

6

Stunning Aesthetics with CSS3

In the previous chapter we learned about some quick and useful CSS3 techniques to aid in building responsive designs. We also made a big difference to the visuals by employing the CSS3 `@font-face` rule to apply custom typography and learned about CSS3's tools for selecting DOM elements. So, with some CSS3 basics covered, let's look at some more advanced features of CSS3; how we can give a responsive design an aesthetic lift by using some of the more exciting CSS3 techniques that, for the vast majority, don't require a single graphics image, making our responsive design as bandwidth friendly as possible.

In this chapter we will cover:

- How to create text-shadows with CSS3
- How to create box-shadows (drop shadows) with CSS3
- Making gradient backgrounds with CSS3
- Using multiple backgrounds with CSS3
- Using CSS3 background gradients to make patterns
- Using the CSS3 `@font-face` rule to make bandwidth friendly icons

At this point I'm going to reiterate why I believe CSS3 is so useful in responsive design: using CSS3, rather than images in a bandwidth design reduces `http` requests (and hence makes the pages load faster) and makes the design more flexible and maintainable. Those benefits would be useful even on a typical fixed-width 'desktop' design but it's even more important with a responsive design as it easily allows different size box or text shadows at different viewports – without needing to make and export a single image. I'm presuming you're with me on this, so let's dig in.



Vendor prefixes

When implementing CSS3, just remember to add relevant vendor prefixes to ensure the broadest cross-browser compatibility. Alternately, if you're happy to add some JavaScript to your code, consider the afore mentioned **-prefix-free** script. It automatically adds relevant vendor prefixes to any CSS3 rules that need them, allowing you to only write the W3C version in your stylesheet. Get it here: <http://leaverou.github.com/prefixfree/>.

Text shadows with CSS3

One of the most widely implemented CSS3 features is 'text-shadow'. Like @font-face, it had a previous life but was dropped in CSS 2.1. Thankfully it's back and widely supported (all modern browsers and Internet Explorer 9 onwards).

Let's look at the basic syntax:

```
.element {
  text-shadow: 1px 1px 1px #cccccc;
}
```

Remember, the values in shorthand rules always go right and then down. Therefore, the first value is the amount of shadow to the right, the second is the amount down, the third value is the amount of blur (the distance the shadow travels before fading to nothing), and the final value is the color.

HEX, HSL, or RGB color allowed

The color value doesn't need to be defined as a HEX value. It can just as easily be HSL(A) or RGB(A) :

```
text-shadow: 4px 4px 0px hsla(140, 3%, 26%, 0.4);
```

However, keep in mind that the browser must then also support HSL/RGB color modes along with text-shadow in order to render the effect. If I'd really like to use HSLA or RGBA (because of the opacity capability) I tend to do this:

```
text-shadow: 4px 4px 0px #404442;
text-shadow: 4px 4px 0px hsla(140, 3%, 26%, 0.4);
```

Define the shadow first with a HEX value (as a fall back for older browsers) and then repeat the rule afterwards using the HSLA or RGBA value.

Pixels, em, or rem

You can also set the shadow values in em or rem. For example, here's the **AND THE WINNER ISN'T** composite:

AND THE WINNER ISN'T...

WHY? SYNOPSIS **STILLS/PHOTOS** VIDEOS/CLIPS QUOTES QUIZ

UNSUNG HEROES...

OVERHYPERED NONSENSE...

EVERY YEAR

WHEN I WATCH THE OSCARS I'M ANNOYED...

that films like **King Kong**, **Moulin Rouge** and **Munich** get the statue whilst the real cinematic heroes lose out. Not very Hollywood is it? We're here to put things right.

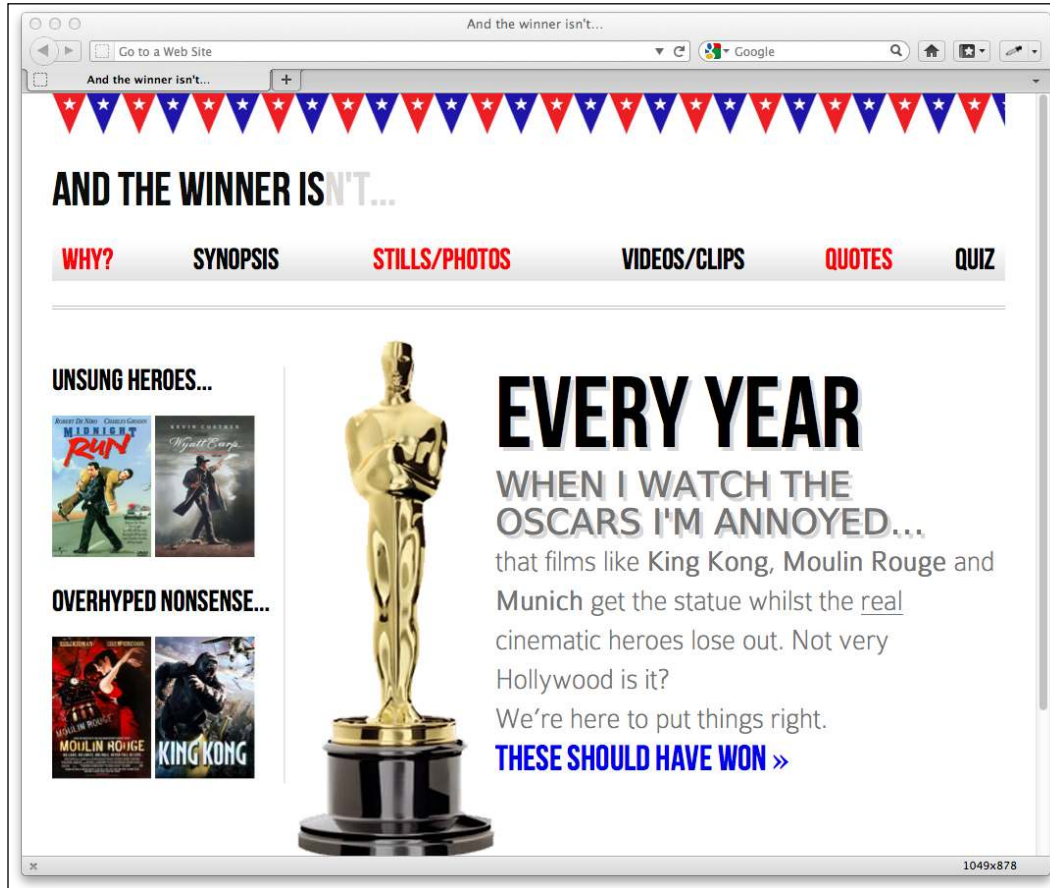
THESE SHOULD HAVE WON >>

NOTE: OUR OPINION IS ABSOLUTELY CORRECT. YOU ARE WRONG, EVEN IF YOU THINK YOU ARE RIGHT. THAT'S A FACT. DEAL WITH IT.

