Pro CSS and HTML Design Patterns

Michael Bowers

Apress®

Pro CSS and HTML Design Patterns

Copyright © 2007 by Michael Bowers

All rights reserved. No part of this work may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage or retrieval system, without the prior written permission of the copyright owner and the publisher.

ISBN-13 (pbk): 978-1-59059-804-7

ISBN-10 (pbk): 1-59059-804-0

Printed and bound in the United States of America 9 8 7 6 5 4 3 2 1

Trademarked names may appear in this book. Rather than use a trademark symbol with every occurrence of a trademarked name, we use the names only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

Lead Editor: Chris Mills
Technical Reviewer: Paul Haine
Editorial Board: Steve Anglin, Ewan Buckingham, Gary Cornell, Jason Gilmore, Jonathan Gennick, Jonathan Hassell, James Huddleston, Chris Mills, Matthew Moodie, Jeff Pepper, Dominic Shakeshaft, Jim Sumser, Matt Wade
Project Manager: Kylie Johnston
Copy Edit Manager: Nicole Flores
Copy Editor: Ami Knox
Assistant Production Director: Kari Brooks-Copony
Production Editor: Laura Esterman
Compositor: Dina Quan
Proofreader: Elizabeth Berry
Indexer: Julie Grady
Cover Designer: Kurt Krames
Manufacturing Director: Tom Debolski

Distributed to the book trade worldwide by Springer-Verlag New York, Inc., 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax 201-348-4505, e-mail `orders-ny@springer-sbm.com`, or visit `http://www.springeronline.com`.

For information on translations, please contact Apress directly at 2560 Ninth Street, Suite 219, Berkeley, CA 94710. Phone 510-549-5930, fax 510-549-5939, e-mail `info@apress.com`, or visit `http://www.apress.com`.

The information in this book is distributed on an "as is" basis, without warranty. Although every precaution has been taken in the preparation of this work, neither the author(s) nor Apress shall have any liability to any person or entity with respect to any loss or damage caused or alleged to be caused directly or indirectly by the information contained in this work.

The source code for this book is available to readers at `http://www.apress.com` in the Source Code/Download section.

CHAPTER 14

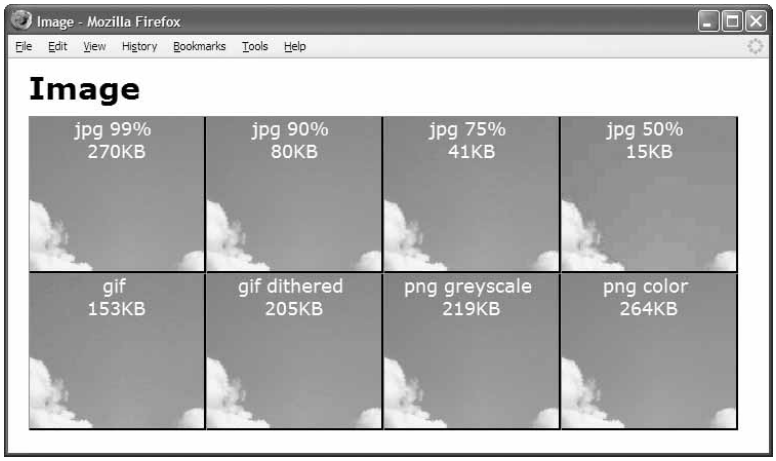
Images

This chapter shows how to use images to create beautiful and functional documents that remain accessible and download quickly.

Chapter Outline

- **Image** shows how to use the `<img>` element. It also contrasts the advantages and disadvantages of the GIF, JPG, and PNG image formats.
- **Image Map** shows how to overlay an image with clickable areas that link to other pages.
- **Fade-out** shows how to use gradient images to add subtle shading behind content. It also shows how to create chameleon gradients that adapt to the current background.
- **Semi-transparent** shows how to put a partially transparent background behind an element so that it stands out from the background below it without obscuring it.
- **Replaced Text** shows how to replace text with an image while remaining accessible to nonsighted users. This technique also shows the text when the image is unavailable.
- **Content over Image** shows how to overlay text and other images on top of an image.
- **Content over Background Image** shows how to overlay text and other images on top of a background image.
- **CSS Sprite** shows how to embed multiple images into one file and display them independently as the background of different elements of a document.
- **Basic Shadowed Image** shows how to create and apply a simple shadow to an image without modifying the image itself.
- **Shadowed Image** shows a generic way of applying a shadow to an image of any size.
- **Rounded Corners** shows how to round the corners of an element's borders and how to create custom borders of any style imaginable.
- **Image Example** showcases these patterns in one document.

Image



HTML

```
<img width="742" height="556" src="cl1-99.jpg" alt="Crater Lake 1" />

<!-- Nonessential markup is not shown. -->
```

CSS

```
img { display:block; width:auto; height:auto; }

/* Nonessential rules are not shown. */
```

Example

The example contains eight different versions of a picture that I took of Crater Lake on August 4, 2003. The source image is 742×556 pixels with a file size of 1,238,822 bytes. I processed the image to create eight separate files—each with a different image type and quality.

The first image is a JPG image at maximum quality, which reduces the file size to 275,798 bytes. This is a reduction of 5 times. At a JPG's highest quality, it is difficult to see any loss of quality. The second image is a JPG at 90% quality, which reduces the file size to 81,248 bytes. This is a reduction of 15 times. At 90% quality, you can barely see a difference with a magnifying glass. You can see a difference in the third and fourth images, which are JPGs at 75% and 50% quality and 41,290 and 14,841 bytes. This is a reduction of 30 and 84 times.

The fifth and sixth images are GIFs. These images have less quality and larger sizes than the JPG images. This is not a fair test of GIFs because they are not designed for real-world images containing thousands of colors. GIFs produce smaller files and have better quality when used for computer-generated images containing 256 or less colors.

The seventh and eight images are PNGs. These images have the best quality with slightly smaller file sizes than the best-quality JPG, but there is no way to increase the compression to shrink the file size.

Image

Problem You want to insert an image into the document because it is part of the content.

Solution You can insert an image into your document using `<img>`. You can use the `src` attribute to specify the URL containing the image.

You should put a brief description of the image in the `alt` attribute. This alternative description should be written specifically for screen readers to read and for displaying when the image fails to download. Decorative images are best displayed as background images, but if you must use a decorative `<img>` element, include the `alt` attribute, but leave it empty.

Because a browser downloads each image separately, it needs to know the image's height and width so it can create a placeholder for the image while the image downloads. Otherwise, after *each* image is downloaded and its real size becomes known, a browser has to reflow the page. This slows the rendering and annoys the user. To set an image's size, you can use the `width` and `height` attributes of `<img>` or the `width` and `height` CSS properties. There is no need to use both. CSS properties override HTML attributes.

Pattern

HTML

```
<img src="FILE.EXT" width="IMAGE_WIDTH" height="IMAGE_HEIGHT"
  alt="BRIEF_IMAGE_DESCRIPTION" />
```

Location This pattern applies to images.

Tips An image is an inline element. It vertically aligns to the baseline of the line in which it occurs. You can use `vertical-align` to adjust the alignment.

When you want to treat an image as a block, you should use `display:block` to display it as a block. This removes a small amount of extra space that a browser places below an image when it is inline, and it preserves the image's size when it fails to download.

JPG, GIF, and PNG are the most common types of images on the Internet.

JPG is the best image format for photographs. JPG supports up to 16 million colors and lossy compression. You control the amount of lossy compression from none to extreme. More compression produces smaller files and poorer quality. JPG does not support transparency.

