2

Speech Bubbles

One of the most fun and easy uses of CSS3 is for layering on visual "frosting"—non-essential visual flair and little details that can push your design from adequate to alluring. We'll use some of the most straightforward and well-supported CSS3 properties to create the appearance of three-dimensional speech bubbles that can be used to style blog comments, pull quotes, and more.

WHAT YOU'LL LEARN

We'll create the appearance of speech bubbles without using any images, just these pieces of pure CSS:

- The `word-wrap` property to contain overflowing text
- The `border-radius` property to create rounded corners
- HSLA to create semitransparent backgrounds
- The `linear-gradient` function to create gradient backgrounds
- The `box-shadow` property to create drop shadows behind objects
- The `text-shadow` property to create drop shadows behind (you guessed it) text
- The `transform` property to rotate objects

The Base Page

Let's say you're working on styling a blog's comments section. Before delving into any CSS3 fanciness, you'd want to get some basic styles in place to take care of older, non-CSS3-supporting browsers. As I mentioned in Chapter 1, it's important to make sure your pages are functional and at least decent-looking in browsers that don't support CSS3 before you add on CSS3 as part of progressive enhancement.

FIGURE 2.1 The comments area before any CSS3 is applied.

Figure 2.1 shows a blog's comments section with some basic styles applied. The text, avatar image, commenter's name, and date for each comment have been laid out neatly, the text is formatted, and we even have some basic backgrounds and borders in place. There's nothing wrong with this comments area; it's usable, it's clean, it's attractive. Anyone seeing it in an older browser would not think they were missing something or that the page was "broken."

But there's a lot we can do with CSS3, without adding a single image or touching the markup, to jazz up the page's appearance. To get started, download the exercise files for this chapter at www.stunningcss3.com, and open speech-bubble_start.html in your code editor of choice. Its CSS is contained in a `style` element in the `head` of the page, for ease of editing.

Corralling Long Text

OK, I know I just said we were going to jazz up the comments' appearance. But before we get into the actual speech bubble styles, let's quickly take care of an old, frustrating text-formatting problem that can be solved with the simplest bit of CSS3 you can imagine.

It's not uncommon for people to include URLs in comments and forum posts, and these URLs often overflow their containers due to their length (**Figure 2.2**). If the URLs have dashes (-) in them, all the major browsers can wrap the text of the URLs just fine. But Webkit-based browsers and IE will not wrap at the forward-slash (/) character, and none of the major browsers will wrap at underscores (_).

NOTE: Here's a pleasant surprise: the `word-wrap` property works in IE, as far back as version 5.5! The property was actually created by Microsoft and later adopted by W3C.

FIGURE 2.2 Long URLs often overflow their containers, especially if they contain underscores.

In CSS3, there's finally an easy way to tell the browser to wrap text within words and stop it from overflowing. All you have to do is give the `word-wrap` property a value of `break-word`, and the browser will wrap text within a word if it has to in order to keep it from overflowing.

THE LOWDOWN ON THE word-wrap PROPERTY

The word-wrap property is part of the Text module found at www.w3.org/TR/css3-text. It controls whether or not text is allowed to break within "words." (The separate text-wrap property controls how lines break between words.) The word-wrap property can be set either to normal (the default) or break-word.

Other than breaking long URLs, you might want to use word-wrap for:

- Keeping data tables from becoming too wide and overflowing or breaking your layout; see www.456bereastreet.com/archive/200704/how_to_prevent_html_tables_from_becoming_too_wide
- Wrapping displayed code snippets in pre elements; see www.longren.org/2006/09/27/wrapping-text-inside-pre-tags

TABLE 2.1 word-wrap **browser support**

IE	FIREFOX	OPERA	SAFARI	CHROME
Yes, 5.5+	Yes, 3.5+	Yes	Yes	Yes

In speech-bubble_start.html, find the blockquote rule in the CSS in the head of the page, and add the word-wrap property:

```
blockquote {
   margin: 0 0 0 112px;
   padding: 10px 15px 5px 15px;
   border-top: 1px solid #fff;
   background-color: #A6DADC;
   word-wrap: break-word;
}
```

Save the page and check it in a very narrow browser window. Ah, much better. The browser will still try to wrap first at normal breakpoints, but if it has to, it will now wrap the text at underscores or even within a word (**Figure 2.3**). Obviously, placing a break within a word is not ideal, but I think in this case it's preferable to the text overflowing and will probably only occur on long URLs, not regular text.

Now that we've taken care of that little annoyance, let's start making these comments look like speech bubbles!

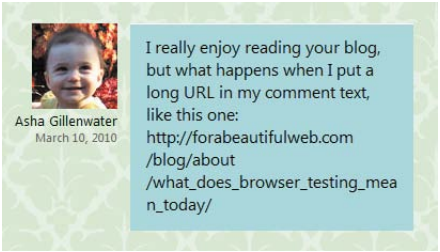

FIGURE 2.3 The browser will now break text between any two characters.

Graphic Effects Sans Graphics

You can create very graphic-looking speech bubbles without using any actual graphics. Avoiding graphics has many benefits beyond just being able to amaze your designer friends. You benefit by saving all the time and effort spent creating, slicing, and optimizing graphics, and then redoing them when your client inevitably wants to make one small change. Your visitors benefit from the increase in page speed that comes from having less data to download and fewer HTTP requests to the server.

NOTE: There's more in-depth information on the benefits of reducing images in Chapter 1, as well as a real-world case study.

Rounding the Corners

Those sharp, rectangular-cornered comments don't look very bubble-y, do they? Let's round the corners to start getting more of a speech-bubble look.

Rounded corners are a simple, common visual effect that used to be surprisingly hard to create in an actual web page. Creating the rounded-corner images in a graphics program was time-consuming, as was creating the actual HTML and CSS. You'd often have to add a bunch of extra nested `div`s to place each corner image separately, since CSS 2.1 allows only one background image per box, and the CSS used to actually control the placement of the images could get complicated. The images, along with the bloated markup and CSS, bulked up the amount that each visitor had to download, slowing down page-loading speeds. Even if you used a script to dynamically create the rounded corners instead of manually creating and applying images, you were still adding to the number of files that users had to download and decreasing your pages' performance. All this trouble for some simple-looking little rounded corners!