In Photoshop, the **EVERY YEAR** text is 102 px with a text shadow of 4 px. Therefore, using the trusty **target ÷ context = result** formula ($4 \div 102 = .039215686$). So this becomes:

```
text-shadow: .039215686em .039215686em 0em #dad7d7; /* 4 ÷ 102 */
```

The following screenshot shows the effect in the browser:



Personally, I rarely use `em` or `rem` for `text-shadow` values. As the values are always really low, using 1 or 2 px generally looks good across all viewports.

Preventing a text shadow

Depending on your eyesight, you *may* notice that we now also have a text shadow on the second sentence, **WHEN I WATCH THE OSCARS I'M ANNOYED...**

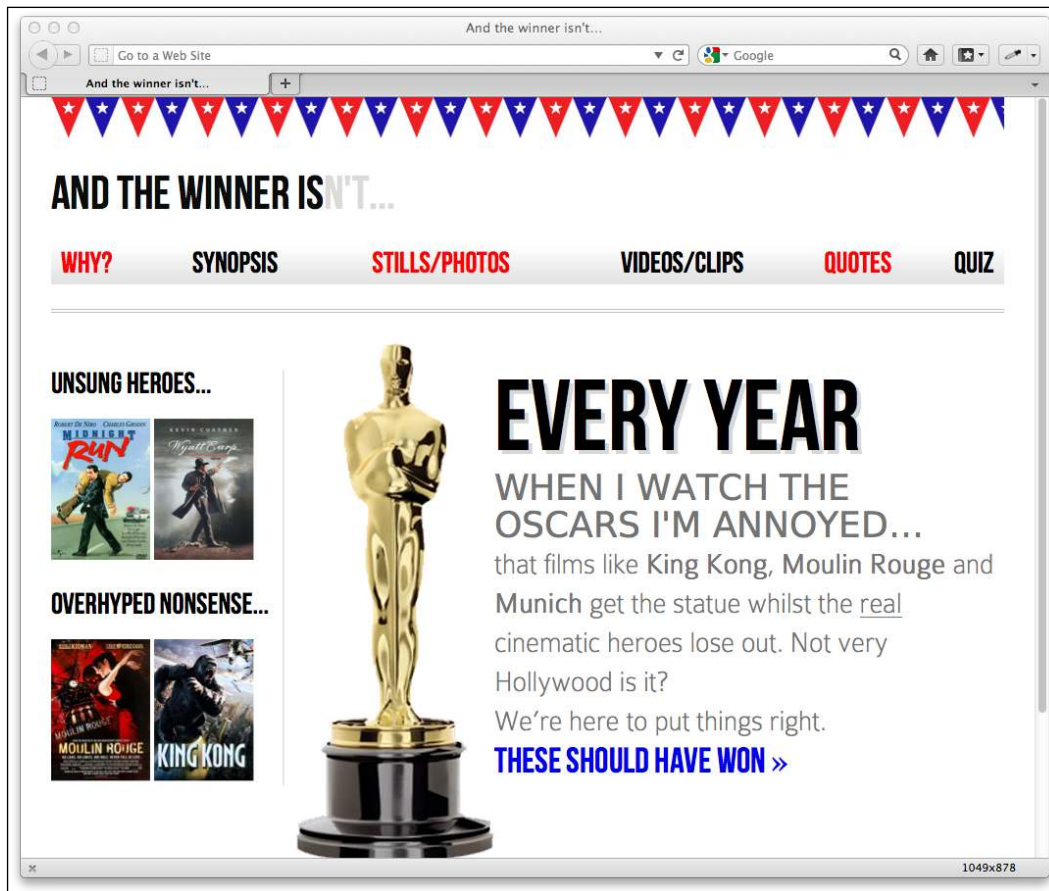
Here's why:

```
<h1>Every year <em>when I watch the Oscars I'm annoyed...</em></h1>
```

The text-shadow is currently applying to the entire <h1> tag (which includes the tag within it) so we need to remove the text-shadow from the tag:

```
#content h1 em {
  font-family: 'BitstreamVeraSansRoman';
  display: block;
  line-height: 1.052631579em; /* 40 ÷ 38 */
  color: #757474;
  font-size: .352941176em; /* 36 ÷ 102 */
  text-shadow: none;
}
```

And now it's looking good:



Left and top shadows

Shadows to the left and above can be achieved using negative values. For example:

```
text-shadow: -4px -4px 0px #dad7d7;
```

Adds an effect like the following:



If there is no blur to be added to a `text-shadow` the value can be omitted from the declaration, for example:

```
text-shadow: -4px -4px #dad7d7;
```

The spec assumes that the first two values are for the offsets if no third value is declared.

Creating an embossed text-shadow effect

I've always felt that `text-shadow` works best for creating embossed text. This effect usually works best with a highlight color (for example, white or close to it) applied to dark text on a non-white background. Let's add an embossed effect to the navigation links:

```
nav ul li a {  
  height: 42px;  
  line-height: 42px;  
  text-decoration: none;  
  text-transform: uppercase;  
  font-family: 'BebasNeueRegular';  
  font-size: 1.875em; /*30 ÷ 16 */  
  color: #000000;  
  text-shadow: 0 1px 0 hsla(0, 0%, 100%, 0.75);  
}
```

And here's the result. Subtle but effective—just a little depth added without shouting LOOK AT MY TEXT-SHADOW!





For the best embossed text, I tend to find that 1 or 2 px in the vertical offset and nothing for blur and horizontal offset works best.

Multiple text-shadows

It's possible to add multiple text shadows by comma separating two values. For example:

```
text-shadow: 0px 1px #ffffff,4px 4px 0px #dad7d7;
```

As ever, subtlety is necessary or type can become illegible. I'm going to use this declaration to combine both the previous embossed effect and the existing text-shadow. Here's the effect in the browser:

EVERY YEAR



Read the W3C specification for the `text-shadow` property here:
<http://www.w3.org/TR/css3-text/#text-shadow>

Box shadows

Once text-shadows are understood, box-shadows will be a piece of cake. Principally, they follow exactly the same syntax: horizontal offset, vertical offset, blur, and color:

```
box-shadow: 0px 3px 5px #444444;
```

However, they aren't as well supported across browsers so it's wise to use vendor prefixes to maximize compatibility. For example:

```
-ms-box-shadow: 0px 3px 5px #444444;  
-moz-box-shadow: 0px 3px 5px #444444;  
-webkit-box-shadow: 0px 3px 5px #444444;  
box-shadow: 0px 3px 5px #444444;
```

We'll use this to add a box shadow to the film posters in the sidebar of the **AND THE WINNER ISN'T** site:

```
.sideBlock img {  
    max-width: 45%;  
    box-shadow: 0px 3px 5px #444444;  
}
```

Here's the effect in the browser:



