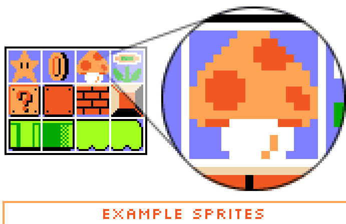
# CSS Sprites: Image Slicing's Kiss of Death

**This article was originally published by A List Apart and is reproduced by kind permission.**

Back when video games were still fun (we're talking about the 8-bit glory days here), graphics were a much simpler matter by necessity. Bitmapped 2-dimensional character data and background scenery was individually drawn, much like today's resurgent pixel art. Hundreds and later thousands of small graphics called sprites were the building blocks for all things visual in a game




As game complexity increased, techniques developed to manage the multitude of sprites while keeping game play flowing. One variation saw sprites being plugged into a master grid, then later pulled out as needed by code that mapped positions of each individual graphic, and selectively painted them on the screen.

## *And what does this have to do with the web?*

Everything old is new again, and though the rise of 3D games has made sprite maps obsolete, the concurrent rise of mobile devices with 2D gaming capabilities have brought them back into vogue. And now, with a bit of math and a lot of CSS, we're going to take the basic concept and apply it to the world of web design.

Specifically, we're going to replace old-school image slicing and dicing (and the necessary JavaScript) with a CSS solution. And because of the way CSS works, we're going to take it further: by building a grid of images and devising a way to get each individual cell out of the grid, we can store all buttons/navigation items/whatever we wish in a single master image file, along with the associated "before" and "after" link states.

## *How do CSS Sprites work?*

As it turns out, the basic tools to do this are built into CSS, given a bit of creative thinking.

Let's start with the master image itself. Dividing a rectangle into four items, you'll observe in this master image (below) that our intended "before" link images are on the top row, with "after" :hover states immediately below.

Copyright © 1998–2004 A List Apart Magazine, Happy Cog Studios, and Dave Shea
Reproduced by Kind Permission.

There's no clear division between the four links at the moment, so imagine that each piece of text is a link for now. (For the sake of simplicity, we'll continue to refer to link images as "before" images and the `:hover` state as "after" for the rest of this article. It's possible to extend this method to `:active`, `:focus`, and `:visited` links states as well, but we won't go into that here.)

Those familiar with Petr Stanicek's (Pixy) Fast Rollovers may already see where we're going with this. This article owes a debt of gratitude to Pixy's example for the basic function we'll be relying on. But let's not get ahead of ourselves.

On to the HTML. Every good CSS trick strives to add a layer of visuals on top of a clean block of code, and this technique is no exception:

```
<ul id="skyline">
  <li id="panel1b"><a href="#1"></a></li>
  <li id="panel2b"><a href="#2"></a></li>
  <li id="panel3b"><a href="#3"></a></li>
  <li id="panel4b"><a href="#4"></a></li>
</ul>
```

This code will serve as a base for our example. Light-weight, simple markup that degrades well in older and CSS-disabled browsers is all the rage, and it's a trend that's good for the industry. It's a great ideal to shoot for. (We'll ignore any text inside the links for the time being. Apply your favorite image replacement technique later to hide the text you'll end up adding.)

Copyright © 1998–2004 A List Apart Magazine, Happy Cog Studios, and Dave Shea
Reproduced by Kind Permission.

## *Applying the CSS*

With those basic building blocks, it's time to build the CSS. A quick note before we start — because of an IE glitch, we'll be tiling the after image on top of the before image when we need it, instead of replacing one with the other. The result makes no real visual difference if we line them up precisely, but this method avoids what otherwise would be an obvious "flicker" effect that we don't want.

```
#skyline {
  width: 400px; height: 200px;
  background: url(test-3.jpg);
  margin: 10px auto; padding: 0;
  position: relative;}
#skyline li {
  margin: 0; padding: 0; list-style: none;
  position: absolute; top: 0;}
#skyline li, #skyline a {
  height: 200px; display: block;}
```

Counter-intuitively, we're not assigning the before image to the links at all, it's applied to the <ul> instead. You'll see why in a moment.

