').getContext('2d');
        var myRadialGradient = myCircle.
        createRadialGradient(0,150,150,0,140,90);
        myRadialGradient.addColorStop(0, 'red');
        myRadialGradient.addColorStop(0.9, 'yellow');
        myRadialGradient.addColorStop(1, 'blue');
        myCircle.fillStyle = myRadialGradient;
        myCircle.fillRect(0,0,450,450);
      }
    </script>
  </head>
  <body onload="draw();">
      <canvas id="myCanvas" width="500" height="500">
</canvas>
    </body>
</html>
```

Gradients are useful for creating depth on an object. Careful use of gradients can simulate a 3D environment.

## Adding Animation to CANVAS Images

Animation can be added to CANVAS images. As you can imagine, animation requires additional work. To make your life easier there is a great JavaScript library called CAKE (Canvas Animation

Kit Experiment) that you can download at *http://code.google.com/p/cakejs/*. Using the CAKE library you can easily create CANVAS-based animation. The following code will create a pulsing blue circle.

```
window.onload = function()
{
   var CAKECanvas = new Canvas(document.body,
600, 400);
   var myCircle = new Circle(100,
        {
             id: 'myCircle',
             x: CAKECanvas.width / 3,
             y: CAKECanvas.height / 2,
             stroke: 'blue',
             strokeWidth: 20,
             endAngle: Math.PI*2
        }
   );
   myCircle.addFrameListener(
     function(t, dt)
        {
             this.scale = Math.sin(t / 1000);
        }
   );
   CAKECanvas.append(myCircle);
};
```

The final step you need to take to ensure that your animation works is to download the CAKE library files to your Web site. The files can be downloaded at *http://glimr.rubyforge.org/cake/*. You will need to save the CAKE JS library to your Web site. In the HEAD section of your Web page you will need to make a linked reference to the CAKE library. It will look like this:

```
<script type="text/javascript" src="cake.js">
</script>
```

To accomplish the animation use the Scale method. The effect is very similar to Adobe's Flash, but with the benefit of running correctly on web browsers found on mobile devices such as the iPhone, MyTouch, and Palm Pre.

The introduction of CANVAS and SVG gives you great opportunities to create complex and compelling illustrations programmatically inside of your HTML5 web pages. It is fair to say that CANVAS is still growing in technical scope. Expect additions and changes to the technology over the next few years. A big addition will be the inclusion of 3D within CANVAS.

## What You Have Learned

This article introduced you to scalable vector graphics (SVG) and the CANVAS element. These two technologies enable you to programmatically build images inside of your web pages without needing graphic tools such as Adobe Illustrator, Flash, or PhotoShop.

focal press books

VISUALIZING THE WEB WITH HTML 5

Download these other great articles in the HTML5: Visualizing the Web series

**HTML5 Rich Media Foundation**

Learn about the new ways in which video and audio can be easily embedded into your HTML5 Web pages. Discover how you can create new Web media content and how JavaScript, CSS, and SVG can be integrated to create a compelling, rich media foundation for your work.

**Rendering HTML5 Illustration**

Scalable Vector Graphics (SVG) and CANVAS are two new tools introduced in HTML5 that you can use to add illustration and interactive animation to your Web pages. Understand why there are two different formats, how you can use them, and where they can be used today.

**Picture CSS3**

See how you can use Cascading Style Sheets 3, or CSS3, to quickly and easily increase your control over visual Web page design.

**The HTML5 JavaScript Model**

JavaScript is the glue that enables HTML to become interactive. Learn how you can leverage JavaScript Libraries to quickly build beautiful Web applications.

**HTML5 Tag Structure**

Take a gander at the new HTML5 elements: how you can take advantage of them and what you need to look out for as you design your new Web sites.

learn • master • create

And look for the HTML5: Visualizing the Web book coming in July 2010.
The book will feature information and resources that expand on the 5-part series.

Focal Press

Visit focalpress.com for video tutorials, sample chapters, podcasts and exclusive competitions and discounts.

focalpress.com