Inset shadow

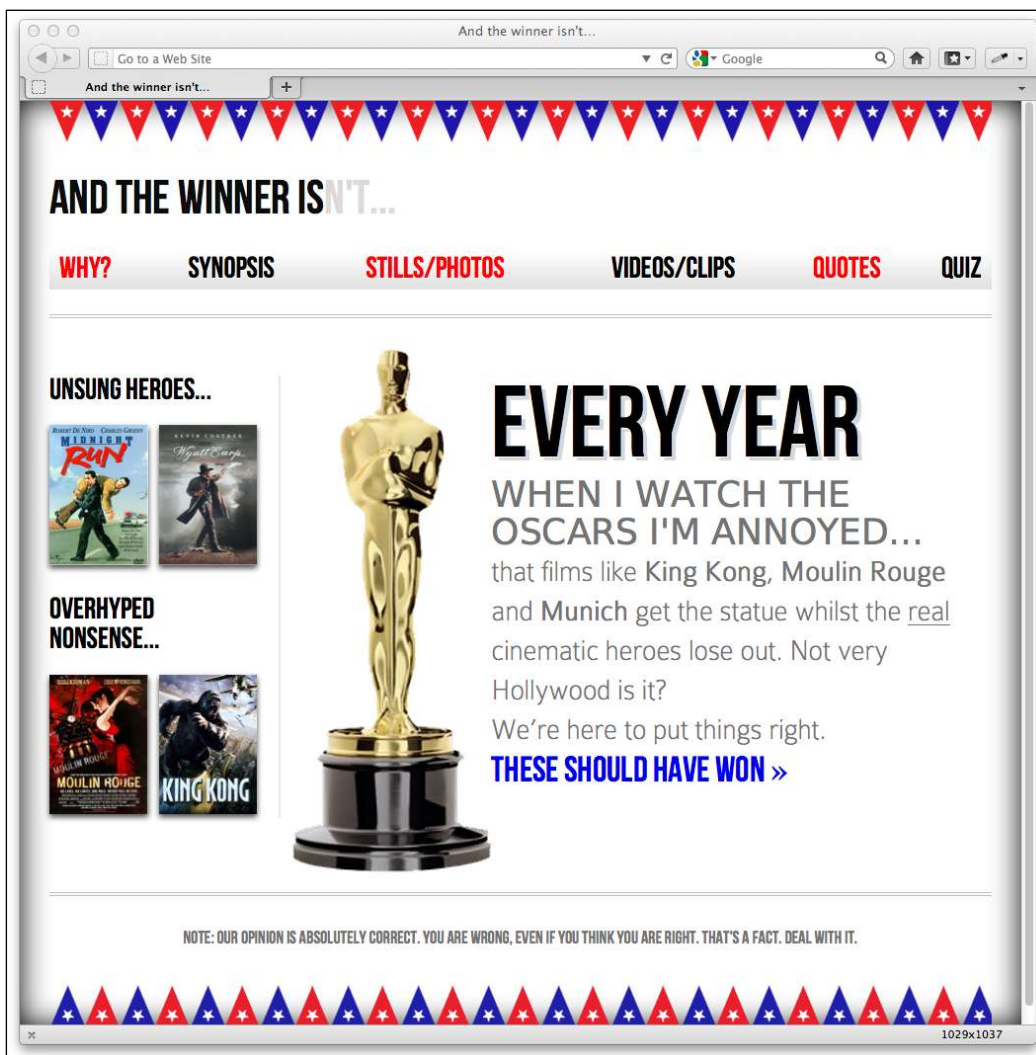
The `box-shadow` property can also be used to create an inset shadow – this applies within the targeted element, as opposed to the outside, as a normal box shadow would. It's useful for creating vignette effects for example. Here is the syntax:

```
box-shadow:inset 0 0 40px #000000;
```


Everything functions as before but the `inset` part of the declaration instructs the browser to set the effect on the **inside**. I'm going to use this rule now on the `<body>` tag to create a vignette effect for the entire page. The idea is make a shadow appear from all the edges of our page.

```
body {
  -moz-box-shadow:inset 0 0 30px #000000;
  -webkit-box-shadow:inset 0 0 30px #000000;
  box-shadow:inset 0 0 30px #000000;
}
```

Here's what the effect looks like in the browser:

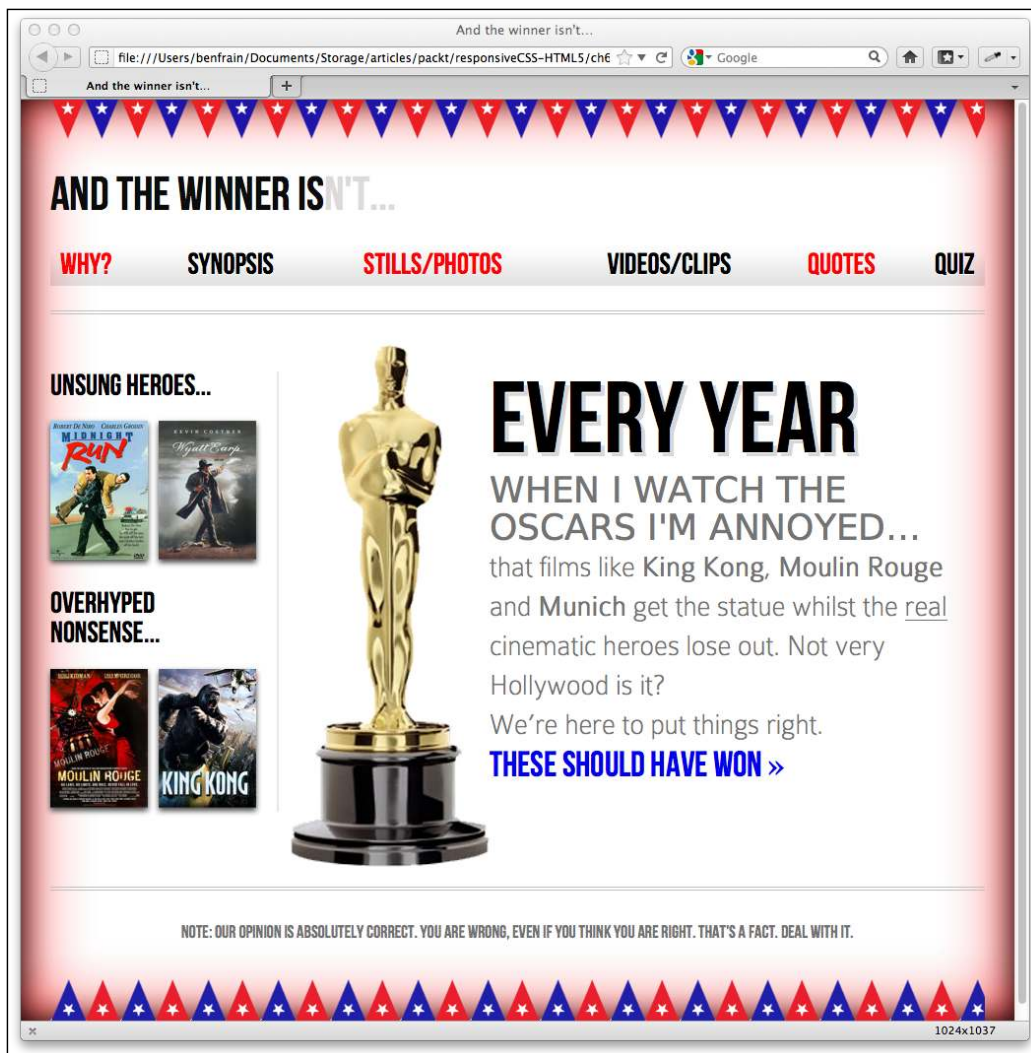


Multiple shadows

Like text-shadows, you can have multiple box-shadows. Again, merely separate the values with a comma and they are applied top to bottom as they are listed. I remind myself of the order by thinking that the declaration nearest to the top in the rule (in the code) appears nearest to the 'top' of the order when displayed in the browser.

```
box-shadow: inset 0 0 30px hsl(0, 0%, 0%),  
           inset 0 0 70px hsla(0, 97%, 53%, 1);
```

I've added this to my body rule and it produces an awful red boudoir effect. Must be the by-product of having an image of Moulin Rouge on the page!