The rest of the CSS in the above example sets things like the dimensions of the #skyline block and the list items, starting positions for the list items, and it turns off the unwanted list bullets.
We'll be leaving the links themselves as empty, transparent blocks (though with specific dimensions) to trigger the link activity, and position them using the containing <li>s. If we were to position the links themselves and effectively ignore the <li>s, we'd start seeing errors in older browsers, so let's avoid this.

## *Positioning the links*

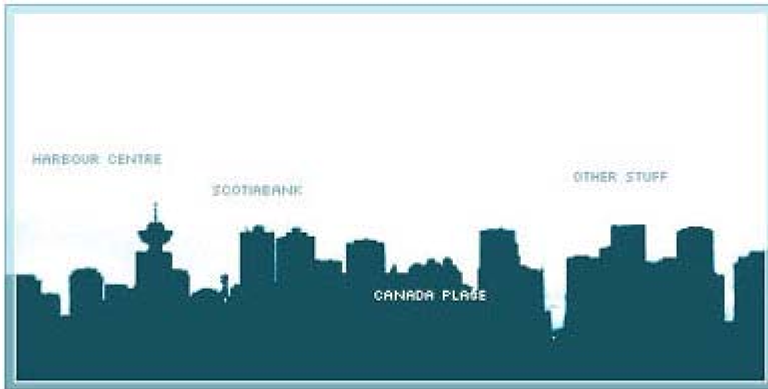
The `<li>`s are absolutely positioned, so why aren't they at the top of the browser window? A quirky but useful property of positioned elements is that all descendent elements contained within them base their absolute position not off the corners of the browser window, but off the corners of the nearest positioned ancestor element. The upshot of this is that since we applied `position: relative;` to `#skyline`, we're able to absolutely position the `<li>`s from the top left corner of `#skyline` itself.

```
#panel1b {left: 0; width: 95px;}
#panel2b {left: 96px; width: 75px;}
#panel3b {left: 172px; width: 110px;}
#panel4b {left: 283px; width: 117px;}
```

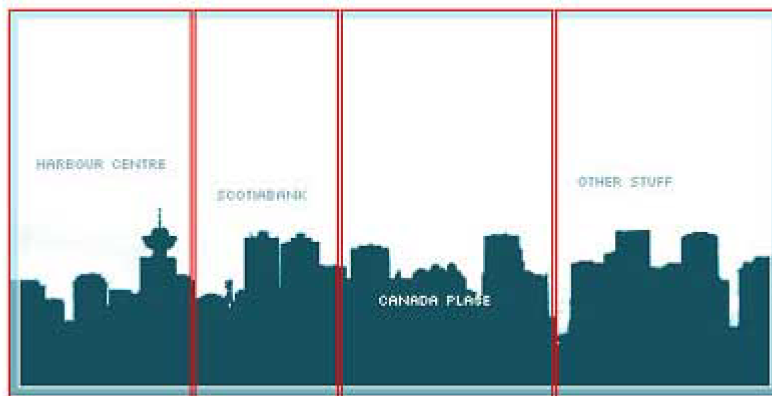
So `#panel1` isn't horizontally positioned at all, `#panel2b` is positioned 96px to the left of `#skyline`'s left edge, and so on. We assigned the links a `display: block;` value and the same height as the `<li>`s in the past listing, so they'll end up filling their containing `<li>`s, which is exactly what we want.

Copyright © 1998–2004 A List Apart Magazine, Happy Cog Studios, and Dave Shea
Reproduced by Kind Permission.

At this point we have a basic image map with links, but no `:hover` states:




It's probably easier to see what's happening with borders turned on:




## *Hovers*

In the past we would have applied some JavaScript to swap in a new image for the after state. Instead our after states are in one image, so all we need is a way to selectively pull each state out for the appropriate link.

If we apply the master image to the :hover state without additional values, we make only the top left corner visible — not what we want, though clipped by the link area, which *is* what we want. We need to move the position of the image somehow.

We're dealing with known pixel values; a little bit of math should enable us to offset that background image enough both vertically and horizontally so that only the piece containing the after state shows.
That's exactly what we'll do:

```
#panel1b a:hover {
    background: transparent url(test-3.jpg)
    0 -200px no-repeat;}
  #panel2b a:hover {
    background: transparent url(test-3.jpg)
    -96px -200px no-repeat;}
```

Copyright © 1998–2004 A List Apart Magazine, Happy Cog Studios, and Dave Shea
Reproduced by Kind Permission.

```
  #panel3b a:hover {
    background: transparent url(test-3.jpg)
    -172px -200px no-repeat;}
  #panel4b a:hover {
    background: transparent url(test-3.jpg)
    -283px -200px no-repeat;}
```

Where did we get those pixel values? Let's break it down: the first value is of course the horizontal offset (from the left edge), and the second is the vertical.