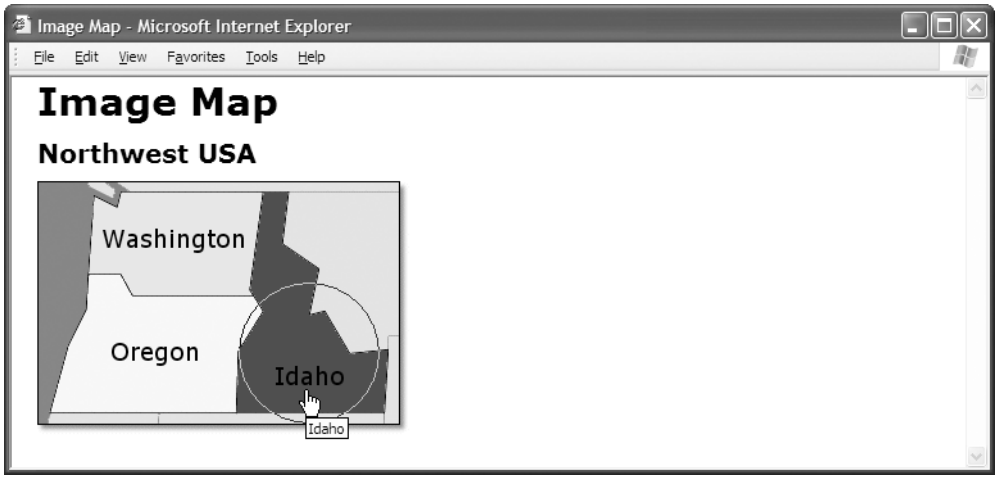
GIF is the best image format for line art and computer-generated images. GIF supports a transparent background, but it does not support an alpha channel. GIF supports up to 256 colors in the palette. To get more colors, a graphics program may use dithering to simulate them. GIF uses lossless compression. You cannot control the amount of compression. The main problem with GIF is its limit of 256 colors and its lack of an alpha channel.

PNG is an improvement over GIF. It supports alpha channel transparency, 16 million colors, grayscale, and palette-based colors. PNG uses lossless compression, which you cannot control. Internet Explorer 7 and other major browsers support PNG transparency. Internet Explorer 6 does not.

Related to Image Map; Inline-block Box (Chapter 4); Width, Height, Sized, Shrinkwrapped, Stretched (Chapter 5); Margin, Border, Padding (Chapter 6); Vertical-aligned Content, Vertical-offset Content (Chapter 12); Left Marginal, Right Marginal (Chapter 13); Flyout Menu (Chapter 17); JavaScript Alert, Tooltip Alert, Popup Alert (Chapter 20)

See also `www.cssdesignpatterns.com/image`

Image Map



HTML

```
<h1>Image Map</h1>

<h2><a id="home" href="example.html">Northwest USA</a></h2>

<img src="nw.gif" usemap="#nw-map" alt="Northwest USA" width="290" height="200" />

<map id="nw-map" name="nw-map">

  <area href="washington.html" alt="Washington"
    shape="poly" coords="176,8, 164,89, 75,89, 40,72, 45,8" />

  <area href="oregon.html" alt="Oregon"
    shape="rect" coords="9,90, 155,180" />

  <area href="idaho.html" alt="Idaho"
    shape="circle" coords="212, 134,55" />
 </map>
```

CSS

```
/* There are no CSS properties for styling image maps. */
```

Image Map

Problem You want to overlay an image with clickable areas that link to other pages.

Solution You can link an image to a `map` element that defines clickable areas and associates each area with a URL. When a user clicks an area, a browser jumps to its associated link. You can add a `usemap` attribute to an image to link the image to the `map` element with the same value in its `name` attribute. Multiple images can be linked to the same `map` element. For easy access to the element through JavaScript, it is a good practice for `map` elements to have an `id` attribute with the same value as its `name` attribute.

A `map` element contains one or more `area` elements. Each `area` defines a region of an image that can be clicked. Areas should not overlap, but if they do, the document order of `area` elements determines the stacking order.

Each `area` has four required attributes: `href`, `alt`, `shape`, and `coords`. `href` is the URL of the link that a browser jumps to when a user clicks the area. `alt` is read by screen readers to describe the link—it is not visible. `shape` is the shape of the area, which is one of three shapes: `rect`, `circle`, and `poly`. `coords` define the location and extent of the shape.

The number and meaning of coordinates in `coords` varies with each type of shape. Rectangles require four comma-delimited numbers. The first two are x, y coordinates of the upper-left corner of the rectangle, and the second two are x, y coordinates of the lower-right corner. Circles require three comma-delimited numbers. The first two are x, y coordinates of the circle's center, and the third is its radius. Polygons require a series of comma-delimited numbers in pairs of x, y coordinates that define the points of the polygon.

This design pattern does not use any CSS styles.

Pattern

HTML

```
<img usemap="MAP_NAME" src="FILE.EXT"
  width="WIDTH" height="HEIGHT" alt="DESCRIPTION" />

<map name="MAP_NAME" id="MAP_NAME">
  <area href="URL" shape="RECT_CIRCLE_POLY" coords="x,y..."
    alt="SCREENREADER_DESCRIPTION" />
</map>
```

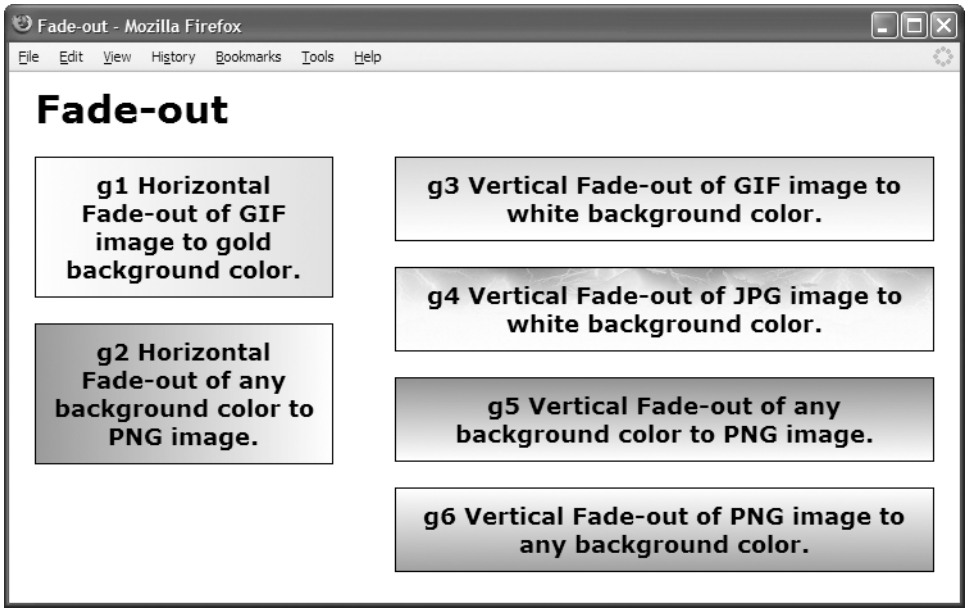
Location This pattern applies to images and image maps.

Tip Image maps work well when you want a user to explore something visual, such as a real-world map. The problem is that image maps are invisible. Other than the mouse pointer changing shape when it is over a clickable area, a user cannot tell where areas are located, how many areas there are, and which areas have already been visited. For this reason, image maps are often paired with redundant links that are absolutely positioned over the image. These links make it clear what is clickable and what has already been visited. The example at the end of the chapter shows how this works.

Related to Image, Content over Image, Content over Background Image

See also `www.cssdesignpatterns.com/image-map`

Fade-out



HTML

```
<h1>Fade-out</h1>

<h2 class="g1">g1 Horizontal Fade-out of GIF image to gold background color.</h2>
<h2 class="g2">g2 Horizontal Fade-out of any background color to PNG image.</h2>

<h2 class="g3">g3 Vertical Fade-out of GIF image to white background color.</h2>
<h2 class="g4">g4 Vertical Fade-out of JPG image to white background color.</h2>
<h2 class="g5">g5 Vertical Fade-out of any background color to PNG image.</h2>
<h2 class="g6">g6 Vertical Fade-out of PNG image to any background color.</h2>
```

CSS

```
*.g1 { background:url("h-white2gold.gif") repeat-y left top gold; }
*.g2 { background:url("h-trans2white.png") repeat-y right top royalblue; }

*.g3 { background:url("v-gold2white.gif") repeat-x left top white; }
*.g4 { background:url("v-lightning.jpg") repeat-x left top white; }
*.g5 { background:url("v-trans2white.png") repeat-x left bottom red; }
*.g6 { background:url("v-white2trans.png") repeat-x left top green; }


/* Nonessential rules are not shown. */
```

Fade-out

Problem You want to create a gradient background behind an element. You want the gradient to work well regardless of how wide or tall the element grows.

Solution There are two keys to creating a scalable background **gradient**: (1) fading the gradient into the background color, and (2) tiling it in the opposite direction of the gradient. For example, when the gradient is horizontal, you can tile the image vertically, and vice versa. This allows the element to grow in any direction while preserving the gradient effect. As an element grows, the background color fills in where the background image ends, and the image tiles to fill in the opposite direction.

