Cascading Style Sheets

CSS 20

Webkits

CSS

Cascading Style Sheets

Mr. K's Technology

CSS

Cascading Style Sheets

Mr. K's Technology

Information for this Unit Was obtained from

Mr. K's Technology

Welcome to HTML.net
Free tutorials on HTML and CSS
http://www.html.net/

Safari Developer Library

tizag.com
unlock your potential

http://www.tizag.com

ABOUT WEB ELEMENTS

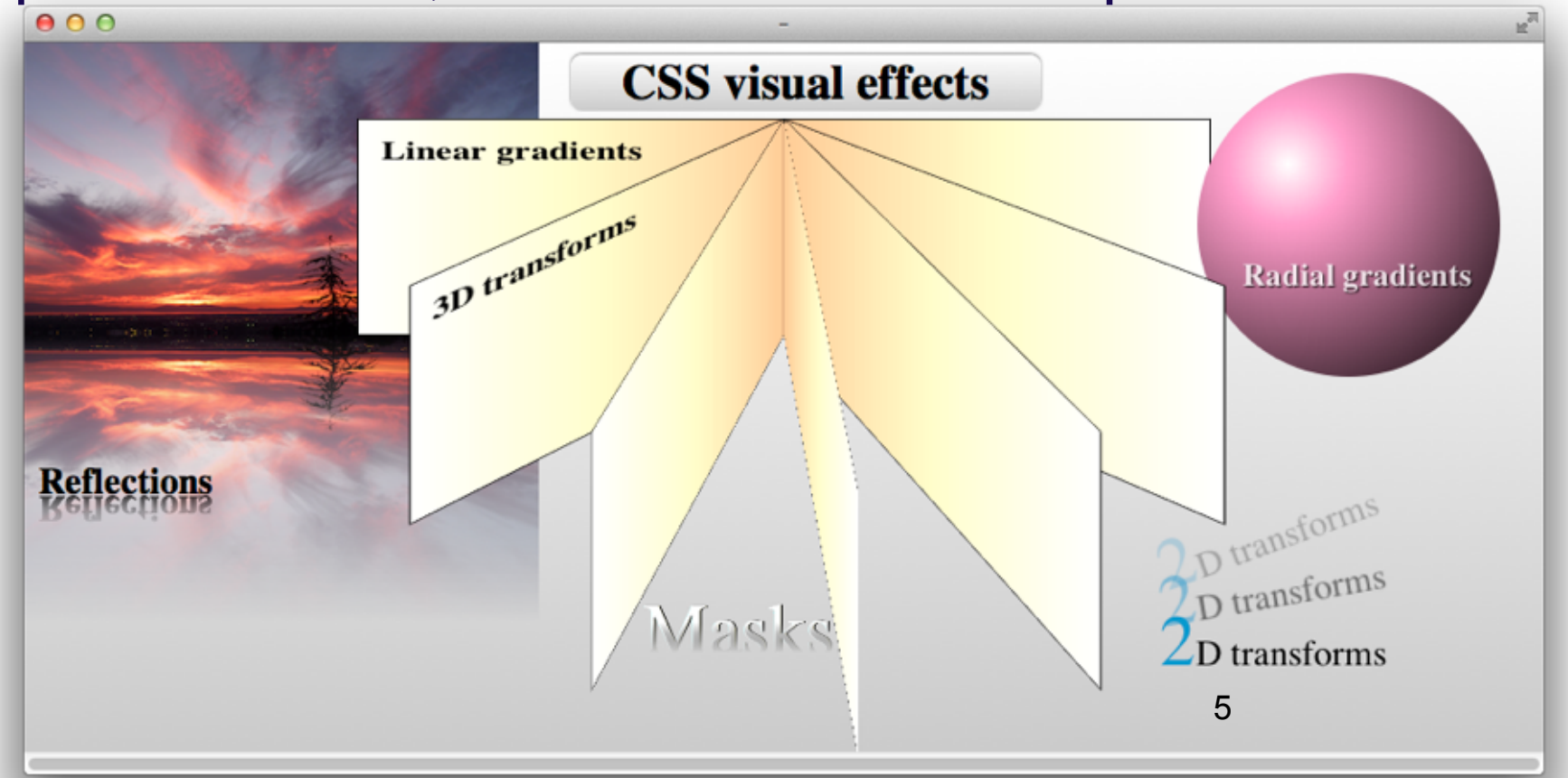
CSS

http://www.w3schools.com/css/

Welcome to the website for the WebKit Open Source Project!

WebKit is an open source web browser engine. WebKit is also the name of the Mac OS X system framework version of the engine that's used by Safari, Dashboard, Mail, and many other OS X applications. WebKit's HTML and JavaScript code began as a branch of the KHTML and KJS libraries from KDE.

Use CSS to create stunning visual effects—masks, gradients, reflections, lighting effects, animations, transitions, 3D rotations, and more. Apply any or all of these effects interactively, triggered by mouse events or touch events; make HTML elements visibly respond to the user—without requiring plug-ins, graphics libraries, or elaborate JavaScript programs.



There are advantages to using CSS instead of graphic images to create visual effects:

- Because they are resolution-independent, CSS effects scale up smoothly when zoomed.
- Text formatted with CSS is searchable; images are not.
- CSS is compact and compresses well compared with graphic images.
- CSS is just text; it can be quickly modified using a text editor or the output of a script.

Safari supports CSS visual effects on Mac OS X and iOS.

Three Ways to Insert CSS

There are three ways of inserting a style sheet:

- External style sheet
- Internal style sheet
- Inline style

Here's what we are planning to demonstrate

Using Gradients (17),

Using a Gradient as a Mask (18),

Using Reflections (18),

Using CSS Filters (18),

Animating CSS Transitions (19),

Animating With Keyframes (19),

Using 2D and 3D Transforms (20),

the rest will be covered in future lessons

Using 2D and 3D Transforms

Use CSS transform properties to give webpages a rich visual appearance without needing image files. Elements can be positioned, rotated, and scaled in 2D and 3D space; perspective can also be applied, giving elements the appearance of depth.

For example, Figure 7-1 shows a simple HTML document, containing only div elements and text but rendered using CSS transform and gradient properties. It appears to be a graphics-intensive page, yet the actual content is less than 2 KB of text, and the elements animate smoothly in 3D under user control.

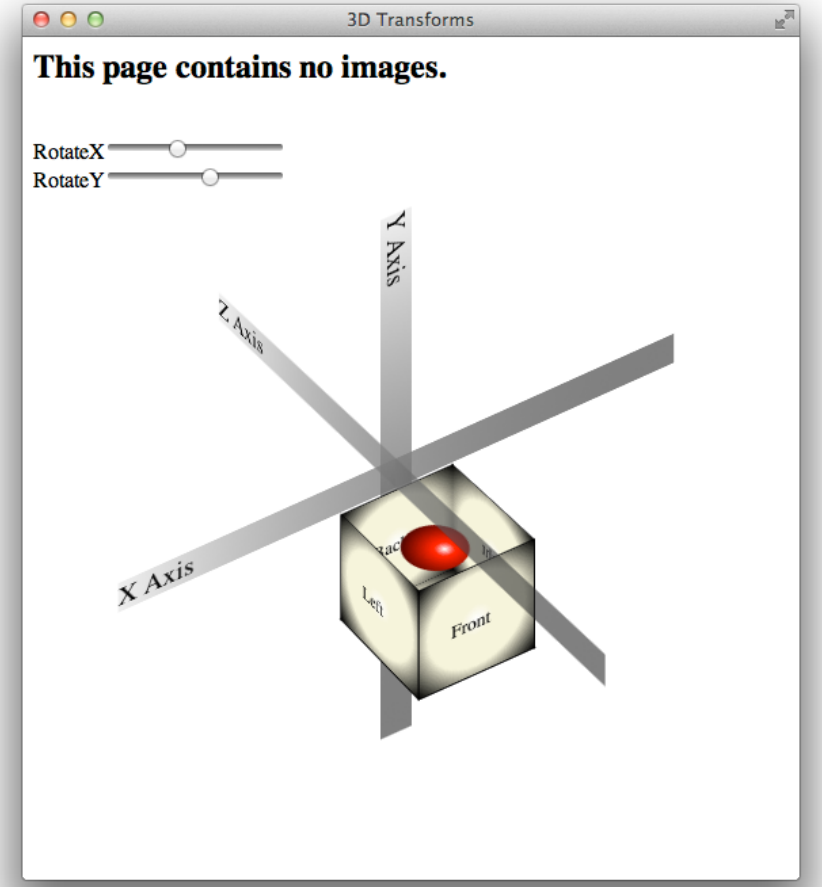


Figure 7-1 *HTML page with rotation and perspective transforms*

How does this work? HTML elements are, after all, inherently two-dimensional, or planar; they have height and width, but no thickness. By rotating planar elements into the third dimension and applying perspective, however, these elements can be combined to create apparently solid objects. For example, five div elements are combined in Figure 7-1 to form the sides and bottom of an open box; the ball inside the box is another div element with rounded borders—a radial gradient gives it the appearance of depth.