Suffice to say, I'm taking that declaration straight out! However, this demonstrates the power of using CSS3 to toy with design ideas. Adding visual flourishes and removing them is a matter of seconds without having to touch a graphics editor.



You can read the W3C specification for the `box-shadow` property here:
<http://www.w3.org/TR/css3-background/#the-box-shadow>

Background gradients

When not using CSS3, if we want an element to have some sort of background gradient, we use a thin graphical slice and then tile it horizontally/vertically. As graphics resources go, it's quite an economical tradeoff. An image, only a pixel or two wide, isn't going to break the bandwidth bank and on a single site it can be used on multiple elements.

Linear background gradients

Let's start with this technique to make a linear background gradient for the sidebar of the **AND THE WINNER ISN'T** site:

```
aside {
    border-right-color: #e8e8e8;
    border-right-style: solid;
    border-right-width: 2px;
    margin-top: 58px;
    padding-left: 1.5%;
    padding-right: 1.0416667%;
    margin-left: 1.0416667%;
    float: left;
    width: 20.7083333%;
    background: url(..img/sidebarBg2.png) 50% repeat-x;
}
```

Here's how it looks in a browser:



However, it still requires trips to the graphics editor when we want to amend the effect. Plus occasionally, content can 'break out' of the gradient background, extending beyond its fixed size limitations. This problem is compounded with a responsive design, as we want the page structure to have the ability to change shape (for example, getting longer or wider) significantly without breaking up the design.

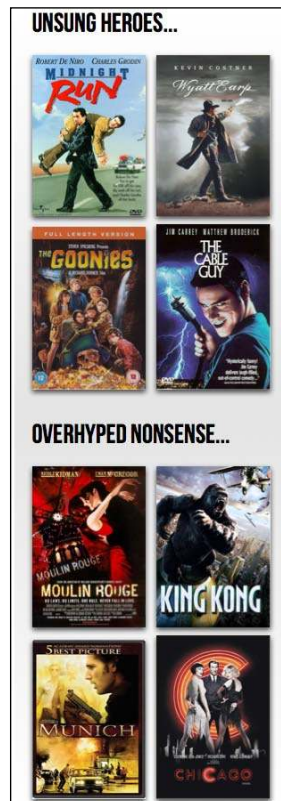
For example, let's suppose I wanted to add another two films in each section. Here's what happens:



It's not terrible but the grey gradient certainly isn't spanning the whole sidebar section, as I'd like. Ordinarily, I'd have to head back to my graphics editor and re-make the graphic. With a CSS3 gradient however, things are far more flexible. Here's the syntax for the same gradient in pure CSS3, instead of using an image:

```
background: linear-gradient(90deg, #ffffff 0%, #e4e4e4 50%,  
                                #ffffff 100%);
```

And here's how it looks in a supporting browser:



No matter how long that section gets (after all, there are plenty of films I could enthuse and moan about in equal measure), the CSS3 gradient will always cover the area.

The only significant fly in the ointment of background gradient nirvana is that they aren't supported as well as some of the other CSS3 features. Internet Explorer 9 doesn't have native support for them for instance (although it is promised for Internet Explorer 10). However, background gradients are supported in most other browsers, albeit with vendor prefixes. It shouldn't stop you from using them to enhance designs for browsers that support them now and others that will in the near future. As a fallback for older browsers, it's sometimes preferable to define a solid background color first so that older browsers at least render a solid background if they don't understand the gradient rules.



Note: there used to be different background gradient syntaxes

Historically, there were a number of different syntaxes employed by different browser vendors to render the same background gradient effect. Webkit was the main offender but thankfully, since Safari 5.1 they have adopted the same conventions as Mozilla – the conventions that the W3C is also using.

Breakdown of linear gradient syntax


The linear background gradient syntax (refer to the following example) is potentially confusing so let's break it down:

- Within the parenthesis the first (optional) value (in this case `90deg`) defines the direction the gradient starts off in. Leaving this out defaults to a vertical top to bottom gradient. You can also use values like `to top right`, which would be a diagonal gradient ending at the top right.
- The next value (`#ffffff 0%` in this example) is the 'starting point' – a color value given as the color and then the position. You could also use something like `blue 20%` which would then start fading from blue to the next color at 20 percent along the imaginary line from beginning to end of the linear gradient. Equally, you could set a negative value for the position so that the gradient begins before it is actually visible. For example:

```
background: linear-gradient(90deg, #ffffff -50%, #e4e4e4 50%,  
                             #ffffff 100%);
```

This line means that the gradient would start 50 percent before the beginning of the visible area the imaginary line travels along.

- The next value is a 'color stop'. Let's recap where we're at: in our example we are moving in an upwards direction at 90 degrees (`90deg`), starting with white (`ffffff 0%`), and moving towards a color value of `#e4e4e4` (a light grey color) at 50 percent along the line. This is our first 'color stop' within the gradient. We can use multiple color stops if we like, (separated by commas) before we define our 'ending point'.
- The final value in parenthesis (`ffffff 100%` in our example) is always the 'ending point' of the gradient. Regardless of how many color stops are placed after the starting point, the final value is always the ending point.


 Read the W3C specification for linear background gradients at:
<http://dev.w3.org/csswg/css3-images/#linear-gradients>

Radial background gradients

CSS3 background gradients aren't limited to linear gradients. It's equally simple to create a radial gradient. These begin from a central point and spread out smoothly in an elliptical or circular shape.

Here's the syntax for a radial background gradient:

```
background: radial-gradient(center, ellipse cover, #ffffff 72%,
                           #dddddd 100%);
```

Adding this declaration to our `#content` rule results in the following effect:



See that subtle darkening at the corners? That's our radial gradient. Let's break the syntax down to see what's going on.

Breakdown of radial gradient syntax

After specifying the property (`background:`) we specify that we'd like a radial-gradient (rather than a linear one). Then, within parenthesis we specify the starting point. In the previous example, we used `center` but we could equally use something like `25px 25px` to start 25 px from the top and left of the element. For example:

```
background: radial-gradient(25px 25px, ellipse cover, #ffffff 72%, #dddddd 100%);
```

This line of code produces the following effect:



The center is 25 px from the top left of the element and then radiates smoothly outwards.

