Lesson 7

Transforms

In this Lesson 7

CSS

Share

With CSS3 came new ways to position and alter elements. Now general layout techniques can be revisited with alternative ways to size, position, and change elements. All of these new techniques are made possible by the `transform` property.

The `transform` property comes in two different settings, two-dimensional and three-dimensional. Each of these come with their own individual properties and values.

Within this lesson we'll take a look at both two-dimensional and three-dimensional transforms. Generally speaking, browser support for the `transform` property isn't great, but it is getting better every day. For the best support vendor prefixes are encouraged, however you may need to download the nightly version of Chrome to see all of these transforms in action.

Transform Syntax

The actual syntax for the `transform` property is quite simple, including the transform property followed by the value. The value specifies the transform type followed by a specific amount inside parentheses.

```
1  div {
2    -webkit-transform: scale(1.5);
3       -moz-transform:
4  scale(1.5);
5          -o-transform:
6  scale(1.5);
7             transform:
   scale(1.5);
   }
```

Notice how the `transform` property includes multiple vendor prefixes to gain the best support across all browsers. The un-prefixed declaration comes last to overwrite the prefixed versions, should a browser fully support the transform property.

In the interest of brevity, the remainder of this lesson will not include vendor prefixes. They are, however, strongly encouraged for any code in a production environment. Over time we will be able to remove these prefixes, however keeping them in is the safest approach for the time being.

2D Transforms

Elements may be distorted, or transformed, on both a two-dimensional plane or a three-dimensional plane. Two-dimensional transforms work on the `x` and `y` axes, known as horizontal and vertical axes. Three-dimensional transforms work on both the `x` and `y` axes, as well as the `z` axis. These three-dimensional transforms help define not only the length and width of an element, but also the depth. We'll start by discussing how to transform elements on a two-dimensional plane, and then work our way into three-dimensional transforms.

2D Rotate

The `transform` property accepts a handful of different values. The `rotate` value provides the ability to rotate an element from `0` to `360` degrees. Using a positive value will rotate an element clockwise, and using a negative value will rotate the element counterclockwise. The default point of rotation is the center of the element, `50% 50%`, both horizontally and vertically. Later we will discuss how you can change this default point of rotation.

HTML

```
<figure class="box-1">Box
1</figure>
<figure class="box-2">Box
2</figure>
```

CSS

```
.box-1 {
  transform: rotate(20deg);
}
.box-2 {
  transform: rotate(-55deg);
}

```

The gray box behind the rotated element symbolizes the original position of the element. Additionally, upon hover the box will rotate 360 degrees horizontally. As the lesson progresses, keep an eye out for the gray box within each demonstration as a reference to the element's original position and the horizontal rotation to help demonstrate an elements alteration and depth.

2D Scale

Using the `scale` value within the `transform` property allows you to change the appeared size of an element. The default scale value is `1`, therefore any value between `.99` and `.01` makes an element appear smaller while any value greater than or equal to `1.01` makes an element appear larger.

HTML

```
<figure class="box-1">Box
1</figure>
<figure class="box-2">Box
2</figure>
```

CSS

```
.box-1 {
  transform: scale(.75);
}
.box-2 {
  transform: scale(1.25);
}

```

It is possible to scale only the height or width of an element using the `scaleX` and `scaleY` values. The `scaleX` value will scale the width of an element while the `scaleY` value will scale the height of an element. To scale both the height and width of an element but at different sizes, the `x` and `y` axis values may be set simultaneously. To do so, use the `scale` transform declaring the `x` axis value first, followed by a comma, and then the `y` axis value.

HTML

```
<figure class="box-1">Box
1</figure>
<figure class="box-2">Box
2</figure>
<figure class="box-3">Box
3</figure>
```

CSS

```
.box-1 {
  transform: scaleX(.5);
}
.box-2 {
  transform: scaleY(1.15);
}
.box-3 {
  transform: scale(.5, 1.15);
}

```

2D Translate

The `translate` value works a bit like that of relative positioning, pushing and pulling an element in different

directions without interrupting the normal flow of the document. Using the `translateX` value will change the position of an element on the horizontal axis while using the `translateY` value will change the position of an element on the vertical axis.

As with the `scale` value, to set both the `x` and `y` axis values at once, use the `translate` value and declare the `x` axis value first, followed by a comma, and then the `y` axis value.

The distance values used within the `translate` value may be any general length measurement, most commonly pixels or percentages. Positive values will push an element down and to the right of its default position while negative values will pull an element up and to the left of its default position.