Safari uses a series of transformation matrices to determine the mapping of every pixel on the screen. You don't need to understand matrices to use them, however. You can apply a transform either by using a matrix or by calling one of the transform functions, such as scale() or rotate().

Transforming an element typically causes its image to be rendered in a different position on the screen, but the position and dimensions of the element on the page are not changed—the top, left, height, and width properties, for example, are not altered by transforms. It is the coordinate system in which the element is drawn that is changed. Consequently, changing transform properties does not affect the layout of a webpage. This means that transforming an element can cause it to visually overlap neighboring elements, even though their positions and dimensions on the page may not overlap.

Note: *Mouse and touch events, such as onclick, track the visual representation of an element, taking 2D and 3D transforms into account.*

By default, transforms are applied using the center point of an element as the origin; rotation spins an object about its center, for example, and scaling expands or contracts an element from the center point. You can change the origin by setting the -webkit-transform-origin property.

A transform can cause part of an element to be displayed in the element's overflow area. If the value of the overflow property is scroll or auto, scroll bars appear as needed if a transform renders part of an object outside the display area.

Safari supports both 2D and 3D transforms. Both 2D and 3D transforms are W3C drafts.

Webkits

All **webkit** elements can be used within an internal style sheet

2D Transform Functions

To apply a 2D transform to an element, use the **-webkit-transform** property. The transform property can be set using predefined transform functions—translation, rotation, and scaling—or it can be set using a matrix.

2D Translation

2D translation shifts the contents of an element by a horizontal or vertical offset without changing the top or left properties. The element's position in the page layout is not changed, but the content is shifted and a shifted coordinate system applies to all descendants of the translated element.

For example, if a div element is positioned at the point 10,10 using the CSS properties top and left, and the element is then translated 100 pixels to the right, the element's content is drawn at 110,10. If a child of that div is positioned absolutely at the point 0,100, the child is also shifted to the right and is drawn at the point 110,110; the specified position of 0,100 is shifted right by 100, and the child is drawn at 100,100 relative to the parent's upper-left corner, which is still at 10,10. Figure 7-2 illustrates this example.

ABOUT WEB ELEMENTS

All **webkit** elements can be used inline within the HTML style tag

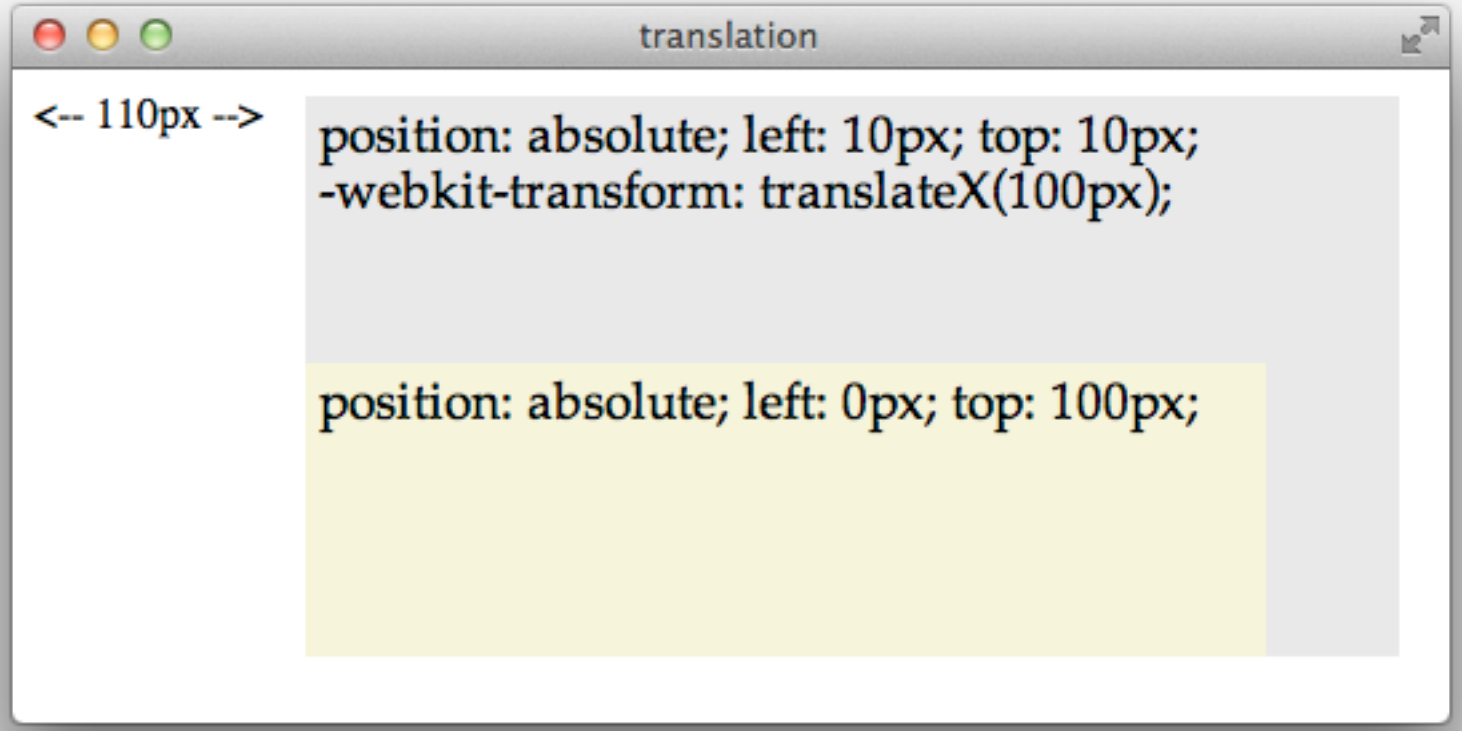


Figure 7-2 A translated element and its offspring

To apply a 2D translation, set the -webkit-transform property to translateX(offset), translateY(offset), or both. For example:

```
<div style="-webkit-transform: translateX(40px);">

<div style="-webkit-transform: translateY(50%);">

<div style="-webkit-transform: translateX(100px) translateY(100px);">
```

2D Rotation

2D rotation is a rotation in the xy plane. By default, 2D rotation spins an object around its center point. To rotate an element around a different point, see "Changing the Origin." Rotation is specified in degrees clockwise from the element's orientation after any inherited rotation; rotation affects the specified element and all of its descendants. The coordinate system of any descendants is likewise rotated.

