7
CSS3 Transitions, Transformations, and Animations

In the last two chapters we looked at some of the new features and functionality that CSS3 provides. However, until now, everything we have looked at has been static. But CSS3 can do more.

At present, chances are, if you need to animate elements on a web page you'll either write your own JavaScript to perform the required action or turn to a popular JavaScript library like jQuery to do the heavy lifting. However, someone involved with CSS3 clearly has issues with JavaScript's ubiquity in this area and they're looking to encroach on JavaScript's dominance. While CSS3 isn't likely to usurp jQuery or the like anytime soon, it's perfectly capable of things like smoothing transitions (for example, on mouse hover) and moving elements around the screen. This is great news for us, as it means for the growing number of devices sporting modern browsers (recent smart phones for example), we can use CSS to provide animations rather than relying on JavaScript. The upshot: you can probably scratch 'learn how to animate elements with jQuery' off the 'to do' list as we can now do all that fun stuff in pure CSS. As ever, these CSS3 features don't break anything for browsers lacking the features; they'll just skip over the rules they don't understand like they weren't there.

In this chapter, we'll cover:

- What CSS3 transitions are and how we can use them
- How to write a CSS3 transition and its shorthand syntax
- CSS3 transition timing functions (ease, cubic-bezier, and so on)

- Fun transitions for responsive web sites
- What CSS3 transformations are and how we can use them
- Understanding different 2D transformations (scale, rotate, skew, translate, and so on)
- Dabbling with 3D transformations
- Animating with CSS3 (using keyframes)

What CSS3 transitions are and how we can use them

When styling hyperlinks in CSS, it's common practice to create a hover state; an obvious way to make users aware that the item they are hovering over is a link. They're of less relevance to the growing number of touch screen devices but for everyone else, they're a great and simple interaction between website and user.

Traditionally, using only CSS, hover states are an on/off affair. There is one state as the default, that instantly changes to a different state on hover. However, CSS3 transitions, as the name implies, allow us to transition between one state and another. It's not specific to hover states but let's start there.

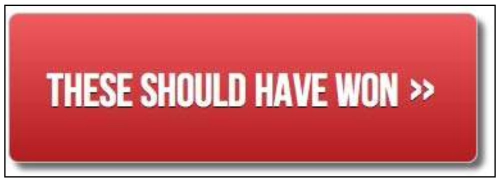
In the previous chapter, we created a CSS3 button with a red gradient background. This is the CSS3 used (with the additional vendor prefixes removed for brevity):

```
#content a {
    text-decoration: none;
    font: 2.25em /* 36px ÷ 16 */ 'BebasNeueRegular';
    background-color: #b01c20;
    border-radius: 8px;
    color: #ffffff;
    padding: 3%;
    float: left;
    background: 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;
}
```

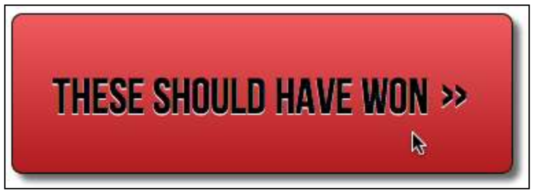
Let's add a hover state:

```
#content a:hover {
    border: 1px solid #000000;
    color: #000000;
    text-shadow: 0px 1px white;
}
```

And here are the two states, first the default:



And then here's the hover state:



It's a simple change of text, text-shadow, and border color on hover. So, as you might imagine, with the current CSS, hovering the mouse over snaps from the first state (white text) button to the second (black text); it's an on/off affair. Let's add a little CSS3 magic to our first rule:

```
#content a {
    /*…existing styles…*/
    transition: all 1s ease 0s;
}
```

Now when we hover over the button, the text, text-shadow, and border color all transition smoothly from one to the other. You'll notice the transition is applied to the original element, not the hover state. This is so that different states such as `:active` can also have different styles set and enjoy the transition. So remember, the transition declaration is added to the element it transitions *away* from. But how do transitions actually work?

The properties of a transition

A **transition** can be declared using up to four properties or a single shorthand declaration including all four:

- `transition-property`: the name of the CSS property to be transitioned (such as `background-color`, `text-shadow`, or `all` to transition every possible property).
- `transition-duration`: the length of time over which the transition should occur (defined in seconds, for example `.3s`, `2s`, or `1.5s`).
- `transition-timing-function`: how the transition changes speed during the duration (for example `ease`, `linear`, `ease-in`, `ease-out`, `ease-in-out`, or `cubic-bezier`).
- `transition-delay`: an optional value to determine a delay before the transition commences. Alternatively, a negative value can be used to commence a transition immediately but part way through its transition 'journey'.

Used separately, the various transition properties can be used to create a transition as follows:

```
#content a {
    ...(more styles)...
    transition-property: all;
    transition-duration: 1s;
    transition-timing-function: ease;
    transition-delay: 0s;
}
```

The transition shorthand property

As we've already seen however, we can roll these individual declarations into a single, shorthand version:

```
transition: all 1s ease 0s;
```

One important point to note when writing the shorthand version is that the first timing value given is always taken to be the `transition-duration`. The second timing value is taken to be the `transition-delay`.

As ever, it's important to use vendor prefixes. For example, a stack of vendor-prefixed versions of the prior shorthand declaration would be as follows:

```
-o-transition: all 1s ease 0s;
-ms-transition: all 1s ease 0s;
-moz-transition: all 1s ease 0s;
-webkit-transition: all 1s ease 0s;
transition: all 1s ease 0s;
```

We've placed the non-prefixed 'official' version last so it will overwrite the others when browsers have fully implemented the standard.

Limitations of transitions

There are some caveats to using transitions; some properties can't be transitioned, despite the specifications (even the latest editor's draft at `http://dev.w3.org/csswg/css3-transitions/`) saying it should be possible. For example, the `background-gradient` property. However, you can, in theory, transition all these properties (`http://www.w3.org/TR/css3-transitions/#properties-from-css-`).

Transition different properties over different periods of time

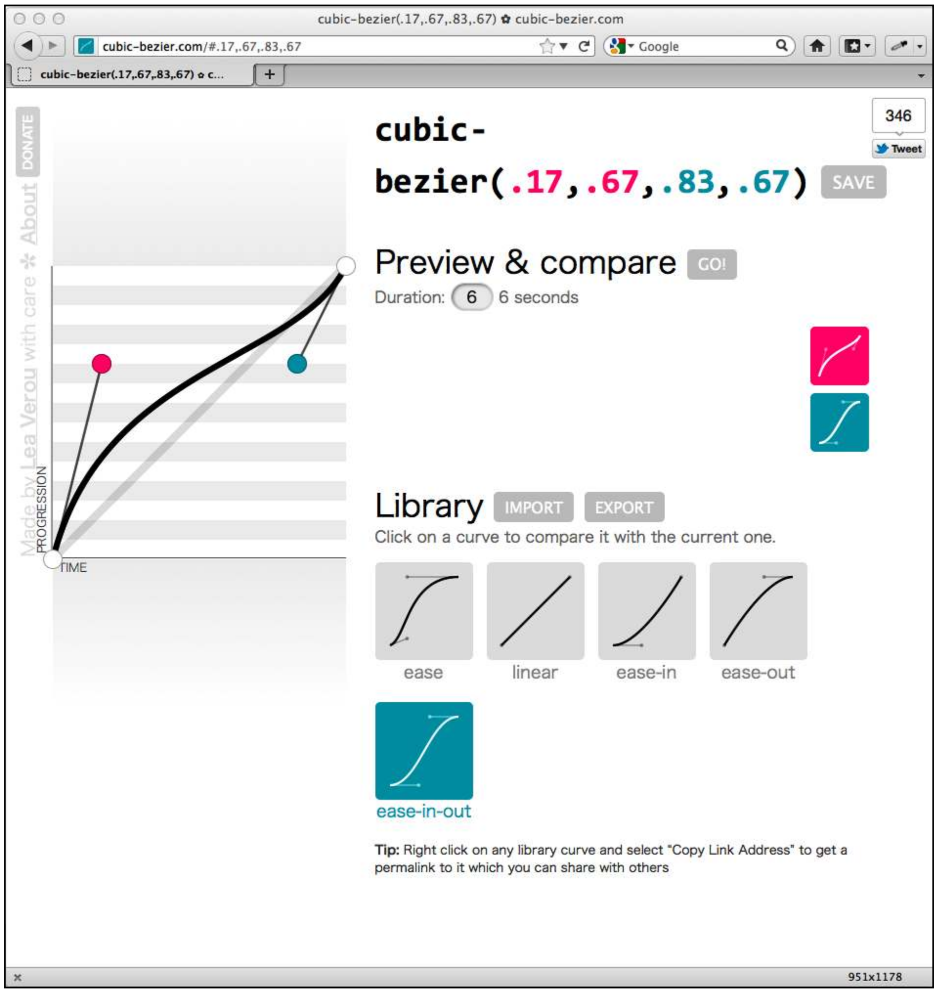
Where a rule has multiple properties declared you don't have to transition all of them in the same way. Consider this rule:

```
#content a {
    ...(more styles)...
    transition-property: border, color, text-shadow;
    transition-duration: 2s, 3s, 8s;
}
```