CREATING OVALS AND CIRCLES WITH `border-radius`

If you want your speech bubbles to be complete ovals instead of rounded rectangles, you'll need to use elliptical-shaped corners instead of perfectly round ones. *Elliptical* just means that the curve of each corner is somewhat flattened out—just like an oval. To specify an elliptical corner, you write two measurements, separated by a slash, such as this: `border-radius: 50px/20px`. (Safari 3 and 4 use the non-standard syntax of no slash, just a space.) This means that the curve will extend horizontally 50 pixels but vertically only 20 pixels, making a flattened, elliptical curve. You can make each corner have different angles; find out how at http://css-tricks.com/snippets/css/rounded-corners.

To create circles, first give your box the same `width` and `height`; use ems as the unit of measurement instead of pixels to ensure it can grow with its text. Then set each corner's `border-radius` to one-half the `width`/`height` value. For instance, if you have a box that is 10 ems wide and tall, use `border-radius: 5em`. See http://blog.creativityden.com/the-hidden-power-of-border-radius-2 for more examples.

In CSS3, creating rounded corners can be as simple as `border-radius: 10px` on a single `div`. No extra markup, no images, no JavaScript.

Of course, while CSS3 continues to be developed and gain browser support, it's a little more complicated in real-world usage. But it's still really, really easy.

In your page, modify the `blockquote` rule to match the following:

```
blockquote {
   margin: 0 0 0 112px;
   padding: 10px 15px 5px 15px;
   -moz-border-radius: 20px;
   -webkit-border-radius: 20px;
   border-radius: 20px;
   border-top: 1px solid #fff;
   background-color: #A6DADC;
   word-wrap: break-word;
}
```

The `border-radius: 20px;` declaration is the W3C standard syntax for rounded corners, specifying that all four corners should be rounded by 20 pixels. This syntax is currently supported by Opera, Chrome, Safari 5, and IE 9. Firefox and Safari 4 and earlier use the `-moz-border-radius` and `-webkit-border-radius` properties, respectively. As explained in Chapter 1, browser vendors use these browser-specific prefixes when the specification is still being worked out and they think it may change. The non-prefixed version of the property (in this case, plain `border-radius`) should always come last, so that when browsers *do* support the non-prefixed property, it overrides the earlier rules, which may use non-standard behavior from an older version of the spec.

NOTE: You don't have to actually declare a border when using `border-radius`. If there is no border, the browser just rounds the background area.

THE LOWDOWN ON THE `border-radius` PROPERTY

The `border-radius` property is part of the Backgrounds and Borders module found at www.w3.org/TR/css3-background. It's shorthand for the properties specifying the rounding amount of each of the four corners, in this order: `border-top-left-radius`, `border-top-right-radius`, `border-bottom-right-radius`, `border-bottom-left-radius`. Mozilla's properties for individual corners have the non-standard syntax of `-moz-border-radius-topleft` and so forth.

You can write out all four values, with spaces in between, in one `border-radius` property, or just use one value to round all four corners the same amount. Safari 4 and Safari on iOS 3 and earlier don't allow you to specify multiple corners in the shorthand `border-radius` property, other than writing one value to specify all four at once.

See the "Creating ovals and circles with `border-radius`" sidebar for the syntax for elliptical curves on corners. Also see www.owlfolio.org/htmletc/border-radius and http://muddledramblings.com/table-of-css3-border-radius-compliance for more `border-radius` syntax details and examples.

Other than speech bubbles, you might want to use `border-radius` for:

- Buttons; see http://blogfreakz.com/button/css3-button-tutorials and http://css-tricks.com/examples/ButtonMaker
- Tabs
- Dialog boxes
- Circular badges
- Bar charts; seewww.marcofolio.net/css/animated_wicked_css3_3d_bar_chart.html
- Smiley faces; see http://ryanroberts.co.uk/_dev/experiments/css-border-faces

TABLE 2.2 `border-radius` **browser support**

IE	FIREFOX	OPERA	SAFARI	CHROME
Yes, 9+	Yes with -moz-	Yes	Yes, 5+; 4+ with -webkit-	Yes

NOTE: See how I keep referring back to Chapter 1? If you skipped it, please go back and read it now. There's some important stuff there.

With these three lines added, the corners are now rounded in all browsers except IE 8 and earlier (**Figure 2.4**). These versions of IE simply ignore the properties and keep the corners straight—no harm done. This is a great example of progressive enhancement, as explained in Chapter 1. Since this is a purely decorative effect, I see no harm in IE users missing it. If you do, read on.

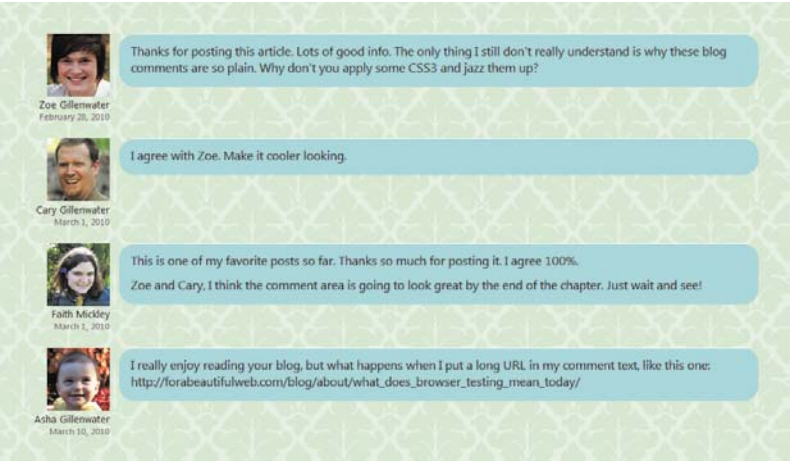

FIGURE 2.4 The `border-radius` **property applied**

WORKAROUNDS FOR IE

If you really must have rounded corners in IE 8 and earlier, you can use one of these scripts:

- "PIE," by Jason Johnston (http://css3pie.com), reads the `border-radius` properties that are already present in your CSS and makes them work in IE 6 and later. It also adds several other CSS3 effects to IE.
- "curved-corner," by Remiz Rahnas (http://code.google.com/p/curved-corner), also reads the `border-radius` properties out of your CSS, but works only when all four corners have the same `border-radius`.