Using a graphics program, you can create a **gradient image**, such as a JPG, GIF, or PNG, that transitions from the forecolor and backcolor of your choosing. For example, if your document's background color is white and you want your forecolor to be gold, you could create a gradient image that transitions from white to gold or vice versa.

Using a graphics program, you can use a **gradient mask** to fade any image, illustration, or graphical text into the background color. In the example, the fourth heading has a background image created from a texture that fades out to the white background color.

You can also create a generic PNG image that fades from a predefined forecolor to whatever background color is currently assigned to the element. In the example, the second, fifth, and sixth headings use PNG images that fade from white to *transparent*. You can change the background color, and the image fades from white to that color. It just takes one of these **chameleon PNG gradients** to transition to *any* background color!

The following design patterns show how to align and tile gradients in all four directions.

Patterns

Horizontal Left-to-Right Fade-out

```
SELECT { background:url("FILE.EXT") repeat-y left top COLOR; }
```

Horizontal Right-to-Left Fade-out

```
SELECT { background:url("FILE.EXT") repeat-y right top COLOR; }
```

Horizontal Top-to-Bottom Fade-out

```
SELECT { background:url("FILE.EXT") repeat-x left top COLOR; }
```

Horizontal Bottom-to-Top Fade-out

```
SELECT { background:url("FILE.EXT") repeat-x left bottom COLOR; }
```

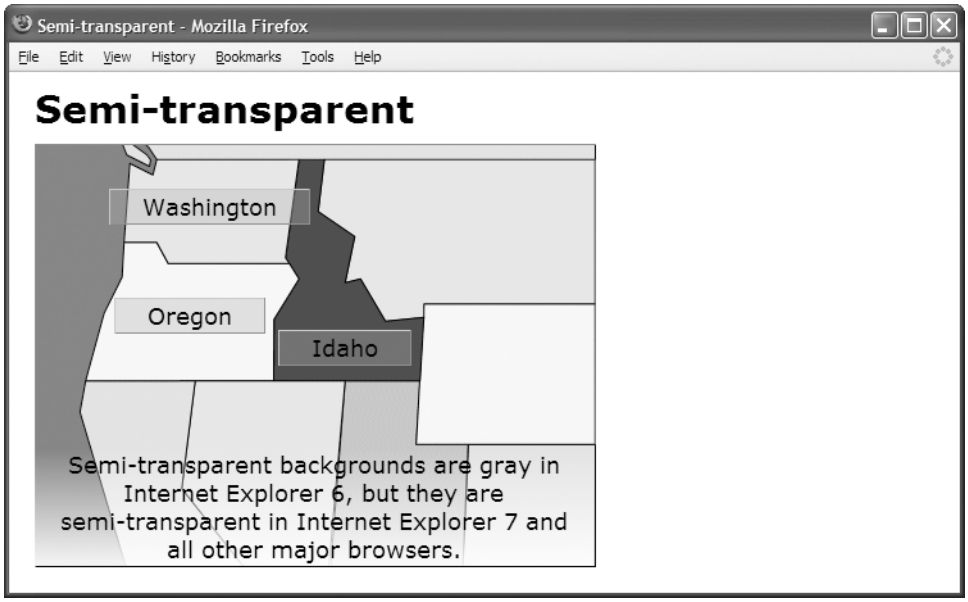
Location This pattern applies to all elements.

Limitations Internet Explorer 6 does not support PNG transparency, but Internet Explorer 7 and the other major browsers do. In the example, the PNG images show up in Internet Explorer 6 as gray gradients, which is not a bad effect in and of itself.

Related to Semi-transparent; Background (Chapter 6)

See also `www.cssdesignpatterns.com/image`

Semi-transparent



HTML

```
<h1>Semi-transparent</h1>

<div id="nw">
  <img src="nw.gif" alt="Northwest" width="437" height="328" />

  <span id="washington" class="overlay">Washington</span>
  <span id="oregon" class="overlay">Oregon</span>
  <span id="idaho" class="overlay">Idaho</span>

  <p id="note1">
    Semi-transparent backgrounds are gray in Internet Explorer 6, but they are
    semi-transparent in Internet Explorer 7 and all other major browsers.</p>
</div>
```

CSS

```
*.overlay { background:url("semi-transparent.png") repeat; }

#note1 { background:url("trans2white.png") bottom left repeat-x; }

/* Nonessential rules are not shown. */
```

Semi-transparent

Alias Translucent

Problem You want an element to have a partially transparent background so that it stands out from the background below it without obscuring it.

Solution You can use a graphics program to create a semi-transparent PNG image. You can set the transparency of its background to some value less than 100% to make it partially transparent. You can also use a gradient mask to fade into transparency. The color or colors you use in this image are important. Semi-transparent grayscale colors are color-neutral when they overlay a background. Nongrayscale semi-transparent colors colorize.

If the image has the same transparency throughout, it only needs to have a height and width of about 10 pixels so a browser can efficiently tile it to fill the background of its container. For example, the `semi-trnsparent.png` image in the example is 10 pixels square, and I use `background:repeat` to tile it throughout the background. If the image contains a vertical transparent gradient, it needs to be about 10 pixels wide and as tall as the gradient. For example, the `trans2white.png` in the example is 10 pixels wide and 100 pixels tall to fit the gradient. I use `background:repeat-x` to tile it horizontally across the background. If the image contains a horizontal gradient, it needs to be about 10 pixels tall and as wide as the gradient, and you can tile it vertically down the background.

Pattern **CSS**

```
SELECT { background:url("SEMI_TRANSPARENT_FILE.png") repeat; }
```

Location This pattern applies to all elements.

Limitations Internet Explorer 6 does not support PNG transparency, but Internet Explorer 7 and the other major browsers do. In the example, the PNG images show up in Internet Explorer 6 as gray gradients, which is a nice way for the effect to degrade.

Advantages Semi-transparency is practical and looks great as long as the color of the text contrasts well with the background. I expect to see more demand for this technique now that Windows Vista has joined the other major operating systems in building transparency effects into the desktop.

Example In the example, the four spans positioned over the image have semi-transparent gray backgrounds. I created this effect by tiling `semi-trnsparent.png` across their background. Since this image is semi-transparent, you can partially see the image of the map behind them.

In the example, paragraph `#note1` has a semi-transparent gradient that starts out transparent at the top and transitions to white at the bottom. This allows the background image to show through at the top of its background and gradually fade out to white at the bottom. This is the same `trans2white.png` image that I used in the Fade-out design pattern.

Related to Fade-out; Background (Chapter 6)

See also `www.cssdesignpatterns.com/semi-transparent`

Replaced Text

HTML

```
<h1>Replaced Text</h1>

<h2 id="h2">Heading 2<span></span></h2>
```

CSS

```
#h2 { position:relative; width:250px; height:76px;
  padding:0; overflow:hidden; }

#h2 span { position:absolute; width:250px; height:76px;
  left:0; top:0; margin:0;
  background:url("heading2.jpg") no-repeat; }
```

Replaced Text

Problem You want to replace text with an image. You also want the text to be read by a screen reader. You also want the text to be visible if the image is unavailable.

Solution You can insert an empty `<span>` into the block element that contains text that you want to replace with an image. You can assign the image to be the span's background image. You can relatively position the block element and absolutely position the span at the top left of the block. This displays the span in front of the block. You can size both the block and the span to fit the image exactly. Since the block and the span are the same size and the span is in front of the block, the background image of the span covers the text in the block. If the span's image is unavailable, the text behind it is visible because the span's background is transparent.

You can assign a unique ID to the block containing the text you want to replace. Using a unique ID is important when text you are replacing with the image is unique in the document. If you repeatedly replace the same text with the same image, you may want to use a class instead.

It is important that the block has no padding and the span has no margin. Otherwise, the hidden text might be visible. In addition, you can use `overflow:hidden` to ensure text does not overflow from behind the image. Also make sure the text fits within the area of the image so that if a user turns off images, the text does not overflow and get cut off.

Pattern

HTML

```
<BLOCK id="UNIQUE-ID"> TEXT <span></span></BLOCK>
```

CSS

```
#UNIQUE-ID { position:relative; padding:0; overflow:hidden;
  width:IMAGE_WIDTH;
  height:IMAGE_HEIGHT; }

#UNIQUE-ID span { position:absolute; margin:0;
  left:0; top:0;
  width:IMAGE_WIDTH;
  height:IMAGE_HEIGHT;
  background:url("FILE.EXT") no-repeat; }
```

Location This pattern applies to any block element.

