A BOOK APART

Brief books for people who make websites

№ 4

SECOND EDITION

SAMPLE CHAPTER

Ethan Marcotte

RESPONSIVE WEB DESIGN

FOREWORD BY Jeremy Keith

3 FLEXIBLE IMAGES

THINGS ARE looking good so far: we've got a grid-based layout, one that doesn't sacrifice complexity for flexibility. I have to admit that the first time I figured out how to build a fluid grid, I was feeling pretty proud of myself.

But then, as often happens with web design, despair set in. Currently, our page is awash in words, and little else. Actually, *nothing* else: our page is nothing *but* text. Why is this a problem? Well, text reflows effortlessly within a flexible container—and I don't know if you've noticed, but the Internet seems to have one or two of those "image" things lying around. None of which we've incorporated into our fluid grid.

So what happens when we introduce fixed-width images into our flexible design?

GOING BACK, BACK TO MARKUP, MARKUP

To find an answer, let's do another quick experiment: let's drop an image directly into our blog module, and see how our layout

responds. The first thing we'll need to do is to clear some space for it in our markup.

Remember our little `blockquote`, comfortably tucked into our blog article? Well, we've got *way* too much text on this darned page, so let's replace it with an inset image:

```
<div class="figure">
  <p>
    <img src="robot.jpg" alt="" />
    <b class="figcaption">Lo, the robot walks</b>
  </p>
</div>
```

Nothing fancy: an `img` element, followed by a brief but descriptive caption wrapped in a `b` element. I'm actually appropriating the HTML5 figure/figcaption tags as class names in this snippet, which makes for a solidly semantic foundation.

(Sharp-eyed readers will note that I'm using a `b` element for a non-semantic hook. Now, some designers might use a `span` element instead. Me, I like the terseness of shorter tags like `b` or `i` for non-semantic markup.)

With that HTML finished, let's drop in some basic CSS:

```
.figure {
  float: right;
  margin-bottom: 0.5em;
  margin-left: 2.55319149%;  /* 12px / 470px */
  width: 49.14893617%;  /* 231px / 470px */
}
```

We're creating a nice inset effect for our figure. It'll be floated to the right, and will span roughly half the width of our article, or four columns of our flexible grid. Markup: check; style: check. Of course, all this HTML and CSS is for naught if there isn't an actual *image* available.

Now, because I love you (and robots) dearly, not just *any* image will do. And after scouring the web for whole minutes, I found a fantastically imposing robo-portrait (**FIG 3.1**). The beautiful thing about this image (aside from the robot, of course) is that

FIG 3.1: An appropriately botty robot pic, courtesy of Jeremy Noble (http://bkaprt.com/rwd2/12/).

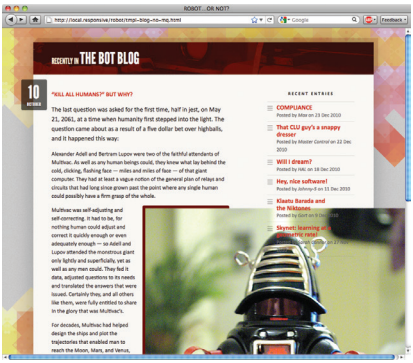

FIG 3.2: Our huge image is huge. Our broken layout is broken.

it's huge. I've cropped it slightly, but I haven't scaled it down at all, leaving it at its native resolution of 655×655. This image is much larger than we know its flexible container will be, making it a perfect case to test how robust our flexible layout is.

So let's drop our oversized image onto the server, reload the page, and—oh. Well. That's pretty much the worst thing on the internet (**FIG 3.2**).

Actually, the result isn't that surprising. Our layout isn't broken *per se*—our flexible container is working just fine, and the proportions of our grid's columns remain intact. But because our image is much wider than its containing `.figure`, the excess

content simply overflows its container, and is visible to the user. There simply aren't any constraints applied to our image that could make it aware of its flexible environment.

FLUID IMAGES

But what if we could introduce such a constraint? What if we could write a rule that prevents images from exceeding the width of their container?

Well, here's the good news: that's very easy to do.

```
img {
  max-width: 100%;
}
```

First discovered by designer Richard Rutter (http://bkaprt.com/rwd2/13/), this one rule immediately provides an incredibly handy constraint for every image in our document. Now, our `img` element will render at whatever size it wants, as long as it's narrower than its containing element. But if it happens to be wider than its container, then the `max-width: 100%` directive forces the image's width to match the width of its container. And as you can see, our image has snapped into place (**FIG 3.3**).

What's more, modern browsers have evolved to the point where they resize the images proportionally: as our flexible container resizes itself, shrinking or enlarging our image, the image's aspect ratio remains intact (**FIG 3.4**).