Here we have specified with the transition-property that we'd like to transition the border, color, and text-shadow. Then with the transition-duration declaration, we are stating that the border should transition over 2 seconds, the color over 3 seconds, and the text-shadow over 8 seconds. The comma-separated durations match the comma-separated order of the transition properties.

Understanding timing functions

Most of the transition properties are self-explanatory. We've covered the list of properties that can be (or should be!) transitioned. Durations and delays are set with seconds (for example `2s`) so they're simple enough to understand but the one property that can cause some head scratching is the timing functions. Just what do `ease`, `linear`, `ease-in`, `ease-out`, `ease-in-out`, and `cubic-bezier` actually do? Each of them is actually a cubic-bezier-curve—essentially the same as an easing function. I realize that perhaps doesn't mean much to you either. So… this is one of those situations where words (and certainly this author's power to wield them well enough) struggle to offer a satisfactory explanation—much like if you have to give your other half a satisfactory explanation for why you've forgotten their birthday! Instead, I recommend you head over to `http://cubic-bezier.com/`.


cubic-bezier(.17,.67,.83,.67) ✿ cubic-bezier.com
cubic-bezier.com/#.17,.67,.83,.67
Google
DONATE
About
Made by Lea Verou with care
PROGRESSION
TIME
346
Tweet
SAVE
GO!
IMPORT
EXPORT
ease
linear
ease-in
ease-out
ease-in-out
951x1178

cubic-bezier(.17,.67,.83,.67)

Preview & compare

Duration: 6 6 seconds

Library

Click on a curve to compare it with the current one.

Tip: Right click on any library curve and select "Copy Link Address" to get a permalink to it which you can share with others

This site lets you compare timing functions and see the difference each one makes. However, even if you can write your own cubic-bezier curves blindfolded (while also counting backwards from a thousand in Mandarin), the likelihood is, for most practical situations, it makes little difference. Here's why…

Like any enhancement, it's necessary to employ transition effects subtly. For 'real world' implementations, transitions that occur over too great a period of time tend to make a site 'feel' slow. For example, navigation links that take 5 seconds to transition are going to frustrate, rather than 'Wow!' your users. Therefore, unless there is a compelling reason to do so, using the default transition (ease) over a short interval (a maximum of 1 second is my own preference) is often best.

Fun transitions for responsive web sites

Once you become a responsive web design junkie, you'll find yourself constantly resizing the browser window on websites you visit to see if it's responsive. Keep in mind this habit infuriates 'normal' people, so best only do it in private.

A great website I often visit that discusses CSS techniques is Chris Coyier's excellent `http://css-tricks.com`. After a re-design I happened to resize the browser window and smiled knowingly as the different on-screen elements whizzed about the screen. What magic had Chris employed to bring this effect about? Something similar to this:

```
* {
   transition: all 1s;
}
```

Here, we are using the CSS universal selector `*` to select everything and then setting a transition on `all` elements over 1 second (`1s`). As we have omitted to specify the timing function, `ease` will be used by default and there will be no delay as again, a default of none is assumed if an alternative value is not specifically added. The effect? Well, most things (links, hover states, and the like) behave as you would expect. However, because *everything* transitions, it also includes any rules within media queries, so as the browser window is resized, elements sort of flow from one state to the next. Is it essential? Absolutely not! Is it fun to watch and play around with? Certainly!

CSS3 2D transformations

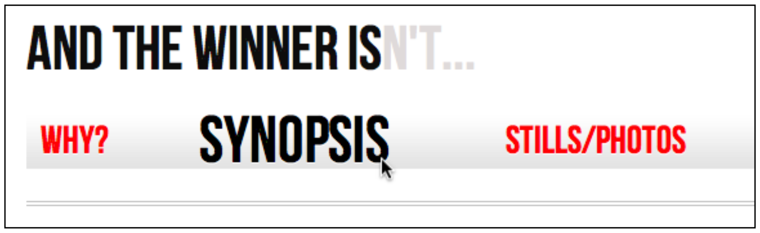
Despite sounding similar, CSS **transformations** (both 2D and 3D variants) are entirely different to CSS transitions. Think of it like this: transitions smooth the change from one state to another, while transformations are defining what the element will become. My own (admittedly childish) way of remembering it is like this:

Imagine a Transformer robot like Optimus Prime. He's a robot that becomes something else (transforms) over a period of time (the transition) into a truck.

In case that tangent muddied the waters further (or you don't have a clue who Optimus Prime is) let's just dig in. Let's add a 2D transition to the hover state of the navigation links on the **AND THE WINNER ISN'T** site:

```
nav ul li a:hover {
    transform: scale(1.7);
}
```

Now, in a modern browser, hovering over a link produces this effect:



We've told the browser that when this element is hovered over, we want the element to scale to 1.7 times its original value.

Now, if you're attempting to add this rule to an element in Safari, be aware that it requires the main element to be displayed as a block. For example:

```
nav ul li a {
    height: 42px;
    text-decoration: none;
    text-transform: uppercase;
    color: black;
    text-shadow: 0 1px 0 hsla(0, 0%, 100%, 1.0);
    font: 1.875em/42px 'BebasNeueRegular';
    display: block;
}
```

Otherwise nothing happens, which is, you know, rubbish.

What can we transform?

There are two groups of CSS3 transforms available: 2D and 3D. 2D variants are far more widely implemented, browser wise, and certainly easier to write so let's look at those first. The CSS3 2D Transforms Module allows us to use the following transformations:

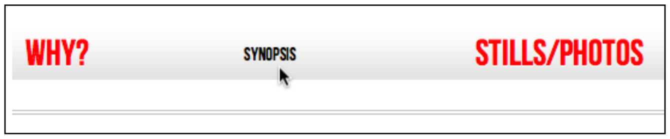
- `scale`: used to scale an element (larger or smaller)
- `translate`: move an element on the screen (up, down, left, and right)
- `rotate`: rotate the element by a specified amount (defined in degrees)
- `skew`: used to skew an element with its X and Y co-ordinates
- `matrix`: allows you to move and shape transformations with pixel precision

Let's try each of these and see what we can achieve.

scale

We've already looked at this transform above. However, besides the positive values we've already used, it's worth knowing that by using values below 1, we can shrink elements; the following will shrink the element to half its size:

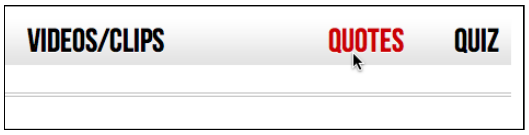
```
transform: scale(0.5);
```



translate

```
transform: translate(40px, 0px);
```

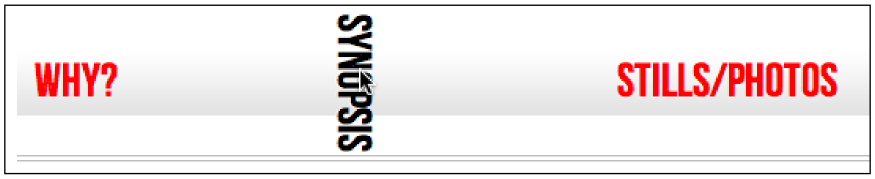
`translate` tells the browser to move the element by an amount, defined in either pixels or percentages. The syntax is applied first from the left to the right (40px here) and then from the top to the bottom (`0px` here so it stays in line with the other navigation elements). Positive values given within parentheses move the element right or down; negative values move it left or up. So using this declaration on our navigation hover state results in this—our link shifting 40 pixels to the right when hovered over:



rotate

```
transform: rotate(90deg);
```

`rotate` allows you to rotate an element. In this example, we've amended the hover link to rotate 90 degrees. In the browser, here's what happens:



The value in parentheses should always be in degrees (for example, `90deg`). That doesn't stop you going crazy—you can make elements spin by specifying a value like the following:

```
transform: rotate(3600deg);
```

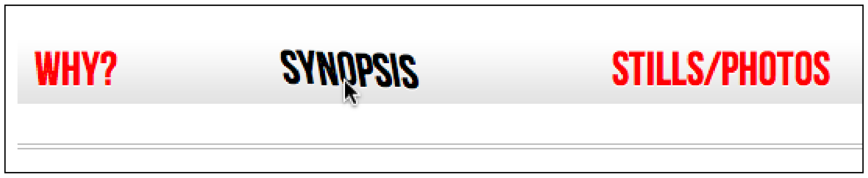
This will rotate the element 10 times in a complete circle. Practical uses for this particular value are few and far between but you know, if you ever find yourself designing websites for a windmill company it may come in handy!

skew

If you've spent any time working in Photoshop, you'll have a good idea what `skew` will do. It allows an element to be skewed on either or both of its axes.

```
transform: skew(10deg, 2deg);
```

Setting this on the hover link produces the following effect on hover:



The first value is the skew applied to the X axis (in our example, `10deg`), while the second (`2deg`) is for the Y axis. Omitting the second value means any value will be applied to the X axis (horizontal). For example:

```
transform: skew(10deg);
```

This is perfectly valid but will only apply skew to the X axis. Values should always be given in degrees. While positive values always apply clockwise, using negative values will rotate the element counter-clockwise.

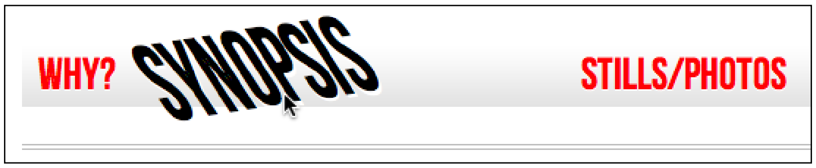
matrix

So, on the subject of over-rated films. What's that? You want to know about the CSS3 `matrix`, not the film? Oh, okay…

The `matrix` transform syntax looks scary:

```
transform: matrix(1.678, -0.256, 1.522, 2.333, -51.533, -1.989);
```

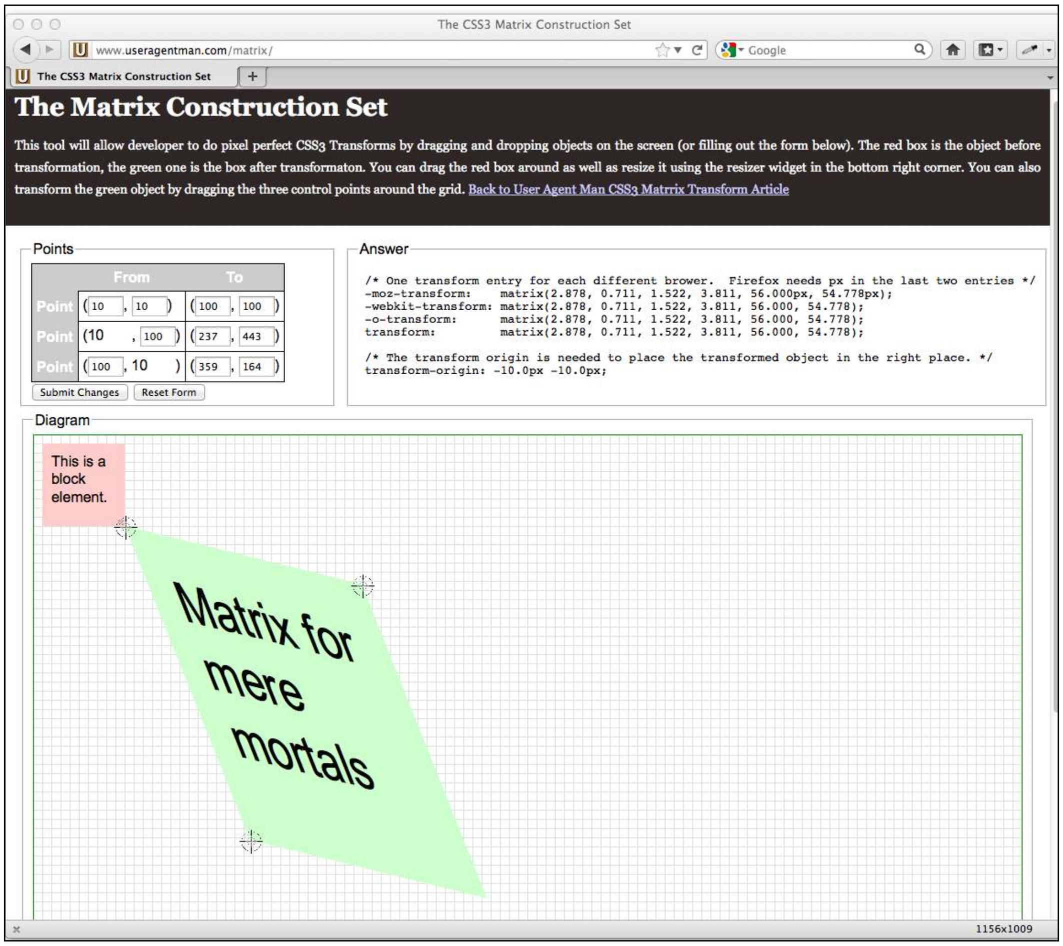
It essentially allows you to combine a number of other transforms (scale, rotate, skew, and so on) into a single declaration. The above declaration results in the following effect in the browser:



Now, I like a challenge like the best of them (unless, you know, it's sitting through Moulin Rouge) but I'm sure we can agree that syntax is a bit testing. It gets worse when you look at the specification and realize that it involves mathematics knowledge to fully understand: `http://www.w3.org/TR/css3-2d-transforms/#cssmatrix-interface`.

Matrix transformations for cheats and dunces

I'm not a mathematician by any stretch of the imagination so when faced with the need to create a matrix based transformation, I cheat. If your mathematical skills are also found wanting, I'd suggest heading over to `http://www.useragentman.com/matrix/`.



The CSS3 Matrix Construction Set

www.useragentman.com/matrix/

The Matrix Construction Set

This tool will allow developer to do pixel perfect CSS3 Transforms by dragging and dropping objects on the screen (or filling out the form below). The red box is the object before transformation, the green one is the box after transformaton. You can drag the red box around as well as resize it using the resizer widget in the bottom right corner. You can also transform the green object by dragging the three control points around the grid. Back to User Agent Man CSS3 Matrrix Transform Article

Points

	From		To	
Point	10	10	100	100
Point	10	100	237	443
Point	100	10	359	164

Answer

```
/* One transform entry for each different brower.  Firefox needs px in the last two entries */
-moz-transform:    matrix(2.878, 0.711, 1.522, 3.811, 56.000px, 54.778px);
-webkit-transform: matrix(2.878, 0.711, 1.522, 3.811, 56.000, 54.778);
-o-transform:      matrix(2.878, 0.711, 1.522, 3.811, 56.000, 54.778);
transform:         matrix(2.878, 0.711, 1.522, 3.811, 56.000, 54.778);

/* The transform origin is needed to place the transformed object in the right place. */
transform-origin: -10.0px -10.0px;
```

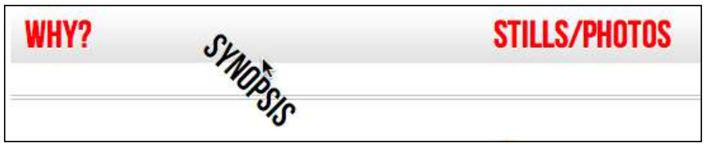
The **Matrix Construction Set** website allows you to drag and drop the element exactly where you want it and then includes good ol' copy and paste code (including vendor-prefixes) for your CSS file.

transform-origin property

Alongside the aforementioned transformations, you can use the `transform-origin` property to amend the point from which transforms are applied:

```
transform: rotate(45deg);
transform-origin: 20% 20%;
```

Setting this on our navigation links results in the following when hovered over:



The `transform-origin` property comes in useful as by default, transformations are applied to the center of an element. This provides a handy means of offsetting it and can produce some great results.