Limitations When a user zooms in on a document in Firefox 2 and Internet Explorer 6, images do not enlarge along with the text. This does not apply to Internet Explorer 7 and Opera 9, which properly zoom images and text. Users typically zoom in because they need to see everything larger. When replaced images do not enlarge, the document is less accessible. This is usually not an issue because replaced text is typically a heading, and the text in the image is large to begin with.

Tips Text replacement works well with links and buttons that use rollover effects.

Related to Width, Height, Sized (Chapter 5); Background (Chapter 6); Positioning Models, Positioned, Closest Positioned Ancestor, Absolute (Chapter 7); Left Aligned, Top Aligned (Chapter 9)

See also `www.cssdesignpatterns.com/replaced-text`

Content over Image

HTML

```
<h1>Content over Image</h1>

<div class="figure">
  <h3 class="caption">Crater Lake North Rim</h3>
  <p id="crater-date"><img src="star.gif" alt="" /> August 4, 2003
    <img src="star.gif" alt="" /></p>
  <img class="framed" width="518" height="389"
    src="crater-lake.jpg" alt="Crater Lake North Rim August 4, 2003" /></div>
```

CSS

```
*.figure { float:left; position:relative;
  color:white; background-color:black; }

*.figure *.caption { position:absolute; margin:15px; left:0; top:0;
  font-size:1.05em; }

*.framed { display:block;
  border-left:1px solid gray; border-right:2px solid black;
  border-top:1px solid gray; border-bottom:2px solid black; }

#crater-date { position:absolute; left:0; bottom:10px; width:518px;
  text-align:center; color:white; font-size:0.8em; }
```

Content over Image

Problem

You want to place text on top of an image. You want to position the text relative to the image. You want the text to be visible if the image does not load. You want search engines to give the text a high priority and to index the *image* because it is part of the content.

Solution

You can embed a heading, an image, and any other type of object in a block element. You can shrinkwrap the block around the image by floating it or absolutely positioning it. This makes this design pattern work with any size of image. You can relatively position the block so it is the closest positioned ancestor of the image. This allows you to position text elements at any location over the image.

You can absolutely position the heading and use the alignment design patterns in Chapter 9 to position it within the image. Aligning the heading to the block is the same as aligning to the image because the block is shrinkwrapped to the image and is the closest positioned ancestor.

Pattern

HTML

```
<BLOCK class="figure">
  <HEADING class="caption"> TEXT_OVER_TEXT </HEADING>

  <p id="UNIQUE_ID"> TEXT_OVER_TEXT </p>

  <img src="FILE.EXT" alt="IMAGE_DESCRIPTION"
    width="IMAGE_WIDTH" height="IMAGE_HEIGHT" />
</BLOCK>
```

CSS

```
*.figure { float:LEFT_OR_RIGHT; position:relative;
  color:COLOR; background-color:COLOR; }
*.figure *.caption { position:absolute; POSITIONING_STYLES; }
*.framed { display:block; border:WIDTH STYLE COLOR; }

#UNIQUE_ID { position:absolute; POSITIONING_STYLES; }
```

Location

This pattern can be used anywhere a block element can be used.

Tips

You can use any type of element for text-over effects. I use a heading because search engines prioritize headings, and speech readers use headings to create an aural table of contents for the page.

You can include any number and type of child elements in the figure. You can assign each to a unique ID so that you can position it within the image.

In case a down-level browser does not shrinkwrap the block around the image, you should put borders around the image instead of the block.

Example

The example assigns text in the block to a white color over a black background. This ensures the text is visible if the image does not load. Also, the `alt` text is purposefully omitted from the two star images because they are meant to be decorative—the Inline Decoration design pattern is a better choice for displaying decorative images, but I wanted to keep the example simple.

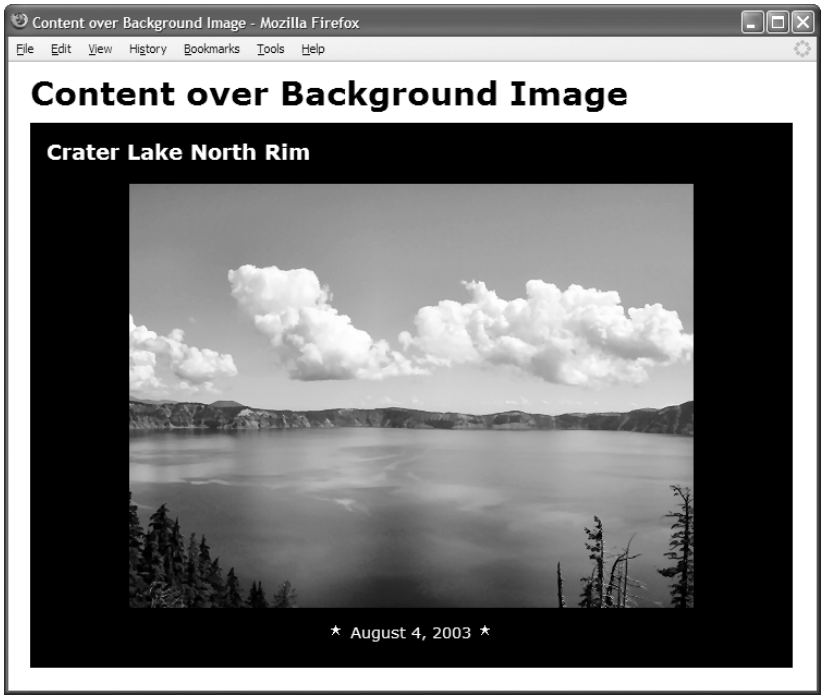
Related to

Content over Background Image; Display, Block Box (Chapter 4); Border, Background (Chapter 6); Positioning Models, Positioned, Closest Positioned Ancestor, Absolute, Float and Clear, Relative Float (Chapter 7); Aligned and Offset Absolute (Chapter 8); Inline Decoration (Chapter 11)

See also

`www.cssdesignpatterns.com/content-over-image`

Content over Background Image



HTML

```
<h1>Content over Background Image</h1>

<div id="crater-lake">
  <h3 class="caption">Crater Lake North Rim</h3>
  <p id="crater-date"><img src="star.gif" alt="" /> August 4, 2003
  <img src="star.gif" alt="" /></p></div>
```

CSS

```
#crater-lake { position:relative; padding:0; width:700px; height:500px;
  background:black url("crater-lake.jpg") no-repeat center center; }

#crater-lake *.caption { position:absolute; margin:15px; left:0; top:0;
  font-size:1.05em; color:white; }

#crater-date { position:absolute; left:0; bottom:10px; width:700px;
  text-align:center; color:white; font-size:0.8em; }

/* Nonessential rules are not shown. */
```

Content over Background Image

Problem

Like the Content over Image design pattern, you want to place text and objects on top of an image, but you do not want the image to be part of the document's content, and you do *not* want search engines to index the image. You want to position the text relative to the image. You want the text to be visible when the image does not load. You want search engines to give the text priority.

Solution

You can assign a background image to a *sized* block element. Unique IDs work well for linking unique background images to these blocks. If you use the same image multiple times, you may want to use a class instead.

You can use `background` to center a nontiled background image in the block. You can size the block to the exact size of the image or to an arbitrary size. If you size it larger than the image, the background color of the block becomes visible and creates a picture-frame effect around the image. The same thing happens if you apply padding to the block. If you size the block smaller than the image, it crops the image.

You can relatively position the block so you can absolutely position its child elements relative to it. You can use the alignment design patterns in Chapter 9 to position child elements within the image.

Pattern

HTML

```
<BLOCK id="IMAGE-NAME">
  <HEADING class="caption"> TEXT_OVER_TEXT </HEADING>
  <p id="UNIQUE_ID"> TEXT_OVER_TEXT </p>
</BLOCK>
```

CSS

```
#IMAGE-NAME {
  position:relative;
  width:IMAGE-WIDTH; height:IMAGE-HEIGHT;
  padding:VALUE;
  background:url("FILE.EXT") COLOR center center no-repeat; }

#IMAGE-NAME *.caption { position:absolute; POSITIONING_STYLES; }
#UNIQUE_ID { position:absolute; POSITIONING_STYLES; }
```

Location

This pattern can be used anywhere a block element can be used.

Advantages

There is less HTML markup than the Content over Image pattern because there is no image element. There is no need for `alt` text because a text-over caption serves the same purpose. This works better when the image fails to download because a browser does not try to display `alt` text in its place, which might get in the way of the content rendered on top of the image.