HTML

```
<figure class="box-1">Box
1</figure>
<figure class="box-2">Box
2</figure>
<figure class="box-3">Box
3</figure>
```

CSS

```
.box-1 {
  transform: translateX(-10px);
}
.box-2 {
  transform: translateY(25%);
}
.box-3 {
  transform: translate(-10px, 25%);
}

```

2D Skew

The last `transform` value in the group, `skew`, is used to distort elements on the horizontal axis, vertical axis, or both. The syntax is very similar to that of the `scale` and `translate` values. Using the `skewX` value distorts an element on the horizontal axis while the `skewY` value distorts an element on the vertical axis. To distort an element on both axes the `skew` value is used, declaring the `x` axis value first, followed by a comma, and then the `y` axis value.%p

The distance calculation of the `skew` value is measured in units of degrees. Length measurements, such as pixels or percentages, do not apply here.

HTML

```
<figure class="box-1">Box
1</figure>
<figure class="box-2">Box
2</figure>
<figure class="box-3">Box
3</figure>
```

CSS

```
.box-1 {
  transform: skewX(5deg);
}
.box-2 {
  transform: skewY(-20deg);
}
.box-3 {
  transform: skew(5deg, -20deg);
}

```

Combining Transforms

It is common for multiple transforms to be used at once, rotating and scaling the size of an element at the same time for example. In this event multiple transforms can be combined together. To combine transforms, list the transform values within the `transform` property one after the other without the use of commas.

Using multiple `transform` declarations will not work, as each declaration will overwrite the one above it. The behavior in that case would be the same as if you were to set the `height` of an element numerous times.

HTML

```
<figure class="box-1">Box
1</figure>
<figure class="box-2">Box
2</figure>
```

CSS

```
.box-1 {
  transform: rotate(25deg) scale(.75);
}
.box-2 {
  transform: skew(10deg, 20deg) translateX(20px);
}

```

Behind every transform there is also a matrix explicitly defining the behavior of the transform. Using the `rotate`, `scale`, `transition`, and `skew` values provide an easy way to establish this matrix. However, should you be mathematically inclined, and prefer to take a deeper dive into transforms, try your hand at using the `matrix` property.

2D Cube Demo

HTML

```
<div class="cube">
  <figure class="side top">1</figure>
  <figure class="side left">2</figure>
  <figure class="side
right">3</figure>
</div>
```

CSS

```
.cube {
  position: relative;
}
.side {
  height: 95px;
  position: absolute;
  width: 95px;
}
.top {
  background: #9acc53;
  transform: rotate(-45deg) skew(15deg, 15deg);
}
.left {
  background: #8ec63f;
  transform: rotate(15deg) skew(15deg, 15deg) translate(-50%, 100%);
}
.right {
  background: #80b239;
  transform: rotate(-15deg) skew(-15deg, -15deg) translate(50%, 100%);
}

```

Demo

Transform Origin

As previously mentioned, the default transform origin is the dead center of an element, both `50%` horizontally and `50%` vertically. To change this default origin position the `transform-origin` property may be used.

The `transform-origin` property can accept one or two values. When only one value is specified, that value is used for both the horizontal and vertical axes. If two values are specified, the first is used for the horizontal axis and

the second is used for the vertical axis.

Individually the values are treated like that of a background image position, using either a length or keyword value. That said, `0` is the same value as `left`, and `100% 100%` is the same value as `right bottom`. More specific values can also be set, for example `20px 50px` would set the origin to 20 pixels across and 50 pixels down the element.

HTML

```
<figure class="box-1">Box 1</figure>
<figure class="box-2">Box 2</figure>
<figure class="box-3">Box 3</figure>
<figure class="box-4">Box 3</figure>
```

CSS

```
.box-1 {
  transform: rotate(15deg);
  transform-origin: 0 0;
}
.box-2 {
  transform: scale(.5);
  transform-origin: 100% 100%;
}
.box-3 {
  transform: skewX(20deg);
  transform-origin: top left;
}
.box-4 {
  transform: scale(.75) translate(-10px, -10px);
  transform-origin: 20px 50px;
}

```

Notably, the `transform-origin` property does run into some issues when also using the `translate` transform value. Since both of them are attempting to position the element, their values can collide. Use the two of these with caution, always checking to make sure the desired outcome is achieved.

Perspective

In order for three-dimensional transforms to work the elements need a perspective from which to transform. The perspective for each element can be thought of as a *vanishing point*, similar to that which can be seen in three-dimensional drawings.

The perspective of an element can be set in two different ways. One way includes using the `perspective` value within the `transform` property on individual elements, while the other includes using the `perspective` property on the parent element residing over child elements being transformed.