Full information on the `transform-origin` property can be found here: `http://www.w3.org/TR/css3-2d-transforms/#transform-origin-property`

That covers the essentials of 2D transforms. They are far more widely implemented in the browser landscape than their 3D brethren and when used sensibly, provide a light-weight means of providing visual flourishes to reward users with modern browsers.

Read the full specification on **CSS3 2D Transforms Module Level 3** here: `http://www.w3.org/TR/css3-2d-transforms/`

Dabbling in CSS3 3D transformations

Although already supported by Webkit browsers (Safari and Chrome) and Firefox 10+, CSS3 3D transforms won't be supported in IE until version 10. However, despite a lack of support in 'desktop' browsers, thanks to their origin in Webkit, they are well supported in Android (v3 onwards) and iOS (all versions).

Suffice to say, from this point on, you'll be best off checking your results in a Webkit based browser such as Chrome or Safari (unless, of course, you're reading this at a time when your browser of choice *does* support 3D transformations).

Now, we're just going to dabble in 3D transformations here. They're a vast subject and the possibilities are virtually infinite. I imagine by the time they are supported widely, most of us will reach for them to create Carousel effects, rather than relying on JavaScript solutions from the likes of jQuery. However, until then, let's just open the lid and take a peek at what's possible.

Let's imagine we're making a simple quiz for the **AND THE WINNER ISN'T** website. It will be composed of images of movie posters and you have to guess whether they are considered 'Hot or Not' by the world's most respected film critic (yep, that's me). Hovering over the images (or tapping on a touch screen) will reveal the answer.

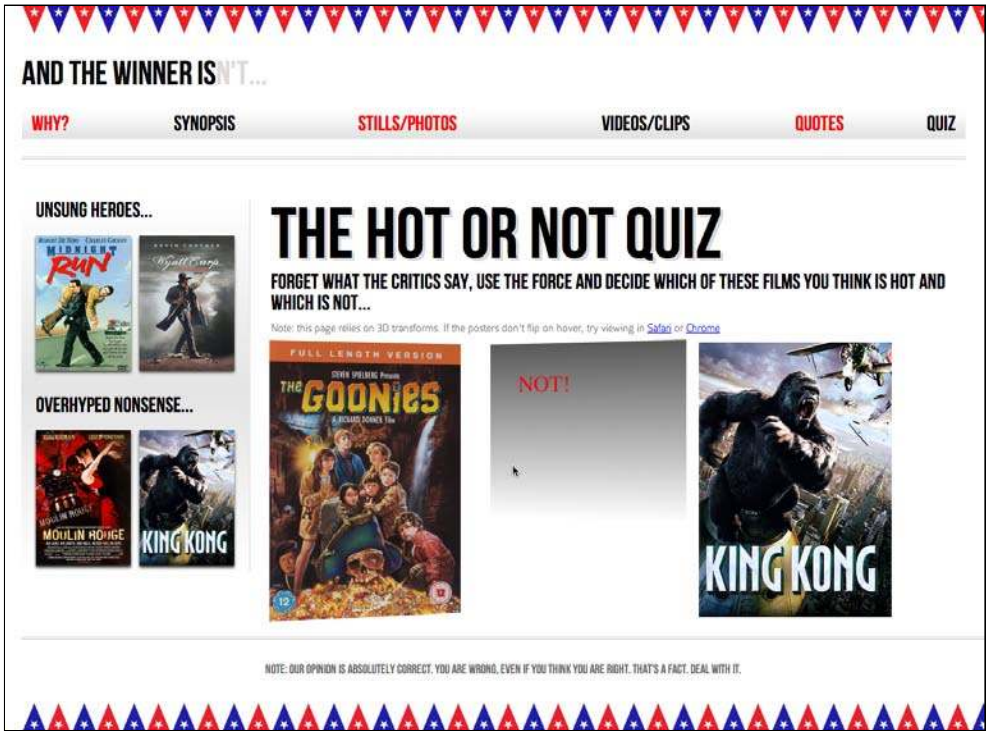
Here's the relevant section of markup; note that I've omitted the repetition of the markup for each image as they follow exactly the same format:

```
<section class="Qcontainer">
  <div class="film">
    <div class="face front">
      <img src="img/goonies.jpg" alt="The Goonies" />
    </div>
    <div class="face back"><h5>HOT!</h5></div>
  </div>
</section>
```

And now here's the CSS. Note, as Webkit is the browser with the greatest support for 3D transformations, the declarations here all use that specific vendor prefix. As ever, when implementing in the real world, vendor-prefixes are your friend.

```
.Qcontainer {
    height: 100%;
    width: 28%;
    position: relative;
    -webkit-perspective: 800;
    float: left;
    margin-right: 2%;
}
.film {
    width: 100%;
    height: 15em;
    -webkit-transform-style: preserve-3d;
    -webkit-transition: 1s;
}
.Qcontainer:hover .film {
    -webkit-transform: rotateY(180deg);
}
.face {
    position: absolute;
    -webkit-backface-visibility: hidden;
}
.back {
    width: 66%;
    height: 127%;
    -webkit-transform: rotateY(180deg);
    background: #3b3b3b;
    background: -webkit-linear-gradient(top,
                               rgba(0,0,0,0.65) 0%,
                               rgba(0,0,0,0) 100%);
    padding: 15%;
}
```

With that in place, hovering over the relevant image makes the poster flip and the simple **HOT** or **NOT** answer is revealed.



THE HOT OR NOT QUIZ

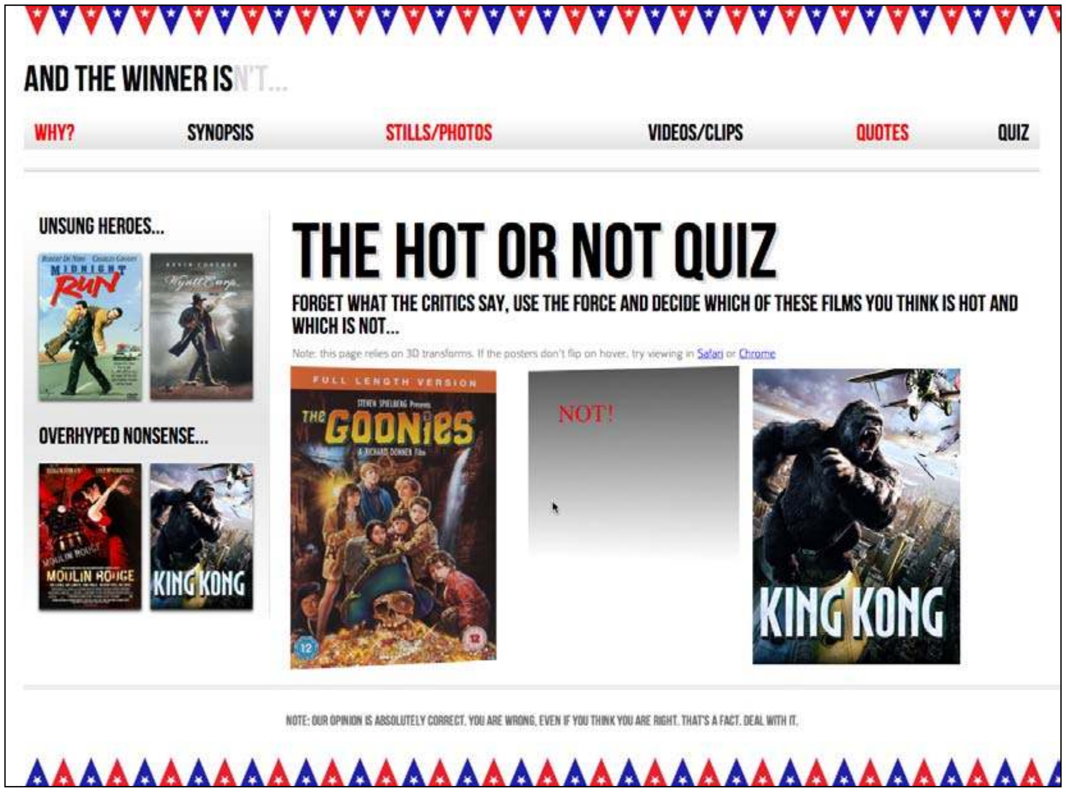
FORGET WHAT THE CRITICS SAY, USE THE FORCE AND DECIDE WHICH OF THESE FILMS YOU THINK IS HOT AND WHICH IS NOT...