The following snippet rotates a div element 45 degrees clockwise, as shown in Figure 7-3. The div element has a beige background and contains a paragraph of text and an image; both the text and the image inherit the div element's rotation. A second div is positioned under the rotated div to show the original div's position prior to rotation.

```
<div style="-webkit-transform: rotate(45deg);" >
```

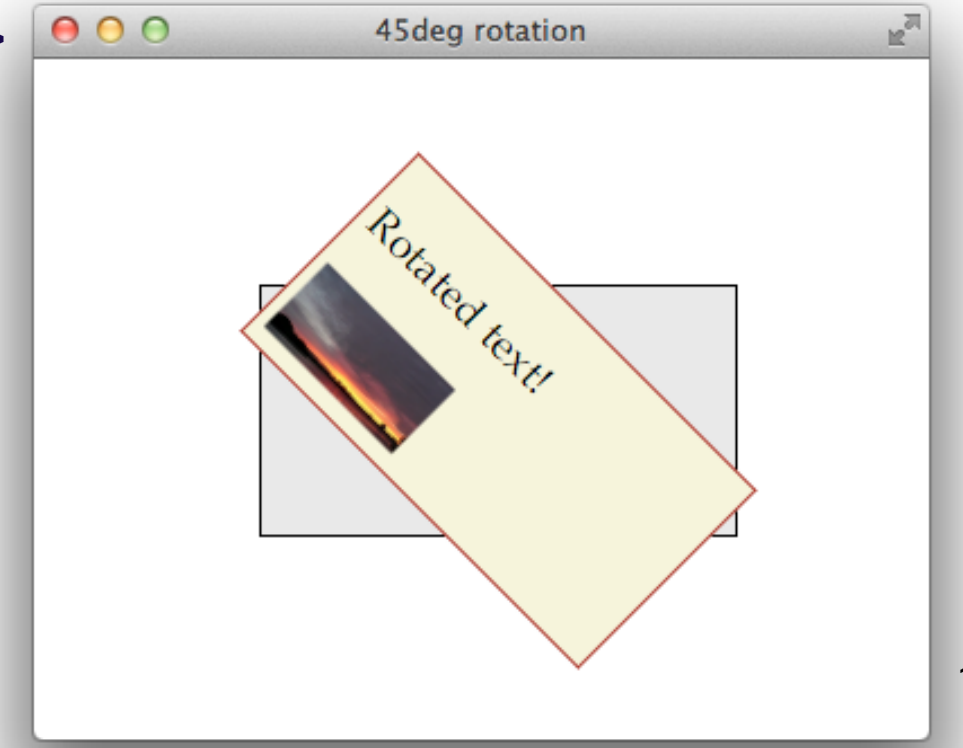


Figure 7-3 Rotating an element

Rotation can be specified in positive or negative degrees. For example, -webkit-transform: rotate(-45deg); specifies a 45 degree rotation counterclockwise. If rotation is animated, specifying a degree of rotation greater than the current degree causes clockwise rotation; specifying a degree of rotation less than the current degree causes counter-clockwise rotation.

When animating rotation, it can be useful to specify a rotation of more than 360 degrees. For example, Listing 7-1 uses JavaScript to set a rotation of 3600deg, causing a div element to spin clockwise ten times. The text spins once on page load, and a button lets the user spin it again.

All **webkit** elements can be used from an external style sheet

Safari Developer Library

We are introducing these concepts now so when we get to JavaScript it will be easier to understand

Listing 7-1 Animating 2D rotation

```
<html>
<head>
   <title>spin-in text</title>
<style>
#spinner {
   background: beige; border: 1px solid brown; padding: 25px;
   position: absolute; left: 100; top: 100; width: 200px;
   -webkit-transition: -webkit-transform 2s;
}
</style>
<script type="text/javascript">
var angle = 0;
function spin() {
   angle = angle + 3600;
   var spin10 = "rotate(" + angle + "deg)";
   document.getElementById("spinner").style.webkitTransform = spin10;
}
</script>
</head>
<body onload="spin();">
<div id="spinner">
   <h2>Spinning text!</h2>
   Whew! I'm dizzy...
</div>
<input type="button" value="Do it again!" onclick="spin();">
</body>
</html>
```

Notice that the CSS property -webkit-transform is addressed in JavaScript as element.style.webkitTransform. Notice also that the spin() function increments the rotation angle by 3600 each time it is called; setting the angle to 3600deg repeatedly would have no effect.

ABOUT WEB ELEMENTS

2D Scaling

2D scaling makes an element smaller or larger in one or two dimensions. Scaling affects the whole element, including any border thickness. By default, the element is scaled up or down relative to its center, which causes all four of the element's corners to be redrawn at new locations. The element's top, left, height, and width properties are unchanged, however, so the layout of the page is not affected. Consequently, scaling an element up can cause it to cover other elements on the page unless you design the layout to allow room for the expansion.

Scaling modifies the coordinate system of an element's descendants, multiplying the x and y values by the specified scaling factor. For example, if a div element contains an image positioned absolutely at 10,10, with a height and width of 100 pixels, scaling-up the div element by a factor of two results in a 200 x 200 image, positioned at 20,20 relative to the div's upper-left corner, which moves up and to the left. Figure 7-4 illustrates this behavior.

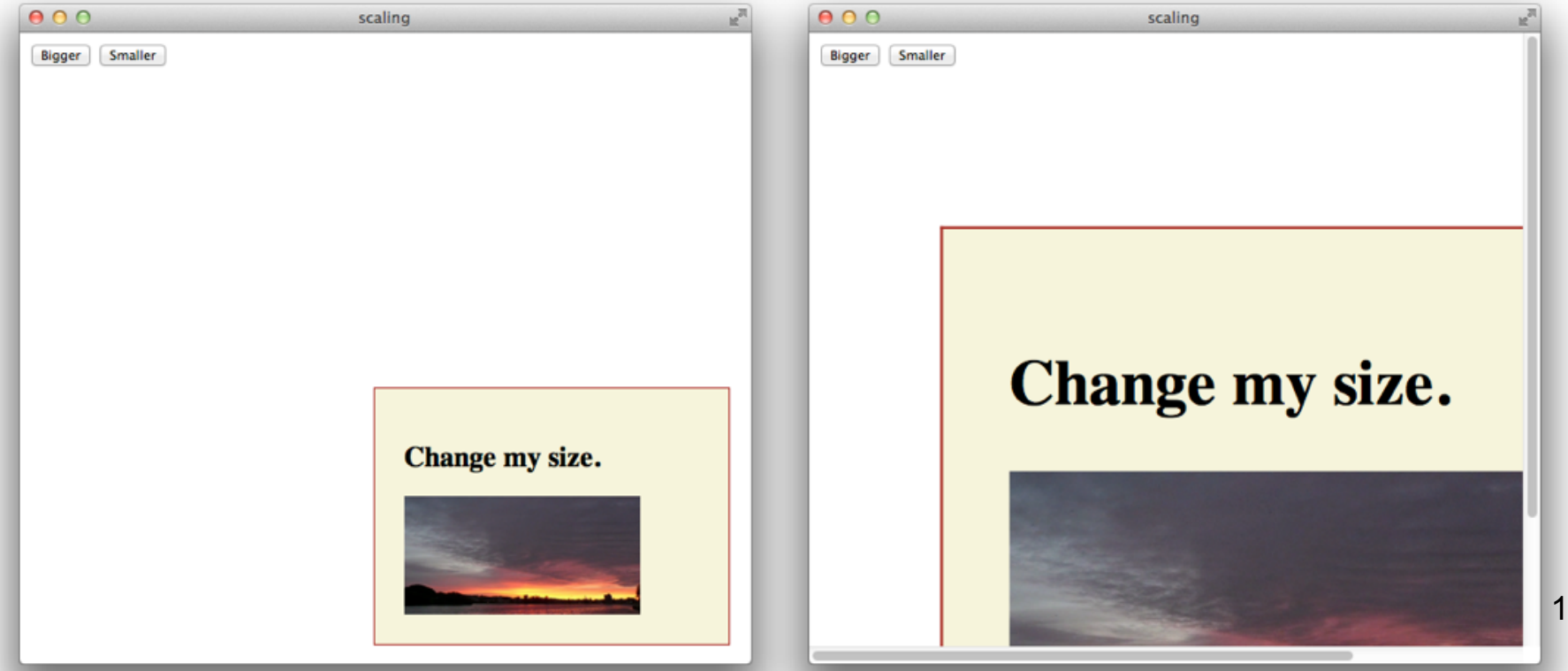


Figure 7-4 Scaling an element up

Apply a scale transformation by setting the -webkit-transform property to scale(x y), where x and y are independent scale factors for width and height, or by setting the transform property to scale(size), where size is the scaling factor in both dimensions. For example:

style="webkit-transform: scale(1, 2)" renders an element the same width, but twice as tall.

style="webkit-transform: scale(2, 0.5)" renders an element twice as wide and half as tall.

style="webkit-transform: scale(1.5)" renders an element 1.5 times larger.

Setting Multiple Transforms

Different transforms, such as rotation and scaling, are applied by setting different values to a single property: -webkit-transform. Consequently, if you apply one transform to an element and then specify another transform, the first transform is no longer applied; the new value overwrites the old one, as with any CSS property.

There are two different ways to perform multiple transforms on an element—both scaling and rotating an element for example:

- Use inheritance to apply multiple transforms: create a scaled div element, for example, add your element as a child, then rotate the child element.
- Set the -webkit-transform property of the element to a space-delimited list of transform functions, such as: -webkit-transform: scale(2) rotate(45deg);
  When a list of functions is provided, the final transformation value for the element is obtained by performing a matrix concatenation of each entry in the list. (Matrix concatenation can have some side effects, such as normalizing the rotation angle modulo 360.)