- "IE-CSS3," by Nick Fetchak (http://fetchak.com/ie-css3), is based off of curved-corner but also adds drop shadows in IE.
- "DD_roundies," by Drew Diller (http://dillerdesign.com/experiment/DD_roundies), lets you round corners individually, but it doesn't read the values from your CSS; you have to manually set the IE values separately.

Besides these IE-specific scripts, there are a number of rounded-corner scripts and image-based techniques out there that were developed before the `border-radius` property gained support, so you could always go back to one of these older techniques for IE. You can choose between dozens of options at www.smileycat.com/miaow/archives/000044.php and http://css-discuss.incutio.com/wiki/Rounded_Corners.

If you do use a script or images for IE, make sure to hide them from other browsers by placing the script references or IE styles within conditional comments, or by using Modernizr, both of which are explained in Chapter 1. That way, only IE users get the performance hit of using an old-school rounded-corner method, and non-IE users get the faster, pure CSS version. You'll have to decide if the extra work and performance hit is worth having IE users see rounded instead of straight corners.

Adding the Bubble's Tail

With rounded corners, each comment box now looks more like a bubble, but a speech bubble isn't complete without a pointer or arrow, commonly called a "tail," pointing to the speaker. We can add that tail without using any graphics. In fact, we can add it without using any CSS3—the technique only uses properties and selectors from CSS 2.

CREATING TRIANGLES OUT OF BORDERS

All we need to create a tail is a triangle, and you can create triangles with pure CSS by using regular old borders. When two borders of a box meet at a corner, the browser draws their meeting point at an angle (**Figure 2.5**). If you reduce that box's `width` and `height` to zero, and give every border a thick width and a different color, you'll end up with the appearance of four triangles pushed together, each pointing in a different direction (**Figure 2.6**).

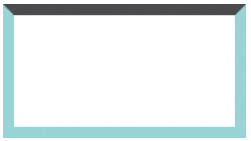

FIGURE 2.5 By making the top border a different color, you can see that borders meet at corners at an angle.

FIGURE 2.6 When a box has no width or height, each border creates the appearance of a triangle.

TIP: Remember, in CSS, when you have four values in a single property, like in the `border-color` property shown in the code here, the first value is for the top, the second for the right, the third for the bottom, and the fourth for the left. Think of going around a clock clockwise.

Here's what the HTML and CSS used to create Figure 2.6 look like:

```
<div class="triangles"></div>

.triangles {
   border-color: red green blue orange;
   border-style: solid;
   border-width: 20px;
   width: 0;
   height: 0;
}
```

What would happen if you made the top, left, and bottom borders transparent instead of colored? Only the right border would show, leaving the appearance of a left-pointing triangle (**Figure 2.7**):

FIGURE 2.7 Making all but one of the borders transparent creates the appearance of a single triangle.

```
<div class="triangle-left"></div>

.triangle-left {
   border-color: transparent green transparent transparent;
   border-style: solid;
   border-width: 20px;
   width: 0;
   height: 0;
}
```

So, to sum that up, all you need to do to create a triangle using CSS is give an element zero `width` and `height`, give it thick borders, and make all but one of those borders transparent. You can vary the angle of the triangle by making the widths of the borders different on different sides.

GENERATING THE TAIL

Now that you know how to make an image-free triangle, let's add a left-pointing triangle to the left side of each comment, pointing to the commenter's avatar. To do this, we could nest a `span` or `div` inside each comment, and then transform this element into our triangle, but let's leave the HTML pristine and use CSS-generated content to make the element we need appear.

Generated content is a CSS 2.1 technique where you place content into your CSS to have it appear in your HTML. It's useful for adding things that you don't want to manually hard-code into the HTML, like numbers before headings or icons after links. It shouldn't be used for essential content that would be missed if the user couldn't access the CSS file.

To create generated content, you need to specify *where* the content is to be inserted, using either the `::before` or `::after` pseudo-elements (also written as `:before` and `:after`), and specify *what* content to insert, using the `content` property.

WHAT'S WITH THE DOUBLE COLONS?

You may have noticed that I wrote the `::before` and `::after` pseudo-elements with double colons instead of the single colons you may be used to seeing. No, it's not a typo. CSS3 changed the syntax for pseudo-elements to use double colons, while pseudo-classes retain the single colons.

You can continue to use the single colon versions if you wish; they still work just fine. In fact, since IE 8 and earlier don't support the double-colon versions, we'll stick with the single colon versions in this book. You could also use both as a grouped selector, such as `.caption:before, .caption::before { content: "Figure: ";}`.

For instance, to insert the word "Figure" before every image caption on your page, you could use the following CSS:

```
.caption:before {
   content: "Figure: ";
}
```

This CSS would turn the HTML `<p class="caption">Isn't my cat cute?</p>` into this text when seen on the page:

Figure: Isn't my cat cute?

In the case of the speech-bubble tail we want to generate, all we want to see are the *borders* of the generated content, not the content itself. So, let's generate a piece of invisible content: a non-breaking space.

The HTML entity for a non-breaking space is ` `, but you can't use HTML entities within the `content` property. Instead, you need to use the hexadecimal part of the character's Unicode code point (or reference). That may sound really confusing and difficult and science-y, but don't be scared—there are lots of handy charts online that allow you to look up this kind of stuff.