Note: this page relies on 3D transforms. If the posters don't flip on hover, try viewing in Safari or Chrome

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

Breaking down the 3D effect

Let's go through the code to understand how this effect is achieved.



THE HOT OR NOT QUIZ

FORGET WHAT THE CRITICS SAY, USE THE FORCE AND DECIDE WHICH OF THESE FILMS YOU THINK IS HOT AND WHICH IS NOT...

Note: this page relies on 3D transforms. If the posters don't flip on hover, try viewing in Safari or Chrome

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

The first important point is to set the perspective on the parent element. This activates 3D space:

```
.Qcontainer {
    height: 100%;
    width: 28%;
    position: relative;
    -webkit-perspective: 200;
    float: left;
    margin-right: 2%;
}
```

The larger this perspective value, the greater the virtual depth of 3D space from your viewing point. Therefore, for a subtler 3D effect, increase the value. For a more dramatic effect, decrease it.

The next noteworthy point:

```
.film {
    width: 100%;
    height: 15em;
    -webkit-transform-style: preserve-3d;
    -webkit-transition: 1s;
}
```

The first perspective declaration added to the `.Qcontainer` class only applies to the first direct descendent (the `div` with a class of `.film` in this example). Therefore, to pass on the parent's perspective we use the `preserve-3d` value.

Now, we'll add a rule to flip the `.film` div when the `.Qcontainer` section is hovered over:

```
.Qcontainer:hover .film {
    -webkit-transform: rotateY(180deg);
}
```

The next rule deals with hiding the opposite side of the poster when it's flipped:

```
.face {
    position: absolute;
    -webkit-backface-visibility: hidden;
}
```

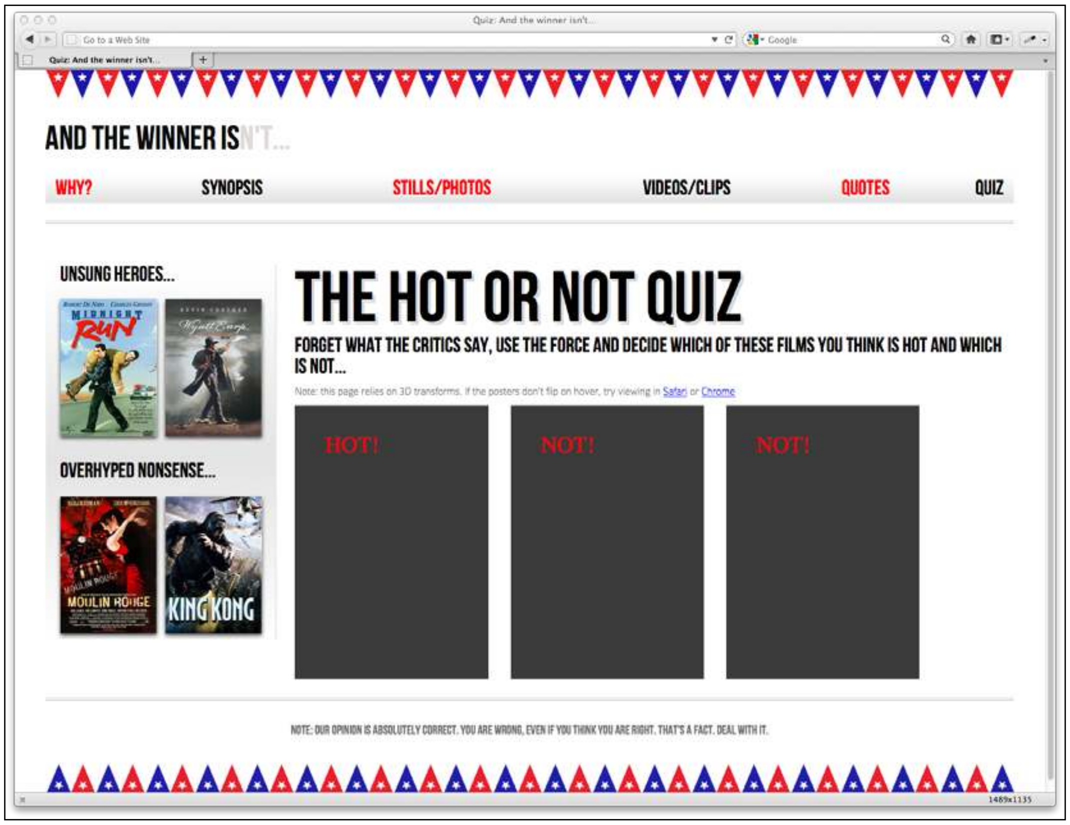
The absolute positioning on the `.face` is necessary to position it on top of the `.back` DIV:

```
.back {
    width: 66%;
    height: 127%;
    -webkit-transform: rotateY(180deg);
    background: #3b3b3b;
    background: -webkit-linear-gradient(top,
                                rgba(0,0,0,0.65) 0%,
                                rgba(0,0,0,0) 100%);
    padding: 15%;
}
```

Finally, we also add a simple `rotateY` on the `.back` DIV. Without this, the `.back` DIV effectively shows through the front.

And that's all there is to it. Now, hovering over any of the posters flips them in a rather dramatic fashion.

However, for any non-Webkit browsers the page functionality is decidedly lame:


Quiz: And the winner isn't...
AND THE WINNER ISN'T...
WHY?
SYNOPSIS
STILLS/PHOTOS
VIDEOS/CLIPS
QUOTES
QUIZ
UNSUNG HEROES...
OVERHYPED NONSENSE...
THE HOT OR NOT QUIZ
FORGET WHAT THE CRITICS SAY, USE THE FORCE AND DECIDE WHICH OF THESE FILMS YOU THINK IS HOT AND WHICH IS NOT...
Note: this page relies on 3D transforms. If the posters don't flip on hover, try viewing in Safari or Chrome
HOT!
NOT!
NOT!
NOTE: OUR OPINION IS ABSOLUTELY CORRECT. YOU ARE WRONG, EVEN IF YOU THINK YOU ARE RIGHT. THAT'S A FACT. DEAL WITH IT.