Tip

GIF and PNG images with transparent backgrounds overlay background images nicely. PNGs can even blend their edges into the background.

Example

In the example, I increase the height and width of the block to create a picture frame around the image.

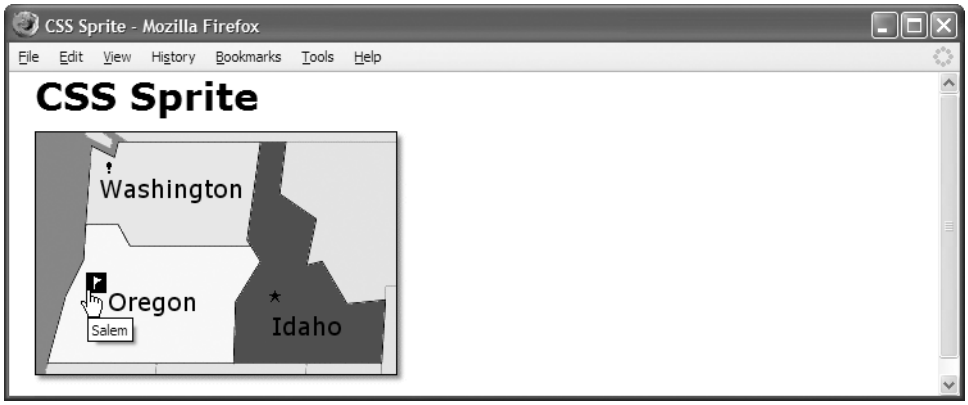
Related to

Content over Image; Width, Height (Chapter 5); Padding, Background (Chapter 6); Positioning Models, Positioned, Closest Positioned Ancestor, Absolute (Chapter 7); Aligned and Offset Absolute (Chapter 8); Inline Decoration (Chapter 11)

See also

`www.cssdesignpatterns.com/content-over-background-image`

CSS Sprite



HTML

```
<h1>CSS Sprite</h1>

<div id="nw">
  <img src="nw.gif" width="290" height="200" alt="Northwest USA" />

  <a id="olympia" class="bang-bg" href="olympia.html" title="Olympia">
    <span class="screenreader-only">Olympia</span></a>

  <a id="salem" class="flag-bg" href="salem.html" title="Salem">
    <span class="screenreader-only">Salem</span></a>

  <a id="boise" class="star-bg" href="boise.html" title="Boise">
    <span class="screenreader-only">Boise</span></a>
</div>
```

CSS

```
*.bang-bg { background:url("bt.gif") -48px -16px; width:16px; height:16px; }
*.flag-bg { background:url("bt.gif") -64px -16px; width:16px; height:16px; }
*.star-bg { background:url("bt.gif") -64px -32px; width:16px; height:16px; }

*.star-bg:hover { background-image:url("wt.gif"); background-color:black; }
*.flag-bg:hover { background-image:url("wt.gif"); background-color:black; }
*.bang-bg:hover { background-image:url("wt.gif"); background-color:black; }

*.screenreader-only { position:absolute; left:-9999px; top:-9999px;
  width:1px; height:1px; overflow:hidden; }

/* Nonessential rules are not shown. */
```

CSS Sprite

Problem

You want to use many images on a page, but you do not want the performance penalty caused by downloading multiple image files. Even on a broadband connection, it is not unusual for latency alone to slow the rendering of a page by 100 milliseconds *per image*. In other words, the latency of downloading ten images will likely delay the rendering of a page by 1 second—no matter how small the image files. Of course, delays caused by latency vary depending on web server proximity and how busy it is.

Solution

You can combine multiple background images into one image file. This file is called a **CSS sprite**. For example, you could include most, if not all, of a page's background images in one file. You could also embed a library of list bullets, icons, and text decorations in a CSS sprite that is shared across your web site.

The key to using a sprite is to display it as the background image of a sized element and to position the background image at the exact horizontal and vertical offset of the embedded image. The element must be the exact width and height of the desired embedded image; otherwise, parts of several embedded images may be visible in its background. The element must be set to the proper horizontal and vertical offset, or the background will show the wrong embedded image or will show parts of several embedded images. The measurements used in `width`, `height`, and `background-position` must all be in pixels because embedded images are measured in pixels. The values in `background-position` are *negative* because they move the composite background image up and to the left to position it.

You can replace `<img>` elements with CSS sprites by displaying them as background images within sized spans or divisions, but unless content images cause performance problems, it is more natural to use `<img>` elements. When replacing an image with a CSS sprite, you can use the Screenreader-only design pattern to embed hidden alternate text that will be read only by screen readers. This makes the CSS sprite accessible.

Pattern

HTML

```
<ELEMENT>
  <span class="screenreader-only"> ALTERNATE_TEXT </span>
</ELEMENT>
```

CSS

```
SELECTOR { width:SPRITE_WIDTH; height:SPRITE_HEIGHT;
  background-image:url("SPRITE_FILE.EXT");
  background-position:-HORIZONTAL_OFFSETpx -VERTICAL_OFFSETpx; }

SELECTOR:hover { background-image:url("HOVER_SPRITE_FILE.EXT");
  background-color:COLOR; }
```

Location

This pattern applies to any type of element.

Limitations

Background images using CSS sprites cannot be tiled because the entire composite image would be tiled rather than just the embedded image.

(Continued)

CSS Sprite (Continued)

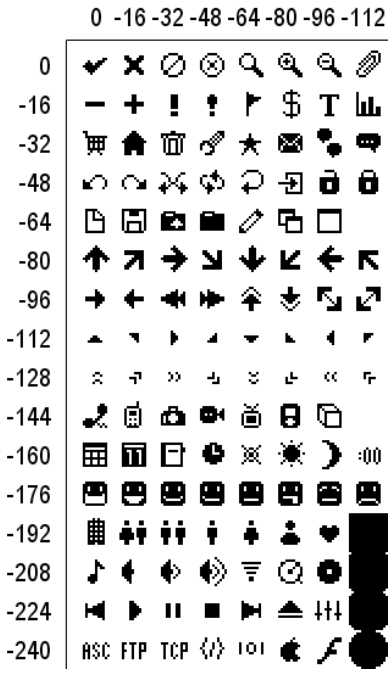


Figure 14-1. *Offsets for 16×16 sprites as used in bt.gif*

Example

I use two CSS sprite files in the example: `bt.gif` (see Figure 14-1) and `wt.gif`. These file names stand for a black image on a transparent background and a white image on a transparent background. When the user mouses over the image, the hover selector switches out the `bt.gif` and replaces it with `wt.gif`, which inverts the color from black to white. The background is also changed to black, which shows through the transparent parts of the image.

I include two other sprite files in the example directory that are not used in the example. They are named `tb.gif` and `tw.gif`. These file names stand for transparent images in black boxes and transparent images in white boxes. These embedded images are little black and white boxes with transparent images in the center, which change color to match the background.

I created these four CSS sprites from an icon set called bitcons. I made all the embedded images exactly 16×16 pixels, like the originals. These icons are freely licensed and are available at `http://somerandomdude.net/srd-projects/bitcons`. Likewise, you are free to use these four CSS sprite files in your projects.

When making your own CSS sprite images, you can embed any image of any size into the sprite. Embedded images do not need to be the same size. All you need to know is the offset and size of each embedded image.

CSS Sprite (Continued)

Advantages By reducing the number of files that are downloaded, you can dramatically speed the loading of a page. Embedding multiple images in a single file typically results in a smaller overall file size than the combined file sizes of separate images.

Disadvantages Combining images to create sprites and tracking their offsets can be time consuming and error prone. This makes managing images harder. It works best when you create a sprite containing a library of images that work together to skin a document. Whenever you want to change the look and feel of a document, you change the sprite.

Tip Managing sprite offsets is easier if all embedded images are the same size.

Latency Over a broadband connection to the Internet, downloading data in a small file is very quick, but the communication latency involved in requesting a small file can often take several times longer than actually downloading the file! HTTP and TCP/IP communications protocols require handshake messages to be sent back and forth before content can be downloaded, messages traveling across the Internet compete for bandwidth, and servers queue requests until they can get to them. My measurements show latency delays the rendering of a page by approximately 100 milliseconds plus the time it takes to download the data.