The next value in our declaration is more straightforward; it's the shape and size the radial gradient should take:

```
background: radial-gradient(center, ellipse cover, #ffffff 72%, #dddddd 100%);
```

For shape, the options are either `circle` (the gradient will radiate uniformly in all directions) or `ellipse` (which will radiate different amounts in different directions). However, there's quite a bit of flexibility in how the shape is sized. The size can be any of the following:

- `closest-side`: the shape meets the side of the box nearest to the center (in the case of circles), or meets both the horizontal and vertical sides that are closest to the center (in the case of ellipses)
- `closest-corner`: the shape meets exactly the closest corner of the box from its center

- `farthest-side`: the opposite of `closest-side`, in that rather than the shape meeting the nearest size, it's sized to meet the one farthest from its center (or both the furthest vertical and horizontal side in the case of an ellipse)
- `farthest-corner`: the shape expands to the farthest corner of the box from the center
- `cover`: identical to `farthest-corner`
- `contain`: identical to `closest-side`

It's then a matter of defining the starting point, color stops, and end point (in exactly the same manner as linear gradients).

For example, if we changed our rule to this:

```
background: radial-gradient(20px 20px, circle cover,
                           hsla(9,69%,85%,0.5) 0%,
                           hsla(9,76%,63%,1) 50%,
                           hsla(10,98%,46%,1) 51%,
                           hsla(24,100%,50%,1) 75%,
                           hsla(10,100%,39%,1) 100%);
```

You can see we are starting 20 pixels from the left and top, using a circle to cover the area and using multiple HSL(A) color stops. Here's how it looks:



Hopefully, while this isn't the best lesson in aesthetics, it demonstrates the power of using pure CSS3 to achieve visual effects.



Read the W3C specification for radial background gradients at: <http://dev.w3.org/csswg/css3-images/#radial-gradients>



The cheat's way to perfect CSS3 linear and radial gradients

If writing out a CSS3 gradient seems like hard work there are some great online gradient generators. My personal favorite is <http://www.colorzilla.com/gradient-editor/>. It uses a graphics editor style GUI, allowing you to pick your colors, stops, gradient style (linear and radial gradients are supported), and even the color space (HEX, RGB(A), HSL(A)) you'd like the final gradient in. There are also loads of preset gradients to use as starting points. If that wasn't enough, it even gives you optional code for fixing up Internet Explorer 9 to show the gradient and a fallback flat color for older browsers. Still not convinced? How about the ability to generate a gradient based on an existing image? Thought that might swing it for you.

Repeating gradients

CSS3 also gives us the ability to create repeating background gradients. Let's take a look at how it's done:

```
background: repeating-linear-gradient(90deg, #ffffff 0px,
                                     hsla(0, 1%, 50%,0.1) 5px);
```

And here's how that looks applied to the sidebar:



Firstly, prefix the `linear-gradient` or `radial-gradient` with 'repeating', then it follows the same syntax as a normal gradient. Here I've used pixel distances between the white and grey colors (`0px` and `5px` respectively) but you could also choose to use percentages. For best results, it's recommended to stick to the same measurement units (such as, pixels or percentages) within a gradient.

Let's try a repeating radial gradient:

```
background: repeating-radial-gradient(2px 2px, ellipse,
    hsla(0,0%,100%,1) 2px, hsla(0,0%,95%,1) 10px,
    hsla(0,0%,93%,1) 15px, hsla(0,0%,100%,1) 20px);
```

It's very similar to the standard radial gradient used earlier. I've merely amended the start point, removed the 'cover' value as it's not needed and then set distances for each color stop in pixels. My end point is `20px` so the pattern repeats every 20 pixels. Here's that rule applied to the `body`. I'll warn you now – it isn't pretty!





Read the W3C information on repeating gradients at: <http://dev.w3.org/csswg/css3-images/#repeating-gradients>

There's one more way of using background gradients I'd like to share with you.

Background gradient patterns

It no doubt depends on your own design sensibilities but although I've often used subtle linear gradients in designs I've found less practical use for radial gradients and repeating gradients. However, clever folks out there have harnessed all these background techniques together to create background gradient patterns. Let's look at an example. Instead of the repeating radial gradient I just added to the `body`, I'll add this:

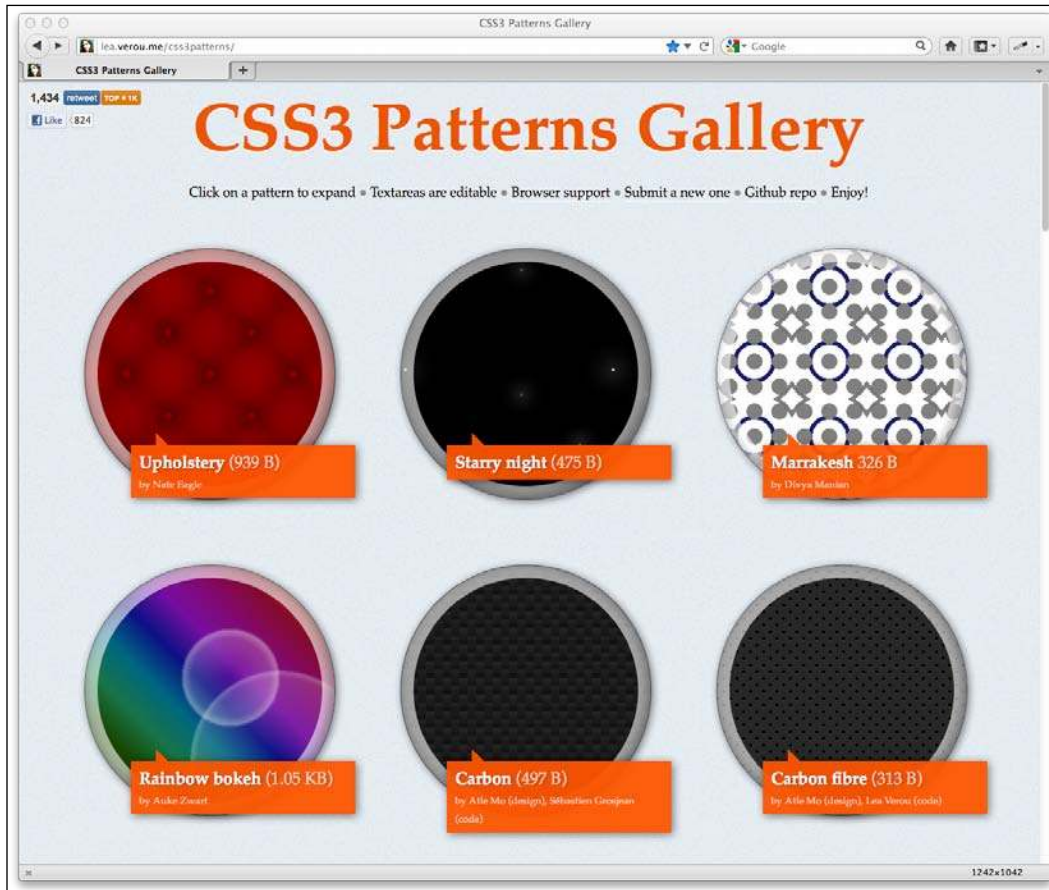
```
body {
  background-color:white;
  background-image:
    radial-gradient(hsla(0, 0%, 87%, 0.31) 9px, transparent 10px),
    repeating-radial-gradient(hsla(0, 0%, 87%, 0.31) 0,
      hsla(0, 0%, 87%, 0.31) 4px, transparent 5px,
      transparent 20px, hsla(0, 0%, 87%, 0.31) 21px,
      hsla(0, 0%, 87%, 0.31) 25px, transparent 26px,
      transparent 50px);
  background-size: 30px 30px, 90px 90px;
  background-position: 0 0;
}
```