Well, we can provide an acceptable fallback for non-Webkit browsers with a little CSS of old:

```
.front {
    z-index: 5;
}
.Qcontainer:hover .front {
    z-index: 0;
}
```

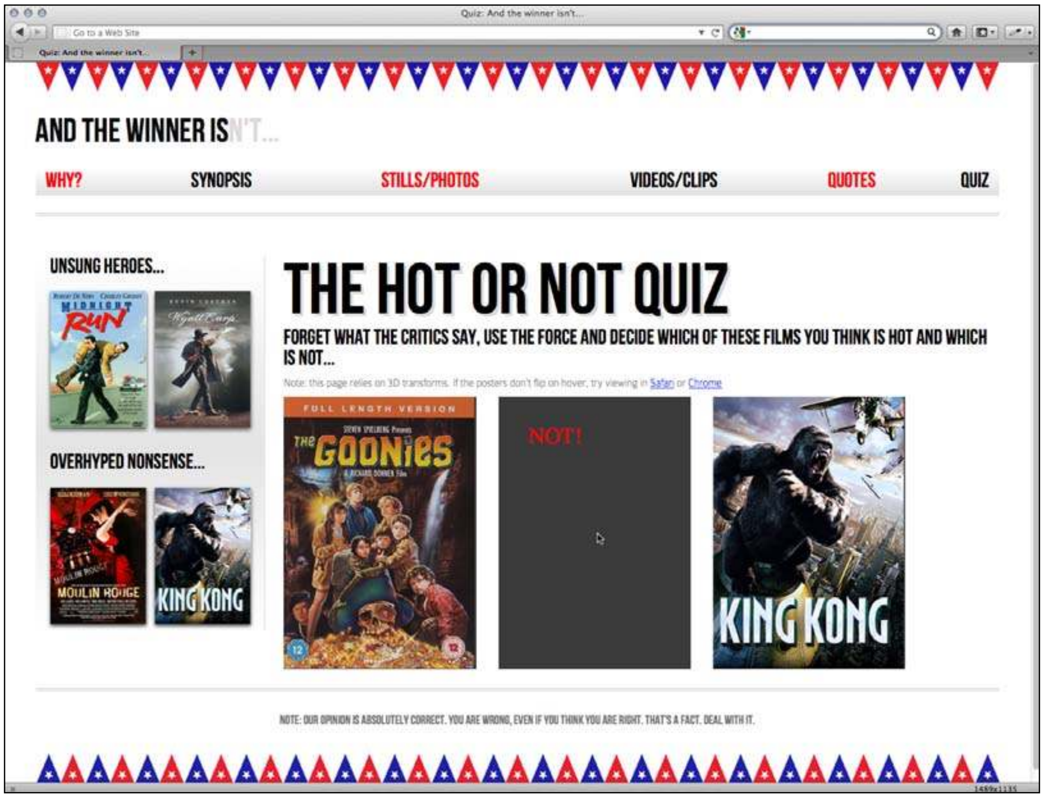
First, we set a `z-index` of `5` on the `.front` DIV so that it sits above the `.back` DIV by default:

```
.front {
    z-index: 5;
}
```

Now, when the `.Qcontainer` section is hovered over, we'll set the `z-index` to `0` so it once more sits behind the `.back` DIV:

```
.Qcontainer:hover .front {
    z-index: 0;
}
```

Now we get a functional question and answer functionality in non-3D transform capable browsers, *sans* the fancy 3D effect.


Quiz: And the winner isn't...
AND THE WINNER ISN'T...
WHY?
SYNOPSIS
STILLS/PHOTOS
VIDEOS/CLIPS
QUOTES
QUIZ
UNSUNG HEROES...
OVERHYPED NONSENSE...
THE HOT OR NOT QUIZ
FORGET WHAT THE CRITICS SAY, USE THE FORCE AND DECIDE WHICH OF THESE FILMS YOU THINK IS HOT AND WHICH IS NOT...
NOT!
NOTE: OUR OPINION IS ABSOLUTELY CORRECT. YOU ARE WRONG, EVEN IF YOU THINK YOU ARE RIGHT. THAT'S A FACT. DEAL WITH IT.

3D transformations not ready for prime time

In my experience, at present, many of the 3D transforms don't play happily with percentage sizes (for example, amending the viewport width with the prior example makes things misbehave severely). So there's often quite a bit of tweaking to be done to make them play happily within a responsive layout. Furthermore, as support is currently so limited, 3D transformations seldom offer the most robust solution when you're building a cross-browser site. So for now, I still err towards jQuery or similar for this kind of functionality.

The possibilities of CSS 3D transforms are, however, extremely promising and when browser support is extended, they offer the opportunity to move many of the fancy effects we currently rely on JavaScript for, to be moved within our stylesheets.

> Read about the latest W3C developments on **CSS 3D Transforms** at `http://dev.w3.org/csswg/css3-3d-transforms/`

Animating with CSS3

If you've ever worked with Flash, you'll have an instant advantage when working with CSS3 animations. CSS3 employs animation keyframing conventions found in Flash and other timeline based applications.

Animations are also more widely implemented than 3D transforms. They are supported in Firefox 5+, Chrome, Safari 4+, Android (all versions), iOS (all versions), and due to be incorporated into Internet Explorer 10.

There are two components to a CSS3 animation; firstly a keyframes declaration and then using that keyframe declaration in an animation property. Let's take a look.

In the previous section we made a simple flip effect for films that I consider **HOT** or **NOT**. Well, the text on the reveal is pretty dull, so let's add a nice pulsing effect to the answer that's revealed after the poster flips.

Firstly the keyframe rule:

```
@keyframes warning {
  0% {
    text-shadow: 0px 0px 4px #000000;
  }
  50% {
    text-shadow: 0 0 20px #000000;
  }
  100% {
    text-shadow: 0px 0px 4px #000000;
  }
}
```

I'm using the non-prefixed version of the code here so if things aren't happening you'll probably need to add a full vendor-prefixed stack (@-webkit-keyframes for example).

Let's break this down:

```
@keyframes warning {
  0% {
    text-shadow: 0px 0px 4px #000000;
  }
  50% {
    text-shadow: 0 0 20px #000000;
  }
  100% {
    text-shadow: 0px 0px 4px #000000;
  }
}
```

First, we are specifying a `@keyframes` declaration. We are then giving this particular keyframes declaration a name—`warning` in this instance. You can name them however you like but as these keyframe declarations can be re-used on multiple elements, name them accordingly.

You can set as many percentage points as you like (for example, 10, 20, 30, 40, and so on) or if you'd rather, define the animation with `from` and `to` values. Be warned however that Webkit browsers don't always play happily with from and to values (preferring 0% and 100%):

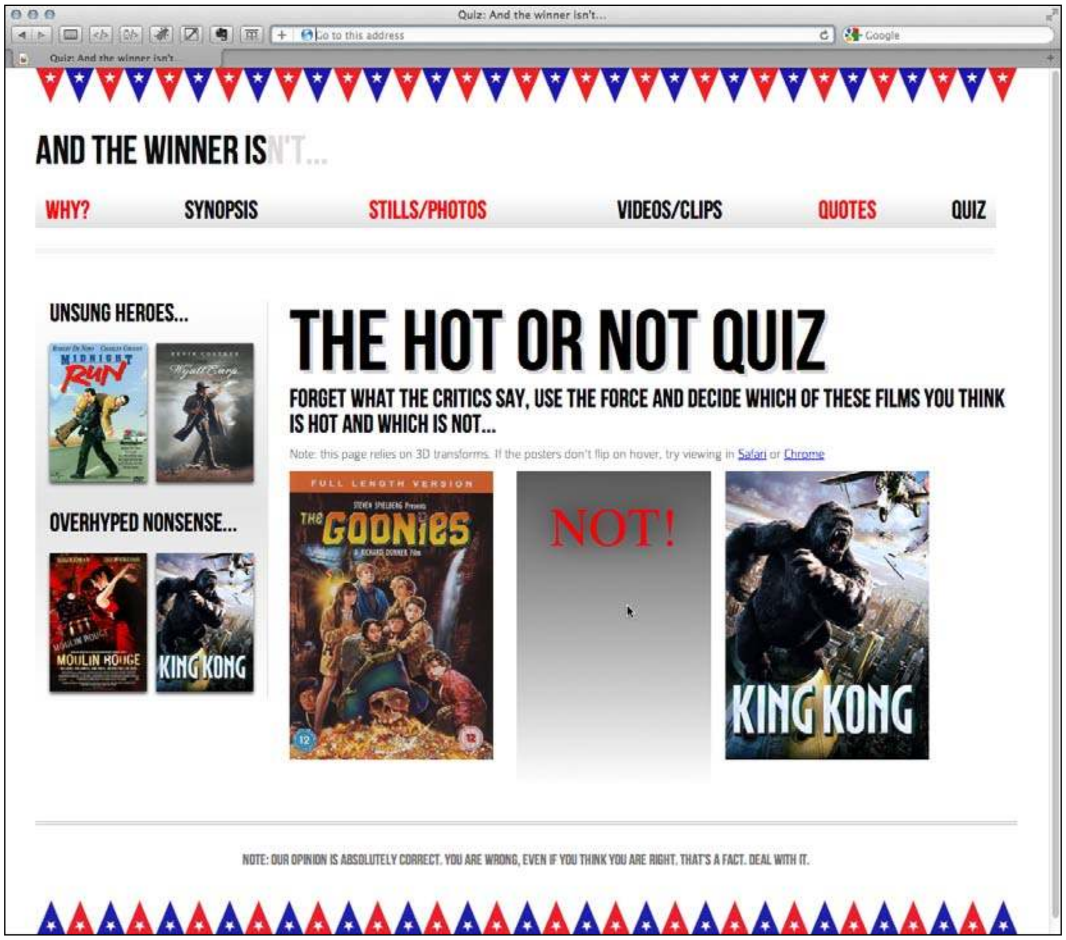
```
@keyframes warning {
  from {
    text-shadow: 0px 0px 4px #000000;
  }
  50% {
    text-shadow: 0 0 40px #000000;
  }
  to {
    text-shadow: 0 0 4px #000000;
  }
}
```