I hope you're not tired of all this good news because as it happens, the `max-width: 100%` rule can also apply to most fixed-width elements, like video and other rich media. In fact, we can beef up our selector to cover other media-ready elements, like so:

```
img,
embed,
object,
video {
  max-width: 100%;
}
```

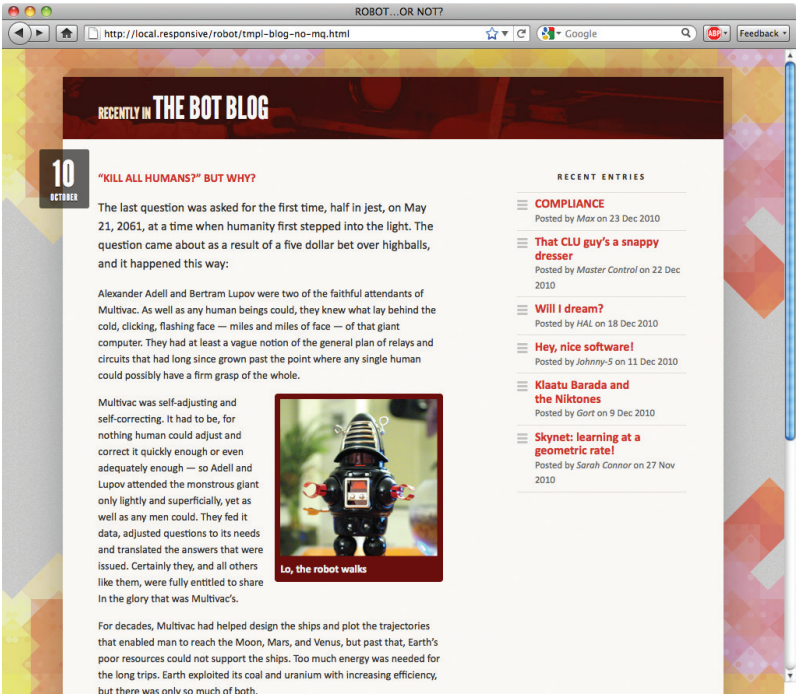

FIG 3.3: Just by including `max-width: 100%`, we've prevented our image from escaping its flexible container. On a related note, I love `max-width: 100%`.

Whether it's a cute little Flash video (**FIG 3.5**), some other embedded media, or a humble `img`, browsers do a fair job of resizing the content to fit a flexible layout. All thanks to our lightweight `max-width` constraint.

So we've cracked the problem of flexible images and media—right? One CSS rule and we're done?

BECAUSE THIS JOB IS NEVER EASY

Time to let the healing begin: we need to work through the pain, the tears, the rending of garments, and talk about a few browser-specific issues around flexible images.

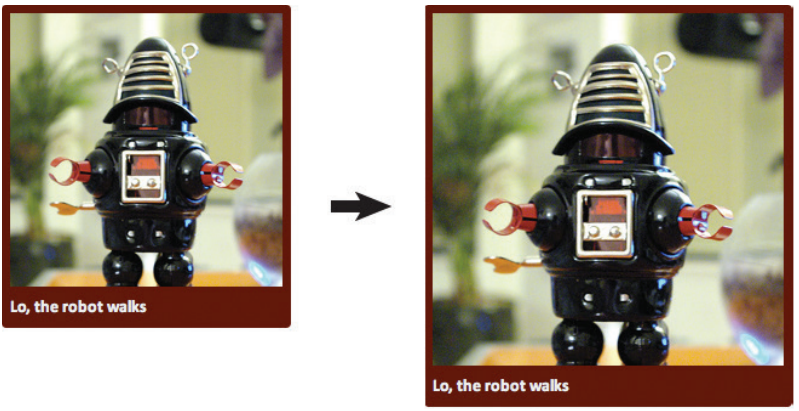

FIG 3.4: Regardless of how wide or small its flexible container becomes, the image resizes proportionally. Magic? Who can say.