Both approaches—transform inheritance and transform function lists—are valid. The following two examples illustrate two ways to apply a set of transforms to an element. Listing 7-2 sets an element's transform property to a list of transform functions. Listing 7-3 produces the same results by applying each transform to a nested element.

Listing 7-2 Setting multiple transforms using a list

```
<div style="-webkit-transform:
        translate(-10px,-20px) scale(2) rotate(45deg);">
</div>
```

Listing 7-2 Setting multiple transforms using a list

```
<div style="-webkit-transform: translate(-10px,-20px) scale(2) rotate(45deg);">
</div>
```

Listing 7-3 Nesting 2D transforms

```
<div style="-webkit-transform:translate(-10px,-20px)">

  <div style="-webkit-transform:scale(2)">

    <div style="-webkit-transform:rotate(45deg)">

    </div>
  </div>
</div>
```

We are introducing these concepts now so when we get to JavaScript it will be easier to understand

Changing the Origin

By default, the origin for transforms is the center of an element's bounding box. Most HTML and CSS entities use the upper-left corner as the default origin, but for transforms, it is usually more convenient to use the center of an element as a reference point. Consequently, elements rotate around their center by default, and scale up or down from the center out.

To change the origin for transforms of a given element, set the **-webkit-transform-origin** property. The new origin is specified as a distance or percentage from the element's upper-left corner. For example, the default center origin can be expressed as -webkit-transform-origin: 50% 50%;. Changing the origin to 0% 0% or 0px 0px causes transformation to occur around the upper-left corner of the element.

The code in Listing 7-4 rotates the second box in Figure 7-5 around the top-right corner.

Listing 7-4 Rotating an element around the top-right corner

```
<div style="width: 300px; -webkit-transform: rotate(-45deg); -webkit-transform-origin: 100% 0%;">
<p>I am rotated about my top-right corner.</p>
</div>
```

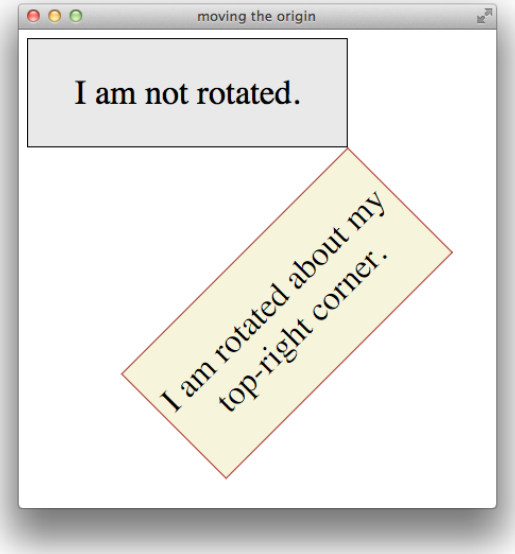


Figure 7-5 Element rotated around the top-right corner

We are introducing these concepts now so when we get to JavaScript it will be easier to understand

3D Transforms

The standard HTML coordinate system has two axes—the x-axis increases horizontally to the right, and the y-axis increases vertically downwards. With 3D transforms, a z-axis is added, with positive values rising out of the window toward the user and negative values falling away from the user, as Figure 7-6 illustrates.

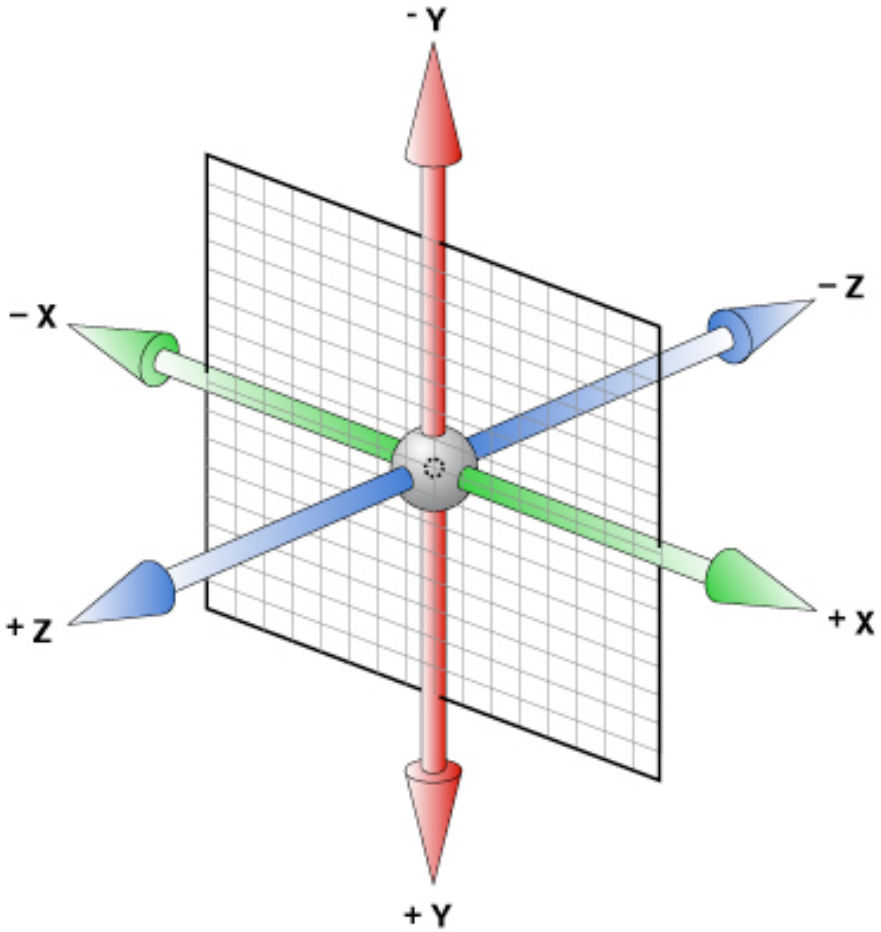


Figure 7-6 3D coordinate space

We are introducing these concepts now so when we get to JavaScript it will be easier to understand

ABOUT WEB ELEMENTS

3D transforms move an element out of the usual xy plane, where z=0—the plane of the display. A transformed element is still two dimensional, but it no longer lies in the usual plane. A transformed object may be translated along the z-axis, rotated around the x- or y-axis, or transformed using some combination of translation and rotation.

All HTML elements have a z-index. The z-index controls the rendering order when elements overlap. An element's z-index has nothing to do with its z-axis coordinate. Transformed objects follow the standard HTML rendering rules—objects with higher z-index values are drawn on top of other objects with lower z-index values—but for elements sharing the same z-index, the areas with higher z-axis coordinate values are drawn on top.

Adding 3D Perspective

To render elements with the appearance of depth, you must specify a perspective. If you apply 3D transforms without setting the perspective, elements appear flattened. For example, if you rotate an element around its y-axis without setting the perspective, the element just appears narrower. If you rotate an element 90 degrees from the default xy plane, it is seen edge-on—the element either disappears entirely or is displayed as a line.

Adding perspective distorts the appearance of objects realistically, making nearby things appear larger and distant things look smaller. The closer the object, the greater the distortion. In order for Safari to create the illusion of depth, it's necessary to specify a point of view, or perspective. Once Safari knows where the user's eye is relative to an element, it knows how much distortion to apply and where.

Use the **-webkit-perspective** property to set the perspective for all the descendants of an element. For example:

```
<div style="-webkit-perspective: 400px;">
```

The perspective is specified in distance from the screen. You may specify the distance in pixels, centimeters, inches, or any CSS distance unit. If no unit type is supplied, px is assumed.

Note: *Perspective distortion can cause part or all of an element to be displayed offscreen, particularly if an object's z-coordinates are positive and the projected image would extend beyond the specified viewpoint, somewhere behind the user's eye.*

Tip: *Perspective settings of 300px or less tend to create intense perspective distortion; settings from 500px to 1000px create moderate perspective distortion, and settings of more than 2000px give very mild perspective distortion.*

Listing 7-5 sets the -webkit-perspective property using a slider.