In this instance I'm altering a `text-shadow`, starting and ending with the same distance of `4px` but going to `40px` blur at `50%`.

Now we have declared the keyframe, we can reference it with the animation property:

```
.back h5 {
    font-size: 4em;
    color: #f2050b;
    text-align: center;
    animation: warning 1.5s infinite ease-in;
}
```

After specifying the animation property, we define the particular keyframe rule we want to use (`warning` in this case), we then specify the `animation-iteration-count` (we've used infinite here so the animation continues continuously) and finally the timing function (`ease-in`). A static image obviously fails to do this justice but hopefully you can imagine the text shadow pulsing back and forth. View this in the browser at `http://www.andthewinnerisnt.com`.


Quiz: And the winner isn't...
AND THE WINNER ISN'T...
WHY?
SYNOPSIS
STILLS/PHOTOS
VIDEOS/CLIPS
QUOTES
QUIZ
UNSUNG HEROES...
OVERHYPED NONSENSE...
THE HOT OR NOT QUIZ
FORGET WHAT THE CRITICS SAY, USE THE FORCE AND DECIDE WHICH OF THESE FILMS YOU THINK IS HOT AND WHICH IS NOT...
Note: this page relies on 3D transforms. If the posters don't flip on hover, try viewing in Safari or Chrome
NOT!
NOTE: OUR OPINION IS ABSOLUTELY CORRECT. YOU ARE WRONG, EVEN IF YOU THINK YOU ARE RIGHT. THAT'S A FACT. DEAL WITH IT.

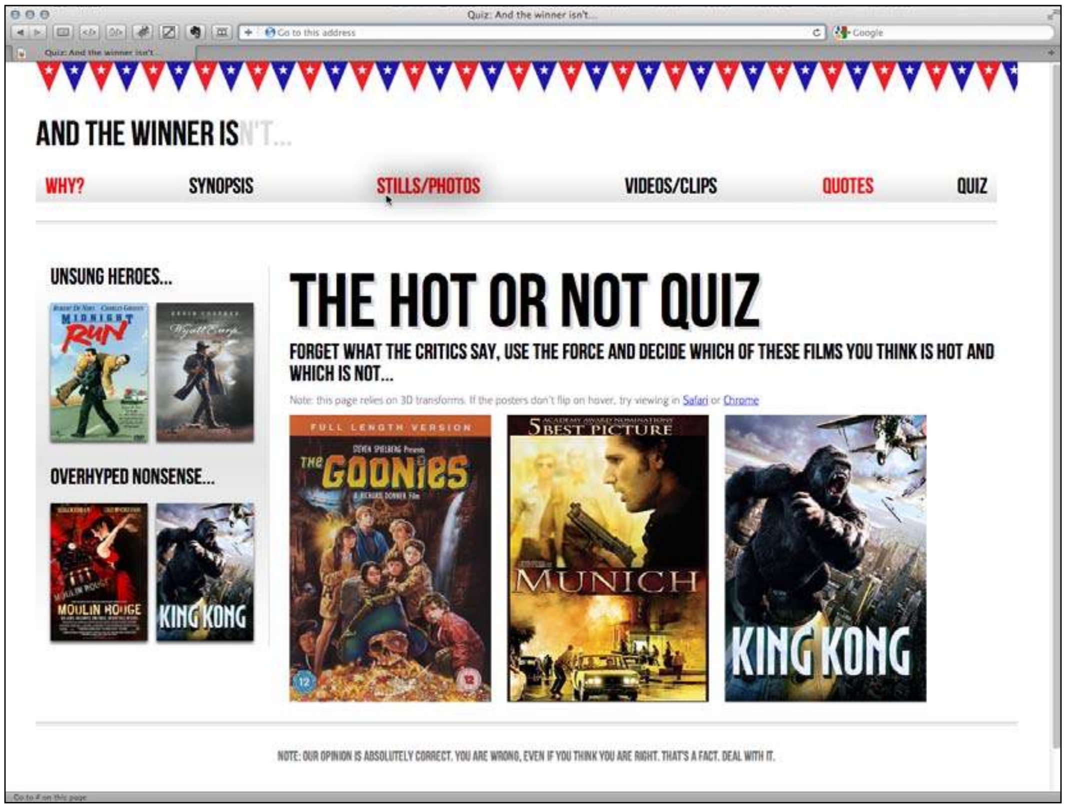
The shorthand property can accept all seven animation properties. In addition to those used in the above example, it's also possible to specify `animation-delay` (for example, if you wanted to delay when the animation starts), `animation-play-state` (can be set to `running` or `paused` to effectively play and pause an animation) and finally `animation-fill-mode`, which I confess, I've yet to find a need to use (the default is `none`). Of course you don't need to use the shorthand property; you can list them individually as follows:

```
    animation-name: warning;
    animation-duration: 1.5s;
    animation-timing-function: ease-in-out;
    animation-iteration-count: infinite;
    animation-play-state: running;
    animation-delay: 0s;
    animation-fill-mode: none;
```

As mentioned previously, it's simple to reuse the animation on other elements. For example:

```
    nav ul li a:hover {
        animation: warning 1.5s infinite ease-in;
    }
```

This gives our navigation links the same pulsing effect. You can (hopefully) see the `STILLS/PHOTOS` link in the screenshot below in the midst of the animation. Try it out for yourself at `http://www.andthewinnerisnt.com`.



THE HOT OR NOT QUIZ

FORGET WHAT THE CRITICS SAY, USE THE FORCE AND DECIDE WHICH OF THESE FILMS YOU THINK IS HOT AND WHICH IS NOT...

Note: this page relies on 3D transforms. If the posters don't flip on hover, try viewing in Safari or Chrome

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

This is just one very simple example of using CSS animations. As virtually anything can be key-framed, the possibilities are pretty endless. There are countless showcases of CSS3 animation techniques around the web. Pages like `http://webdesignerwall.com/trends/47-amazing-css3-animation-demos` should give you more than enough inspiration to be getting on with.

[Read about the latest developments on CSS3 Animations at `http://dev.w3.org/csswg/css3-animations/`.]