Using the `perspective` value within the `transform` property works great for transforming one element from a single, unique perspective. When you want to transform a group of elements all with the same perspective, or vanishing point, apply the `perspective` property to their parent element.

The example below shows a handful of elements all transformed using their individual perspectives with the `perspective` value.

HTML

```
<figure class="box">Box
1</figure>
<figure class="box">Box
2</figure>
<figure class="box">Box
3</figure>
```

CSS

```
.box {
  transform: perspective(200px) rotateX(45deg);
}

```

The following example shows a handful of elements, side by side, all transformed using the same perspective, accomplished by using the `perspective` property on their direct parent element.

HTML

```
<div class="group">
  <figure class="box">Box
1</figure>
  <figure class="box">Box
2</figure>
  <figure class="box">Box
3</figure>
</div>
```

CSS

```
.group {
  perspective: 200px;
}
.box {
  transform: rotateX(45deg);
}

```

Perspective Depth Value

The perspective value can be set as `none` or a length measurement. The `none` value turns off any perspective, while the length value will set the depth of the perspective. The higher the value, the further away the perspective appears, thus creating a fairly low intensity perspective and a small three-dimensional change. The lower the value the closer the perspective appears, thus creating a high intensity perspective and a large three-dimensional change.

Imagine yourself standing 10 feet away from a 10 foot cube as compared to standing 1,000 feet away from the same cube. At 10 feet, your distance to the cube is the same as the dimensions of the cube, therefore the perspective shift is much greater than it will be at 1,000 feet, where the dimensions of the cube are only one one-hundredth of your distance to the cube. The same thinking applies to perspective depth values.

HTML

```
<figure class="box-1">Box
1</figure>
<figure class="box-2">Box
2</figure>
```

CSS

```
.box-1 {
  transform: perspective(100px) rotateX(45deg);
}
.box-2 {
  transform: perspective(1000px) rotateX(45deg);
}

```

Perspective Origin

As with setting a `transform-origin` you can also set a `perspective-origin`. The same values used for the `transform-origin` property may also be used with the `perspective-origin` property, and maintain the same relationship to the element. The large difference between the two falls where the origin of a transform determines the coordinates used to calculate the change of a transform, while the origin of a perspective identifies the coordinates of the vanishing point of a transform.

HTML

```
<div class="original original-1">
  <figure class="box">Box
1</figure>
</div>
<div class="original original-2">
  <figure class="box">Box
2</figure>
</div>
<div class="original original-3">
  <figure class="box">Box
3</figure>
</div>
```

css

```
.original {
  perspective: 200px;
}
.box {
  transform: rotateX(45deg);
}
.original-1 {
  perspective-origin: 0 0;
}
.original-2 {
  perspective-origin: 75% 75%;
}
.original-3 {
  perspective-origin: 20px 40px;
}

```

3D Transforms

Working with two-dimensional transforms we are able to alter elements on the horizontal and vertical axes, however there is another axis along which we can transform elements. Using three-dimensional transforms we can change elements on the `z` axis, giving us control of depth as well as length and width.

3D Rotate

So far we've discussed how to rotate an object either clockwise or counterclockwise on a flat plane. With three-dimensional transforms we can rotate an element around any axes. To do so, we use three new `transform` values, including `rotateX`, `rotateY`, and `rotateZ`.

Using the `rotateX` value allows you to rotate an element around the `x` axis, as if it were being bent in half horizontally. Using the `rotateY` value allows you to rotate an element around the `y` axis, as if it were being bent in half vertically. Lastly, using the `rotateZ` value allows an element to be rotated around the `z` axis.

As with the general `rotate` value before, positive values will rotate the element around its dedicated axis clockwise,

while negative values will rotate the element counterclockwise.

HTML

```
<figure class="box-1">Box
1</figure>
<figure class="box-2">Box
2</figure>
<figure class="box-3">Box
3</figure>
```

CSS

```
.box-1 {
  transform: perspective(200px) rotateX(45deg);
}
.box-2 {
  transform: perspective(200px) rotateY(45deg);
}
.box-3 {
  transform: perspective(200px) rotateZ(45deg);
}

```

3D Scale

By using the `scaleZ` three-dimensional transform elements may be scaled on the `z` axis. This isn't extremely exciting when no other three-dimensional transforms are in place, as there is nothing in particular to scale. In the demonstration below the elements are being scaled up and down on the `z` axis, however the `rotateX` value is added in order to see the behavior of the `scaleZ` value. When removing the `rotateX` in this case, the elements will appear to be unchanged.

