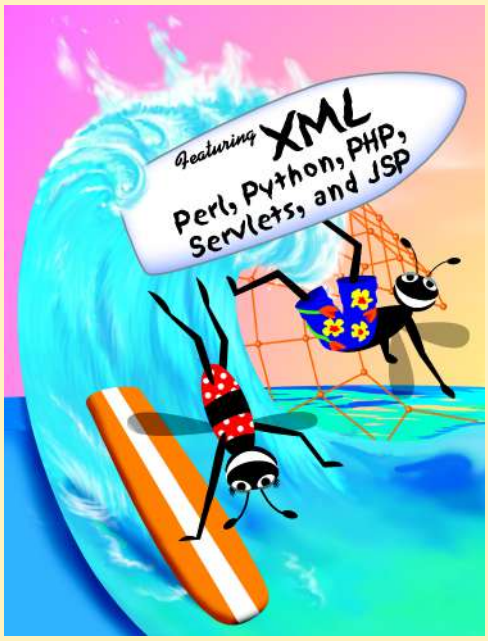
15

Dynamic HTML: Filters and Transitions

Objectives

- To use filters to achieve special effects.
- To combine filters to achieve an even greater variety of special effects.
- To be able to create animated visual transitions between Web pages.
- To be able to modify filters dynamically, using DHTML.



Between the motion and the act falls the shadow.
Thomas Stearns Eliot, *The Hollow Men*

...as through a filter, before the clear product emerges.
F. Scott Fitzgerald

There is strong shadow where there is much light.
Johann Wolfgang von Goethe

When all things are equal, translucence in writing is more effective than transparency, just as glow is more revealing than glare.
James Thurber

...one should disdain the superficial and let the true beauty of one's soul shine through.
Fran Lebowitz

Modernity exists in the form of a desire to wipe out whatever came earlier, in the hope of reaching at least a point that could be called a true present, a point of origin that marks a new departure.
Paul de Man

Outline

15.1 Introduction

Just a few years ago it was not realistic to offer the kinds of dramatic visual effects you will see in this chapter, because desktop processing power was insufficient. Today, with powerful processors, these visual effects are realizable without delays. Just as you expect to see dramatic visual effects on TV weather reports, Web users appreciate visual effects when browsing Web pages.

In the past, achieving these kinds of effects, if you could get them at all, demanded frequent trips back and forth to the server. With the consequent delays, the beauty of the effects was lost.

Performance Tip 15.1

With Dynamic HTML, many visual effects are implemented directly in the client-side browser (Internet Explorer 5.5 for this book), so no server-side processing delays are incurred. The DHTML code that initiates these effects is generally quite small and is coded directly into the XHTML Web page.

You will be able to achieve a great variety of effects. You may transition between pages with *random dissolves* and *horizontal and vertical blinds* effects, among others. You can convert colored images to gray in response to user actions; this could be used, for example, to indicate that some option is not currently selectable. You can make letters *glow* for emphasis. You can create *drop shadows* to give text a three-dimensional appearance.

In this chapter, we discuss both *filters* and *transitions*. Applying filters to text and images causes changes that are persistent. Transitions are temporary phenomena; applying a transition allows you to transfer from one page to another with a pleasant visual effect such as a random dissolve. Filters and transitions do not add content to your pages—rather,

they present existing content in an engaging manner to help hold the user's attention.

Each of the visual effects achievable with filters and transitions is programmable, so these effects can be adjusted dynamically by programs that respond to user-initiated events like mouse clicks and keystrokes. Filters and transitions are so easy to use that virtually any Web page designer or programmer can incorporate these effects with minimal effort.

Look-and-Feel Observation 15.1

Experiment by applying combinations of filters to the same element. You may discover some eye-pleasing effects that are particularly appropriate for your applications.

Part of the beauty of DHTML filters and transitions is that they are built right into Internet Explorer. You do not need to spend time working with sophisticated graphics packages, preparing images that will be downloaded (slowly) from servers. When Internet Explorer renders your page, it applies all the special effects and does this while running on the client computer, without lengthy waits for files to download from the server.

Look-and-Feel Observation 15.2

DHTML's effects are programmable. They can be applied dynamically to elements of your pages in response to user events such as mouse clicks and keystrokes.

Filters and transitions are included with the *CSS* **`filter`** *property.* They give you the same kind of graphics capabilities you get through presentation software like Microsoft's PowerPoint. You can have new pages or portions of pages fade in and fade out. You can have a page randomly dissolve into the next page. You can make portions of the page transparent or semitransparent so that you can see what is behind them. You can make elements glow for emphasis. You can blur text or an image to give it the illusion of motion. You can create drop shadows on elements to give them a three-dimensional effect. And you can combine effects to generate an even greater variety of effects.

Software Engineering Observation 15.1

Filters and transitions can be applied to block-level elements such as **`div`** *or* **`p`***, but can be applied only to inline-level elements such as* **`strong`** *or* **`em`** *if the element has its* **`height`** *or* **`width`** *CSS properties set.*

Portability Tip 15.1

Filters and transitions are a Microsoft technology available only in Windows-based versions of Internet Explorer 5.5. Do not use these capabilities if you are writing for other browsers. If you are writing for an audience with a diversity of browsers and you use DHTML filters and transitions, you should also make alternate provisions.

15.2 Flip filters: `flipv` and `fliph`

The **`flipv`** and **`fliph`** *filters* mirror text or images vertically and horizontally, respectively. In Fig. 15.1 we demonstrate these effects, using both filters to flip text.

```
1  <?xml version = "1.0"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
3     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

Fig. 15.1 Using the **`flip`** filter (part 1 of 3).

```
 4
 5  <!-- Fig. 15.1: flip.html   -->
 6  <!-- Using the flip filters -->
 7
 8  <html xmlns = "http://www.w3.org/1999/xhtml">
 9     <head>
10        <title>The flip filter</title>
11
12        <style type = "text/css">
13           body  { background-color: #CCFFCC }
14
15           table { font-size: 3em;
16                   font-family: Arial, sans-serif;
17                   background-color: #FFCCCC;
18                   border-style: ridge ;
19                   border-collapse: collapse }
20
21           td    { border-style: groove;
22                   padding: 1ex }
23        </style>
24     </head>
25
26     <body>
27
28        <table>
29
30           <tr>
31              <!-- Filters are applied in style declarations -->
32              <td style = "filter: fliph">Text</td>
33              <td>Text</td>
34           </tr>
35
36           <tr>
37              <!-- More than one filter can be applied at once -->
38              <td style = "filter: flipv fliph">Text</td>
39              <td style = "filter: flipv">Text</td>
40           </tr>
41
42        </table>
43
44     </body>
45  </html>
```

Fig. 15.1 Using the **flip** filter (part 2 of 3).

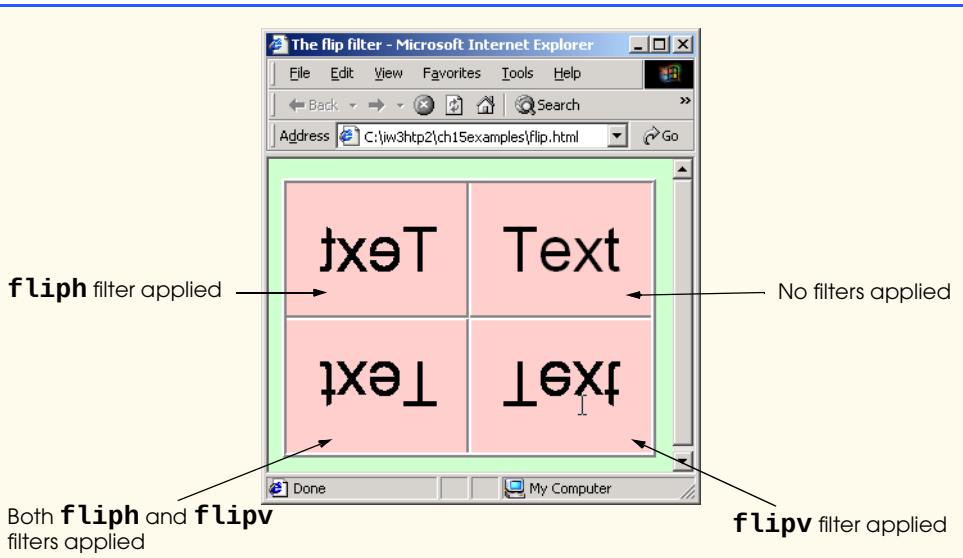


Fig. 15.1 Using the **flip** filter (part 3 of 3).

In line 32

```
<td style = "filter: fliph">Text</td>
```

filters are applied in the **style** attribute. The value of the **filter** property is the name of the filter. In this case, the filter is **fliph**, which flips the affected object horizontally.

In line 38

```
<td style = "filter: flipv fliph">Text</td>
```

we see that more than one filter can be applied at once. Enter multiple filters as values of the **filter** attribute, separated by spaces. In this case, the **flipv** filter is also applied, which flips the affected object vertically.

15.3 Transparency with the chroma Filter

The ***chroma*** *filter* applies *transparency effects* dynamically, without using a graphics editor to hard-code transparency into the image. In Fig. 15.2 ,we alter the transparency of an image, using object model scripting based on a user selection from a **select** element.

```
1  <?xml version = "1.0"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
3     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
4
5  <!-- Fig 15.2: chroma.html                              -->
6  <!-- Applying transparency using the chroma filter  -->
7
```

Fig. 15.2 Changing values of the **chroma** filter (part 1 of 3).