Using Google Load Time Analyzer for Firefox, I tracked web page download times on my high-speed broadband connection. For example, the home page of MSN.com took 5 seconds to download 41 files: 1 HTML document, 3 CSS stylesheets, 4 JavaScript files, 15 GIFs, 10 JPGs, and 8 ad callbacks. The total download size was 136K, which took 1742 ms to download. The time it took to send messages to the server and to wait for replies was 15,960 ms! In other words, for each millisecond that data was downloaded, 9 milliseconds were spent waiting: 3 milliseconds were lost waiting for messages to travel back and forth across the Internet, and 6 milliseconds were lost due to server latency. I have documented the results in an Excel spreadsheet included in this design pattern's example directory.

If all 25 images in the MSN homepage were merged into one composite file, latency would be reduced from 9000 ms to 500 ms. This would save 8500 ms! Since a browser downloads using three connections simultaneously, the actual savings are one-third of 8500 ms, or 2800 ms. This one change alone would reduce the download time of the MSN homepage from 5.2 seconds to 2.4 seconds—more than doubling its download speed!

Sprite history A sprite gets its name from a technique used in two-dimensional video games of compositing multiple images into one file where each image is a frame of animation. You can animate a sprite simply by rotating the display through offsets in the composite image. Animated GIFs use this technique, and you can use this technique to create rollover effects.

Related to Image; Width, Height (Chapter 5); Background (Chapter 6)

See also `www.cssdesignpatterns.com/css-sprite`

Basic Shadowed Image

HTML

```
<h1>Basic Shadowed Image</h1>

<img class="shadowed"
  src="crater-lake.jpg"
  alt="Crater Lake"
  width="518"
  height="389" />
```

CSS

```
img.shadowed { padding-right:20px;
  padding-bottom:20px;
  background-image:url("shadow.jpg");
  background-position:right bottom;
  background-repeat:no-repeat; }
```

Basic Shadowed Image

Problem

You want to place a shadow behind an image without having to modify the original image. You also want to control the distance the shadow is offset from behind the image.

Solution

You can create a shadow image that is the same size as the image it is shadowing. You can assign the shadow as the nontiled background of the image. You can use `background-position` to move the background shadow to the bottom right of the padding area. You can use `padding-right:+VALUE` and `padding-bottom:+VALUE` to control how much the shadow extends below the bottom right of the image.

Shadows are traditionally displayed in the bottom-right corner, but if you want to display them in a different corner, you can extend the padding into that corner and position the shadow there.

Pattern

HTML

```
<img class="shadowed"
  src="FILE.EXT" alt="DESCRIPTION"
  width="WIDTH" height="HEIGHT" />
```

CSS

```
*.shadowed { padding-right:+VALUE;
  padding-bottom:+VALUE;
  background-image:url("FILE.EXT");
  background-position:right bottom;
  background-repeat:no-repeat; }
```

Location

This pattern applies to images.

Advantages

Because the shadow is an image, there is no limit to what you can do with the shadow. You can use any color, blur, and texture to fit the style of your document.

This pattern is simple and does not require you to process images to embed shadows in them. You can also change the look and feel of all shadows on a web site by simply changing the shadow image.

Disadvantages

This pattern requires you to create a shadow image for each size of image. If all your images are the same size or have a limited number of sizes, this pattern works well. If your images come in unpredictable sizes, you may want to use the more complicated, yet more versatile, Shadowed Image pattern.

The *latency* caused by a browser checking to see whether the shadowed image has already been downloaded slows the rendering of a page—even on broadband connections.

Related to

Image, Shadowed Image; Padding, Background (Chapter 6)

See also

`www.cssdesignpatterns.com/basic-shadowed-image`

Shadowed Image

Figure 14-2. *shadow.jpg*

Figure 14-3. *shadow-rt.jpg and shadow-lb.jpg are created by extracting them from shadow.jpg.*

Figure 14-4. *shadow-rt.jpg indents and closes off the top-right edge of the shadow.*

Figure 14-5. *shadow-lb.jpg indents and closes off the left-bottom edge of the shadow.*

Shadowed Image

Problem You want to place a shadow behind an image without having to modify the original image. You also want to control how much the shadow is offset from the image. You also want the shadow to work automatically with any size of image.

Solution You can use three image files to create a shadow that will automatically fit any image. This can be a great timesaver because you do not need to embed shadows within images, and it makes it easy to change the style of the shadows on the fly.

Like the Basic Shadowed Image pattern, the first step is to create a shadowed image, as shown in Figure 14-2, or reuse one previously created like the one in the example. I name this file `shadow.jpg`. Unlike the Basic Shadowed Image pattern, `shadow.jpg` should be as large as the *largest* image it will shadow.

In addition, you need to create two additional images by extracting them from the shadowed image (see Figure 14-3). One indents and closes off the right-top edge of the shadow (see Figure 14-4), and one indents and closes off the left-bottom edge of the shadow (see Figure 14-5). These images are the key to creating an automatically sized shadow because they create the illusion that the shadow is indented on the right-top and the left-bottom, as shown in Figure 14-6. I call these the **indentor images**.

In the example, I created the two indentor images as follows. I extracted the right-top corner of the shadow image and saved it as `shadow-rt.jpg` (see Figure 14-4). I also extracted the left-bottom corner of the shadow image and saved it as `shadow-lb.jpg` (see Figure 14-5). I made `shadow-rt.jpg` 100 pixels *wide* and only as tall as needed to capture the shadow's blur. I made `shadow-lb.jpg` 100 pixels *tall* and only as wide as needed to capture the shadow's blur. I then expanded the canvas of each of these two images to make them 100 pixels square. I put the background color in the expanded part of these images. This allows the indentors to indent up to 100 pixels of the shadow by covering it with the background color (see Figure 14-6).

You need to stack the images in the following order from bottom to top: `shadow.jpg`, `shadow-rt.jpg`, and `shadow-lb.jpg`. The image receiving the shadow gets stacked on top of them all, as shown in Figure 14-6. You can stack these three background images by assigning them to three nested block elements. I typically use divisions. The order is important. You can assign `shadow.jpg` to the outermost block element. You can assign `shadow-rt.jpg` to the second nested element. You can assign `shadow-lb.jpg` to the third nested element. You can place the `<img>` element inside the third nested block.

To shrinkwrap these three elements to the size of the image, you need to float them or absolutely position them.

(Continued)

Shadowed Image (Continued)

Figure 14-6. *Composite view of the shadowed image*

Shadowed Image (Continued)

Apply styles to your chosen class or ID as follows:

-You can use `background-image` to load the shadow images into the backgrounds of their respective elements.

-You can use `background-position:right bottom;` to position the shadow image in the right-bottom corner of the image.

-You can use `background-position:right TOP_OFFSET;` to position `shadow-rt.jpg` at an offset from the right-top corner of the image. You can calculate the value of `TOP_OFFSET` by adding `BOTTOM_OFFSET` to the negative of the height of `shadow-rt.jpg`. For example, if the height of `shadow-rt.jpg` is 100 pixels and `BOTTOM_OFFSET` is 20 pixels, you would add 20 to –100 to get a `TOP_OFFSET` of `-80px`. By offsetting `shadow-rt.jpg` by the inverse of its height, you are aligning its bottom to the top of the background. By adding back in the `BOTTOM_OFFSET`, you move it down the same amount that you move down the shadow.

-You can use `background-position:LEFT_OFFSET bottom;` to `position shadow-lb.jpg` at an offset from the left-bottom corner of the image. You can calculate the value of `LEFT_OFFSET` by adding `RIGHT_OFFSET` to the negative of the width of `shadow-lb.jpg`. For example, if the width of `shadow-lb.jpg` is 100 pixels and `RIGHT_OFFSET` is 20 pixels, you would add 20 to –100 to get a `LEFT_OFFSET` of `-80px`. By offsetting `shadow-lb.jpg` by the inverse of its width, you are aligning its right side to the left side of the background. By adding back in the `LEFT_OFFSET`, you move it to the right by the same amount that you move the shadow to the right.

-You can use `background-repeat:no-repeat` to prevent each background image from being tiled.

-You can use `padding-right:RIGHT_OFFSET` to move the shadow image past the right side of the image.

-You can use `padding-bottom:BOTTOM_OFFSET` to move the shadow image below the bottom of the image.