Listing 7-5 Adding a perspective slider

```
<html>
<head>
   <title>applying perspective</title>
<style>
<script type="text/javascript">
function setView(value) {
   document.getElementById("showVal").innerHTML=value+"px";
   document.getElementById("view3d").style.webkitPerspective=value+"px";
}
</script>
</head>
<body>
<div id="view3d" style="-webkit-perspective: 400px;">
     <div class="tilted" style= "-webkit-transform: rotateX(45deg); -webkit-transform-origin: 50% 80%;">
     <p>
     Once upon a time, in a third dimension far, far, away, a paragraph of text went
     on and on and on and on and on and on and on and on and on and on and on and on
     on and on and on and on and on and on and on and on and on and on and on and on...
     </p>
     </div>
</div>
<p>Perspective:</p>
<input type="range" min="150" max="800" value="400" step="5" onchange="setView(value);">
<p id="showVal"> 400px </p>
</body>
</html>
```

We are introducing these concepts now so when we get to JavaScript it will be easier to understand

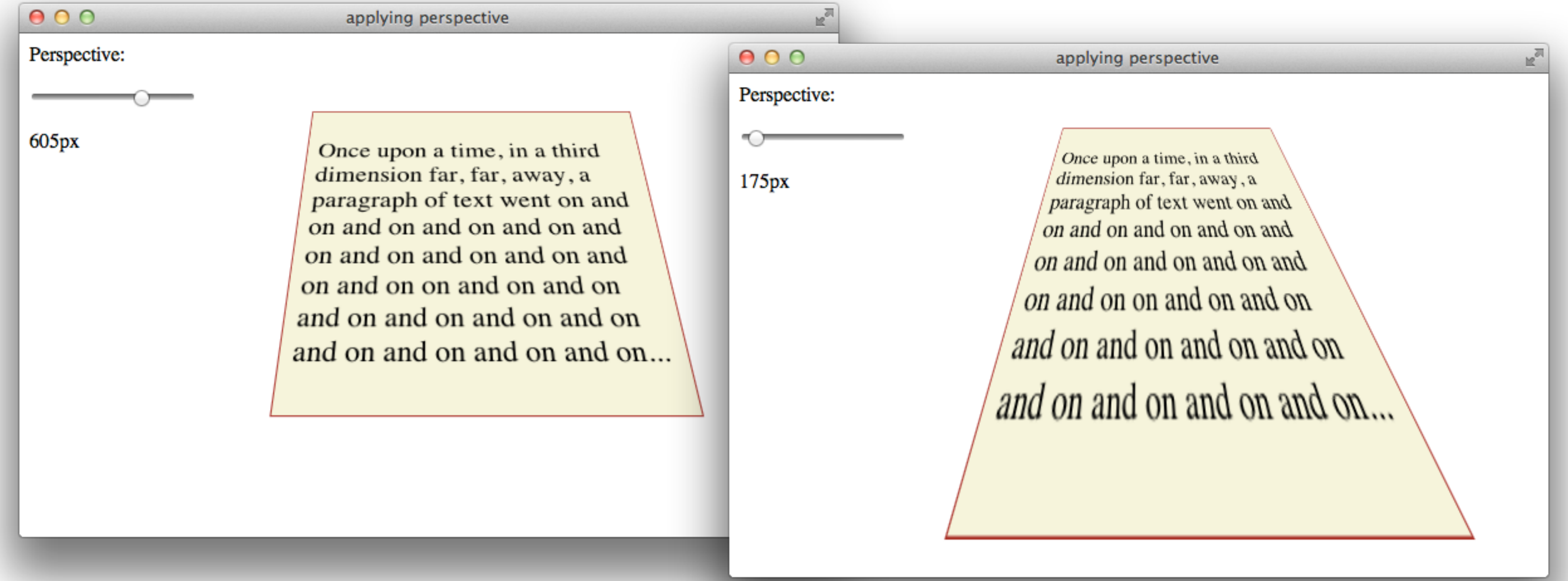


Figure 7-7 shows two perspective settings the from the previous example, in which a child element is rotated 45 degrees around the x-axis. The elements are shown at the same rotation and position in both cases, but with different perspective settings.

Figure 7-7 Setting the perspective above...

Important The -webkit-perspective property applies to the descendants of an element, not to transforms applied to the element itself. Generally speaking, you need to create a container that sets the -webkit-perspective property, enclosing all the elements that you want to have the appearance of depth. You can set the perspective on the body element if you like.

You can envision perspective view as a pyramid, with the point of the pyramid centered at the user's eye, and the base of the pyramid extending into the infinite distance. By setting the -webkit-perspective property, you specify the viewpoint's z-coordinate—how far the point of the pyramid lies above the screen. By default, the x- and y-coordinates of the viewpoint are the center of the element to which the -webkit-perspective property belongs.

Shift the viewpoint horizontally or vertically by setting the -webkit-perspective-origin property. The default setting is -webkit-perspective-origin: 50% 50%. To set the viewpoint above the top-left corner of an element, for example, set the element's style to -webkit-perspective-origin: 0px 0px.

The perspective origin can affect the visibility of elements. For example, if an element is rotated 90 degrees out of the default plane, it is viewed edge-on: if the perspective origin is directly in front of the element, the element is invisible; but if the origin is off-center from the element, one side of the element can be seen. For example, Listing 7-6 creates three div elements, all rotated 90 degrees. The elements positioned on the left and right of the window are visible. When the window is sized appropriately, however, the element in the center of the window is edge-on to the viewer and cannot be seen, as Figure 7-8 illustrates.

We are introducing these concepts now so when we get to JavaScript it will be easier to understand

Listing 7-6 Effects of perspective origin

```
<html>
<head>
   <title>perspective origin</title>
</head>
<style>
.body { width: 200px; height: 200px;
       background-color: beige; border: 1px solid brown;
       font-size: 150%; padding: 10px;
       -webkit-transform: rotateY(90deg);}
</style>
<body style="-webkit-perspective: 10cm;">
    <div class="body" style="position:absolute; left:50px; top: 70;">
    <h2>Left</h2>
   </div>
   <div class="body" style="position:absolute; left:250px; top: 70;">
    <h2>Center</h2>
   </div>
   <div class="body" style="position:absolute; left:450px; top: 70;">
    <h2>Right</h2>
   </div>
   <p align="center">The center element is edge-on, and not visible.</p>
</body>
</html>
```

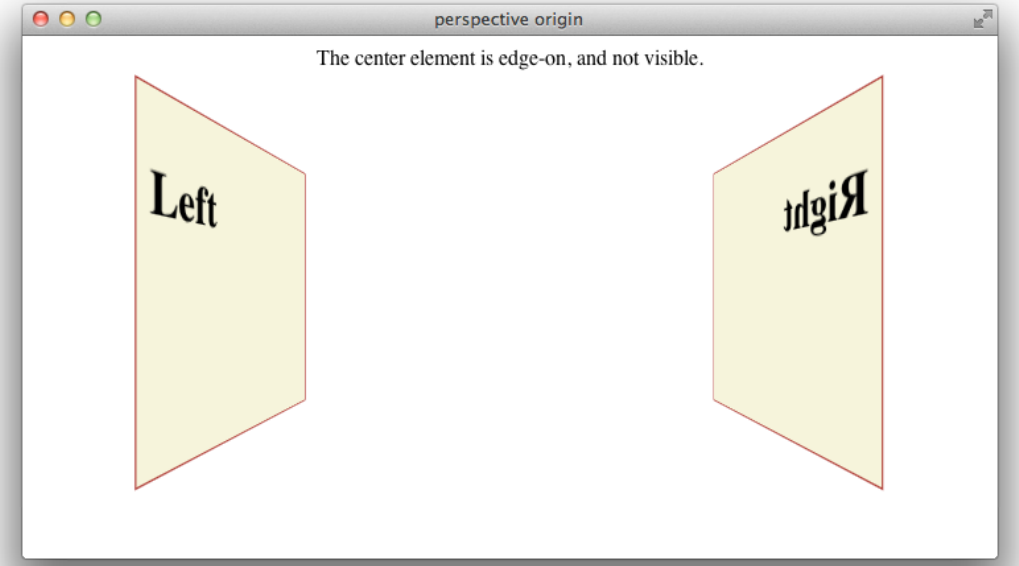


Figure 7-8 Perspective origin effects

Safari Developer Library

ABOUT WEB ELEMENTS

Creating a 3D Space

By default, the descendants of an element are flattened into the plane of their parent. When you apply a 3D transform to an element, that element's plane is no longer the default xy plane—the plane of the display. All descendants of the element share that element's new plane. In order to further transform the children of an element relative to that element's plane, you must set the -webkit-transform-style property to preserve-3d, creating a 3D space. For example:

```
<div id="space3d" style="-webkit-transform-style: preserve-3d;">
```

Setting the transform style to preserve-3d in an element makes that element into a 3D container; all the element's immediate children can be manipulated independently in 3D, relative to the parent. Because HTML elements are flat, a transformed child also occupies a plane in 3D space. Each child can occupy a separate plane, or multiple children can share the same plane. By default, any descendants of these transformed children are flattened into their parental plane; setting the transform style to preserve-3d affects only an element's immediate children.