```
<html xmlns = "http://www.w3.org/1999/xhtml">
   <head>
      <title>Chroma Filter</title>

      <script type = "text/javascript">
         <!--
         function changecolor()
         {
            if ( colorSelect.value ) {
               // if the user selected a color, parse the
               // value to hex and set the filter color.
               chromaImg.filters( "chroma" ).color =
                  parseInt( colorSelect.value, 16 );
               chromaImg.filters( "chroma" ).enabled = true;
            }
            else // if the user selected "None",
               // disable the filter.
               chromaImg.filters( "chroma" ).enabled = false;
         }
         // -->
      </script>
   </head>

   <body>

      <h1>Chroma Filter:</h1>

      <img id = "chromaImg" src = "trans.gif" style =
         "position: absolute; filter: chroma"  alt =
         "Transparent Image" />

      <form action = "">
         <!-- The onchange event fires when -->
         <!-- a selection is changed        -->
         <select id = "colorSelect" onchange = "changecolor()">
            <option value = "">None</option>
            <option value = "#00FFFF">Cyan</option>
            <option value = "#FFFF00">Yellow</option>
            <option value = "#FF00FF">Magenta</option>
            <option value = "#000000" selected = "selected">
               Black</option>
         </select>
      </form>

   </body>
</html>
```

Fig. 15.2 Changing values of the **chroma** filter (part 2 of 3).

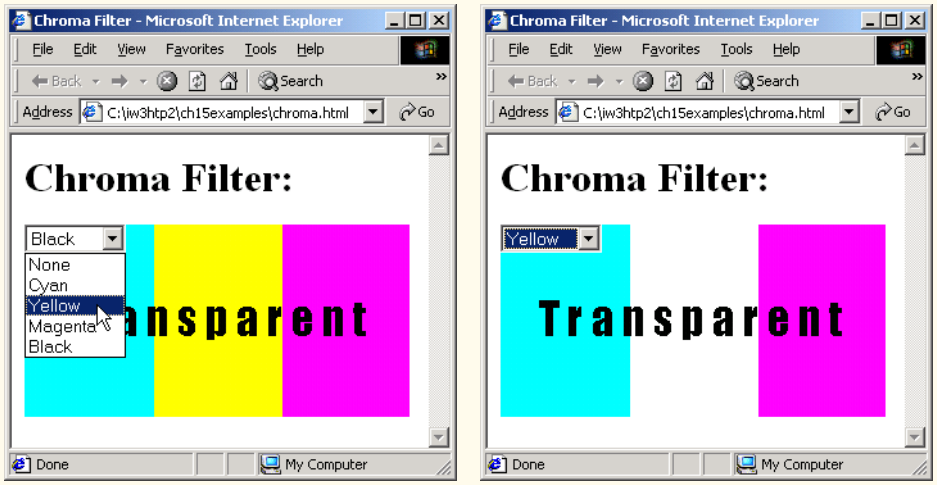


Fig. 15.2 Changing values of the **chroma** filter (part 3 of 3).

In lines 19–20

```
chromaImg.filters( "chroma" ).color =
   parseInt( colorSelect.value, 16 );
```

we set the filter properties dynamically, using JavaScript. In this case, **colorSelect.value**, the value of the **colorSelect** drop-down list (line 42), is a string. We use the **parseInt** function to convert the value to a hexadecimal integer for setting the **color** property of the **chroma** filter. The second parameter of **parseInt**, **16** in this case, specifies the base of the integer (base 16 is hexadecimal).

In line 21

```
chromaImg.filters( "chroma" ).enabled = true;
```

we turn on the filter. Each filter has a property named **enabled**. If this property is set to **true**, the filter is applied. If it is set to **false**, the filter is not applied. So, in line 25

```
chromaImg.filters( "chroma" ).enabled = false;
```

we indicate that, if the user selected **None** (line 43) from the drop-down list, the filter is disabled.

In line 42

```
<select id = "trSelect" onchange = "changecolor()">
```

we introduce a new event, ***onchange***. This event fires whenever the **value** of a form field changes, which in this case happens whenever the user makes a different selection in the **colorSelect** drop-down list.

15.4 Creating Image masks

Applying the **mask** *filter* to an image allows you to create an *image mask*, in which the background of an element is a solid color and the foreground of an element is transparent to the image or color behind it. In Fig. 15.3 we add the **mask** filter to an **h1** element which overlaps an image. The foreground of that **h1** element (the text inside it) is transparent to the image behind it.

```
1  <?xml version = "1.0"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
3     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
4
5  <!-- Fig 15.3: mask.html          -->
6  <!-- Placing a mask over an image -->
7
8  <html xmlns = "http://www.w3.org/1999/xhtml">
9     <head>
10       <title>Mask Filter</title>
11    </head>
12
13    <body>
14
15       <h1>Mask Filter</h1>
16
17       <!-- Filter parameters are specified in parentheses, -->
18       <!-- in the form param1 = value1, param2 = value2,   -->
19       <!-- etc.                                            -->
20       <div style = "position: absolute; top: 125; left: 20;
21          filter: mask( color = #CCFFFF )">
22       <h1 style = "font-family: Courier, monospace">
23       AaBbCcDdEeFfGgHhIiJj<br />
24       KkLlMmNnOoPpQqRrSsTt
25       </h1>
26       </div>
27
28       <img src = "gradient.gif" width = "400" height = "200"
29          alt = "Image with Gradient Effect" />
30    </body>
31 </html>
```

Fig. 15.3 Using the **mask** filter (part 1 of 2).

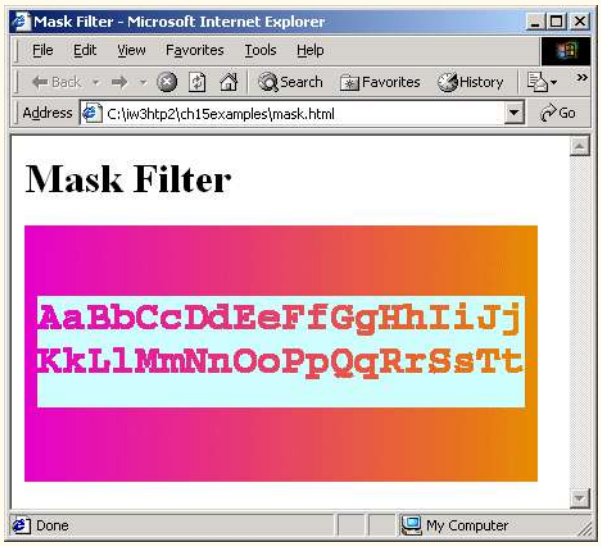


Fig. 15.3 Using the **mask** filter (part 2 of 2).

In line 21

```
filter: mask( color = #CCFFFF )
```

is a color parameter for the **mask** filter that specifies what the mask's color will be. Parameters are always specified in the format *param* = *value*.

15.5 Miscellaneous Image filters: invert, gray and xray

The following three image filters apply simple image effects to images or text. The ***invert*** *filter* applies a *negative image effect*—dark areas become light, and light areas become dark. The ***gray*** *filter* applies a *grayscale image effect*, in which all color is stripped from the image and all that remains is brightness data. The ***xray*** *filter* applies an x-ray effect, which is basically just an inversion of the grayscale effect. Figure 15.4 demonstrates applying these filters, alone and in combination, to a simple image.

```
1  <?xml version = "1.0"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
3     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
4
5  <!-- Fig 15.4: misc.html                                    -->
6  <!-- Image filters to invert, grayscale or xray an image -->
7
8  <html xmlns = "http://www.w3.org/1999/xhtml">
9     <head>
10       <title>Misc. Image filters</title>
```

Fig. 15.4 Filters **invert**, **gray** and **xray** (part 1 of 3).

```
11
12          <style type = "text/css">
13             .cap { font-weight: bold;
14                    background-color: #DDDDAA;
15                    text-align: center }
16          </style>
17       </head>
18
19       <body>
20          <table class = "cap">
21             <tr>
22                <td>Normal</td>
23                <td>Grayscale</td>
24             </tr>
25             <tr>
26                <td><img src = "hc.jpg" alt =
27                       "normal scenic view" /></td>
28                <td><img src = "hc.jpg" style = "filter: gray"
29                         alt = "gray scenic view"/>
30                </td>
31             </tr>
32             <tr>
33                <td>Xray</td>
34                <td>Invert</td>
35             </tr>
36             <tr>
37                <td><img src = "hc.jpg" style = "filter: xray"
38                         alt = "xray scenic view"/>
39                </td>
40                <td><img src = "hc.jpg" style = "filter: invert"
41                     alt = "inverted scenic view"/>
42                </td>
43             </tr>
44          </table>
45
46       </body>
47    </html>
```

Fig. 15.4 Filters **invert**, **gray** and **xray** (part 2 of 3).

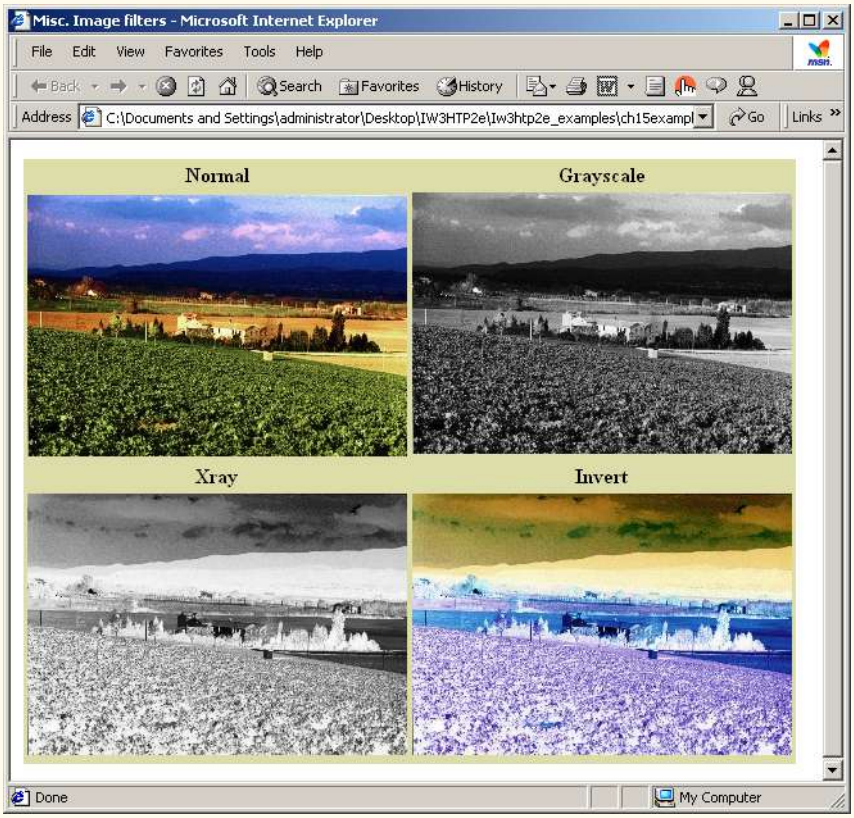


Fig. 15.4 Filters **`invert`**, **`gray`** and **`xray`** (part 3 of 3).

Each of our filters in lines 26–41 applies a separate image effect to **`hc.jpg`**.

Look-and-Feel Observation 15.3

A good use of the **`invert`** *filter is to signify that something has just been clicked or selected.*

15.6 Adding shadows to Text

A simple filter that adds depth to your text is the ***shadow*** *filter*. This filter creates a shadowing effect that gives your text a three-dimensional look (Fig. 15.5).

