TEXT WRAP WITH CSS SHAPES

Excerpt from:
Learning Web Design, 5e

by Jennifer Robbins

Author's Note: *This article originally appeared in "Chapter 15, Floating and Positioning" in Learning Web Design, 5e. It was removed from the 6th edition due to space limitations, but I have made it available here to provide additional detail and practice. Be aware that some information may be out of date.*

Look at the previous float examples and you will see that the text always wraps in a rectangular shape around a floated image or element box. However, you can change the shape of the wrapped text to a circle, ellipse, polygon, or any image shape using the **`shape-outside`** property. This is an up-and-coming CSS feature, so be sure to check the Browser Support Note. Following is a quick introduction to CSS Shapes, which should inspire and prepare you for more exploration on your own.

`shape-outside`

Values:	`none` | `circle()` | `ellipse()` | `polygon()` | `url()` | [`margin-box` | `padding-box` | `content-box`]
Default:	none
Applies to:	floats
Inherits:	no

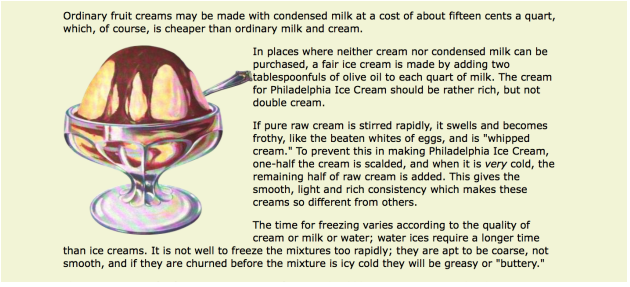
FIGURE A shows the default text wrap around a floated image (left) and the same wrap with **`shape-outside`** applied right). This is the kind of thing you'd expect to see in a print magazine, but now we can do it on the web!

It is worth noting that you can change the text wrap shape around any floated element (see **Note**), but I will focus on images in this discussion, as text elements are generally boxes that fit nicely in the default rectangular wrap.

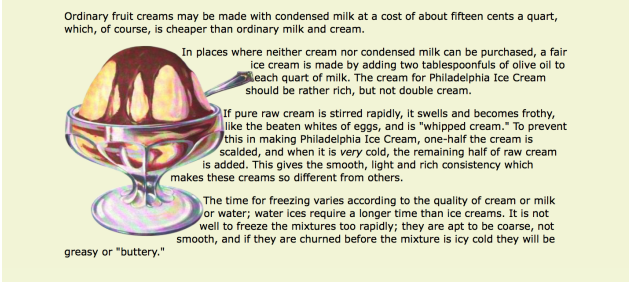
IN THIS ARTICLE

NOTE

`shape-outside` *only works on floated elements for now, but it is believed that will change in the future.*



Default text wrap



Text wrap with **shape-outside** using the transparent areas of the image as a guide

FIGURE A. Example of text wrapping around an image with **shape-outside**

There are two approaches to making text wrap around a shape. One way is to provide the path coordinates of the wrap shape with **circle()**, **ellipse()**, or **polygon()**. Another way is to use **url()** to specify an image that has transparent areas (such as a GIF or a PNG). With the image method, text flows into the transparent areas of the image and stops at the opaque areas. This is the shape method shown in FIGURE A and the method I'll introduce first.

Using a transparent image

In the example in FIGURE A, I placed the *sundae.png* image in the HTML document to display on the page, and I've specified the same image in the style rule using **url()** so that its transparent areas define the wrap shape. It makes sense to use the same image in the document and for the CSS shape, but it is not required. You could apply a wrap shape derived from one image to another image on the page.

THE MARKUP

```
<p><img src="sundae.png" class="wrap" alt=""> In places…</p>
```

THE STYLES

```
img.wrap {
  float: left;
  width: 300px;
  height: 300px;
  -webkit-shape-outside: url(sundae.png);   /* prefix required
in 2018 */
   shape-outside: url(sundae.png);
```

Notice that the wrapped text is now bumping right into the image. How about we give it a little extra space with **shape-margin**?

WARNING

There is a security setting in Chrome and Opera that make image-based text wraps a little tricky to use. Without getting into too much sys-admin detail, the browser restricts the use of the image used to create the CSS shape if it isn't on the same domain as the file requesting it. This is not a bug; they are following the rules set out in the specification.

The rule also means that compliant browsers won't allow images to be used for shapes when the files are served locally (i.e. on your own computer). They need to be uploaded to a server to work, which makes the design process a little more cumbersome, especially for beginners.