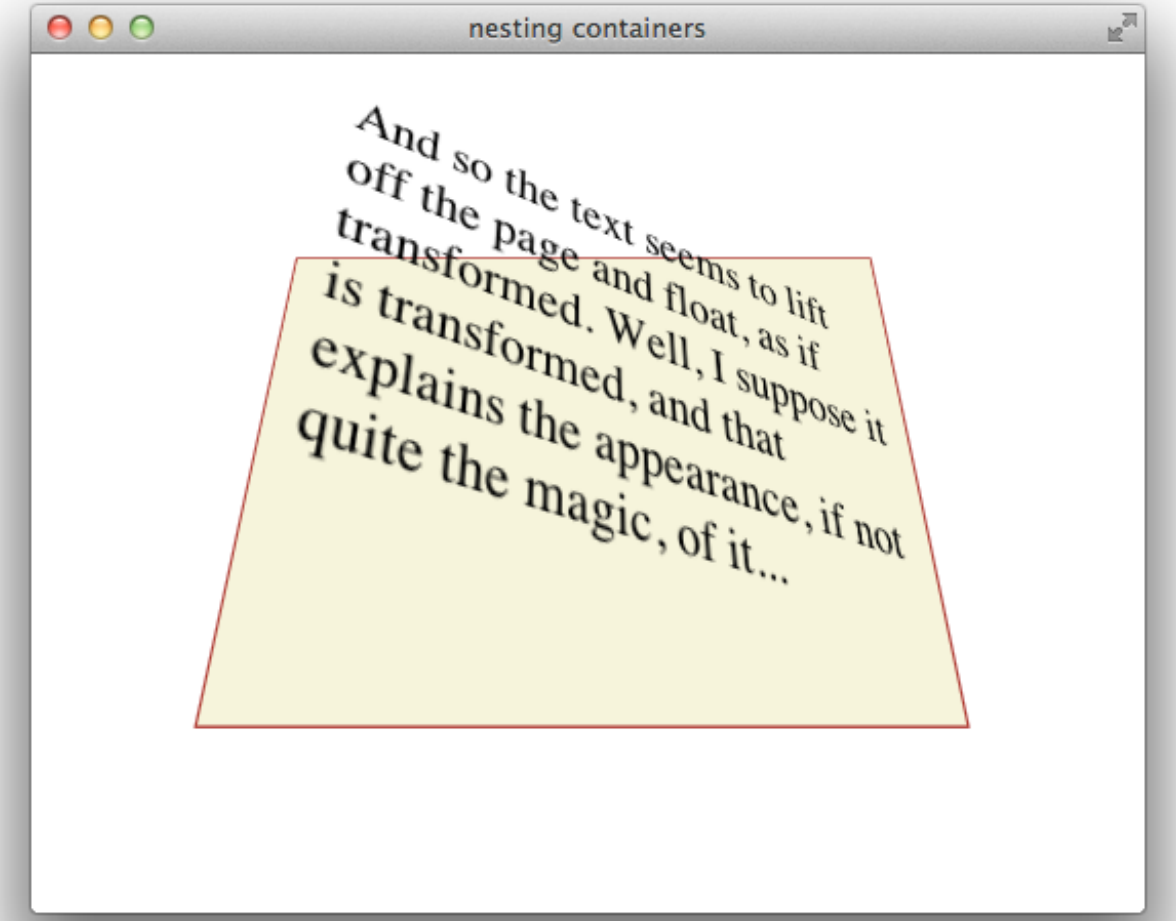
3D containers can be nested. Enabling 3D transforms in one of a container element's descendants creates a nested 3D layer; children of that descendant can be transformed in 3D, relative to their container's plane. You need to enable 3D in a particular element only if the element's children are to be transformed in 3D relative to that element's plane.

Any transform applied to a 3D container is inherited by all of its descendants. By applying a rotation to the highest level 3D container, for example, you are able to rotate the view of all of the container's contents at once.

Listing 7-7 gives an example of a nested pair of 3D containers, illustrated in Figure 7-9. The topmost container has a child div element rotated 45 degrees on its x-axis, so it appears to be tilted away from the viewer. This child div is also a 3D container, containing a paragraph of text rotated 35 degrees on its right edge away from the container, causing the text to appear to lift off the page.

Listing 7-7 Nested 3D rotations

```
<body>
<div id="view3d" style="-webkit-perspective: 15cm;   -webkit-transform-style: preserve-3d;">
    <div id="nested3dContainer" class="tilted" style= " -webkit-transform-style: preserve-3d;
-webkit-transform: rotateX(45deg); ">
<p style="-webkit-transform: rotateY(35deg); -webkit-transform-origin: 100% 50%;">
        And so the text seems to lift off the page and float, as if transformed.
        Well, I suppose it is transformed, and that explains the appearance,
        if not quite the magic, of it...
        </p>
    </div>
</div>
</body>
```



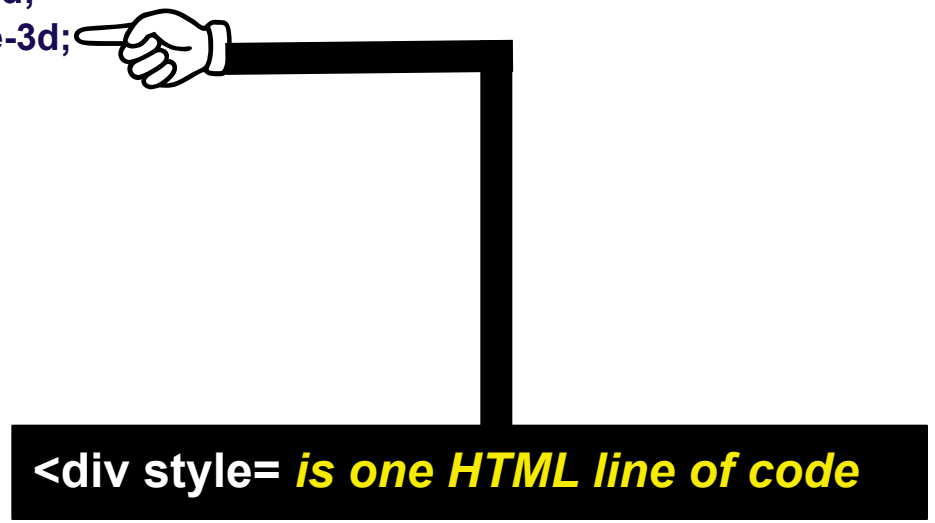


Figure 7-9 Text rotated relative to 3D backdrop

We are introducing these concepts now so when we get to JavaScript it will be easier to understand

Generally speaking, you need to create only a single 3D container—all 3D transforms can be applied relative to the default xy plane, and global transforms can be applied to the top-level container and inherited by all its descendants. Sometimes it may be convenient to manipulate a subgroup of transformed elements as a unit, however; in such a case, it makes sense to create a nested container.

To disable 3D for an element's children dynamically, set the -webkit-transform-style property to flat. Applying a 3D transform when the transform style is set to flat does not move the element out of its parent's plane.

Note: *Elements that have overflow set to hidden are unable to render their child elements in 3D, and are rendered as though the transform style were set to flat.*

We are introducing these concepts now so when we get to JavaScript it will be easier to understand

3D Transform Functions

Like 2D transforms, 3D transforms are set using the -webkit-transform property. You can apply a transform to an element by specifying a transform function, a list of transform functions, or by passing in a 3D matrix. There are several functions that perform 3D transforms:

- translateZ(distance)—Moves an element closer or farther away.
- translate3d(x, y, z)—Moves an element in three dimensions.
- rotateX(degrees)—Rotates an element around the x-axis, moving the top and bottom closer or farther away.
- rotateY(degrees)—Rotates an element around the y-axis, moving the left and right sides closer or farther away.
- perspective(distance)—Sets the 3D perspective for a single element.