Putting CSS3 transformations and animations together

Let's try one more thing to flex our CSS3 muscles. I'd like to try placing all the aside sidebar images at varying angles and then animating them. The aim is to have them 'shake' when the page is first visited. Here's the markup for the sidebar:

```
<aside>
  <div role="complementary">
    <div class="sideBlock unSung">
      <h1>Unsung heroes...</h1>
      <a href="#"><img src="img/midnightRun.jpg"
                       alt="Midnight Run" /></a>
      <a href="#"><img src="img/wyattEarp.jpg"
                       alt="Wyatt Earp" /></a>
    </div>
  </div>
  <div role="complementary">
    <div class="sideBlock overHyped">
      <h1>Overhyped nonsense...</h1>
      <a href="#"><img src="img/moulinRouge.jpg"
                       alt="Moulin Rouge" /></a>
      <a href="#"><img src="img/kingKong.jpg"
                       alt="King Kong" /></a>
    </div>
  </div>
</aside>
```

Now let's create the CSS3 to make this work. First, let's create a new keyframe declaration called `swing`:

```
@-webkit-keyframes swing {
  from {
    transform: rotate(3deg);
  }
  20% {
    transform: rotate(7deg);
  }
  60% {
    transform: rotate(10deg);
  }
  80% {
    transform: rotate(7deg);
  }
  to {
    transform: rotate(3deg);
  }
}
```

The animation will use the 2D rotate transform to move the item from 3 degrees to 10 and back again. And here's how the `animation` property is added:

```
#quiz .unSung a:nth-child(odd) img {
    transform: rotate(3deg);
    animation: swing 0.1s 5 ease-in;
}
#quiz .unSung a:nth-child(even) img {
    transform: rotate(-3deg);
    animation: swing 0.1s 5 0.3s ease-in;
}
#quiz .overHyped a:nth-child(odd) img {
    transform: rotate(3deg);
    animation: swing 0.1s 5 0.2s ease-in;
}
#quiz .overHyped a:nth-child(even) img {
    transform: rotate(-3deg);
    animation: swing 0.1s 5 0.5s ease-in;
}
```

Let's break this down. Firstly by relying on CSS specificity we can target these rules only at the **QUIZ** page (which has a `<body id="quiz">` tag).

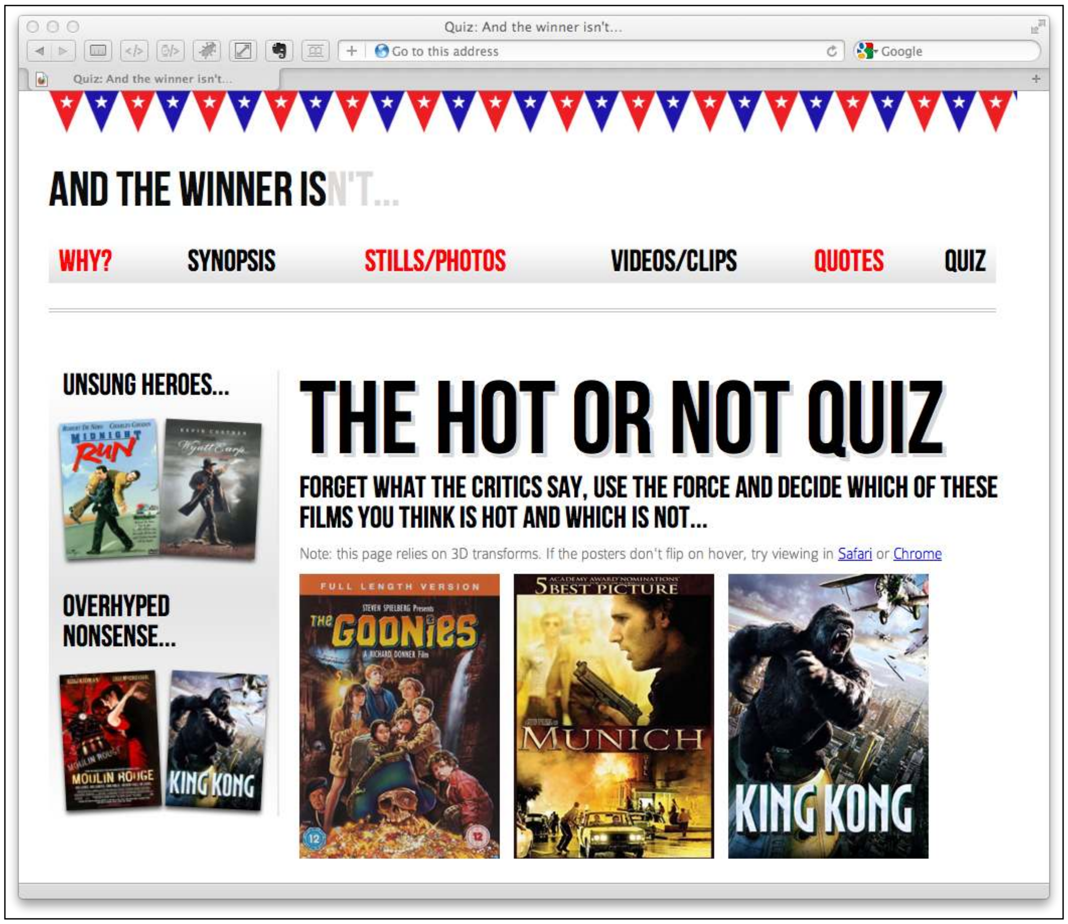
Before adding the animation property, I want to set a default rotate transform so that they remain off-kilter when the animation completes. I don't want them all at the same angle—so let's use the `nth-child` selector we learned about in *Chapter 5, CSS3: Selectors, Typography, and Color Modes* to select the odd and even images and apply different rotation transforms to them:

```
 #quiz .unSung a:nth-child(odd) img {
    transform: rotate(3deg);
    animation: swing 0.1s 5 ease-in;
}
#quiz .unSung a:nth-child(even) img {
    transform: rotate(-3deg);
    animation: swing 0.1s 5 0.3s ease-in;
}
#quiz .overHyped a:nth-child(odd) img {
    transform: rotate(3deg);
    animation: swing 0.1s 5 0.2s ease-in;
}
#quiz .overHyped a:nth-child(even) img {
    transform: rotate(-3deg);
    animation: swing 0.1s 5 0.5s ease-in;
}
```

Then the `animation` property is added for each instance. You'll notice slight variations in each of the rules. The shorthand property also takes into account that the second time value given (`0.5s`) is assigned to the animation delay. By utilizing this value we can effectively fire off each different instance separately.

```
#quiz .overHyped a:nth-child(even) img {
    transform: rotate(-3deg);
    animation: swing 0.1s 5 0.5s ease-in;
}
```

Again, when writing about animations, it's a little difficult to convey the effect. If you're not near an Internet connection, the best I can tell you is that the films rapidly shake from side to side and then settle off-kilter as shown in the following image:



FORGET WHAT THE CRITICS SAY, USE THE FORCE AND DECIDE WHICH OF THESE FILMS YOU THINK IS HOT AND WHICH IS NOT...

Note: this page relies on 3D transforms. If the posters don't flip on hover, try viewing in Safari or Chrome

Summary

It would be entirely possible to fill multiple books covering the possibilities of CSS transformations, transitions, and animations. However, hopefully, by dipping your toe in the water with this chapter you'll be able to pick up the basics and run with them. Ultimately, by embracing the new features and techniques of CSS3 the aim is to make a responsive design even leaner and richer than ever by using CSS3, rather than JavaScript for some of the fancier aesthetic enhancements. In this chapter we've learned what CSS3 transitions are and how to write them, got a handle on timing functions like 'ease' and 'linear', and then used them to create simple but fun effects with our responsive design. We then learned all about 2D transformations like scale and skew and then how to use them in tandem with transitions. We also looked briefly at 3D transformations before learning all about the power and relative simplicity of CSS animations. You'd better believe our CSS3 muscles are growing!

However, if there's one area of site design that I always avoid where possible (as desperately as I avoid *Munich* or *King Kong* if they're showing), it's making forms. I don't know why, I've just always found making them a tedious and frustrating task. Imagine my joy when I learned that HTML5 and CSS3 can make the whole form building, styling, and even validating (yes, validating!) process easier than ever before. I was quite joyous. As joyous as you can be about building web forms that is. In the next chapter I'd like to share this knowledge with you.