```
1  <?xml version = "1.0"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
3     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
4
5  <!-- Fig 15.5: shadow.html      -->
6  <!-- Applying the shadow filter -->
```

Fig. 15.5 Applying a **`shadow`** filter to text (part 1 of 2).

```
 7
 8  <html xmlns = "http://www.w3.org/1999/xhtml">
 9     <head>
10        <title>Shadow Filter</title>
11
12        <script type = "text/javascript">
13           <!--
14           var shadowDirection = 0;
15
16           function start()
17           {
18              window.setInterval( "runDemo()", 500 );
19           }
20
21           function runDemo()
22           {
23              shadowText.innerText =
24                 "Shadow Direction: " + shadowDirection % 360;
25              shadowText.filters( "shadow" ).direction =
26                 ( shadowDirection % 360 );
27              shadowDirection += 45;
28           }
29           // -->
30        </script>
31     </head>
32
33     <body onload = "start()">
34
35        <h1 id = "shadowText" style = "position: absolute; top: 25;
36           left: 25; padding: 10; filter: shadow( direction = 0,
37           color = red )">Shadow Direction: 0</h1>
38     </body>
39  </html>
```

Fig. 15.5 Applying a **shadow** filter to text (part 2 of 2).

In lines 35–37

```
<h1 id = "shadowText" style = "position: absolute; top: 25;
         left: 25; padding: 10; filter: shadow(direction = 0,
          color = red )">Shadow Direction: 0</h1>
```

we apply the **shadow** filter to text. Property **direction** of the **shadow** filter determines in which direction the shadow effect is applied—this can be set to any of eight directions expressed in angular notation: **0** (up), **45** (above-right), **90** (right), **135** (below-right), **180** (below), **225** (below-left), **270** (left) and **315** (above-left). Property **color** specifies the color of the shadow that is applied to the text. Lines 23–27 in function **runDemo**

```
shadowText.innerText =
   "Shadow Direction: " + shadowDirection % 360;
shadowText.filters( "shadow" ).direction =
   ( shadowDirection % 360 );
shadowDirection += 45;
```

cycle through all values of the **direction** property, from **0** to **315**, and update property **innerText** of the **h1** element (**shadowText**) to match the current shadow direction.

Note that we apply a **padding** CSS style to the **h1** element. Otherwise, the shadow effect is partially cut off by the border of the element. Increasing the **padding** provides greater distance between the text and the border of the element, allowing the full effect to be displayed.

Software Engineering Observation 15.2

Some filters may be cut off by element borders—make sure to increase the padding in that element if this happens.

15.7 Creating Gradients with alpha

In Chapter 3, we saw a brief example of the gradient effect, which is a gradual progression from a starting color to a target color. Internet Explorer 5.5 allows you to create the same type of effect dynamically, using the ***alpha*** *filter* (Fig. 15.6). It is also often used for transparency effects not achievable with the **chroma** filter.

```
 1  <?xml version = "1.0"?>
 2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
 3     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
 4
 5  <!-- Fig 15.6: alpha.html                    -->
 6  <!-- Applying the alpha filter to an image -->
 7
 8  <html xmlns = "http://www.w3.org/1999/xhtml">
 9     <head>
10        <title>Alpha Filter</title>
11     <script type = "text/javascript">
12        <!--
13        function run()
14        {
15           pic.filters( "alpha" ).opacity = opacityButton.value;
16           pic.filters( "alpha" ).finishopacity =
```

Fig. 15.6 Applying the **alpha** filter (part 1 of 3).

```
17              opacityButton2.value;
18           pic.filters( "alpha" ).style = styleSelect.value;
19        }
20        // -->
21     </script>
22     </head>
23
24     <body>
25
26        <div id = "pic"
27             style = "position: absolute; left:0; top: 0;
28                      filter: alpha( style = 2, opacity = 100,
29                      finishopacity = 0 )">
30           <img src = "flag.gif" alt = "Flag" />
31        </div>
32
33        <table style = "position: absolute; top: 250; left: 0;
34           background-color: #CCFFCC" border = "1">
35
36           <tr>
37              <td>Opacity (0-100):</td>
38              <td><input type = "text" id = "opacityButton"
39                 size = "3" maxlength = "3" value = "100" /></td>
40           </tr>
41
42           <tr>
43             <td>FinishOpacity (0-100):</td>
44              <td><input type = "text" id = "opacityButton2"
45                 size = "3" maxlength = "3" value = "0" /></td>
46           </tr>
47
48           <tr>
49              <td>Style:</td>
50              <td><select id = "styleSelect">
51                 <option value = "1">Linear</option>
52                 <option value = "2" selected = "selected">
53                    Circular</option>
54                 <option value = "3">Rectangular</option>
55                 </select></td>
56           </tr>
57
58           <tr>
59              <td align = "center" colspan = "2">
60                 <input type = "button" value = "Apply"
61                 onclick = "run()" />
62              </td>
63           </tr>
64        </table>
65
66     </body>
67  </html>
```

Fig. 15.6 Applying the **alpha** filter (part 2 of 3).

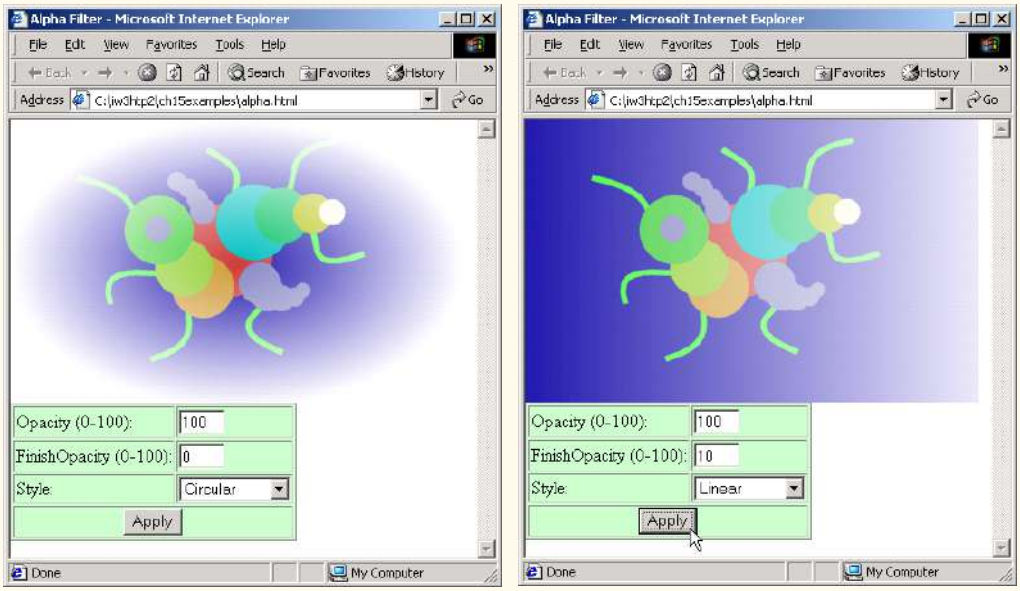


Fig. 15.6 Applying the **alpha** filter (part 3 of 3).

In lines 26–29

```
<div id = "pic"
     style = "position: absolute; left:0; top: 0;
              filter: alpha( style = 2, opacity = 100,
               finishopacity = 0 )">
```

we apply the **alpha** filter to a **div** element containing an image. The **style** property of the filter determines in what style the opacity is applied; a value of 0 applies *uniform opacity*, a value of 1 applies a *linear gradient*, a value of 2 applies a *circular gradient* and a value of 3 applies a *rectangular gradient*.

The **opacity** and **finishopacity** properties are both percentages that determine at what percent opacity the specified gradient starts and finishes, respectively. Additional attributes are **startX**, **startY**, **finishX** and **finishY**. These specify at what *x-y* coordinates the gradient starts and finishes in that element.

15.8 Making Text glow

The ***glow*** *filter* adds an aura of color around text. The color and strength can both be specified (Fig. 15.7).

```
1  <?xml version = "1.0"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
```

Fig. 15.7 Applying changes to the **glow** filter (part 1 of 3).