(Continued)

Shadowed Image (Continued)

HTML

```
<h1>Shadowed Image</h1>

<div class="shrinkwrapped">
  <div class="shadowed">
    <div class="shadowed-rt">
      <div class="shadowed-lb">
        <img src="crater-lake.jpg" alt="Crater Lake" width="518" height="389" />
</div></div></div></div>
```

CSS

```
*.shrinkwrapped { float:left; }

*.shadowed { background-image:url("shadow.jpg");
  background-position:right bottom; background-repeat:no-repeat; }

*.shadowed-rt { background-image:url("shadow-rt.jpg");
  background-position:right -80px; background-repeat:no-repeat; }

*.shadowed-lb { padding-right:20px; padding-bottom:20px;
  background-image:url("shadow-lb.jpg");
  background-position:-80px bottom; background-repeat:no-repeat; }
```

Shadowed Image (Continued)

Pattern

HTML

```
<div class="shrinkwrapped">
  <div class="shadowed">
    <div class="shadowed-rt">
      <div class="shadowed-lb">
        <img src="FILE.EXT" alt="" width="WIDTH" height="HEIGHT" />
</div></div></div></div>
```

CSS

```
*.shrinkwrapped { float:LEFT_OR_RIGHT; }

*.shadowed { background-image:url("FILE.EXT");
  background-position:right bottom;
  background-repeat:no-repeat; }

*.shadowed-rt { background-image:url("FILE-rt.EXT");
  background-position:right TOP_OFFSET;
  background-repeat:no-repeat; }

*.shadowed-lb {
  padding-right:RIGHT_OFFSET;
  padding-bottom:BOTTOM_OFFSET;
  background-image:url("FILE-lb.EXT");
  background-position:LEFT_OFFSET bottom;
  background-repeat:no-repeat; }
```

Location

This pattern applies to images. Because this pattern wraps the image in block elements, it cannot be used inline.

Advantages

Because the shadow is an image, there is no limit to what you can do with the shadow. You can use any color, amount of blur, and texture to fit the style of your document. Because this pattern automatically fits the shadow to the size of the image, you only need to create three images to put a shadow behind any image of any size. The browser only has to download three image files to create an unlimited number of shadows.

Disadvantages

This pattern requires you to insert extra divisions into the markup to create this shadow effect.

This pattern requires you to shrinkwrap the parent division to the image. Otherwise, it will be stretched to the width of its container, and the nested background images will extend beyond the image to fill the width of the container. This breaks the shadow effect. In the pattern, I floated the element to shrinkwrap it. You could also position it to shrinkwrap it. The only block element that shrinkwraps naturally is the table.

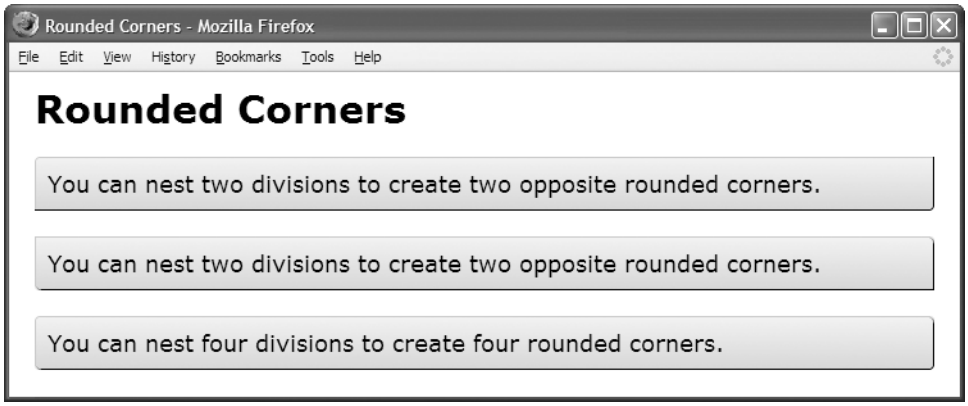
Related to

Image, Basic Shadowed Image, Rounded-corners; Padding, Background (Chapter 6); Float and Clear (Chapter 7)

See also

`www.cssdesignpatterns.com/shadowed-image`

Rounded Corners



HTML

```
<div class="bg"><div class="tl"><div class="br pad">
  You can nest two divisions to create two opposite rounded corners.
</div></div></div>

<div class="bg"><div class="tr"><div class="bl pad">
  You can nest two divisions to create two opposite rounded corners.
</div></div></div>

<div class="bg">
  <div class="tl"><div class="br"><div class="trc"><div class="blc pad">
  You can nest four divisions to create four rounded corners.
</div></div></div></div></div>
```

CSS

```
*.bg { background:url("bg.gif") bottom left repeat-x white; margin-top:20px; }

*.tl { background:url("rc.gif") top left no-repeat; }
*.br { background:url("rc.gif") bottom right no-repeat; }
*.tr { background:url("rc.gif") top right no-repeat; }
*.bl { background:url("rc.gif") bottom left no-repeat; }

*.trc { background:url("rc-trc.gif") top right no-repeat; }
*.blc { background:url("rc-blc.gif") bottom left no-repeat; }

*.pad { padding:10px; }
```

Rounded Corners

Problem

You want to round the corners of an element's box. You want the corners to expand and shrink with the box so it will work with any amount of content.

Solution

You can create rounded corners by embedding background images of rounded corners inside an element. These images also include the borders that connect the rounded corners to each other. Because these are images, you can create any style of corner and border you can imagine.

Since CSS only allows one background image per element, you can insert extra divisions inside the element you want to have rounded corners—one division for each rounded corner. Embedded divisions with no margins and padding are located in exactly the same position as their parent. This allows you to layer background images on top of each other. Note that when a parent element has a fixed height, its child divisions must also have the same fixed height.

The first two boxes in the example have two rounded corners and two nested divisions. The third box has four rounded corners and four nested divisions. A detailed explanation follows on the next page.

Patterns

HTML

```
<div class="bg"><div class="tl"><div class="br">
  CONTENT
</div></div></div>
```

or

```
<div class="bg"><div class="tr"><div class="bl">
  CONTENT
</div></div></div>
```

or

```
<div class="bg"><div class="tl"><div class="br">
  <div class="trc"><div class="blc">
    CONTENT
 </div></div></div></div></div>
```

CSS

```
*.bg { background:BACKGROUND_STYLES; margin-top:20px; }

*.tl { background:url("RC_FILE.EXT") top left no-repeat; }
*.tr { background:url("RC_FILE.EXT") top right no-repeat; }
*.br { background:url("RC_FILE.EXT") bottom right no-repeat; }
*.bl { background:url("RC_FILE.EXT") bottom left no-repeat; }

*.trc { background:url("TRC_FILE.EXT") top right no-repeat; }
*.blc { background:url("BLC_FILE.EXT") bottom left no-repeat; }
```

Location

This pattern applies to block elements and inline elements that are positioned, floated, or displayed as blocks.

(Continued)

Rounded Corners (Continued)

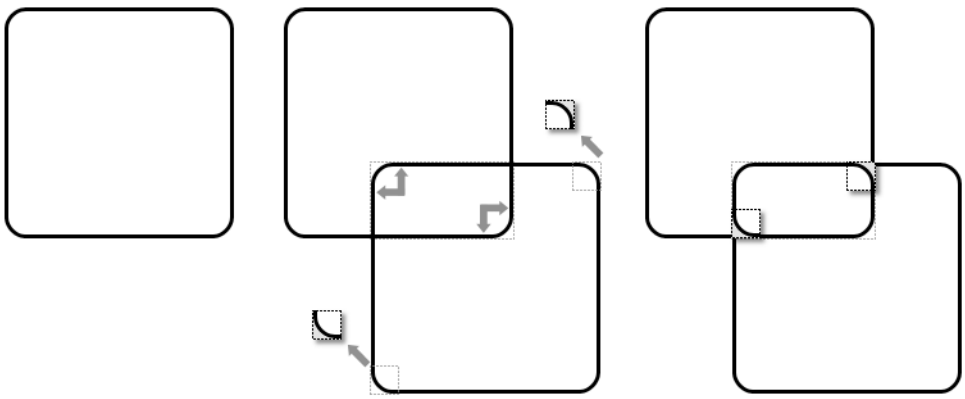


Figure 14-7. *Creating rounded corners from rounded rectangle images*

Creating the Three Rounded Rectangle Images

In the example, I started with a 1600×1600 transparent canvas. I added a rounded rectangle that hugged the edges of the canvas. The rounded rectangle had a transparent interior. I filled in the *exterior* pixels of each rounded corner with the external background color, which is white in my example. This makes them opaque so the outside of each corner overlays the interior background with the background color. Notice in Figure 14-7 how the outside of the top-left corner of the first rounded rectangle and the outside of the bottom-right corner of the second rounded rectangle would display the internal background if they were not opaque. Lastly, I saved the image as `rc.gif`.