For instance, at www.digitalmediaminute.com/reference/entity you can see 252 little boxes, each showing one of the allowed entities in (X)HTML. In the "Filter entities by keyword" box, type "non-breaking

TIP: Another useful tool is the Unicode Code Converter at http://rishida.net/tools/conversion, where you can put in the character or its HTML entity name and convert it into a bunch of different formats, including its hexadecimal code point.

space." 251 of the boxes will disappear, leaving you with one box showing , the HTML entity name. Position your cursor over the box (**Figure 2.8**). Two other codes will appear: its numerical code (in this case,) and its Unicode code (u00A0). You just want the hexadecimal part of the Unicode code, which is the part after the "u." Copy the text "00A0" onto your clipboard.

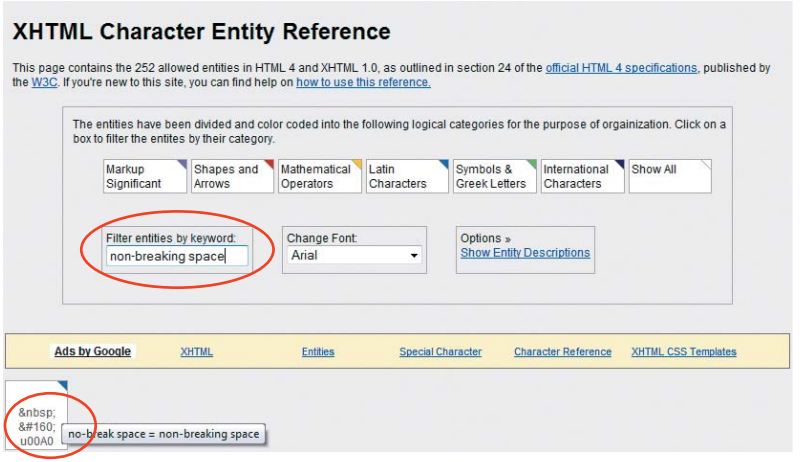

FIGURE 2.8 **Use the XHTML Character Entity Reference to look up the Unicode code points of various entities.**

NOTE: Unicode code points are often written with a prefix of "U+" instead of just "u." In either of these cases, the part you want to include in the content property is just the four-digit hexadecimal part that comes after the prefix.

Now we're almost there; but even though we now have the Unicode code we need, we can't put it straight into the content property, like so:

```
blockquote:after {
   content:"00A0";
}
```

If we did this, the browser would quite logically make the text "00A0" show up, instead of the non-breaking space. To tell the browser that we're putting in a special character code, we need to *escape* the code. If you're a programmer, you'll be familiar with this term, but for the rest of us, all it means is that you have to put a backslash in front of the code. This alerts the browser that what follows the slash is not to be taken as literal text, but is instead a code for something else.

With the backslash, we finally have all the correct characters and punctuation needed to insert a simple non-breaking space:

```
blockquote:after {
   content:"\00A0";
}
```

Once you do this, the page will look exactly the same; the non-breaking space is invisible, of course. Let's add the borders around it to make it show up. We also need to set its `width` and `height` to zero and make it display as a block element so we can move it around to place the tail against the side of the speech bubble:

```
blockquote:after {
   content: "\00a0";
   display: block;
   width: 0;
   height: 0;
   border-width: 10px 20px 10px 0;
   border-style: solid;
   border-color: transparent #000 transparent transparent;
}
```

NOTE: The `:before` pseudo-element would have worked just as well as `:after` in this case. We're going to be moving it from its default position regardless, as you'll soon see.

If we had made all four borders the same width, we'd end up with a rather fat triangle, like the one shown in Figure 2.7. To make the triangle a little longer and thinner, we've set the top and bottom borders to only 10 pixels, and the left border is nonexistent at zero pixels. The right border—the one we use to create the appearance of a left-pointing triangle—is a nice, wide 20 pixels. All the borders except the right one are transparent; here I've set the right border's color to black temporarily just so we can see it in order to place it correctly (**Figure 2.9**).

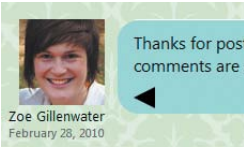

FIGURE 2.9 The black right border creates the appearance of a left-pointing triangle.

The triangle is currently placed right after the `blockquote`'s content—not the right spot for a speech bubble's tail. You can correct this by moving it with absolute positioning. First, add `position: relative;` to the `blockquote` rule; this establishes it as the reference point for the absolute element's positioning:

```
blockquote {
   position: relative;
   margin: 0 0 0 112px;
   padding: 10px 15px 5px 15px;
   -moz-border-radius: 20px;
   -webkit-border-radius: 20px;
   border-radius: 20px;
   border-top: 1px solid #fff;
   background-color: #A6DADC;
   word-wrap: break-word;
}
```

Then, add the absolute positioning to the generated content, along with `top` and `left` values:

```
blockquote:after {
   content: "\00a0";
   display: block;
   position: absolute;
   top: 20px;
   left: -20px;
   width: 0;
   height: 0;
   border-width: 10px 20px 10px 0;
   border-style: solid;
   border-color: transparent #000 transparent transparent;
}
```

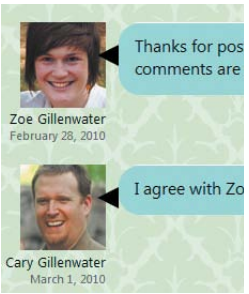

FIGURE 2.10 Absolute positioning places the triangle where we want it.

You can set the `top` value to whatever you want; just make sure it's equal to or greater than the `border-radius` value so it lands on the straight edge of the box, below the corner curve. The `left` value should be a negative value in order to pull the triangle to the left, and it should match the width of the triangle. In this case, the width of the triangle is 20 pixels, because that's the width of the right border, so we're using a `left` value of `–20px`. This places the triangle right up against the left edge of the comment box (**Figure 2.10**).