```
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<!-- Fig 15.7: glow.html      -->
<!-- Applying the glow filter -->

<html xmlns = "http://www.w3.org/1999/xhtml">
   <head>
      <title>Glow Filter</title>
      <script type = "text/javascript">
         <!--
         var strengthIndex = 1;
         var counter = 1;
         var upDown = true;
         var colorArray = [ "FF0000", "FFFF00", "00FF00",
                            "00FFFF", "0000FF", "FF00FF" ];
         function apply()
         {
            glowSpan.filters( "glow" ).color =
               parseInt( glowColor.value, 16 );
            glowSpan.filters( "glow" ).strength =
               glowStrength.value;
         }

         function startdemo()
         {
            window.setInterval( "rundemo()", 150 );
         }

         function rundemo()
         {
            if ( upDown ) {
               glowSpan.filters( "glow" ).strength =
                  strengthIndex++;
            }
            else {
               glowSpan.filters( "glow" ).strength =
                  strengthIndex--;
            }

            if ( strengthIndex == 1 ) {
               upDown = !upDown;
               counter++;
               glowSpan.filters( "glow" ).color =
                  parseInt( colorArray[ counter % 6 ], 16 );
            }

            if ( strengthIndex == 10 ) {
               upDown = !upDown;
            }
         }
         // -->
      </script>
   </head>

```

Fig. 15.7 Applying changes to the **glow** filter (part 2 of 3).

```
57     <body style = "background-color: #00AAAA">
58        <h1>Glow Filter:</h1>
59
60        <span id = "glowSpan" style = "position: absolute;
61           left: 200;top: 100; padding: 5; filter: glow(
62           color = red, strength = 5 ); font-size: 2em">
63           Glowing Text
64        </span>
65
66        <table border = "1" style = "background-color: #CCFFCC">
67           <tr>
68              <td>Color (Hex)</td>
69              <td><input id = "glowColor" type = "text" size = "6"
70                 maxlength = "6" value = "FF0000" /></td>
71           </tr>
72           <tr>
73              <td>Strength (1-255)</td>
74              <td><input id = "glowStrength" type = "text"
75                    size = "3" maxlength = "3" value = "5" />
76              </td>
77           </tr>
78           <tr>
79              <td colspan = "2">
80                 <input type = "button" value = "Apply"
81                    onclick = "apply()" />
82                 <input type = "button" value = "Run Demo"
83                    onclick = "startdemo()" /></td>
84           </tr>
85        </table>
86
87     </body>
88  </html>
```

Fig. 15.7 Applying changes to the `glow` filter (part 3 of 3).

Lines 16–17

```
var colorArray = [ "FF0000", "FFFF00", "00FF00",
                   "00FFFF", "0000FF", "FF00FF" ];
```

establish an array of color values to cycle through in the demo.

Lines 45–46

```
glowSpan.filters( "glow" ).color =
   parseInt( colorArray[ counter % 6 ], 16 );
```

change the **color** attribute of the **glow** filter based on **counter**, which is incremented (line 44) every time the value of **strengthIndex** becomes 1. As in the example with the **chroma** filter, we use the **parseInt** function to assign a proper hexadecimal value (taken from the **colorArray** we declared in lines 16–17) to the **color** property.

Lines 33–40

```
if ( upDown ) {
   glowSpan.filters( "glow" ).strength =
      strengthIndex++;
}
else {
   glowSpan.filters( "glow" ).strength =
      strengthIndex--;
}
```

are an **if**/**else** structure that increments or decrements the **strength** property of the **glow** filter based on the value of **upDown**, which is toggled in the **if** structures at lines 42 and 49 when **strengthIndex** reaches either 1 or 10.

Clicking the **Run Demo** button starts a cycle that oscillates the filter **strength**, cycling through the colors in **colorArray** after every loop.

Common Programming Error 15.1

*When the **glow** filter is set to a large **strength**, the effect is often cut off by the borders of the element. Add CSS **padding** to prevent this.*

15.9 Creating Motion with `blur`

The ***blur** filter* creates an illusion of motion by blurring text or images in a certain direction. As we see in Fig 15.8, the **blur** filter can be applied in any of eight directions, and its strength can vary.

```
1  <?xml version = "1.0"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
3     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
4
5  <!-- Fig 15.8: blur.html -->
6  <!-- The blur filter    -->
7
8  <html xmlns = "http://www.w3.org/1999/xhtml">
9     <head>
10       <title>Blur Filter</title>
11       <script type = "text/javascript">
12          <!--
13          var strengthIndex = 1;
```

Fig. 15.8 Using the **blur** filter (part 1 of 4).

```
14            var blurDirection = 0;
15            var upDown = 0;
16            var timer;
17
18            function reBlur()
19            {
20               blurImage.filters( "blur" ).direction =
21                  document.forms( "myForm" ).Direction.value;
22               blurImage.filters( "blur" ).strength =
23                  document.forms( "myForm" ).Strength.value;
24               blurImage.filters( "blur" ).add =
25                  document.forms( "myForm" ).Add.checked;
26            }
27
28            function startDemo()
29            {
30               timer = window.setInterval( "runDemo()", 5 );
31            }
32
33            function runDemo( )
34            {
35               document.forms( "myForm" ).Strength.value =
36                  strengthIndex;
37               document.forms( "myForm" ).Direction.value =
38                  ( blurDirection % 360 );
39
40               if ( strengthIndex == 35 || strengthIndex == 0 )
41                  upDown = !upDown;
42
43               blurImage.filters( "blur" ).strength =
44                  ( upDown ? strengthIndex++ : strengthIndex-- );
45
46               if ( strengthIndex == 0 )
47                  blurImage.filters( "blur" ).direction =
48                     ( ( blurDirection += 45 ) % 360 );
49            }
50            // -->
51         </script>
52      </head>
53
54      <body>
55         <form name = "myForm" action = "">
56
57         <table border = "1" style = "background-color: #CCFFCC">
58         <caption>Blur filter controls</caption>
59
60            <tr>
61               <td>Direction:</td>
62               <td><select name = "Direction">
63                  <option value = "0">above</option>
64                  <option value = "45">above-right</option>
65                  <option value = "90">right</option>
66                  <option value = "135">below-right</option>
67                  <option value = "180">below</option>
```

Fig. 15.8 Using the **blur** filter (part 2 of 4).

```
68                <option value = "225">below-left</option>
69                <option value = "270">left</option>
70                <option value = "315">above-left</option>
71             </select></td>
72          </tr>
73
74          <tr>
75             <td>Strength:</td>
76             <td><input name = "Strength" size = "3"
77                maxlength = "3" value = "0" /></td>
78          </tr>
79
80          <tr>
81             <td>Add original?</td>
82             <td><input type = "checkbox" name = "Add" /></td>
83          </tr>
84
85          <tr>
86             <td align = "center" colspan = "2">
87                <input type = "button" value = "Apply"
88                   onclick = "reBlur();" /></td>
89          </tr>
90
91          <tr>
92             <td colspan = "2">
93             <input type = "button" value = "Start demo"
94               onclick = "startDemo();" />
95             <input type = "button" value = "Stop demo"
96               onclick = "window.clearInterval( timer );" /></td>
97          </tr>
98
99       </table>
100      </form>
101
102      <div id = "blurImage" style = "position: absolute;
103         top: 0; left: 300; padding: 0; filter: blur(
104         add = 0, direction = 0, strength = 0 );
105         background-color: white;">
106         <img align = "middle" src = "shapes.gif"
107            alt = "Shapes" />
108      </div>
109
110   </body>
111 </html>
```

Fig. 15.8 Using the **blur** filter (part 3 of 4).

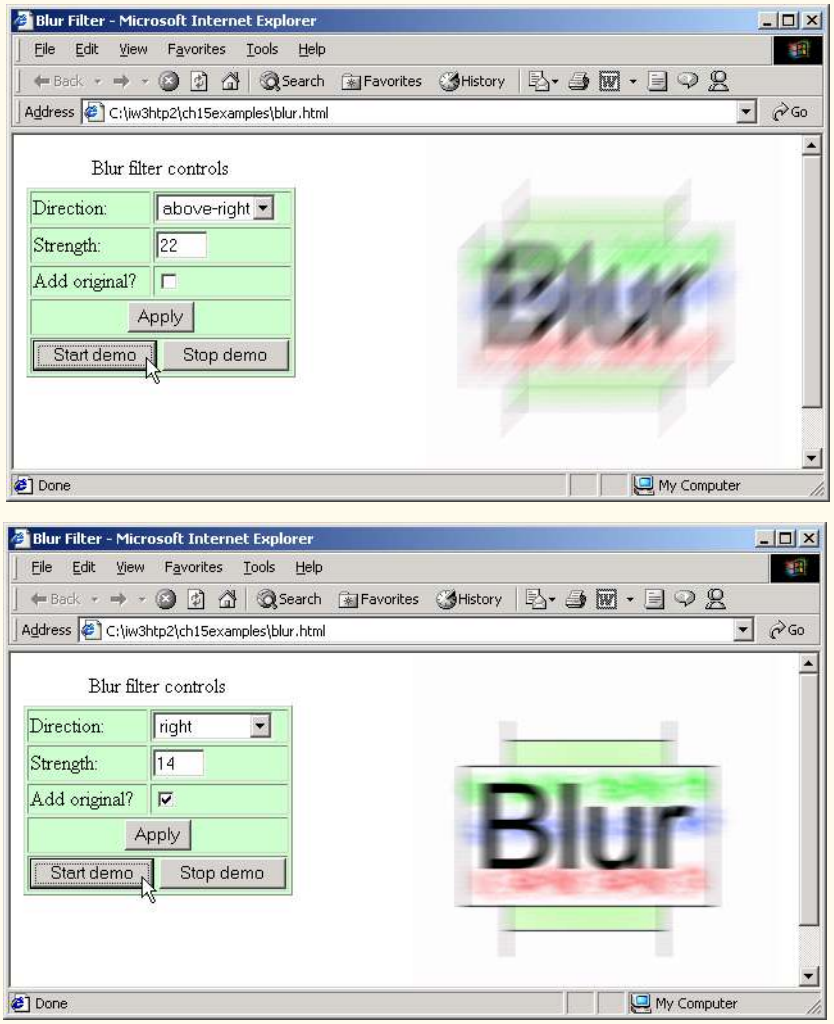


Fig. 15.8 Using the **blur** filter (part 4 of 4).

The three properties of the **blur** filter are ***add***, ***direction*** and ***strength***. The **add** property, when set to **true**, adds a copy of the original image over the blurred image, creating a more subtle blurring effect; Fig. 15.8 demonstrates the contrast between setting this to **true** or **false**.

The **direction** property determines in which direction the **blur** filter is applied. This is expressed in angular form (as we saw in Fig. 15.5 with the **shadow** filter). The **strength** property determines how strong the blurring effect is.

Lines 24–25

```
blurImage.filters( "blur" ).add =
     document.forms( "myForm" ).Add.checked;
```

assign to the **add** property of the **blur** filter the boolean **checked** property of the **Add** checkbox—if the box was checked, the value is **true**.

Lines 47–48

```
blurImage.filters( "blur" ).direction =
   ( ( blurDirection += 45 ) % 360 )
```

increment the **direction** property whenever the **strength** of the **blur** filter is 0 (i.e., whenever an iteration has completed). The value assigned to the **direction** property cycles through all the multiples of 45 between 0 and 360.

15.10 Using the wave Filter

The ***wave*** *filter* allows you to apply *sine-wave distortions* to text and images on your Web pages (Fig. 15.9).