If you use image-based text wraps, you know your CSS is written correctly, but you aren't seeing wrapping in the browser, this security setting (related to Cross Origin Resource Sharing, CORS, if you're curious) is probably the culprit.

shape-margin

Values: *length* | *percentage*

Default: 0

Applies to: floats

Inherits: no

The **shape-margin** property specifies an amount of space to hold between the shape and the wrapped text. In FIGURE B, you can see the effect of adding 1em of space between the opaque image areas and the wrapped text lines. It gives it a little breathing room the way any good margin should.

```
-webkit-shape-margin: 1em;
shape-margin: 1em;
```

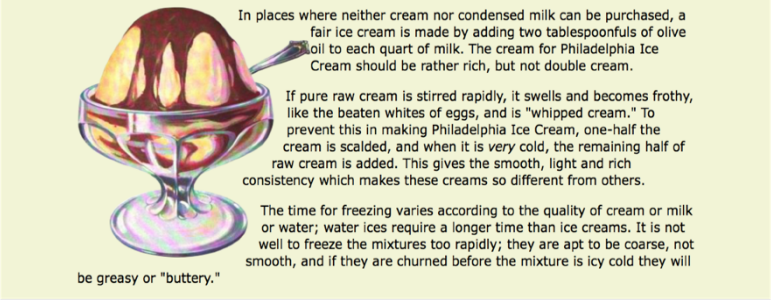


FIGURE B. Adding a margin between the shape and the wrapped text.

Using a path

The other method for creating a text wrap shape is to define it using one of the path keywords: **circle()**, **ellipse()**, and **polygon()**.

Here is a code sample that creates a circle shape for the text to wrap around. The value provided in the **circle()** notation represents the length of the radius of the circle.

```
circle(radius)
```

In this example, the radius is 150px, half of the image width of 300 pixels. By default, the circle is centered vertically and horizontally on the float.

```
img.round {
  float: left;
  -webkit-shape-outside: circle(150px);
  shape-outside: circle(150px);
}
```

FIGURE C shows this style rule applied to different images. Notice that the transparency of the image is not at play here. It's just a path overlaid on the image that sets the boundaries for text wrap. Any path can be applied to any image or other floated element.

Opacity Threshold

If you have a source image with multiple levels of transparency, such as the gradient shadow the **shape-image-threshold** property allows text to creep into the image but stop when it encounters a specific transparency level. The value of this property is a number between 0 and 1, representing a percentage of transparency. For example, if you set the threshold to .2, text will wrap into areas that are up to 20% transparent, but stop when it gets to more opaque levels.

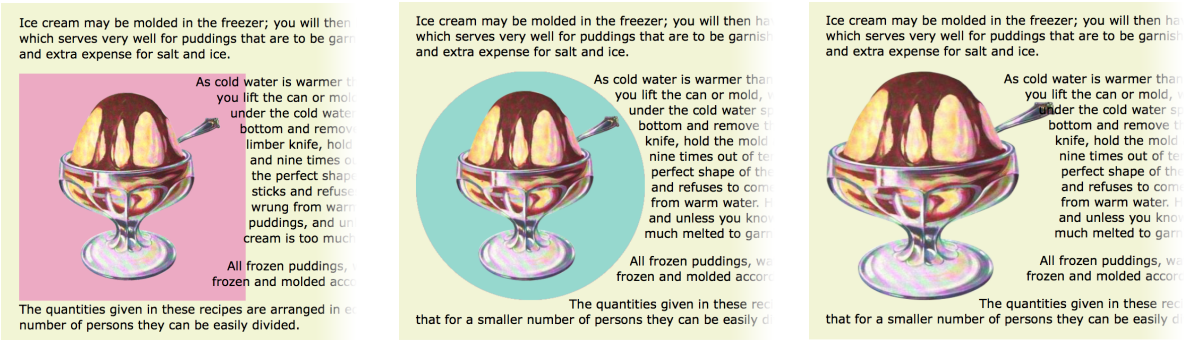


FIGURE 15-C. The same `circle()` shape applied to different images in the source.

This is a good point to demonstrate a critical behavior of wrap shapes. They allow text to flow *into* the floated image or element, but they cannot hold space free beyond it.

In the example in FIGURE D, I've increased the diameter of the circle path from 150px to 200px. Notice that the text lines up along the right edge of the image, even though the circle is set 50 pixels beyond the edge. The path does not push text away from the float. If you need to keep wrapped text away from the outside edge of the floated image or element, apply a margin to the element itself (it will be the standard rectangular shape, of course).

```
img.round {
  float: left;
  -webkit-shape-outside: circle(200px);
  shape-outside: circle(200px);
}
```

CSS shapes allows text to wrap into floated elements, but do not push text away from them.