It's possible that a comment might be so short that the tail hangs off the bottom, as seen in the second comment in Figure 2.10. To fix this, add `min-height: 42px;` to the `blockquote` rule.

```
blockquote {
   position: relative;
   min-height: 42px;
   margin: 0 0 0 112px;
   padding: 10px 15px 5px 15px;
   -moz-border-radius: 20px;
   -webkit-border-radius: 20px;
   border-radius: 20px;
   border-top: 1px solid #fff;
   background-color: #A6DADC;
   word-wrap: break-word;
}
```

Now that the triangle isn't layered over the `blockquote`, we can change its color to match the `blockquote`:

```
blockquote:after {
   content: "\00a0";
   display: block;
   position: absolute;
   top: 20px;
   left: -20px;
   width: 0;
   height: 0;
   border-1width: 10px 20px 10px 0;
   border-style: solid;
   border-color: transparent #A6DADC transparent
                 transparent;
}
```

This creates a seamless appearance between the bubble and the tail parts of each speech bubble (**Figure 2.11**).

NOTE: The page with all the changes to this point is named speech-bubble_1.html in the exercise files that you downloaded for this chapter.

FIGURE 2.11 Each tail is now colored and placed correctly.

WORKAROUNDS FOR IE

Our tail shows up fine in IE 8 and later versions, but IE 7 and earlier versions don't support generated content, so they don't see the tail. I think this is fine in this case, as there's no reason users of those browsers would see the plain rectangles and think, "Hey wait a second! Why isn't there a little triangle sticking out of each comment block?"

To add tails in IE 7 and earlier, you'd need to manually add another element to the HTML of each comment, such as an empty `span`, and turn this element into the triangle.

Semitransparent Backgrounds with RGBA or HSLA

There's nothing more that we have to do to create the appearance of a speech bubble—we've got the rounded corners and the tail—but it would be nice to add a little more depth and visual richness with some extra graphic details.

One great way to add depth is to make backgrounds semitransparent (also called alpha transparency). By letting a little bit of the page background show through, you create more of a layered appearance, as if the semitransparent element is floating over the background. I think this look is especially well-suited to speech bubbles, because, well, they're bubbles—light and airy.

Before CSS3, you could create semitransparent backgrounds using an alpha-transparent PNG as a tiling background image. Using a background image has the disadvantage of adding another hit to your server, making pages load a little slower for your users. Performance is impacted even more if you need to support IE 6, since it needs a script to be able to understand alpha-transparent PNGs. Plus, you can't use a background image on a border, so you wouldn't be able to make the speech bubble's tail semitransparent. It would look pretty weird for the body of the bubble to be semitransparent and the tail to be totally opaque.

CSS3'S RGBA AND HSLA SYNTAX

Luckily, in CSS3 we have both RGBA and HSLA to turn to. Both are methods for specifying a color and its level of transparency at the same time. RGBA stands for red-green-blue-alpha (for alpha transparency) and HSLA stands for hue-saturation-lightness-alpha.

We could specify the shade of blue that we're using as the speech bubble's background using any of these syntaxes:

- Hexadecimal: #A6DADC
- RGB: 166, 218, 220
- RGBA: 166, 218, 220, 1
- HSL: 182, 44%, 76%
- HSLA: 182, 44%, 76%, 1

They all get us to the same exact color, just through different routes. It's a "you say toe-may-toe, I say toe-mah-toe" sort of thing.

In the RGBA syntax, the first three values are the amounts of red, green, and blue, either from 0–255 or 0%–100%. (You'll most often see the 0–255 values, not the percentages.) In the HSLA syntax, the first three values are the hue value, from 0 to 360; the percentage level of saturation; and the percentage level of lightness. In both RGBA and HSLA, the fourth value is the opacity level, from 0 (completely transparent) to 1 (completely opaque).

NOTE: CSS3 also has an `opacity` property, but it makes the entire element semitransparent, including its content, instead of just the background.

You can use most graphic editors to determine the correct red, green, and blue values needed to create your chosen color. Use the color picker to choose a color, and in the color dialog box or picker window, most graphic editors will tell you that color's hexadecimal code as well as RGB values (**Figure 2.12**). Finding HSL values can be a little trickier, as not all image-editing software uses HSL; for instance, Photoshop uses HSB (also called HSV), which is similar, but not quite the same. If you're on a Mac running Snow Leopard, check out the free app Colors by Matt Patenaude (http://mattpatenaude.com), which lets you pick colors from anywhere on your screen and can display values in HSLA as well as other syntaxes. If you're not on a Mac, I recommend you use one of the online HSL color picker or converter tools (see the "Online color tools" sidebar).

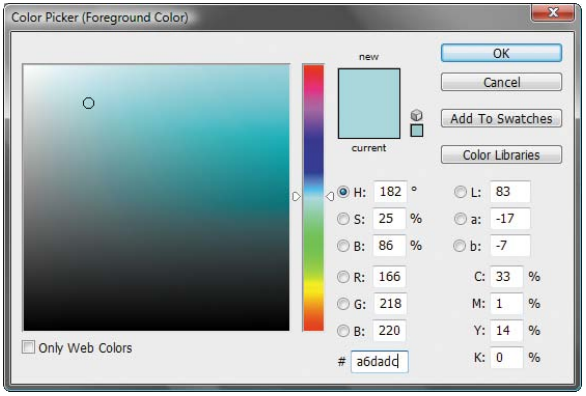

FIGURE 2.12 Photoshop's Color Picker dialog box shows the equivalent RGB values for the chosen hex color.

ONLINE COLOR TOOLS

There are many free web-based color picker and converter tools that you can find through Googling, but here are a couple that are particularly handy for working with RGB and HSL values:

- The color converter tool at http://serennu.com/colour/hsltorgb.php allows you to convert color values you already have into hex, RGB, and HSL syntaxes.
- The Doughnut Color Picker at www.workwithcolor.com/doughnut-color-picker-01.htm lets you both pick and convert colors. The picker uses HSL, but gives the hex and RGB equivalents, and lets you input colors in any of the three syntaxes.

Some browser-based color pickers make finding HSL or RGB values even easier and faster. I'm a big fan of the Rainbow extension for Firefox (https://addons.mozilla.org/en-US/firefox/addon/14328). After you install the extension, you can tell it which syntax to use to display color values (**Figure 2.13**). Then, when you use its Inspector tool to choose colors from a web page, it gives you the option to automatically copy those values to your clipboard (**Figure 2.14**), and you can then easily paste them into your CSS. Note that, as of this writing, the extension doesn't include the "A" part of either RGBA or HSLA, so you'll have to add that part in by hand. But I think you can handle all that typing.