```
1  <?xml version = "1.0"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
3     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
4
5  <!-- Fig 15.9: wave.html      -->
6  <!-- Applying the wave filter -->
7
8  <html xmlns = "http://www.w3.org/1999/xhtml">
9     <head>
10       <title>Wave Filter</title>
11
12       <script type = "text/javascript">
13          <!--
14          var wavePhase = 0;
15
16          function start()
17          {
18             window.setInterval( "wave()", 5 );
19          }
20
21          function wave()
22          {
23             wavePhase++;
24             flag.filters( "wave" ).phase = wavePhase;
25          }
26          // -->
27       </script>
28    </head>
29
30    <body onload = "start();">
31
32       <span id = "flag"
33          style = "align: center; position: absolute;
34          left: 30; padding: 15;
35          filter: wave(add = 0, freq = 1, phase = 0,
```

Fig. 15.9 Adding a **wave** filter to text (part 1 of 2).

```
36                strength = 10); font-size: 2em">
37          Here is some waaaavy text
38          </span>
39
40       </body>
41    </html>
```

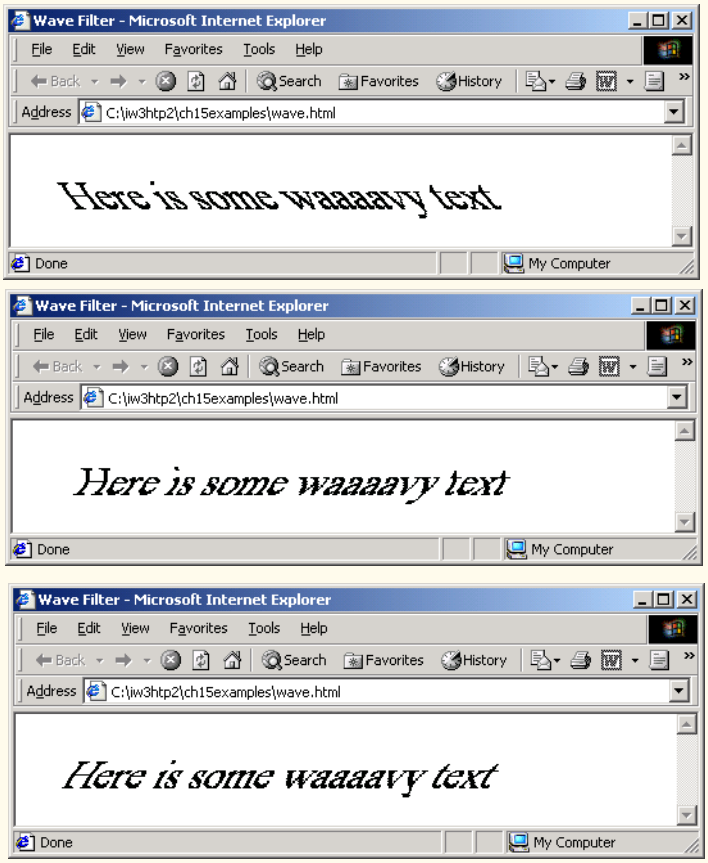


Fig. 15.9 Adding a **wave** filter to text (part 2 of 2).

The **wave** filter, as seen in lines 35–36,

```
filter: wave(add = 0, freq = 1, phase = 0,
   strength = 10);
```

has many properties. The **add** property, as in the case of the **blur** filter, adds a copy of the text or image underneath the filtered effect. The**add** property is useful only when applying the **wave** filter to images.

Performance Tip 15.2

*Applying the **wave** filter to images is processor intensive—if your viewers have inadequate processor power, your pages may act sluggishly on their systems.*

The **freq** property determines the *frequency of the wave* applied—i.e., how many complete sine waves are applied in the affected area. Increasing this property creates a more pronounced wave effect, but makes the text harder to read.

The **phase** property indicates the *phase shift of the wave*. Increasing this property does not modify any physical attributes of the wave, but merely shifts it in space. This property is useful for creating a gentle waving effect, as we do in this example. The last property, **strength**, is the amplitude of the sine wave that is applied.

In the script, lines 23–24

```
wavePhase++;
flag.filters( "wave" ).phase = wavePhase;
```

increment the **phase** shift of the wave in every call to the **wave** function.

15.11 Advanced Filters: dropShadow and light

Two filters that apply advanced image processing effects are the ***dropShadow*** and ***light*** *filters*. The **dropShadow** filter, as you can probably tell, applies an effect similar to the drop shadow we applied to our images with Photoshop Elements in Chapter 3—it creates a blacked-out version of the image, and places it behind the image, offset by a specified number of pixels.

The **light** filter is the most powerful and advanced filter available in Internet Explorer 5.5. It allows you to simulate the effect of a light source shining on your page. With scripting, this filter can be used with dazzling results. Figure 15.10 combines these two filters to create an interesting effect.

```
1  <?xml version = "1.0"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
3     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
4
5  <!-- Fig. 15.10: dropshadow.html                          -->
6  <!-- Using the light filter with the dropshadow filter  -->
7
8  <html xmlns = "http://www.w3.org/1999/xhtml">
9     <head>
10       <title>DHTML dropShadow and light Filters</title>
11
12       <script type = "text/javascript">
13          <!--
14          function setlight( )
15          {
16             dsImg.filters( "light" ).addPoint( 150, 150,
17                125, 255, 255, 255, 100 );
18          }
19
20          function run()
21          {
22             eX = event.offsetX;
23             eY = event.offsetY;
```

Fig. 15.10 Applying **light** filter with a **dropShadow** (part 1 of 2).

```
24
25             xCoordinate = Math.round(
26                eX-event.srcElement.width / 2, 0 );
27             yCoordinate = Math.round(
28                eY-event.srcElement.height / 2, 0 );
29
30             dsImg.filters( "dropShadow" ).offx =
31                xCoordinate / -3;
32             dsImg.filters( "dropShadow" ).offy =
33                yCoordinate / -3;
34
35             dsImg.filters( "light" ).moveLight(
36                0, eX, eY, 125, 1 );
37          }
38          // -->
39       </script>
40    </head>
41
42    <body onload = "setlight()" style = "background-color: green">
43
44       <img id = "dsImg" src = "circle.gif"
45          style = "top: 100; left: 100; filter: dropShadow(
46          offx = 0, offy = 0, color = black ) light()"
47          onmousemove = "run()" alt = "Circle Image" />
48
49    </body>
50 </html>
```

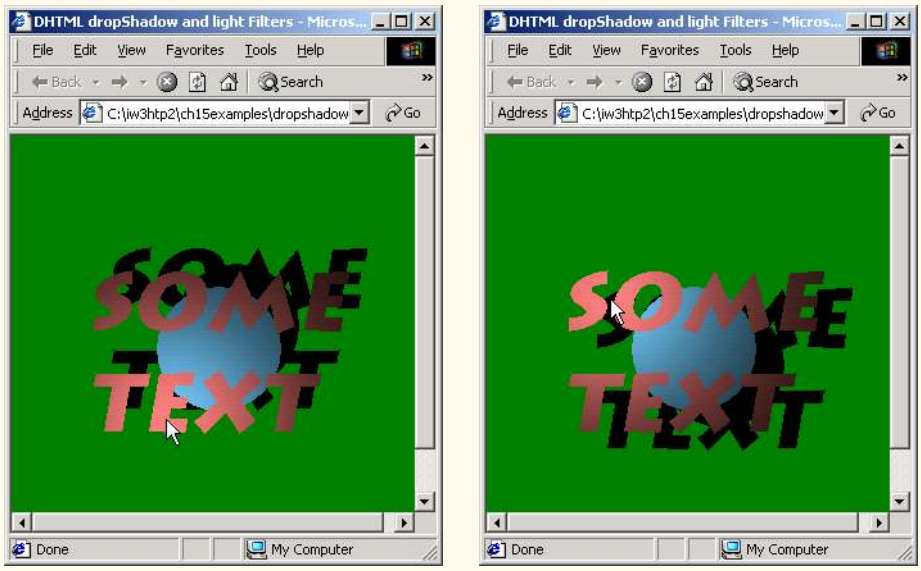


Fig. 15.10 Applying **light** filter with a **dropShadow** (part 2 of 2).

Let us begin by examining the **dropShadow** filter. In lines 44–47

```
<img id = "dsImg" src = "circle.gif"
```

```
style = "top: 100; left: 100; filter: dropShadow(
offx = 0, offy = 0, color = black ) light()"
 onmousemove = "run()" alt = "Circle Image" />
```

we apply the **dropShadow** filter to our image. The **offx** and **offy** properties determine by how many pixels the drop shadow is offset. The **color** property specifies the color of the drop shadow. Note that we also declare the **light** filter in line 46, although we do not give it any initial parameters—all the parameters and methods of the **light** filter are set by scripting. Lines 16–17