Important *You must set a perspective for 3D transforms to have a visible 3D effect.*

We are introducing these concepts now so when we get to JavaScript it will be easier to understand

3D Translation

3D translation moves an element closer to or farther from the viewer by changing its position on the z-axis. Use the translateZ() function to shift an element on the z-axis, or the translate3d(x, y, z) function to shift an element on two or three axes. For example:

```
<div style="-webkit-transform: translateZ(100px);">

<div style="-webkit-transform: translate3d(100px 10cm -100px);">
```

The 3D translation functions work like their 2D counterparts, except that the z offset cannot be specified as a percentage. Z-axis units may be positive (towards the viewer) or negative (away from the viewer). Figure 7-10 shows two identical div elements with same height, width, and x and y positions, one translated on the z-axis by 100 px and the other translated by -100 px.

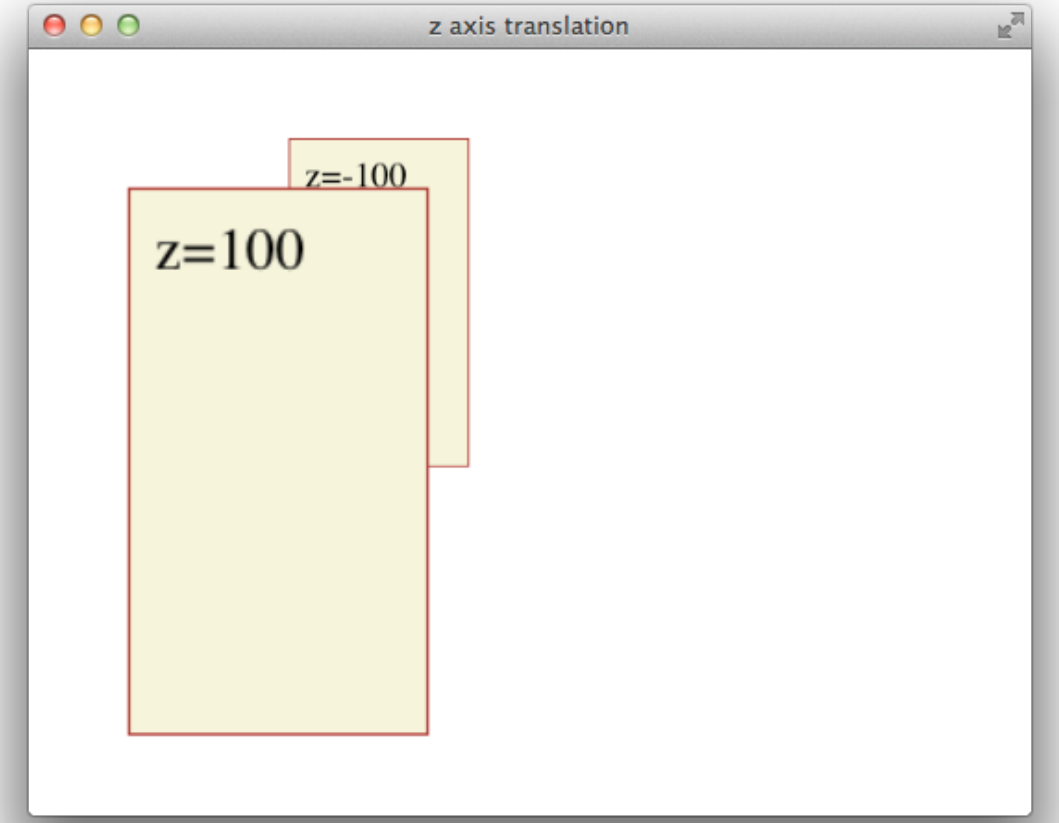


Figure 7-10 Z-axis translation in perspective

All descendants of an element inherit its z-axis translation. Note that the text in the previous illustration is translated along with its parent.

We are introducing these concepts now so when we get to JavaScript it will be easier to understand

3D Rotation

You can rotate an element in 3D either around the y-axis, so that the right and left edges get nearer and farther away, or around the x-axis, so that the top and bottom edges get nearer and farther away. For example:

```
<div style="-webkit-transform: rotateX(45deg);">

<div style="-webkit-transform: rotateY(-45deg);">
```

Rotation is around an imaginary x- or y-axis that passes through the element's origin. By default, the origin is at the center of an object. Positive x units rotate the top edge away. Positive y units rotate the right edge away. This is illustrated in Figure 7-11.

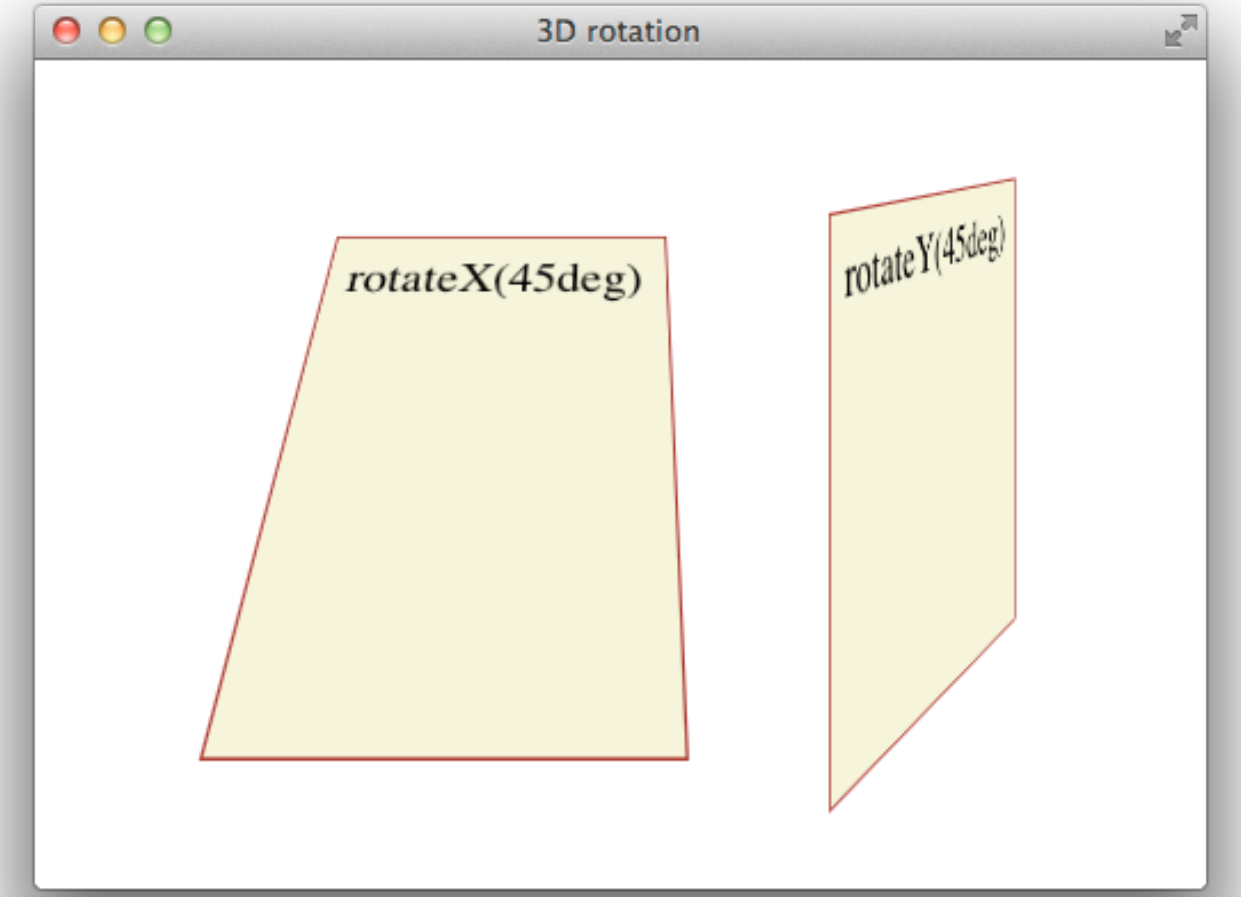


Figure 7-11 X and y rotation

All descendants of an element inherit its 3D rotation. Note that the text in the previous illustration is rotated along with its parent.