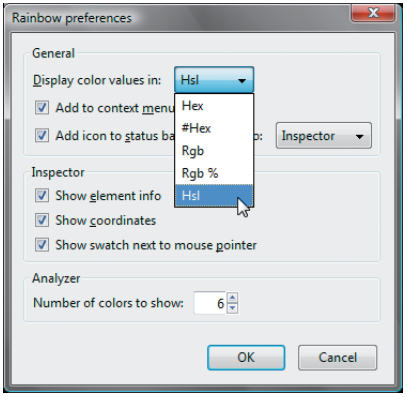

FIGURE 2.13 In the options for the Rainbow extension, set the "Display color values in" option to "Hsl."

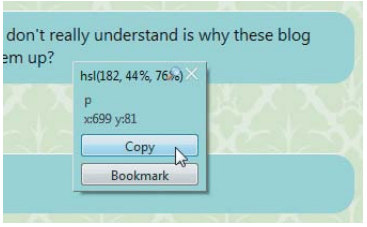

FIGURE 2.14 Using Rainbow's Inspector tool, you can click on a color to display and copy its color code.

RGBA VERSUS HSLA

The main reason I recommend the Rainbow Firefox extension over some other color picker extensions is that many others don't include HSL values, while Rainbow does, and I prefer HSLA over RGBA.

I'm in the minority here. Many more people use RGBA than HSLA, but I think that's mainly because most people haven't heard of HSLA. It's a shame, because the majority of people who use HSLA find it more intuitive.

With RGB and RGBA values, it's hard to tell at a glance what the color is going to be. If you take a minute to study a whole RGB or RGBA value, such as `rgb(166,218,220)`, you can get a fair idea of the resulting color, based on which of the three component color values (red, green, or blue) are highest. But I'm not a big fan of taking that minute to parse it out while I'm trolling through my style sheet trying to track down where some mysterious color is coming from. And even after I determine that an RGB value is producing a greenish-blue hue, for instance, it's hard to tell how muted or dark that greenish-blue is by looking at only its red, green, and blue values.

HSL AND HSLA HUE VALUES CHEAT SHEET

If you're going to use HSLA, it's helpful to memorize the hue values of a few key colors (or at least approximately where they are between 0 and 360, so you can tweak your way to the shade you want).

- 0 or 360 = red
- 30 = orange
- 60 = yellow
- 120 = green
- 180 = cyan
- 240 = blue
- 270 = purple
- 300 = magenta

To get black in HSL and HSLA, just set the lightness value to zero percent. For white, set the lightness value to 100 percent. In both cases, the hue and saturation values can be whatever you want.

To get gray in HSL and HSLA, just set the saturation value to zero percent. The lightness value will control the shade of the gray, and the hue value is irrelevant.

Another problem with RGB and RGBA is that if you want to tweak a color—make it a little darker or brighter or greener—you have to guess at how to change each of the values to get to the hue you want. In web design, it's common to use multiple shades of the same hue in different places in the page, such as a brightened version of a color on the current tab in a nav bar. But with RGB, different shades of the same hue don't necessarily have very similar color values. For instance, a just slightly darker version of the shade of blue we've been working with would have an RGB value of 155, 209, 211 instead of the original 166, 218, 220. All three numbers have to change to produce a very slight shift in darkness.

With HSL and HSLA, you don't have to add amounts of red, green, and blue to get a specific hue, but instead set that hue as one specific number. All you have to do is remember that both 0 and 360 equal the same shade of pure red. As you increase the hue value from 0, you simply move through the rainbow from red to purple and then back around to red again, as if you were going around a color wheel (**Figure 2.15**).

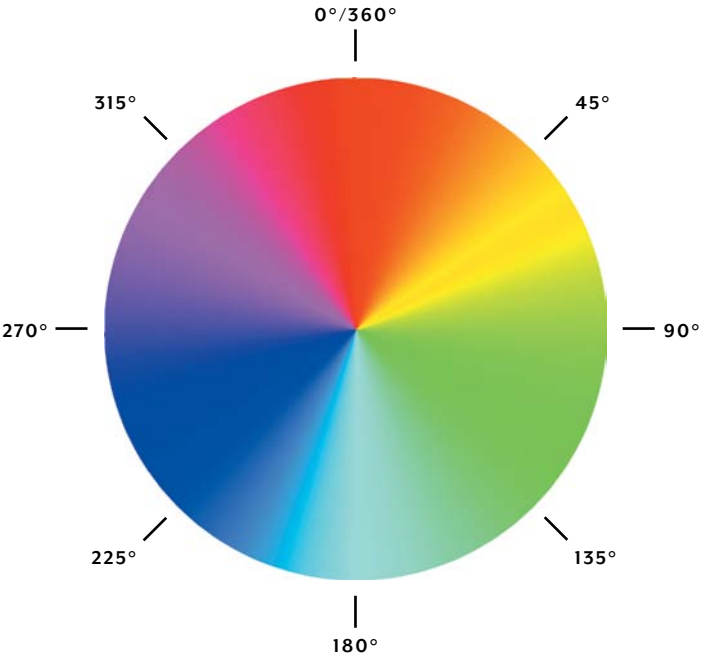

FIGURE 2.15
The 360 hue values in the HSL color syntax

THE LOWDOWN ON RGBA AND HSLA

RGBA and HSLA are part of the Color module found at www.w3.org/TR/css3-color. Both allow you to set a color and its level of transparency at the same time.

In the RGBA syntax, the first three values are the amounts of red, green, and blue, either from 0–255 or 0%–100%. In the HSLA syntax, the first three values are the hue value from 0 to 360, the percentage level of saturation, and the percentage level of lightness. In both RGBA and HSLA, the fourth value is the opacity level from 0 (completely transparent) to 1 (completely opaque).