Here's what that gives me in the browser:



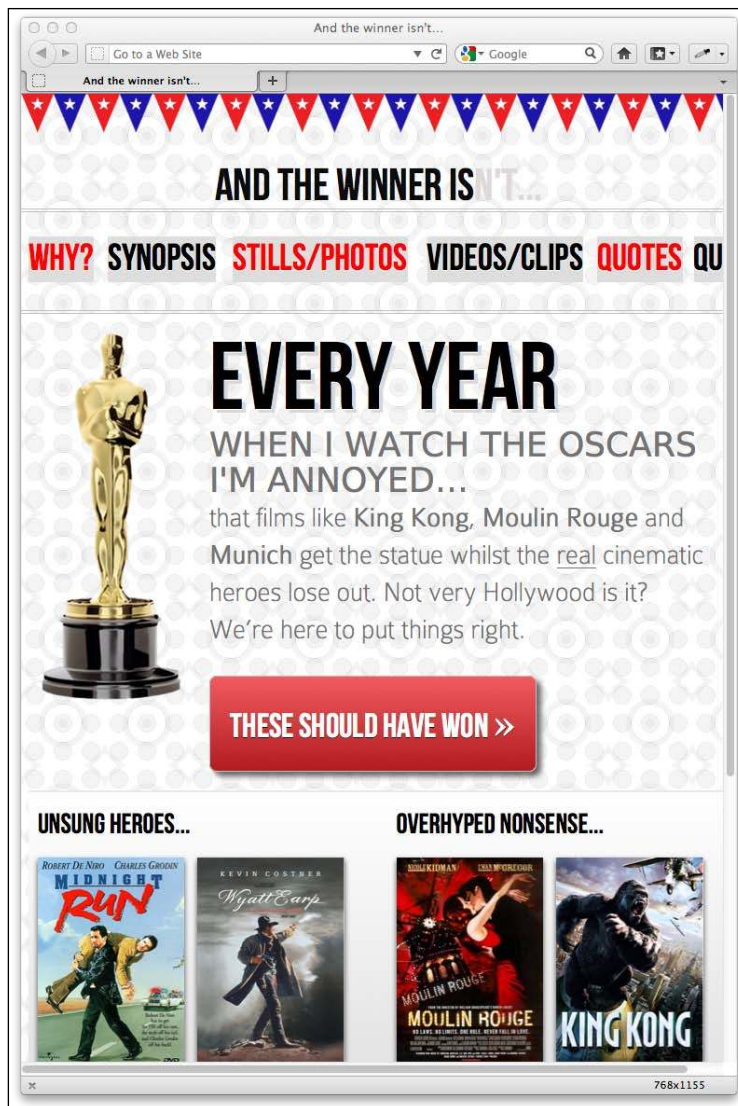
How about that? Just a few lines of CSS3 and we have an easily editable, scalable background pattern by using the background gradient techniques we've already looked at.

CSS Ninja, Lea Verou has collated a growing resource of CSS3 background patterns, available at <http://lea.verou.me/css3patterns/>.



Responsive considerations for CSS3

It's worth remembering that different declarations can be used for different viewports. For example, although I might not mind the way the gradient pattern looks on smaller viewports:



I may choose not to use it for larger viewports (for example 768 px wide and greater). I can therefore just create a specific rule for the background gradient using media queries:

```
@media screen and (max-width: 768px) {
  body {
    background-color:white;
    background-image:
      radial-gradient(hsla(0, 0%, 87%, 0.31) 9px, transparent 10px),
```

```
    repeating-radial-gradient(hsla(0, 0%, 87%, 0.31) 0,
    hsla(0, 0%, 87%, 0.31) 4px, transparent 5px, transparent 20px,
    hsla(0, 0%, 87%, 0.31) 21px, hsla(0, 0%, 87%, 0.31) 25px,
    transparent 26px, transparent 50px);
background-size: 30px 30px, 90px 90px;
background-position: 0 0;
}
}
```

Remember that media queries will allow you to specify every element differently for different viewports if you wish. It's all about presenting the best experience.

Writing CSS3 easily with CSS pre-processors



CSS3 rules currently require multiple vendor prefix properties. An alternative to storing clippings of these prefixes for every declaration, or using a JavaScript file to add prefixes in the browser are CSS pre-processors like SASS and LESS. For example, using SASS with the Compass plugin allows you to write a simple box shadow rule like this: `element { @include box-shadow; }`. When the CSS is generated, it includes a full stack of vendor specific rules along with the relevant Internet Explorer hacks (if available). If this wasn't a big enough reason to take a look, consider that pre-processors also add the ability to use variables and programming conventions like `if/while` statements. Find out more about SASS at <http://sass-lang.com> and LESS at <http://lesscss.org>

Bringing CSS3 properties together

Until now, we've largely been looking at abstract implementations of various CSS3 features. Let's use them together now to create our **THESE SHOULD HAVE WON>>** link. On the original Photoshop composite file for the **AND THE WINNER ISN'T** website, the button text uses custom typography, which we've already dealt with in *Chapter 5, CSS3: Selectors, Typography, and Color Modes*. However, it also has a red gradient background with rounded corners and a drop shadow behind it. This is what we have defined in the stylesheet currently:

```
#content a {
    text-decoration: none;
    font: 2.25em /* 36px ÷ 16 */ 'BebasNeueRegular';
}
```


First, let's add a solid background color for older browsers. That way, should they be unable to render the gradient, they will at least get a solid red background. I've purposely used a HEX value here because if the older browser doesn't understand gradients, it's unlikely to support RGB and HSL color modes:

```
#content a {
  text-decoration: none;
  font: 2.25em /* 36px ÷ 16 */ 'BebasNeueRegular';
  background-color: #b01c20;
}
```

Next, let's add our rounded corners. Note that, as in the rest of this chapter, for all the CSS3 properties I'll be adding it may be necessary to define vendor prefixes. I have omitted them here for the sake of brevity:

```
#content a {
  text-decoration: none;
  font: 2.25em /* 36px ÷ 16 */ 'BebasNeueRegular';
  background-color: #b01c20;
  border-radius: 8px;
}
```

Here's what we've got at this point:



Now, let's make the text white (again, as I want this viewable on older browsers, I've stuck to a simple color definition) and add padding (you could use percentage based padding too) so there's always a little space around the text:

```
#content a {
  text-decoration: none;
  font: 2.25em /* 36px ÷ 16 */ 'BebasNeueRegular';
  background-color: #b01c20;
  border-radius: 8px;
  color: white;
  padding: 30px;
}
```