```
dsImg.filters( "light" ).addPoint( 150, 150,
   125, 255, 255, 255, 100 );
```

call the **addPoint** method of the **light** filter. This adds a *point light source*—a source of light which emanates from a single point and radiates in all directions. The first two parameters (**150, 150**) set the *x-y* coordinates at which to add the point source. In this case we place the source at the center of the image, which is 300-by-300 pixels.

The next parameter (**125**) sets the *height* of the point source. This simulates how far above the surface the light is situated. Small values create a small but high-intensity circle of light on the image, while large values cast a circle of light which is darker, but spreads over a greater distance.

The next three parameters (**255**, **255**, **255**) specify the RGB value of the light, in decimal. In this case we set the light to a color of white (**#FFFFFF**).

The last value (**100**), is a strength percentage—we set our light in this case to radiate with 100% strength.

This point light source creates a pleasant lighting effect, but it is static. We can use scripting to animate the light source in response to user actions. We use the **onmousemove** event (line 47) to have the light source follow the mouse cursor as the user moves it over the image. Lines 22–36

```
eX = event.offsetX;
eY = event.offsetY;

xCoordinate = Math.round(
   eX-event.srcElement.width / 2, 0 );
yCoordinate = Math.round(
   eY-event.srcElement.height / 2, 0 );

dsImg.filters( "dropShadow" ).offx =
   xCoordinate / -3;
dsImg.filters( "dropShadow" ).offy =
   yCoordinate / -3;

dsImg.filters( "light" ).moveLight(
   0, eX, eY, 125, 1);
```

of the **run** function animate both the **dropshadow** and **light** filters in response to user actions. First we set the variables **xCoord** and **yCoord** to the distance between the current cursor position (**eX** and **eY**, which were set to **event.offsetX** and **event.offsetY** on lines 22–23) and the middle of the image (**event.srcElement.width / 2** or **event.srcElement.height / 2**). In the next lines of code, we set the **offx** and

offy properties of the **dropShadow** filter relative to the current *x*-*y* coordinates of the image. We divide by a certain amount to create an effect of height (shadows cast by objects far from light sources only move a small amount when the light source moves by a larger amount).

We then call the **moveLight** method to update the position of the light source as well. The first parameter (**0**) is the index of the light source on the page. Multiple light sources have index numbers assigned to them in the order in which they are added. The next two parameters (**event.offsetX**, **event.offsetY**) specify the *x*-*y* coordinates to which we should move the light source. We use the **offsetX** and **offsetY** properties of the **event** object to move the light source to the current mouse cursor position over the image. The next parameter (**125**) specifies the height to which we move the light source. In this case, we keep the light source as the same level it was when we declared it. The last parameter (**1**) indicates that the values we are using are absolute. To move the light source by relative amounts instead, use a value of **0** for the last parameter of the **moveLight** function.

As you can see, combining the **dropShadow** and **light** filters creates a stunning effect that responds to user actions. The point source is not the only type of light source available for the light filter. Figure 15.11 demonstrates the use of a *cone light source* for illuminating an image.

```
<?xml version = "1.0"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<!-- Fig 15.11: conelight.html        -->
<!-- Automating the cone light source -->

<html xmlns = "http://www.w3.org/1999/xhtml">
   <head><title>Cone lighting</title>

   <script type = "text/javascript">
      var upDown = true;
      var counter = 0;
      var moveRate = -2;

      function setLight()
      {
         marquee.filters( "light" ).addCone( 0, marquee.height,
            8, marquee.width / 2, 30, 255, 150, 255, 50, 15 );
         marquee.filters( "light" ).addCone( marquee.width,
            marquee.height, 8, 200, 30, 150, 255, 255, 50, 15 );
         marquee.filters( "light" ).addCone( marquee.width / 2,
            marquee.height, 4, 200, 100, 255, 255, 150, 50, 50 );

         window.setInterval( "moveLight()", 100 );
      }

      function moveLight()
      {
         counter++;

```

Fig. 15.11 Dynamic cone source lighting (part 1 of 2).

```
32             if ( ( counter % 30 ) == 0 )
33                upDown = !upDown;
34
35             if ( ( counter % 10 ) == 0 )
36                moveRate *= -1;
37
38             if ( upDown ) {
39                marquee.filters( "light" ).moveLight(
40                   0, -1, -1, 3, 0 );
41                marquee.filters( "light" ).moveLight(
42                   1, 1, -1, 3, 0 );
43                marquee.filters( "light" ).moveLight(
44                   2, moveRate, 0, 3, 0);
45             }
46             else {
47                marquee.filters( "light" ).moveLight(
48                   0, 1, 1, 3, 0 );
49                marquee.filters( "light" ).moveLight(
50                   1, -1, 1, 3, 0 );
51                marquee.filters( "light" ).moveLight(
52                   2, moveRate, 0, 3, 0);
53             }
54          }
55       </script>
56       </head>
57       <body style = "background-color: #000000"
58          onload = "setLight()">
59
60          <img id = "marquee" src = "marquee.gif"
61             style = "filter: light; position: absolute; left: 25;
62             top: 25" alt = "Deitel movie marquee" />
63
64       </body>
65    </html>
```

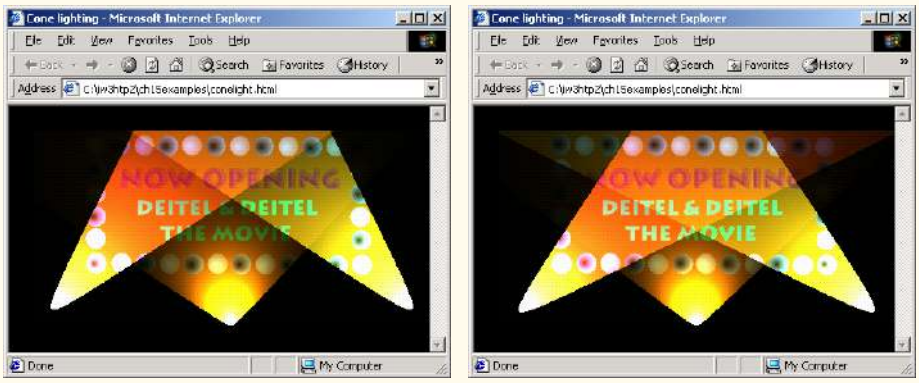


Fig. 15.11 Dynamic cone source lighting (part 2 of 2).

In lines 18–19

```
marquee.filters( "light" ).addCone( 0, marquee.height,
   8, marquee.width / 2, 30, 255, 150, 255, 50, 15 );
```

we add our first cone light source, using the **addCone** method. The parameters of this method are similar to the **addPoint** method. The first two parameters specify the *x-y* coordinates of the light source, and the third parameter specifies the simulated height above the page at which the light should be placed. The next two parameters (**marquee.width / 2**, **30**) are new—they specify the *x-y* coordinates at which the cone source is targeted. The next three parameters (**255**, **150**, **255**) specify the RGB value of the light which is cast, just as we did in the **addPoint** method. The next parameter (**50**) specifies the strength of the cone source, in a percentage (also equivalent to the strength parameter in the **addPoint** method). The last value (**15**) specifies the *spread* of the light source, in degrees (this can be set in the range **0**–**90**). In this case we set the spread of the cone to **15** degrees, illuminating a relatively narrow area.

In lines 39–40

```
marquee.filters( "light" ).moveLight(
   0, -1, -1, 3, 0 );
```

we use the **moveLight** method once again. When used on cone sources, the **moveLight** method moves the target of the light. In this case we set the last parameter to **0** to move the light by a relative amount, not an absolute amount, as we did in Fig 15.10.

15.12 Transitions I: Filter `blendTrans`

The transitions included with Internet Explorer 5.5 give the author control of many scriptable PowerPoint type effects. Transitions are set as values of the **filter** CSS property, just as regular filters are. We then use scripting to begin the transition. Figure 15.12 is a simple example of the ***blendTrans*** *transition*, which creates a smooth fade-in/fade-out effect.

```
1  <?xml version = "1.0"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
3     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
4
5  <!-- Fig 15.12: blendtrans.html -->
6  <!-- Blend transition           -->
7
8  <html xmlns = "http://www.w3.org/1999/xhtml">
9     <head>
10       <title>Using blendTrans</title>
11
12       <script type = "text/javascript">
13          <!--
14          function blendOut()
15          {
16             textInput.filters( "blendTrans" ).apply();
17             textInput.style.visibility = "hidden";
18             textInput.filters( "blendTrans" ).play();
```

Fig. 15.12 Using the **blendTrans** transition (part 1 of 2).

```
19            }
20            // -->
21         </script>
22      </head>
23
24      <body>
25
26         <div id = "textInput" onclick = "blendOut()" style =
27            "width: 300; filter: blendTrans( duration = 3 )">
28            <h1>Some fading text</h1>
29         </div>
30
31      </body>
32   </html>
```

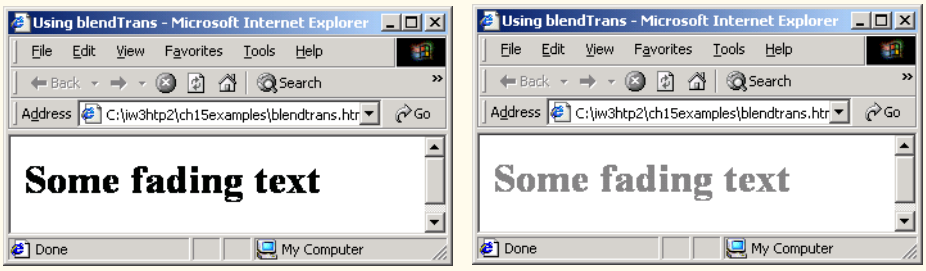


Fig. 15.12 Using the **blendTrans** transition (part 2 of 2).

First, in lines 26–27

```
style = "width: 300; filter:blendTrans( duration = 3 )">
```

sets the **filter** to **blendTrans** and the **duration** parameter to **3**. This determines how long the transition takes. All the rest of our work is done by scripting. In lines 16–18

```
textInput.filters( "blendTrans" ).apply();
textInput.style.visibility = "hidden";
textInput.filters( "blendTrans" ).play();
```

we invoke two methods of **blendTrans**. The **apply** method (line 16) initializes the transition for the affected element. Once this is done, we set the **visibility** of the element to **hidden**—this takes effect when we invoke the **play** method on line 18.

Figure 15.13 is a more complex example of the **blendTrans** transition. We use this to transition between two separate images.