Other than ghostly bubble backgrounds, you might want to use RGBA or HSLA for:

- Drop shadows that tint the background beneath them (you'll learn how to do this later in this chapter)
- Gradient highlights on buttons or any other objects (again, you'll learn how to do this soon)
- Tinting the chosen link in a nav bar a slightly lighter or darker shade of the main color
- Semitransparent caption boxes laid over photos; see http://css-tricks.com/text-blocks-over-image and www.htmldrive.net/items/show/381/Snazzy-Hover-Effects-Using-CSS3.html
- Semitransparent dialog boxes, modal windows, or tooltips laid over content

TABLE 2.3 RGBA and HSLA browser support

IE	FIREFOX	OPERA	SAFARI	CHROME
Yes, 9+	Yes	Yes	Yes	Yes

Once you have the hue you want, you can then adjust its saturation if you want it duller or brighter, or adjust its lightness if you want it darker or lighter. It's easy to get multiple shades of the same color, or to tweak the color's hue just a little bit in one direction. Once you've worked with HSLA for a while and are more familiar with what each hue value computes out to, it's easier to tell at a glance what color you're going to get when you're glancing through the HSLA values in your style sheets.

The bottom line is this: RGBA and HSLA both have the same browser support and produce the same colors. I'm using HSLA throughout this book because it's more intuitive to me, but if you find RGBA easier, it's perfectly fine to use it instead.

CREATING SEMITRANSPARENT SPEECH BUBBLES

Now that we've gotten all that syntax out of the way, we can switch the speech bubbles' background color from hexadecimal to HSLA notation and make them semitransparent.

The speech bubbles' background color is currently set to #A6DADC. We can figure out the HSLA equivalent using the Rainbow extension. Just open your speech-bubble page in Firefox, and use the Rainbow Inspector to click on the speech bubble background color. It will show you that the HSL value is hsl(182, 44%, 76%). Copy this value, go back to your code editor, and paste it over the current hexadecimal background color:

NOTE: Having spaces after the commas between the three HSL values is completely optional—it works the same way with or without spaces. (I took them out.)

```
blockquote {
   position: relative;
   min-height: 40px;
   margin: 0 0 0 112px;
   padding: 10px 15px 5px 15px;
   -moz-border-radius: 20px;
   -webkit-border-radius: 20px;
   border-radius: 20px;
   border-top: 1px solid #fff;
   background-color: hsl(182,44%,76%);
   word-wrap: break-word;
}
```

If you save and refresh the page after this change, it will look exactly the same. You haven't changed the color yet—just changed the syntax for specifying it.

Now we'll modify this new syntax to make the speech bubbles semitransparent. Change `background-color: hsl(182,44%,76%);` to `background-color: hsla(182, 44%,76%,.5);` . Make sure to add the "a" of "hsla"!

NOTE: I've written the alpha value as ".5," but "0.5" is also perfectly fine.

You also want to change the tail to match. Copy and paste the HSLA value over the hexadecimal value in the `border-color` declaration:

```
blockquote:after {
   content: "\00a0";
   display: block;
   position: absolute;
   top: 20px;
   left: -20px;
   width: 0;
   height: 0;
   border-width: 10px 20px 10px 0;
```

```
    border-style: solid;
    border-color: transparent hsla(182,44%,76%,.5)
                  transparent transparent;
}
```

Save and refresh the page in your browser. You can now see the page background pattern showing through the speech bubbles slightly, as well as each commenter's avatar showing through the little bit of the tail that overlaps each picture (**Figure 2.16**).

FIGURE 2.16 Each speech bubble's background is the same shade of blue, but now semitransparent.

WORKAROUNDS FOR IE

You have a few options for working around the lack of HSLA/RGBA support in IE 8 and earlier.

- Provide a replacement solid background color (in hexadecimal, RGB, or HSL syntax). If you declare the solid background color before the HSLA/RGBA version, using the `background` shorthand property on either both the colors or just the HSLA/RGBA one, IE 8 and earlier will use it and correctly ignore the HSLA/RGBA one. But if you use the `background-color` property instead of `background` to declare the HSLA/RGBA color, IE 7 and 6 won't use the solid color; they try to apply the HSLA/RGBA color and can't, so they display no color at all. In some pages, where the text is still readable even without a background color behind it, this would be acceptable. In those cases where it's not, and where you can't use the `background` shorthand property, you would need to feed IE 7 and earlier the solid background color in a rule that only IE can read. See Chapter 1 for your IE-feeding options.
- Tile a tiny semitransparent PNG image as the background image. This has the advantage over the first option of actually making the background semitransparent, instead of opaque. It works in IE 8 and 7, but not IE 6 and earlier, since those versions don't support alpha-transparent PNGs. To work around this, you could use IE's AlphaImageLoader filter (or one of the many IE transparency scripts

TIP: You can use server-side programming to generate the PNGs for you. See http://leaverou.me/2009/02/bulletproof-cross-browser-rgba-backgrounds for a PHP script that generates them based on the RGBA values in your CSS.

NOTE: The PIE script mentioned earlier can also be used to make RGBA work in IE, but only in limited contexts. See http://css3pie.com/documentation/supported-css3-features for more information.

that makes use of the filter), feed IE 6 a solid background color, or feed IE 6 a GIF or PNG8 image. But all of this is a lot of extra work and could have a big impact on the performance of your pages—the AlphaImageLoader filter is horribly slow and an image is another HTTP request. (Plus, in our case, we couldn't use it on the speech bubbles' tails, since they are just borders and don't have background images.) I don't recommend using a PNG background image unless you don't need to worry about IE 6 and thus won't be providing any workarounds for its lack of alpha-transparent PNG support.

- Use IE's Gradient filter, which works since version 5.5, and allows semitransparent colors (using its own proprietary syntax, of course). Just set both the starting and ending colors to the same color so you don't create the appearance of a gradient.