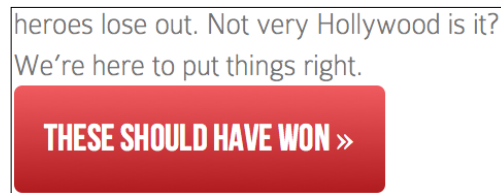
Here's what that gives us:



At this point the padding is encroaching on the text above so we'll add a `float: left` declaration along with the gradient:

```
#content a {
  text-decoration: none;
  font: 2.25em /* 36px ÷ 16 */ 'BebasNeueRegular';
  background-color: #b01c20;
  border-radius: 8px;
  color: white;
  padding: 30px;
  float: left;
  background: linear-gradient(90deg, #b01c20 0%, #f15c60 100%);
}
```

Now it's starting to take shape in the browser:



Besides adding a little margin above, I'll also go ahead and add the box shadow:

```
#content a {
  text-decoration: none;
  font: 2.25em /* 36px ÷ 16 */ 'BebasNeueRegular';
  background-color: #b01c20;
  border-radius: 8px;
  color: white;
  padding: 30px;
  float: left;
  background: -moz-linear-gradient(90deg, #b01c20 0%,
                                     #f15c60 100%);

  margin-top: 30px;
  box-shadow: 5px 5px 5px hsla(0, 0%, 26.6667%, 0.8);
}
```

And a quick check in the browser reveals we're almost done:



Now, although it's not in the Photoshop file, I'm going to add a little `text-shadow` and a thin white border, just to give it a slightly embossed feel. That's the beauty of using CSS rather than image files – it's easy to evaluate changes on the fly!

```
#content a {
  text-decoration: none;
  font: 2.25em /* 36px ÷ 16 */ 'BebasNeueRegular';
  background-color: #b01c20;
  border-radius: 8px;
  color: white;
  padding: 30px;
  float: left;
  background: -moz-linear-gradient(90deg, #b01c20 0%,
                                   #f15c60 100%);

  margin-top: 30px;
  box-shadow: 5px 5px 5px hsla(0, 0%, 26.6667%, 0.8);
  text-shadow: 0px 1px black;
  border: 1px solid #bfbfbf;
}
```

Now here's how our button looks in Firefox 8:



The only issue left is that our double angle quotes symbol (`»` in HTML) in the Photoshop file is in a different font from the main text. I don't feel that loading an extra font for the single character is worthwhile in this instance, so I'm going to wrap that symbol in an inline tag so that I can increase the size. Here's the amended markup:

```
<a href="#">these should have won <span>&raquo;</span></a>
```

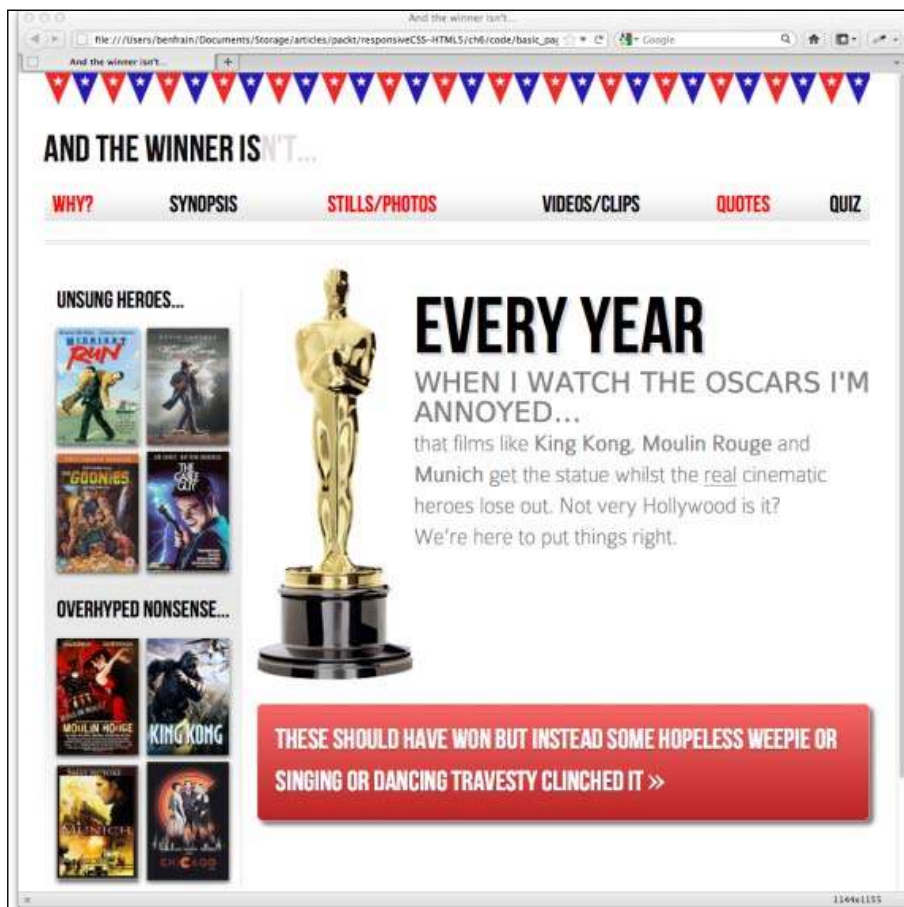
Additionally, here's the extra CSS rule to adjust the size:

```
#content a span {
    font-size: 1.3em;
}
```

Which finishes things off nicely:



What's great about this as CSS3, rather than an image, is that it can contain whatever content is needed, and it will never break up:



Multiple background images

A common design requirement is to build a page with a different background image at the top of the page than at the bottom. Or perhaps different images for the top and bottom of a content section within a page. It seems such a straightforward requirement, that it's understandable to assume this could be easily achieved with CSS. However, with CSS2.1, achieving the effect typically required additional markup. For example, until CSS3, this is how I've always solved the problem:

```
<body class="headerBackgroundHere">

<div class="footerBackground">
  <div id="container">
    <header>
      // Header content here
    </header>
    <div id="main" role="main">
      // Main content here
    </div>
    <footer>
      // Footer content here
    </footer>
  </div>

</div> <!--! end of .footerBackground -->

</body>
```

You'll notice the entire content container (which is the `div` with an `id` of `container`) is wrapped in a `div` with the class `footerBackground`. With this in place we can target a CSS rule to set the background image for the top of the page on the `body` tag:

```
body {
  background-image: url("../img/topSlice.png");
  background-repeat: repeat-x;
}
```

Then another rule for `footerBackground`. This is where we'll place the image we want for the bottom of the page.

```
.footerBackground {
  background-image: url("../img/bottomSlice.png");
  background-repeat: repeat-x;
  background-position: bottom;
}
```

This technique works well and consistently across most browsers. However, I'm never a fan of adding additional markup merely to solve presentational problems.

Thankfully this problem is easily solved with the CSS3 as it allows multiple backgrounds for an element (part of the **CSS Backgrounds and Borders Module Level 3**). It's well supported, with Internet Explorer 8 and below being the only notable exceptions. Here's the syntax:

```
background:
  url('../img/1.png'),
  url('../img/2.png'),
  url('../img/3.png');
```

As with the stacking order of multiple shadows, the image listed first appears nearest to the 'top' in the browser. You can also add a general color for the background in the same declaration if you wish, like this:

```
background:
  url('../img/1.png'),
  url('../img/2.png'),
  url('../img/3.png') left bottom, black;
```

Specify the color last and this will show below every image specified above.