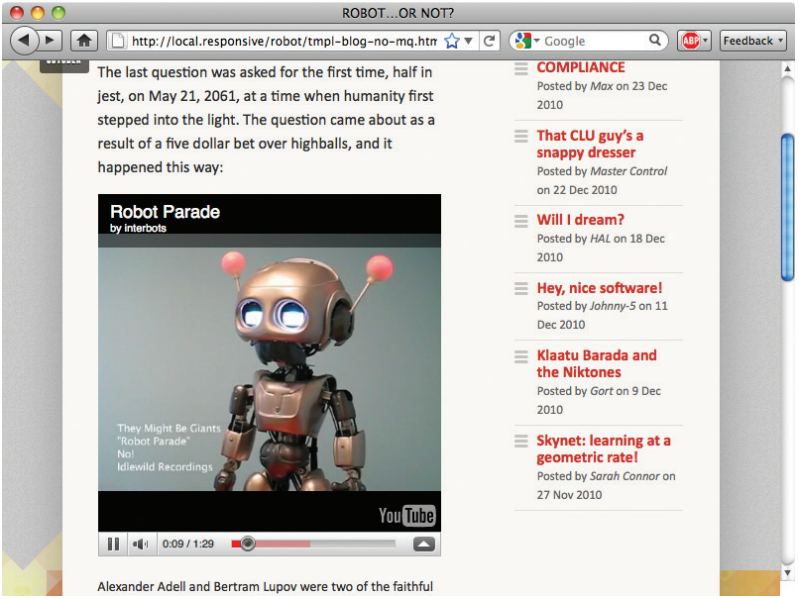

FIG 3.5: Other media play nicely with `max-width: 100%`, becoming flexible themselves. Did I mention I love `max-width: 100%`?

max-width in Internet Explorer

The cold, hard truth is that Internet Explorer 6 and below don't support the `max-width` property. IE7 version and above? Oh, it is positively brimming with support for `max-width`. But if you're stuck supporting the (cough) venerable IE6 or lower, our approach needs refinement.

Now, there are several documented ways to get max-width support working in IE6. Most are JavaScript-driven, usually relying on Microsoft's proprietary `expression` filter to dynamically evaluate the width of an element, and to manually resize it if it exceeds a certain threshold. For an example of these decidedly non-standard workarounds, I'd recommend Cameron Moll's classic blog entry on the subject (http://bkaprt.com/rwd2/14/).

Me? I tend to favor a more lo-fi, CSS-driven approach. Namely, all modern browsers get our max-width constraint:

```
img,
embed,
object,
video {
  max-width: 100%;
}
```

But in a separate IE6-specific stylesheet, I'll include the following:

```
img,
embed,
object,
video {
  width: 100%;
}
```

See the difference? IE6 and lower get `width: 100%`, rather than the `max-width: 100%` rule.

A word of warning: tread carefully here, for these are drastically different rules. Whereas `max-width: 100%` instructs our

images to never exceed the width of their containers, `width: 100%` forces our images to *always match* the width of their containing elements.

Most of the time, this approach will work just fine. For example, it's safe to assume that our oversized `robot.jpg` image will always be larger than its containing element, so the `width: 100%` rule works beautifully.

But for smaller images like thumbnails, or most embedded movies, it might not be appropriate to blindly up-scale them with CSS. If that's the case, then a bit more specificity might be warranted for IE:

```
img.full,
object.full,
.main img,
.main object {
  width: 100%;
}
```

If you don't want the `width: 100%` rule to apply to every piece of fixed-width media in your page, we can simply write a list of selectors that target certain kinds of images or video (`img.full`), or certain areas of your document where you know you'll be dealing with oversized media (`.main img`, `.main object`). Think of this like a whitelist: if images or other media appear on this list, then they'll be flexible; otherwise, they'll be fixed in their stodgy old pixel-y ways.

So if you're still supporting legacy versions of Internet Explorer, a carefully applied `width: 100%` rule can get those flexible images working beautifully. But with that bug sorted, we've still got one to go.

And boy, it's a doozy.

In which it becomes clear that Windows hates us

If you look at our blog module with certain Windows-based browsers, our `robot.jpg` has gone from looking imposing to looking, well, broken (**FIG 3.6**). But this isn't a browser-specific issue as much as a platform-specific one: Windows doesn't scale

FIG 3.6: Seen here in IE6, our robot image has developed some unsightly artifacts. Guess Windows doesn't much care for our flexible images.

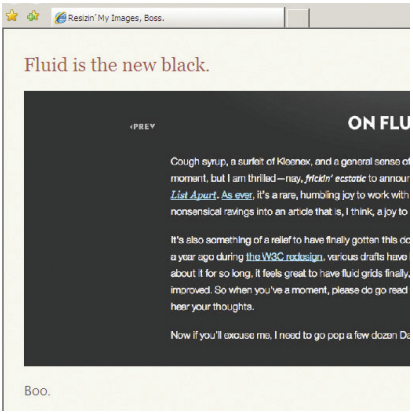

FIG 3.7: In certain Windows-based browsers, the image quickly develops too many artifacts to be readable.

images that well. In fact, when they're resized via CSS, images quickly develop artifacts on Windows, dramatically impacting their quality. And not in a good way.

For a quick test case, I've tossed a text-heavy graphic into a flexible container, and then resized our image with the `max-width: 100%` fix, while IE6 and below receive the `width: 100%` workaround. Now, you'd never actually put this amount of text in an image. But it perfectly illustrates just how badly things can get in IE7 or lower. As you can see, the image looks—if you'll pardon the technical term—downright nasty (**FIG 3.7**).