Setting Perspective for a Single Element

To create a 3D space with a shared perspective, you need to create a 3D container that has the -webkit-perspective property set; but if you just want to render a single element with the appearance of depth, you can set the -webkit-transform property to a list of transform functions that includes the perspective(distance) function as the first transform. For example:

```
<div style="-webkit-transform: perspective(10cm)  rotateX(45deg);">
    <p> This text is rotated 45 degrees around its x-axis. </p>
</div>
```

The foregoing snippet performs two 3D transforms on an element—rotation about the x-axis and perspective distortion, as if the user were viewing the object from 10 cm in front of the screen.

Important The perspective() transform function must be the first function in the list of transforms—you must establish the perspective prior to applying the other transforms.

In almost all cases, it is better to create a 3D container and set the -webkit-perspective property for the container element than it is to apply the perspective() transform to an element directly.

We are introducing these concepts now so when we get to JavaScript it will be easier to understand

Back Face Visibility

If an element is rotated 90 degrees or more around the x- or y-axis, the back face of the element faces the user. The back face of an element is always transparent, so the user sees a reversed image of the front face through the transparent back face, like a sign painted on a glass door and seen from behind. To prevent the mirror image of the front face from being displayed, set the **-webkit-backface-visibility** property to hidden. For example:

```
-webkit-backface-visibility: hidden;
```

When -webkit-backface-visibility is set to hidden, an element is not displayed where its back face would be visible. One reason to do this is to create the illusion that an element has two faces, each with their own content. For example, to create the illusion of a card with different contents on the front and back face, two elements are positioned back to back in the same location. The two elements are then rotated together, progressively hiding the front element and revealing the back element. If the back face of the top element were visible, it would obscure the element beneath it instead of revealing the element beneath it as it rotates. Listing 7-8 creates the illusion of a card with content on both sides, as Figure 7-12 shows.

Listing 7-8 Hiding the back side of a card

Safari Developer Library

```
<html>
<head>
   <title>flip card</title>
<style>
body  { -webkit-transform-style: preserve-3d; -webkit-perspective: 600px; }
.card { position: absolute; left: 100px; top: 100px; padding: 50px;
        height: 200px; width: 100px; border-radius: 10px; border: 1px solid tan;
        -webkit-backface-visibility: hidden;
        -webkit-transition: -webkit-transform 1s;}
.front { background: seashell; }
.back  { background: beige; -webkit-transform: rotateY(180deg);}
</style>
<script type="text/javascript">
var state = 1;
function flipCard() {
   if (state) {
      document.getElementById("front").style.webkitTransform="rotateY(-180deg)";
      document.getElementById("back").style.webkitTransform="rotateY(0deg)";
      state = 0;
   } else {
      document.getElementById("front").style.webkitTransform="rotateY(0deg)";
      document.getElementById("back").style.webkitTransform="rotateY(180deg)";
      state = 1;
   }
}
</script>
</head>
<body>
<div class="card back" id="back" onclick="flipCard();">Back of card</div>
<div class="card front" id="front" onclick="flipCard();">Front of card</div>
</body>
</html>
```

We are introducing these concepts now so when we get to JavaScript it will be easier to understand

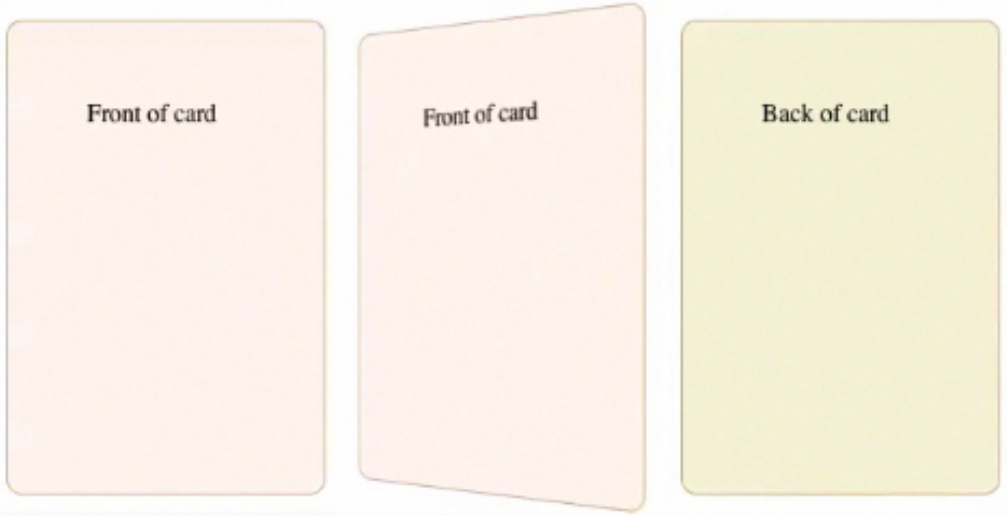
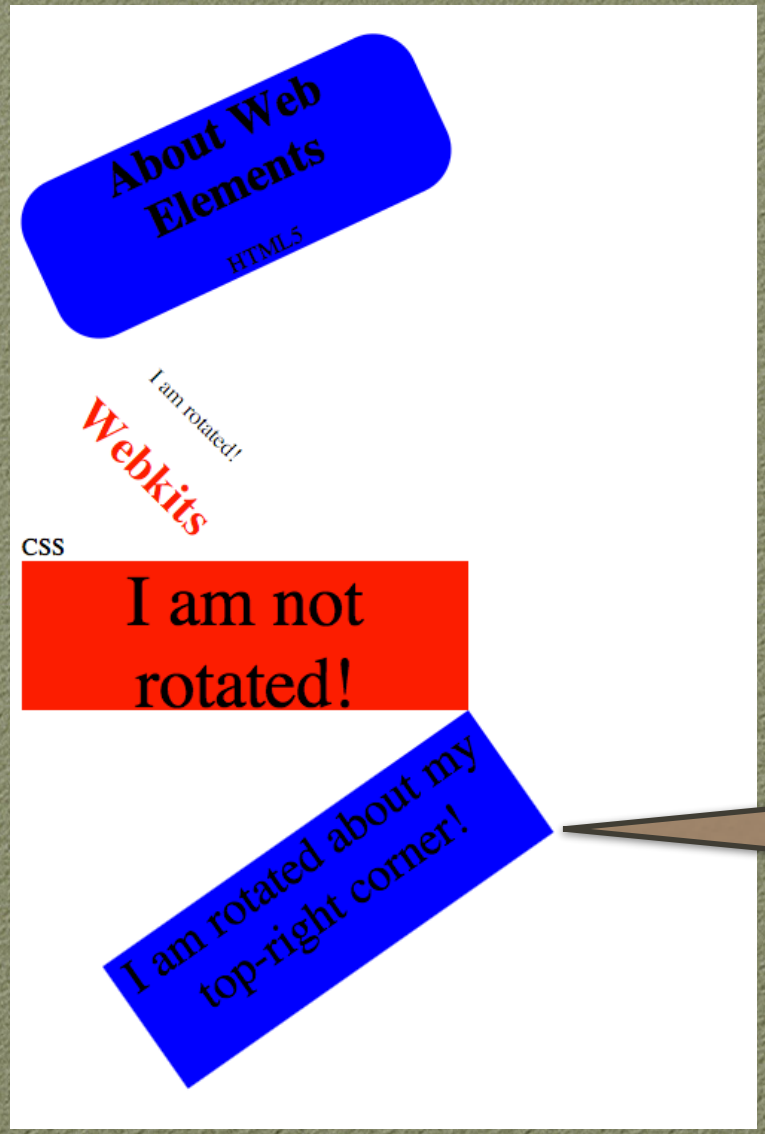


Figure 7-12 Cardflip example

ABOUT WEB ELEMENTS

Safari Developer Library



CSS
Cascading Style Sheets
Mr. K's Technology

Using the information you learned about perspective and rotation, build a page with both

Webkits

ABOUT WEB ELEMENTS

SAVE YOUR WORK

CLOSE ALL OPEN PROGRAMS

ABOUT WEB ELEMENTS

LOG OFF

CSS

Cascading Style Sheets

Mr. K's Technology

Safari Developer Library