HTML

```
<figure class="box-1">Box
1</figure>
<figure class="box-2">Box
2</figure>
```

CSS

```
.box-1 {
  transform: perspective(200px) scaleZ(1.75) rotateX(45deg);
}
.box-2 {
  transform: perspective(200px) scaleZ(.25) rotateX(45deg);
}

```

3D Translate

Elements may also be translated on the `z` axis using the `translateZ` value. A negative value here will push an element further away on the `z` axis, resulting in a smaller element. Using a positive value will pull an element closer on the `z` axis, resulting in a larger element.

While this may appear to be very similar to that of the two-dimensional transform `scale` value, it is actually quite different. The transform is taking place on the `z` axis, not the `x` or `y` axes. When working with three-dimensional transforms, being able to move an element on the `z` axis does have great benefits, like when building the cube below for example.

HTML

```
<figure class="box-1">Box
1</figure>
<figure class="box-2">Box
2</figure>
```

CSS

```
.box-1 {
  transform: perspective(200px) translateZ(-50px);
}
.box-2 {
  transform: perspective(200px) translateZ(50px);
}

```

3D Skew

Skew is the one two-dimensional transform that **cannot** be transformed on a three-dimensional scale. Elements may be skewed on the `x` and `y` axis, then transformed three-dimensionally as wished, but they cannot be skewed on the `z` axis.

Shorthand 3D Transforms

As with combining two-dimensional transforms, there are also properties to write out shorthand three-dimensional transforms. These properties include `rotate3d`, `scale3d`, `transition3d`, and `matrix3d`. These properties do

require a bit more math, as well as a strong understanding of the matrices behind each transform. Should you be interested in looking a bit deeper into them, please do!

Transform Style

On occasion three-dimensional transforms will be applied on an element that is nested within a parent element which is also being transformed. In this event, the nested, transformed elements will not appear in their own three-dimensional space. To allow nested elements to transform in their own three-dimensional plane use the `transform-style` property with the `preserve-3d` value.

The `transform-style` property needs to be placed on the parent element, above any nested transforms. The `preserve-3d` value allows the transformed children elements to appear in their own three-dimensional plane while the `flat` value forces the transformed children elements to lie flat on the two-dimensional plane.

HTML

```
<div class="rotate three-d">
  <figure class="box">Box
1</figure>
</div>
<div class="rotate">
  <figure class="box">Box
2</figure>
</div>
```

CSS

```
.rotate {
  transform: perspective(200px) rotateY(45deg);
}
.three-d {
  transform-style: preserve-3d;
}
.box {
  transform: rotateX(15deg) translateZ(20px);
  transform-origin: 0 0;
}

```

To see an additional example of the `transform-style` property in action check out the WebKit explanation.

Backface Visibility

When working with three-dimensional transforms, elements will occasionally be transformed in a way that causes them to face away from the screen. This may be caused by setting the `rotateY(180deg)` value for example. By default these elements are shown from the back. So if you prefer not to see these elements at all, set the `backface-visibility` property to `hidden`, and you will hide the element whenever it is facing away from the screen.

The other value to `backface-visibility` is `visible` which is the default value, always displaying an element, no matter which direction it faces.

In the demonstration below notice how the second box isn't displayed because `backface-visibility: hidden;` declaration has been set. The `backface-visibility` property takes even more significance when using animations.

HTML

```
<figure class="box-1">Box 1</figure>
<figure class="box-2">Box 2</figure>
```

CSS

```
.box-1 {
  transform: rotateY(180deg);
}
.box-2 {
  backface-visibility: hidden;
  transform: rotateY(180deg);
}

```

3D Cube Demo

HTML

```
<div class="cube-container">
  <div class="cube">
    <figure class="side front">1</figure>
    <figure class="side back">2</figure>
    <figure class="side left">3</figure>
    <figure class="side right">4</figure>
    <figure class="side top">5</figure>
    <figure class="side bottom">6</figure>
  </div>
</div>
```

CSS

```
.cube-container {
  height: 200px;
  perspective: 300;
  position: relative;
  width: 200px;
}
.cube {
  height: 100%;
  position: absolute;
  transform: translateZ(-100px);
  transform-style: preserve-3d;
  width: 100%;
}
.side {
  background: rgba(45, 179, 74, .3);
  border: 2px solid #2db34a;
  height: 196px;
  position: absolute;
  width: 196px;
}
.front {
  transform: translateZ(100px);
}
.back {
  transform: rotateX(180deg) translateZ(100px);
}
.left {
  transform: rotateY(-90deg) translateZ(100px);
}
.right {
  transform: rotateY(90deg) translateZ(100px);
}
.top {
  transform: rotateX(90deg) translateZ(100px);
}
.bottom {
  transform: rotateX(-90deg) translateZ(100px);
}

```

Demo

Resources & Links