But before you give up on the promise of scalable, flexible images, it's worth noting that this bug doesn't affect every Windows-based browser. In fact, only Internet Explorer 7 and lower are affected, as is Firefox 2 and lower on Windows. More modern browsers like Safari, Firefox 3+, and IE8+ don't exhibit a single problem with flexible images. What's more, the bug seems to have been fixed in Windows 7, so that's more good news.

So with the scope of the problem defined, surely there's a patch we can apply? Thankfully, there is—with the exception of Firefox 2.

Now, this grizzled old browser was released in 2006, so I think it's safe to assume it isn't exactly clogging up your site's traffic logs. At any rate, a patch for Firefox 2 would require some fairly involved browser-sniffing to target specific versions on a specific platform—and browser-sniffing is unreliable at best. But even if we *did* want to perform that kind of detection, these older versions of Firefox don't have a switch that could fix our busted-looking images.

Internet Explorer, however, *does* have such a toggle. (Pardon me whilst I swallow my pride for this next section title.)

Hail AlphaImageLoader, the conquering hero

Ever tried to get transparent PNGs working in IE6 and below? Chances are good you've encountered `AlphaImageLoader`, one of Microsoft's proprietary CSS filters (http://bkaprt.com/rwd2/15/). There have since been more robust patches created for IE's lack of support for the PNG alpha channel (Drew Diller's DD_belatedPNG library is an old favorite of mine: http://bkaprt.com/rwd2/16/), but historically, if you had a PNG attached to an element's background, you could drop the following rule into an IE-specific stylesheet:

```
.logo {
  background: none;
  filter: progid:DXImageTransform.Microsoft. »
    AlphaImageLoader(src="/path/to/logo.png", »
    sizingMethod="scale");
}
```

This `AlphaImageLoader` patch does a few things. First, it removes the background image from the element, then inserts it into an `AlphaImageLoader` object that sits "between" the proper background layer and the element's content. But the `sizingMethod` property (http://bkaprt.com/rwd2/17/) is the clever bit, dictating whether the `AlphaImageLoader` object should `crop` any parts of the image that overflow its container, treat it like a regular `image`, or `scale` it to fit it within its containing element.

I can hear you stifling your yawns by now: after all, what does an IE-specific PNG fix have to do with our broken image rendering?

Quite a bit, as it turns out. At one point I discovered that applying `AlphaImageLoader` to an image dramatically improves its rendering quality in IE, bringing it up to par with, well, every other browser on the planet. Furthermore, by setting the `sizingMethod` property to scale, we can use our `AlphaImageLoader` object to create the illusion of a flexible image.

So I whipped up some JavaScript to automate that process. Simply download the script (available at http://bkaprt.com/rwd2/18/) and include it on any page with flexible images; it will scour your document to create a series of flexible, high-quality `AlphaImageLoader` objects.

And with that fix applied, the difference in our rendered images is noticeable (**FIG 3.8**): in our example we've gone from an impossibly distorted image to an immaculately rendered one. And it works wonderfully in a flexible context.

(It's worth mentioning that many of Microsoft's proprietary filters, and `AlphaImageLoader` in particular, have some performance overhead associated with them—Stoyan Stefanov covers the pitfalls in more detail on the YUI blog: http://bkaprt.com/rwd2/19/. What does this mean for you? Just be sure to test the fix thoroughly on your site, gauge its effect on your users, and evaluate whether or not the improved rendering is worth the performance tradeoff.)

With the `max-width: 100%` fix in place (and aided by our `width: 100%` and `AlphaImageLoader` patches), our inset image is resizing beautifully across our target browsers. No matter the

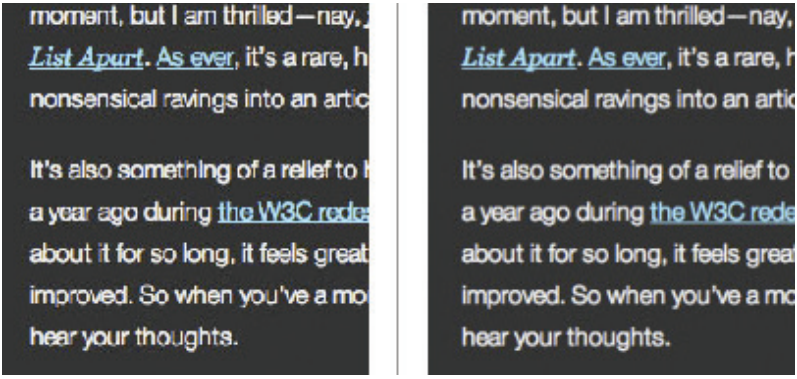

FIG 3.8: Our image is now perfectly legible, and resizing wonderfully. A dab of `AlphaImageLoader`'ll do ya.

size of the browser window, our image scales harmoniously along with the proportions of our flexible grid.

But what about images that aren't actually in our markup?

** Read the rest of this chapter and more when you buy the book!*