To create the cutout images, I cut out the bottom-left corner and the top-right corner of the rounded rectangle image and saved them as separate GIF images named `tr.gif` and `bl.gif`. I made sure the exterior part of the corner remained opaque and the interior remained transparent. Otherwise, they would not do their job of hiding the external square borders on the outside and letting the background show through on the inside. I sized each cutout just large enough to cover the square corner with a rounded corner.

Creating the three rounded rectangle images is simple: create a transparent rounded rectangle; fill in the exterior of its rounded corners; and save the bottom-left and top-right corners as separate images.

Rounded Corners (Continued)

Detailed Solution

You can assign a background image to each nested division. I use six classes for that purpose: `tl`, `br`, `tr`, `trc`, `bl`, and `blc`, which stand for top left, bottom right, top right, top-right corner, bottom left, and bottom-left corner.

To create two opposite rounded corners, you can apply the same background image to two child divisions. The image should be a large rounded rectangle with a transparent *interior* so the background image or color will show through. The *exterior* of its rounded corners should be opaque and should be the same color as the exterior background color.

The key is to position the same rounded rectangle image in the top-left corner and in the bottom-right corner (see Figure 14-7). This creates two overlapping rounded rectangles. As the element expands or contracts, so do the rounded rectangles. The content of the element can grow as large as the size of the rounded rectangle before the illusion breaks. This is not a problem because you can make this rectangle as large as you want. In the example, I made the rounded rectangle image 1600×1600 pixels, and yet it has a file size of only 8,278 bytes because most of it is transparent.

To create four rounded corners, you can position the same rounded rectangle image in the top-left corner and in the bottom-right corner. You then assign two additional background images to two additional nested divisions: one is positioned in the top-right corner, and the other in the bottom-left. These new images are tiny rounded corners that cover up the square intersections of the two overlapping rounded rectangles, as shown in Figure 14-7. It is important that these two corner divisions are placed *after* the first two rounded rectangle divisions. This allows the corner divisions to be stacked on top of the others.

You can set the interior `background` by assigning a background color or image to the parent of the rounded corner box. In the example and the pattern, I use the `bg` class to assign this background. Likewise, the best place to set the `margin` is the parent. The best place to set the `padding` is the last embedded division. In the example, I assign the `pad` class to the last embedded division to set the padding for the interior of the rounded corner box. You should not apply a `border` to any of these elements because it would conflict with the rounded corners.

Limitations

The exterior of the cutout corner images must not be transparent. When they are transparent, they show the intersection of the rounded rectangle borders. This breaks the illusion. Since the exterior of the cutout corner images must be opaque, the opaque exterior needs to match the background color that surrounds the *outside* of the rounded rectangle. This requires that you create a different set of cutout corner images for each different external background color you intend to use.

There is a bug in Internet Explorer 6 that sometimes causes the background to leak out from behind the element. You can assign `zoom:1` to the parent element to give it "layout," which prevents the background from leaking out. See the Atomic pattern in Chapter 7 for more details on "layout."

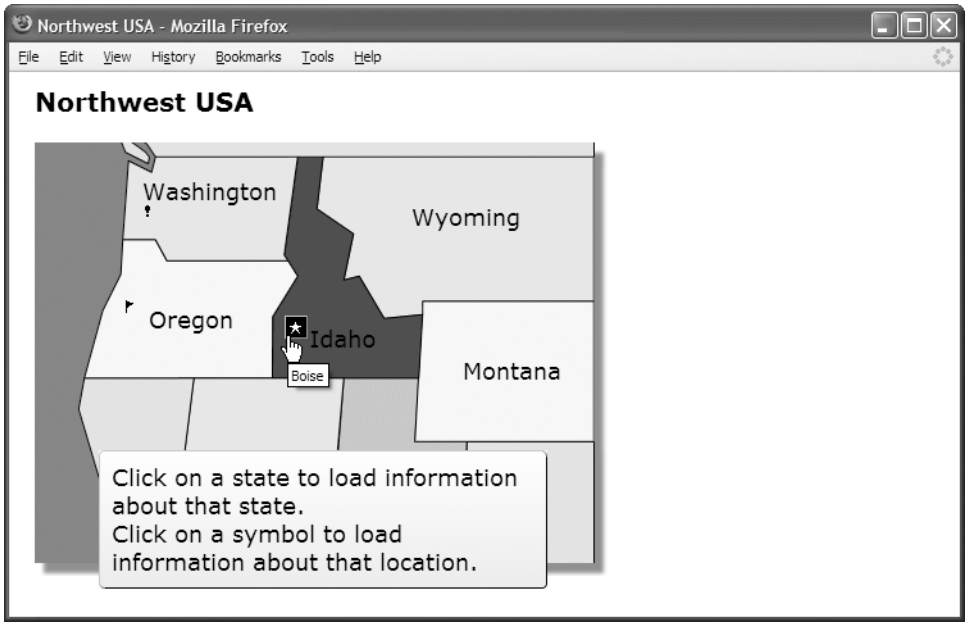
Related to

Image, Basic Shadowed Image, Shadowed Image; Margin, Background (Chapter 6)

See also

`www.cssdesignpatterns.com/rounded-corners`

Image Example



Representative Excerpts from the HTML

```
<h1>Northwest USA</h1>

<div id="states">
  <img src="nw.gif" width="437" height="328"
    alt="Northwest" usemap="#nw-map" class="shadowed" />

  <a id="washington" href="washington.html" class="overlay">Washington</a>
  <a id="oregon" href="oregon.html" class="overlay">Oregon</a>
  <a id="idaho" href="idaho.html" class="overlay">Idaho</a>

  <a id="olympia" class="bang-bg" href="olympia.html" title="Olympia">
    <span class="screenreader-only">Olympia</span></a>
  <a id="salem" class="flag-bg" href="salem.html" title="Salem">
    <span class="screenreader-only">Salem</span></a>

  <div id="info" class="bg">
    <div class="tl"><div class="br"><div class="trc"><div class="blc pad">
      <p>Click on a state to load information about that state.</p>
      <p>Click on a symbol to load information about that location.</p>
      </div></div></div></div></div>
</div>
```

Image Example

Example	This is not a design pattern but an example that illustrates how the design patterns in the chapter can work together.
Explanation	The main image in the example is a map of the Pacific Northwest. I used the Basic Shadowed Image design pattern to put a shadow behind it. The image is linked to the `nw-map` element to make areas on the map clickable. I used the Content over Image design pattern to put links on top of the map. When the user hovers over these links, the background displays a semi-transparent PNG image, which partially hides the content under the image. I also use the CSS Sprite design pattern to put clickable rollover images on top of the map. I also use the Rounded Corners and Fade-out design patterns to style the message below the map.
See also	`www.cssdesignpatterns.com/image-example`

Representative Excerpts from the CSS

```
*.shadowed { padding-right:12px; padding-bottom:12px;
  background:url("shadow.jpg") right bottom no-repeat; }

*.screenreader-only { position:absolute; left:-9999px; top:-9999px;
  width:1px; height:1px; overflow:hidden; }

a { text-decoration:none; color:black; }
a:hover { border-left:1px solid silver; border-right:1px solid gray; color:white;
  border-top:1px solid silver; border-bottom:1px solid gray;
  background-image:url("semi-transparent.png"); background-repeat:repeat-x; }
*.overlay { padding:2px 4px; }

*.bg { background:url("white2trans.png") top left repeat-x yellow;
  margin-top:20px; }
*.tl { background:url("rc.gif") top left no-repeat; }
*.br { background:url("rc.gif") bottom right no-repeat; }
*.trc { background:url("rc-trc.gif") top right no-repeat; }
*.blc { background:url("rc-blc.gif") bottom left no-repeat; }
*.pad { padding:10px; }

*.bang-bg { background:url("bt.gif") -48px -16px; width:16px; height:16px; }
*.flag-bg { background:url("bt.gif") -64px -16px; width:16px; height:16px; }
*.star-bg { background:url("bt.gif") -64px -32px; width:16px; height:16px; }

*.bang-bg:hover { background-image:url("wt.gif"); background-color:black; }
*.star-bg:hover { background-image:url("wt.gif"); background-color:black; }
*.flag-bg:hover { background-image:url("wt.gif"); background-color:black; }

#states { position:relative; float:left; }
  #washington { position:absolute; top:35px; left:80px; }
  #oregon { position:absolute; top:135px; left:85px; }
  #idaho { position:absolute; top:150px; left:210px; }
```