```
1  <?xml version = "1.0"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
3     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
4
5  <!-- Fig 15.13: blendtrans2.html -->
6  <!-- Blend Transition            -->
7
```

Fig. 15.13 Blending between images with **blendTrans** (part 1 of 3).

```
 8  <html xmlns = "http://www.w3.org/1999/xhtml">
 9     <head>
10        <title>Blend Transition II</title>
11
12        <script type = "text/javascript">
13           <!--
14           var whichImage = true;
15
16           function blend()
17           {
18              if ( whichImage ) {
19                 image1.filters( "blendTrans" ).apply();
20                 image1.style.visibility = "hidden";
21                 image1.filters( "blendTrans" ).play();
22              }
23              else {
24                 image2.filters( "blendTrans" ).apply();
25                 image2.style.visibility = "hidden";
26                 image2.filters( "blendTrans" ).play();
27              }
28           }
29
30           function reBlend ( fromImage )
31           {
32              if ( fromImage ) {
33                 image1.style.zIndex -= 2;
34                 image1.style.visibility = "visible";
35              }
36              else {
37                 image1.style.zIndex += 2;
38                 image2.style.visibility = "visible";
39              }
40
41              whichImage = !whichImage;
42              blend();
43           }
44           // -->
45        </script>
46     </head>
47
48     <body style = "color: darkblue; background-color: lightblue"
49           onload = "blend()">
50
51        <h1>Blend Transition Demo</h1>
52
53        <img id = "image2" src = "cool12.jpg"
54        onfilterchange = "reBlend( false )"
55        style = "position: absolute; left: 50; top: 50;
56        width: 300; filter: blendTrans( duration = 4 );
57        z-index: 1" alt = "First Transition Image"  />
58
59        <img id = "image1" src = "cool8.jpg"
60        onfilterchange = "reBlend( true )"
61        style = "position: absolute; left: 50; top: 50;
```

Fig. 15.13 Blending between images with **`blendTrans`** (part 2 of 3).

```
62          width: 300; filter: blendTrans( duration = 4 );
63          z-index: 2" alt = "Second Transition Image"  />
64
65       </body>
66    </html>
```

Fig. 15.13 Blending between images with **blendTrans** (part 3 of 3).

We begin by placing two overlapping images on the page, with **id**s **image1** and **image2** (lines 53–63). The **body** tag's **onload** event (line 49) calls function **blend** as the **body** loads. The **blend** function checks the value of the **whichImage** variable, and, because it is set to **true**, begins a fade transition on **image1**. Because there are two

images in the same place, when **image1** fades out, it appears that **image2** fades in to replace it. When the transition is complete, **image1**'s **onfilterchange** event (line 60) fires. This calls function **reBlend**, which in lines 33–34

```
image1.style.zIndex -= 2;
image1.style.visibility = "visible";
```

changes the **zIndex** (the JavaScript version of the **z-index** CSS property) of **image1** so that it is now below **image2**. Once this is done, the image is made visible again. The function then toggles the **whichImage** property, and calls function **blend** so that the whole process starts again, now transitioning from **image2** back to **image1**.

15.13 Transitions II: Filter `revealTrans`

The ***revealTrans*** *filter* allows you to transition by using professional-style transitions, from *box out* to *random dissolve*. Figure 15.14 cycles through all 24 of these, transitioning from one image to another..

```
1  <?xml version = "1.0"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
3     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
4
5  <!-- Fig. 15.14: revealtrans.html   -->
6  <!-- Cycling through 24 transitions -->
7
8  <html xmlns = "http://www.w3.org/1999/xhtml">
9     <head>
10       <title>24 DHTML Transitions</title>
11
12    <script type = "text/javascript">
13       <!--
14       var transitionName =
15         ["Box In", "Box Out",
16          "Circle In", "Circle Out",
17          "Wipe Up", "Wipe Down", "Wipe Right", "Wipe Left",
18          "Vertical Blinds", "Horizontal Blinds",
19          "Checkerboard Across", "Checkerboard Down",
20          "Random Dissolve",
21          "Split Vertical In", "Split Vertical Out",
22          "Split Horizontal In", "Split Horizontal Out",
23          "Strips Left Down", "Strips Left Up",
24          "Strips Right Down", "Strips Right Up",
25          "Random Bars Horizontal", "Random Bars Vertical",
26          "Random"];
27
28       var counter = 0;
29       var whichImage = true;
30
31       function blend()
32       {
33          if ( whichImage ) {
```

Fig. 15.14 Transitions using **revealTrans** (part 1 of 4).

```
34             image1.filters( "revealTrans" ).apply();
35             image1.style.visibility = "hidden";
36             image1.filters( "revealTrans" ).play();
37          }
38          else {
39             image2.filters( "revealTrans" ).apply();
40             image2.style.visibility = "hidden";
41             image2.filters( "revealTrans" ).play();
42          }
43       }
44
45       function reBlend( fromImage )
46       {
47          counter++;
48
49          if ( fromImage ) {
50             image1.style.zIndex -= 2;
51             image1.style.visibility = "visible";
52             image2.filters("revealTrans").transition =
53                counter % 24;
54          }
55          else {
56             image1.style.zIndex += 2;
57             image2.style.visibility = "visible";
58             image1.filters("revealTrans").transition =
59                counter % 24;
60          }
61
62          whichImage = !whichImage;
63          blend();
64          transitionDisplay.innerHTML = "Transition " +
65             counter % 24 + ": " + transitionName[ counter % 24 ];
66       }
67       // -->
68    </script>
69    </head>
70
71    <body style = "color: white; background-color: lightcoral"
72          onload = "blend()">
73
74       <img id = "image2" src = "icontext.gif"
75            style = "position: absolute; left: 10; top: 10;
76            width: 300; z-index:1; visibility: visible;
77            filter: revealTrans( duration = 2, transition = 0 )"
78            onfilterchange = "reBlend( false )" alt =
79            "Programming Tips" />
80
81       <img id = "image1" src = "icons2.gif"
82            style = "position: absolute; left: 10; top: 10;
83            width: 300; z-index:1; visibility: visible;
84            filter: revealTrans( duration = 2, transition = 0 )"
85            onfilterchange = "reBlend( true )" alt = "Icons" />
86
87       <div id = "transitionDisplay" style = "position: absolute;
```

Fig. 15.14 Transitions using **revealTrans** (part 2 of 4).

```
88            top: 70; left: 80">Transition 0: Box In</div>
89
90       </body>
91    </html>
```

Fig. 15.14 Transitions using **revealTrans** (part 3 of 4).

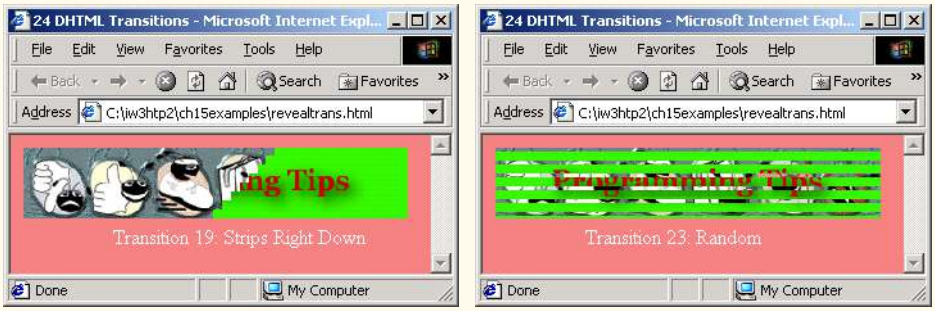


Fig. 15.14 Transitions using **revealTrans** (part 4 of 4).

The script in this example is almost the same as the script in the **blendTrans** example. In lines 52–53