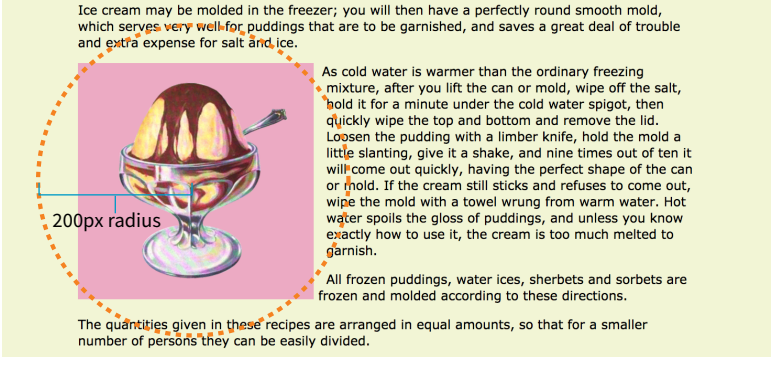


FIGURE D. CSS shapes allow text to wrap into the floated element but do not hold space beyond it.

Elliptical shapes are created with the **ellipse()** statement that provides the horizontal and vertical radius lengths followed by the word "at" then the x,y coordinates for the center of the shape. Here is the syntax:

```
ellipse(rx ry at x y);
```

The position coordinates can be listed as a specific measurement or a percentage. Here I've created an ellipse with a 100-pixel horizontal radius and a 150-pixel vertical radius, centered in the floated element it is applied to (FIGURE E):

```
img.round {
  float: left;
  -webkit-shape-outside: ellipse(200px 100px at 50% 50%);
  shape-outside: ellipse(200px 100px at 50% 50%);
}
```

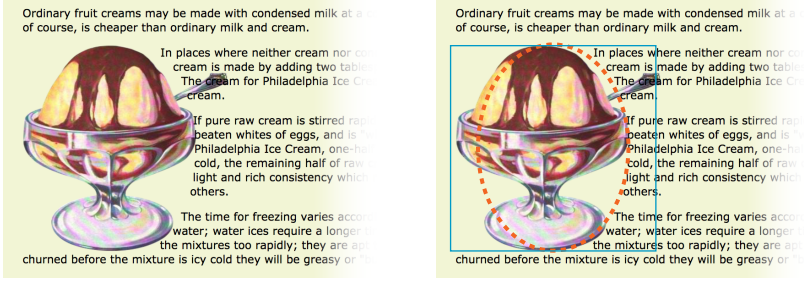


FIGURE E. An elliptical text wrap created with **ellipse()**.

Finally, we come to **polygon()**, which lets you create a custom path using a series of comma-separated x,y coordinates along the path. The style rule below creates the wrap effect shown in FIGURE F.

```
img.wrap {
  float: left;
  width: 300px;
  height: 300px;
  shape-outside: polygon(0px 0px, 186px 0px, 225px 34px,
300px 34px, 300px 66px, 255px 88px, 267px 127px, 246px
178px, 192px 211px, 226px 236px, 226px 273px, 209px 300px,
0px 300px);
}
```

FIGURE F. A custom path created with **`polygon()`**

Holy coordinates! That's a lot of numbers, and my path was fairly simple. I'd like to be able to point you to a great tool for drawing and exporting polygon paths, but sadly, as of this writing I have none to recommend (see Note). I gathered the coordinates for my polygon examples by opening the image in Photoshop and gathering them manually, which although possible, is definitely not ideal.

CSS Shapes resources

There are some finer points regarding CSS Shapes that I must leave to you to research further. Here are a few resources to get you started.

- CSS Shapes Module, Level 1 (w3.org/TR/css-shapes-1/)
- "Getting Started with CSS Shapes" by Razvan Caliman (html5rocks.com/en/tutorials/shapes/getting-started)
- CSS Shapes at The Experimental Layout Lab of Jen Simmons (labs.jensimmons.com/#shapes)
- "A Redesign with CSS Shapes" by Eric Meyer (alistapart.com/article/redesign-with-css-shapes)

TIP

If you search for "CSS Shapes" you will certainly come across that term used for a technique that uses CSS to draw geometric shapes such as triangles, arrows, circles, and so on. It's a little confusing, although those other "CSS shapes" are pretty nifty and something you might want to tinker with.