Browsers that don't understand the multiple backgrounds rule (such as Internet Explorer 8 and below) will ignore the rule altogether so you may wish to declare a 'normal' background property immediately before a CSS3 multiple background rule as a fallback for older browsers.

With the multiple backgrounds, as long as you're using PNG files with transparency, any partially transparent background images that sit on top of another will show through below. However, background images don't have to sit on top of one another, nor do they all have to be the same size.

Background size

To set different sizes for each image, use the `background-size` property. When multiple images have been used, the syntax works like this:

```
background-size: 100% 50%, 300px 400px, auto;
```

The size values (first width, then height) for each image are declared, separated by commas in the order they are listed in the background property. As in the example above, you can use percentage or pixel values for each image alongside the following:

- `auto`: which sets the element at its native size
- `cover`: which expands the image, preserving its aspect ratio, to cover the area of the element

- `contain`: which expands the image to fit its longest side within the element while preserving the aspect ratio

Background position

Another thing that's possible is to specify different positions for the different images. We could do that by amending the rule like this:

```
background:
  url('../img/1.png') center,
  url('../img/2.png'),
  url('../img/3.png') left bottom, black;
```

Where no position is declared, as in the second image, the default position of top left is used.

Background shorthand

There is a shorthand method of combining the different background properties together. However, my experience so far has been that it produces erratic results. Therefore, I tend to use the longhand method and declare the multiple images first, then the size, and then the position.



Read the W3C documentation on multiple background elements here:

<http://www.w3.org/TR/css3-background/#layering>

Read about background sizing here: <http://www.w3.org/TR/css3-background/#the-background-size>

And background positions here: <http://www.w3.org/TR/css3-background/#the-background-position>

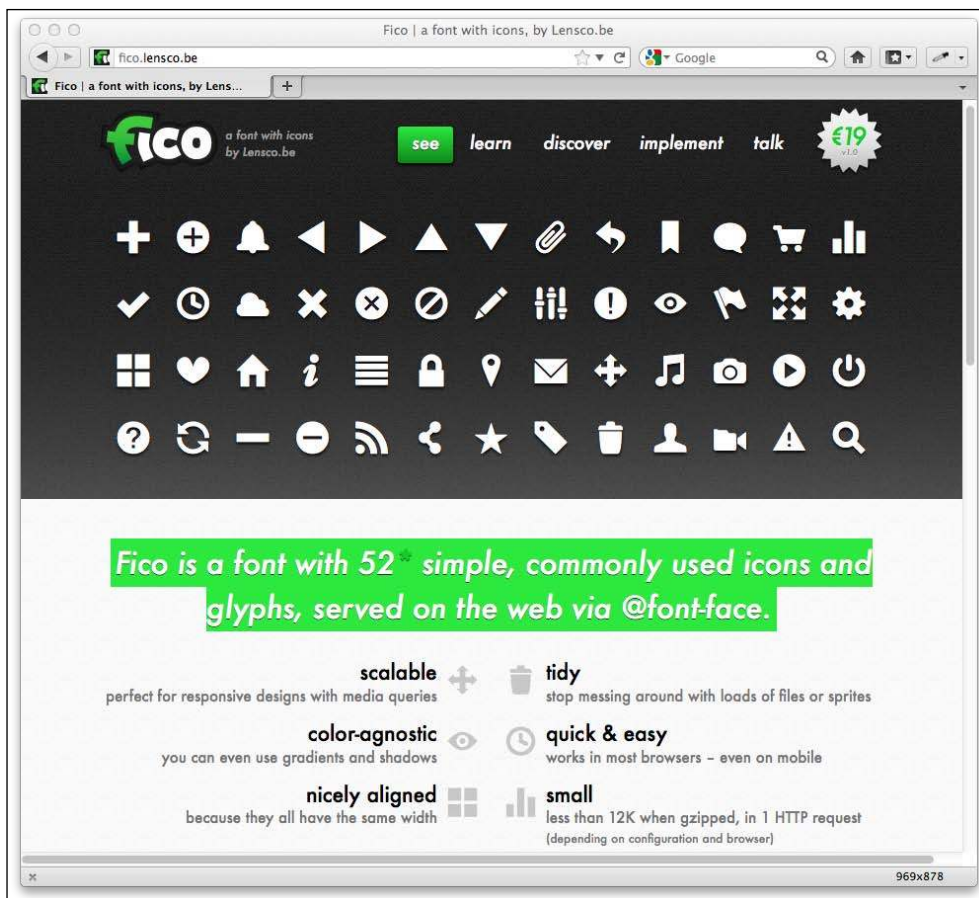
More CSS3 features

We've by no means covered all the goodies CSS3 has to offer. However, these are the ones that seem to be finding most traction in the real world. They are also the techniques I feel most benefit generating visual effects economically and flexibly for responsive designs. However, as always, keep an eye on the various CSS3 modules as there's sure to be something that will ignite your interest beyond the slice we've covered here.

Sizeable icons which are perfect for responsive designs

Smart people are already extending what's possible with CSS3 to great effect. One technique I've seen implemented that I love and now use regularly myself is using @font-face icons in a design.

"What are they?" I hear you cry. Well, my inquisitive friend, I'll tell you. Remember we used the CSS3 @font-face rules in the previous chapter to apply custom typography to our design? @font-face icons are merely fonts specifically made to create commonly used icons. Instead of using lots of separate graphics files for each icon, or even grouping them together into a single, larger sprite image, @font-face icons allow you to apply a single font for every included icon (that's just one http request – woo hoo!). What's more, as it's a font, it scales beautifully – perfect for responsive designs. Fico is a great example, check it out here: <http://fico.lensco.be/>.



Summary

In this chapter we've used a broader selection of CSS3's new features. CSS3's background gradients have enabled us to create some great looking background effects with pure code. We even used them to create background patterns. We've also learned how to use `text-shadow` to create an embossed effect on text and `box-shadow` to add drop-shadow effects to the outside and inside of elements.

When designing responsively, creating these aesthetic effects with pure CSS3 is a huge bonus; it means elements will not break out of any constraints usually associated with more resource heavy and inflexible images. That said, there are times when the use of images is unavoidable. But CSS3 gives us greater flexibility here too. For example, in this chapter we used CSS3's multiple background images feature to add multiple backgrounds and position them independently on the page; a technique that negates the need for extra markup, as has historically always been required. And remember, we're mostly using these effects to add visual flourishes to our responsive design, the kind of subtleties and niceties that modern browsers, regardless of their viewport size, can enjoy. Whilst tiring older browsers like Internet Explorer can't render them, they equally do them no harm.

So far however, all our forays into CSS3 have been static; elements that sit in place and remain stationary on the page in one state or another. However, CSS3 can do much more. In the next chapter we'll look at ways to transition from one state to another and take our CSS where it's never gone before: the domain of animation.