```
image2.filters("revealTrans").transition =
   counter % 24;
```

we set the **transition** property of the image, which determines what visual transition is used here. There are 24 different visual transitions (their names are listed in the **transitionName** array) for updating the **div** element **transitionDisplay**.

SUMMARY

- Applying filters to text and images causes changes that are persistent.
- Transitions are temporary phenomena; applying a transition allows you to transfer from one page to another with a pleasant visual effect, such as a random dissolve.
- Filters and transitions do not add content to your pages—rather, they present existing content in an engaging manner to help hold the user's attention.
- Each of the visual effects achievable with filters and transitions is programmable, so these effects can be adjusted dynamically by programs that respond to user-initiated events like mouse clicks and keystrokes.
- When Internet Explorer renders your page, it applies all the special effects and does this while running on the client computer without lengthy waits for files to download from the server.
- The **flipv** and **fliph** filters mirror text or images vertically and horizontally, respectively.
- Filters are applied in the **style** attribute. The **filter** property's value is the name of the filter.
- More than one filter can be applied at once. Enter multiple filters as values of the **filter** attribute, separated by spaces.
- The **chroma** filter applies transparency effects dynamically, without using a graphics editor to hard-code transparency into the image.
- Use the **parseInt** function to convert a string to a hexadecimal integer for setting the **color** property of the **chroma** filter. The second parameter of **parseInt** specifies the base of the integer.
- Each filter has a property named **enabled**. If this property is set to **true**, the filter is applied. If it is set to **false**, the filter is not applied.

- The **onchange** event fires whenever the **value** of a form field changes.
- Applying the **mask** filter to an image allows you to create an image mask, in which the background of an element is a solid color and the foreground of an element is transparent to the image or color behind it.
- Parameters for filters are always specified in the format *param = value*.
- The **invert** filter applies a negative image effect—dark areas become light, and light areas become dark.
- The **gray** filter applies a grayscale image effect, in which all color is stripped from the image and all that remains is brightness data.
- The **xray** filter applies an x-ray effect which is basically just an inversion of the grayscale effect.
- A simple filter that adds depth to your text is the **shadow** filter. This filter creates a shadowing effect that gives your text a three-dimensional look. The **direction** property of the **shadow** filter determines in which direction the shadow effect will be applied—this can be set to any of eight directions, expressed in angular notation: **0** (up), **45** (above-right), **90** (right), **135** (below-right), **180** (below), **225** (below-left), **270** (left) and **315** (above-left). The **color** property of the **shadow** filter specifies the color of the shadow that is applied to the text.
- Internet Explorer 5.5 allows you to create gradient effects dynamically, using the **alpha** filter. The **style** property of the filter determines in what style the opacity is applied; a value of **0** applies uniform opacity, a value of **1** applies a linear gradient, a value of **2** applies a circular gradient and a value of **3** applies a rectangular gradient. The **opacity** and **finishopacity** properties are both percentages determining at what percent opacity the specified gradient will start and finish, respectively. Additional attributes are **startX**, **startY**, **finishX** and **finishY**. These allow you to specify at what *x-y* coordinates the gradient starts and finishes in that element.
- The **glow** filter allows you to add an aura of color around your text. The **color** and **strength** can both be specified.
- The **blur** filter creates an illusion of motion by blurring text or images in a certain direction. The **blur** filter can be applied in any of eight directions, and its strength can vary. The **add** property, when set to **true**, adds a copy of the original image over the blurred image, creating a more subtle blurring effect. The **direction** property determines in which direction the **blur** filter will be applied. This is expressed in angular form (as with the **shadow** filter). The **strength** property determines how strong the blurring effect is.
- The **wave** filter allows you to apply sine-wave distortions to text and images on your Web pages.
- The **add** property, as in the case of the **blur** filter, adds a copy of the text or image, but underneath the filtered effect. The **add** property is useful when applying the **wave** filter to images. The **freq** property determines the frequency of the wave applied—i.e., how many complete sine waves are applied in the affected area. Increasing this property would create a more pronounced wave effect, but makes the text harder to read. The **phase** property indicates the phase shift of the wave. Increasing this property does not modify any physical attributes of the wave, but merely shifts it in space. This property is useful for creating a gentle waving effect, as we do in this example. The last property, **strength**, is the amplitude of the sine wave that is applied.
- Two filters that apply advanced image processing effects are the **dropShadow** and **light** filters. The **dropShadow** filter applies an effect similar to the drop shadow we applied to our images in Chapter 3—it creates a blacked-out version of the image, and places it behind the image, offset by a specified number of pixels.
- The **light** filter is the most powerful and advanced filter available in Internet Explorer 5.5. It allows you to simulate the effect of a light source shining on your page.

- The **offx** and **offy** properties of the **dropShadow** filter determine by how many pixels the drop shadow offsets. The **color** property specifies the color of the drop shadow.
- All the parameters and methods of the light **filter** are done by scripting. The **addPoint** method adds a point light source—a source of light which emanates from a single point and radiates in all directions. The first two parameters set the *x*-*y* coordinates at which to add the point source. The next parameter sets the height of the point source.This simulates how far above the surface the light is situated. Small values create a small but high-intensity circle of light on the image, while large values cast a circle of light which is darker, but spreads over a greater distance. The next three parameters specify the RGB value of the light, in decimal.The last parameter is a strength percentage.
- The **moveLight** method updates the position of the light source. The first parameter is the index of the light source on the page. Multiple light sources have index numbers assigned to them in the order they are added. The next two parameters specify the *x*-*y* coordinates to which we should move the light source. The next parameter specifies the height to which we move the light source. Setting the last parameter to **1** indicates that the values we are using are absolute. To move your light source by relative amounts instead, use a value of **0** for the last parameter of the **moveLight** function.
- The parameters of the **addCone** method are similar to the **addPoint** method. The first two parameters specify the *x*-*y* coordinates of the light source, and the third parameter specifies the simulated height above the page at which the light should be placed. The next two parameters specify the *x*-*y* coordinates at which the cone source is targeted. The next three parameters specify the RGB value of the light which is cast, just as in the **addPoint** method. The next parameter specifies the strength of the cone source, in a percentage. The last value specifies the spread of the light source, in degrees (this can be set in the range **0–90**).
- The transitions included with Internet Explorer 5.5 give the author control of scriptable PowerPoint type effects. Transitions are set as values of the **filter** CSS property, just as regular filters are.
- The **duration** parameter of **blendTrans** determines how long the transition will take.
- The **apply** method initializes the transition for the affected element. The **play** method then begins the transition.
- The **revealTrans** filter allows you to transition using professional-style transitions, from Box Out to Random Dissolve. The **transition** property determines what visual transition is used. There are 24 different visual transitions.

TERMINOLOGY

add property of **blur** filter
add property of **wave** filter
addCone method of **light** filter
addPoint method of **light** filter
alpha filter
blendTrans filter
color property of **shadow** filter
combining filters
cone light source
CSS **filter** property
direction property of **blur** filter
direction property of **shadow** filter
dropShadow filter
duration of **blendTrans** filter
enabled property of each filter
fade-in/fade-out effect
filter
filter property with **style** attribute
filter **strength**
filter:alpha
filter:blur
filter:chroma
filter:dropshadow
filter:fliph
filter:flipv
filter:glow
filter:gray
filter:invert
filter:light
filter:mask
filter:shadow
filter:wave
filter:xray
finishopacity property of **alpha** filter
finishx property of **alpha** filter
finishy property of **alpha** filter
flipH filter
flipV filter
freq property of **wave** filter
glow filter
gradient
gray filter
grayscale image effect
height of light source
blur filter
chroma filter
circular gradient
color property of **chroma** filter
color property of **dropshadow** filter
color property of **glow** filter
horizontal blinds transition
illusion of motion by blurring
image mask
invert filter
light filter
linear opacity
mask filter
moveLight property of **light** filter
negative image effect with **invert** filter
offx property of **dropshadow** filter
offy property of **dropshadow** filter
opacity property of **alpha** filter
padding (CSS)
phase property of **wave** filter
phase shift of a wave
point light source
radial opacity
random dissolve transition
rectangular opacity
revealTrans filter
shadow filter
sine-wave distortions
spread of cone light source
startx property of **alpha** filter
starty property of **alpha** filter
strength property of **blur** filter
strength property of **glow** filter
strength property of **wave** filter
style property of **alpha** filter
three-dimensional effect with **shadow** filter
transition effects
transparency effects
uniform opacity
vertical blinds transition
visibility
visual filters
wave filter
xray filter

SELF-REVIEW EXERCISES

15.1 State whether the following are *true* or *false*. If *false*, explain why.

a) You can determine the strength of the **shadow** filter.
b) The **flip** filter flips text horizontally.
c) The **mask** filter makes the foreground of an element transparent.
d) The **freq** property of the wave filter determines how many sine waves are applied to that element.
e) Increasing the margin of an element prevents the **glow** filter from being clipped by the element's border.
f) The **apply** method begins a transition.
g) The **invert** filter creates a negative image effect.

h) The **add** property adds a duplicate image below the affected image.

15.2 Fill in the blanks in the following statements:

a) You must use the _____ function to pass a value to the **color** property.
b) The last parameter of the **moveLight** method determines whether the move is or _____.
c) The amplitude of the **wave** filter is controlled by the _____ property.
d) There are _____ **direction**s in which the **blur** filter can be applied.
e) There are two coordinate pairs in the parameters of the **addCone** method: the _____ and the _____.
f) There are _____ different transition styles for the **revealTrans** transition.
g) The two properties of the **dropShadow** filter that specify the offset of the shadow are _____ and _____.
h) The four styles of opacity are _____, _____, _____ and _____.
i) The _____ filter creates a grayscale version of the effected image.

ANSWERS TO SELF-REVIEW EXERCISES

15.1 a) False; there is no **strength** property for the **shadow** filter. b) False; the **flipH** filter flips text horizontally. c) True. d) True. e) False; increasing the padding of an element prevents clipping. f) False; the **play** method begins a transition. g) True. h) True.

15.2 a) **parseInt**. b) relative, absolute. c) **strength**. d) eight. e) source, target. f) 24. g) **offx**, **offy**. h) uniform, linear, circular, rectangular. i) **gray**.

EXERCISES

15.3 Create a Web page that applies the **invert** filter to an image if the user moves the mouse over the image.

15.4 Create a Web page that applies the **glow** filter to a hyperlink if the user moves the mouse over the link.

15.5 Write a script that **blur**s images and slowly unblurs them when they are finished loading into the browser (use event **onload** for the image).

15.6 Write a script that creates a cone **light** filter that tracks mouse movements across the page.

15.7 Write a script that uses the **blendTrans** filter to transition into an image after the image fully loads (use event **onload** for the image).

15.8 Write a script that changes the attributes of an **alpha** filter every 20 seconds (see **setInterval** in Chapter 13). Change both the color and the style of the **alpha** filter every time.

15.9 *(Slide Show)* Use the **revealTrans** filter to present your own slide show in a Web page. On each transition, display a new image.

15.10 *(Image Selector)* Design a Web page that allows the user to choose from a series of images and allows the user to view the image in color and in grayscale.