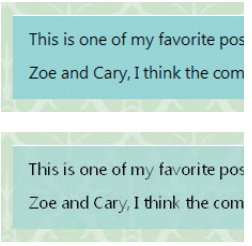

FIGURE 2.17
Before (top) and after (bottom) the Gradient filter is applied in IE 8. With the filter, the background color is semitransparent, but the anti-aliasing of the text is now a little uneven-looking.

I recommend either the first or third option. The third more closely resembles the appearance we're going for, since the background will be semitransparent instead of solid. However, it's worth noting that the Gradient filter can do strange things to the anti-aliasing of the element's text and make it look a little uneven (peek ahead at **Figure 2.17**). You'll have to decide if the less pretty text is worth the more pretty background. Also, adding the filter will make the generated content tail disappear in IE 8 (it never appeared in 7 and 6 to begin with). I can't give you any explanation for this—it's just one of those weird IE bugs.

In this case, I say let's go for the semitransparent background using the filter. Since we don't have rounded corners in IE to create the speech-bubble appearance, I don't mind losing the speech bubble's tail.

We could add the filter right inside the `blockquote` rule—non-IE browsers will just ignore it—but as discussed in Chapter 1, it's always nice to keep hacks and workaround separate from the standard rules. To keep the filters separate, we should either create a separate IE sheet, or use the conditional comments `html` tag trick described in Chapter 1. Let's use the `html` tag trick.

TIP: If you don't want to type all this by hand (I don't blame you), open speech-bubble_final.html from this chapter's exercise files, and copy and paste it from there.

Go to the opening `html` tag of the page, and change it to the following HTML:

```
<!--[if lt IE 7]><html lang="en" class="ie6"><![endif]-->
<!--[if IE 7]><html lang="en" class="ie7"><![endif]-->
<!--[if IE 8]><html lang="en" class="ie8"><![endif]-->
<!--[if IE 9]><html lang="en" class="ie9"><![endif]-->
<!--[if gt IE 9]><html lang="en"><![endif]-->
<!--[if !IE]>--><html lang="en"><!--<![endif]-->
```

Now we can create one rule for IE 5.5, 6 and 7, and another rule for IE 8, since its filter syntax is a little different than that used in earlier versions of IE. Add the IE 7 and earlier rule first:

```
.ie6 blockquote, .ie7 blockquote {
   background: none;
   filter: progid:DXImageTransform.Microsoft.gradient
      (startColorstr=#99A6DADC, endColorstr=#99A6DADC);
   zoom: 1;
}
```

NOTE: The line breaks in the `filter` value are there just to make it easier to read. You can add or remove line breaks within it without affecting how the code functions.

The Gradient filter simply declares a starting and ending color, both the same. The color values look strange, though, don't they? They're not your standard six-digit hexadecimal codes. The first two digits are the alpha transparency value. You can use any hexadecimal value between 00 and FF, with 00 being transparent and FF being opaque. The last six digits are the standard hexadecimal code for a color. So, the color `#99A6DADC` sets the alpha transparency to `99`, the hexadecimal equivalent of the .6 level of transparency we're using in HSLA, and the color to `A6DADC`, the same blue we've been using all along.

In addition to applying the filter, this IE 7 and earlier rule removes the background color, which would override the filter. Also, IE 6 and earlier need to have `hasLayout` triggered on the `blockquote`s to make the filter work, which `zoom: 1;` accomplishes.

NOTE: Understanding `hasLayout` is important when working with IE. If you need a refresher on this strange "property," see Ingo Chao's article "On having layout" at www.satzansatz.de/cssd/onhavinglayout.html.

CONVERTING HSLA AND RGBA TO IE'S GRADIENT FILTER

To use the exact same level of transparency in the IE filter as the HSLA notation, you need to multiply the level of HSLA transparency value, .6 in this case, with 255, and then convert this into hex. Robert Nyman explains how to do this at http://robertnyman.com/2010/01/11/css-background-transparency-without-affecting-child-elements-through-rgba-and-filters.

A much easier way to do this is to use Michael Bester's "RGBa & HSLa CSS Generator for Internet Explorer" at http://kimili.com/journal/rgba-hsla-css-generator-for-internet-explorer. Put in an RGBA or HSLA value and it will automatically convert it to the Gradient filter equivalent.

IE 8 doesn't need the background color removed, as it correctly ignores the HSLA background color on the main `blockquote` rule. It also doesn't need `hasLayout` triggered. But, it does have a slightly different syntax for filter properties. Add the following rule for IE 8:

```
.ie8 blockquote {
   -ms-filter: "progid:DXImageTransform.Microsoft.gradient
   (startColorstr=#99A6DADC, endColorstr=#99A6DADC)";
}
```

The differences in the filter syntax are that it's called `-ms-filter` instead of `filter`, and the value of the `-ms-filter` property is put in quotation marks. This syntax is more in line with the CSS specifications and how other browsers designate their proprietary properties.

Image-free Gradients

We can enhance the speech bubbles' backgrounds even further by giving each a subtle gradient to make them appear more rounded and three-dimensional. CSS3 allows you to create gradients without images, speeding up your development time and decreasing page-loading times, just as our image-free rounded corners can do. CSS-generated gradients also have the advantage of being able to scale with their containers in ways that image gradients can't, making them more versatile.

Unfortunately, CSS3 gradients are still very much in development at the time of this writing; their syntax is laid out only in a W3C editor's draft, not a more finalized working draft or candidate recommendation. Thus, be aware that the syntax for gradients is more likely to change than most of the CSS I'll describe in this book. Still, I think it's fine to add CSS that is a little experimental if you're using it in a very limited manner; non-supporting browsers won't be harmed by its lack, and supporting browsers won't be harmed if the syntax later changes. The (unlikely) worst-case scenario is that the syntax will totally change, making the gradients fail to appear in all browsers. I think I can live with this.

You can create both linear (straight) gradients and radial (circular or elliptical) gradients; we're just going to focus on linear gradients here. There is no `gradient` property; you specify a gradient using the `linear-gradient` or `radial-gradient` function as the *value* for any property that allows an image value, such as `background-image` and `list-style image`