Each vertical value is equal; since the master image is 400 pixels high and the after states sit in the bottom half, we've simply divided the height. Shifting the whole background image up by 200px requires us to apply the value as a negative number. Think of the top edge of the link as the starting point, or 0. To position the background image 200 pixels above this point, it makes sense to move the starting point -200px.

Likewise, if the left edge of each link is effectively 0, we'll need to offset the background image horizontally by the width of all <li>s prior to the one we're working with. So the first link doesn't require an offset, since there are no pixels before its horizontal starting point. The second link requires an offset the width of the first, the third link requires an offset of the combined width of the first two links, and the last requires an offset of the combined width of all three previous links.

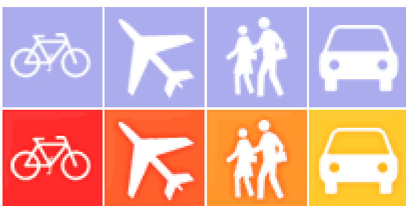
It's a bit cumbersome to explain the process, but playing around with the values will quickly show you how the offsets work, and once you're familiar it's not all that hard to do.

So there you have it. Single-image CSS rollovers, degradable to a simple unordered list.

## *Buttons*

There's no reason why we have to leave the links touching each other, side-by-side as they were in the previous example. Image maps may be convenient in some spots, but what about separating each link into its own stand-alone button? That way we can add borders and margins, let the underlying background show through, and generally treat them as separately as we need to.

In fact, the building blocks are already in place. We really don't need to modify our code too radically; the main change is in creating a new background image that doesn't continue from link to link like the last example did. Since we can't rely on the <ul> for placing the original background image, we'll end up applying it to all <li>s instead and offsetting each the same way we offset the after states in the prior example.  With an appropriate image (below)



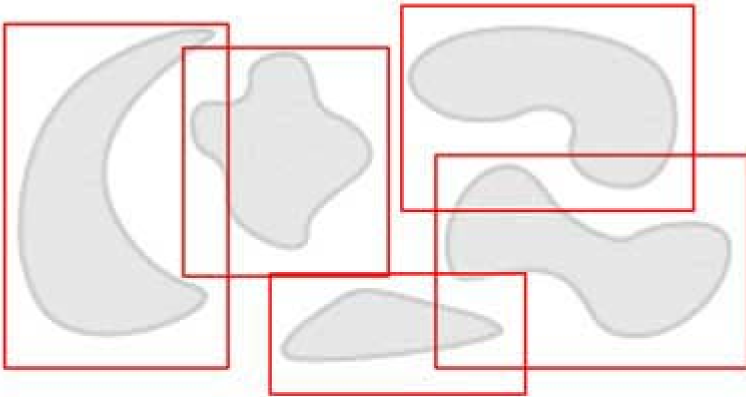
and a bit of spacing between each `<li>`, we've got buttons.

Note that in this example we've added 1px borders which, of course, count toward the final width of the links. This affects our offset values; we've compensated by adding 2px to the offsets where appropriate.

Copyright © 1998–2004 A List Apart Magazine, Happy Cog Studios, and Dave Shea
Reproduced by Kind Permission.

### *Irregular shapes*

Up till now we've focused only on rectangular, non-overlapping shapes. What about the more complex image maps that image slicers like Fireworks and ImageReady export so easily? Relax, we've got you covered there too.
We'll start the same way as the first example, by applying the background image to the <ul> and turning off list item bullets and setting widths and so forth. The big difference is where we position the <li>s; the goal is to surround each graphical element with a box that tightly hugs the edges:

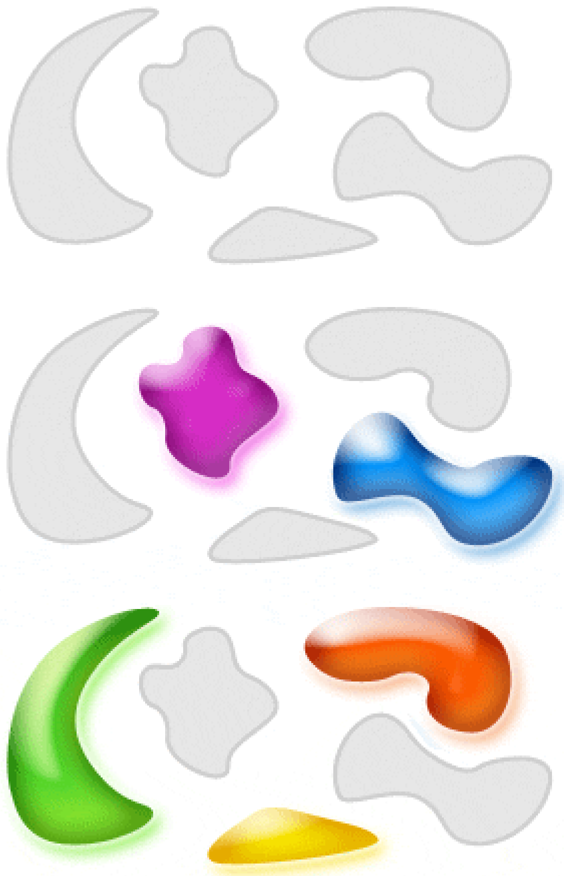


Again, because of the ability to use absolute positioning relative to the top left corner of the <ul>, we're able to precisely place our links exactly where we want them. Now all that's left is to set up the hover states (click to see them).

Worth noting is that in this case, a single set of before and after images wasn't enough. Because of the overlapping objects, relying on only one after state would show pieces of surrounding objects' after states. In fact, it would show precisely the pieces that fall within the link's borders. (Easiest to just see it in action.)

How to avoid this? By adding a second after state, and carefully selecting which objects go where. The master image in this case has split the purple and blue objects into the first after state, and the green, orange and yellow objects into the second:

Copyright © 1998–2004 A List Apart Magazine, Happy Cog Studios, and Dave Shea
Reproduced by Kind Permission.



This order allows boxes to be drawn around each object's after state without including pieces of the surrounding objects. And the illusion is complete (click to see it).

## *Benefits and pitfalls*

A couple of final thoughts. Our new CSS Sprite method tests well in most modern browsers. The notable exception is Opera 6, which doesn't apply a background image on link hover states. Why, we're not sure, but it means that our hovers don't work. The links still do, and if they've been labeled properly, the net result will be a static, but usable image map in Opera 6. We're willing to live with that, especially now that Opera 7 has been around for a while.

The other concern is familiar to anyone who has spent time with FIR. In the rare cases in which users have turned off images in their browsers but retained CSS, a big empty hole will appear in the page where we expect our images to be placed. The links are still there and clickable, but nothing visually appears. At press time, there was no known way around this.

Then there's file size. The natural tendency is to assume that a full double-sized image must be heavier than a similar set of sliced images, since the overall image area will usually be larger. All image formats have a certain amount of overhead though (which is why a 1px by 1px white GIF saves to around 50 bytes), and the more slices you have, the more quickly that overhead adds up. Plus, one master image requires only a single color table when using a GIF, but each slice would need its own. Preliminary tests suggest that all this indicates smaller total file sizes for CSS Sprites, or at the very least not appreciably larger sizes.

Copyright © 1998–2004 A List Apart Magazine, Happy Cog Studios, and Dave Shea
Reproduced by Kind Permission.

And lastly, let's not forget that our markup is nice and clean, with all the advantages that go along with that. HTML lists degrade wonderfully, and a proper image replacement technique will leave the text links accessible to screenreaders. Replacing the sprite imagery is dead simple, since all of our dimensions and offsets are controlled in a single CSS file, and all of our imagery sits in a single image.

## *About The Author*

Creator of the popular css Zen Garden, Dave Shea is a graphic designer who thinks this CSS thing may just be going places. Dave writes daily for mezzoblue, and just started consulting for his own Bright Creative. He was interviewed by DMXzone recently and has contributed to a book on CSS, Cascading Style Sheets: Separating Content from Presentation, Second Edition

Copyright © 1998–2004 A List Apart Magazine, Happy Cog Studios, and Dave Shea
Reproduced by Kind Permission.