**JavaScript Basics #8: Drawing on Canvas**

Remember when we talked about HTML and CSS, we briefly introduced something called SVG? It allows us to create beautiful images by simply using HTML tags. Today, we are going to introduce something similar called canvas, except it allows us to use javascript to create graphics on web pages. And because it uses a programming language instead of a simple markup language, that makes canvas much more flexible and powerful compared to SVG.

Table of Contents

* [The Canvas](https://www.ericsdevblog.com/index.php/2021/11/javascript-basics-8-drawing-on-canvas/#The_Canvas)
* [Lines](https://www.ericsdevblog.com/index.php/2021/11/javascript-basics-8-drawing-on-canvas/#Lines)
* [Paths](https://www.ericsdevblog.com/index.php/2021/11/javascript-basics-8-drawing-on-canvas/#Paths)
* [Curves](https://www.ericsdevblog.com/index.php/2021/11/javascript-basics-8-drawing-on-canvas/#Curves)
* [Texts](https://www.ericsdevblog.com/index.php/2021/11/javascript-basics-8-drawing-on-canvas/#Texts)
* [Transformations](https://www.ericsdevblog.com/index.php/2021/11/javascript-basics-8-drawing-on-canvas/#Transformations)
* [Bitmap Graphics](https://www.ericsdevblog.com/index.php/2021/11/javascript-basics-8-drawing-on-canvas/#Bitmap_Graphics)

**THE CANVAS**

We know that the SVG has a DOM tree structure, and the shape, colour and position are all represented using HTML tags. The canvas, however, is one single HTML node, but it encapsulates a space on the web page, where you can create beautiful artworks using JavaScript. This space can be defined using the <canvas> tag. Here is an example where we create a simple rectangle inside the canvas space:

<canvas width="300px" height="200px"></canvas>

<script>

let canvas = document.querySelector("canvas");

let context = canvas.getContext("2d");

// Define the colour of the rectangle

context.fillStyle = "red";

// The first two parameters means that the top left corner of the ractagle is at coordinate (10, 10)

// The last two parameters define the width and height of the ractangle (width:100px, height:50px)

context.fillRect(10, 10, 100, 50);

</script>

The getContext() method is used to access the drawing interface, which is like a toolbox where your digital pens and pencils are stored. The parameter "2d" stands for two-dimensional graphics. If you are interested in creating three-dimensional graphics, you should use “webgl” instead. But we are only focusing on the 2D system for now.

Also, notice that the defined the size of the canvas at the beginning. If you don’t do that, the canvas element will take a default width of 300 pixels and a height of 150 pixels.

**LINES**

The rectangle we just created is solid, the inside of the rectangle is filled. What if we want something different? It is also possible for us to create a rectangle that is stroked instead, by using a very similar method, strokeRect(). This method also takes four parameters, the first two define the position and the last two define the size.

<canvas></canvas>

<script>

let canvas = document.querySelector("canvas");

let context = canvas.getContext("2d");

// Define the colour, position and size

context.strokeStyle = "blue";

context.strokeRect(10, 10, 100, 50);

// Define the width of the strok and create a new rectangle

context.lineWidth = 5;

context.strokeRect(150, 10, 100, 50);

</script>

**PATHS**

Now you might be wondering, that’s not so exciting, we can create rectangles using SVGs just as easily. Don’t worry, the real power of the canvas starts now.

First, we need to understand what a path is. A path is a sequence of lines. For example, we have a line that starts from coordinate (0, 0) to (0, 50), the second line from (0, 50) to (80, 50) and the third line from (80, 50) to (80, 100). These three lines will form a path.

The canvas allows us to do something like this:

<canvas width="500px" height="500px"></canvas>

<script>

let canvas = document.querySelector("canvas");

let context = canvas.getContext("2d");

context.lineWidth = 5;

context.strokeStyle = "green";

context.beginPath();

// The path starts at (10, 10)

context.moveTo(10, 10);

// Drawing the path: (10, 10) -> (150, 10) -> (150, 150) -> (10, 150) -> (10,10)

context.lineTo(150, 10);

context.lineTo(150, 150);

context.lineTo(10, 150);

context.lineTo(10, 10);

context.stroke();

</script>

With paths, we can create any shape we want. For example, the following code creates a triangle:

<canvas width="500px" height="500px"></canvas>

<script>

let canvas = document.querySelector("canvas");

let context = canvas.getContext("2d");

context.beginPath();

context.fillStyle = "red";

context.moveTo(200, 10);

context.lineTo(250, 100);

context.lineTo(150, 100);

context.lineTo(200, 10);

context.fill();

</script>

**CURVES**

A path could be formed by straight lines, and it could also be formed by curves. A curve, however, is a little bit more difficult to define.

To define a curve, we need a start point, a destination point, and a control point. The curve will not go through the control point directly, but instead, it defines a point where the tangent of the start and destination point goes through.

This is a little hard to understand. I suggest you get familiar with the pen tool in Photoshop or the path tool in GIMP first. They share the same concept, except when you are coding, you need to imagine what the curve looks like.

Here is another example. We’ll first draw the curve, and then draw the tangent lines and the control point, so that it helps you understand what’s going on here:

<canvas width="500px" height="500px"></canvas>

<script>

let canvas = document.querySelector("canvas");

let context = canvas.getContext("2d");

context.beginPath();

// start point = (10, 90)

context.moveTo(10, 90);

// control point = (60,10); destination point = (90,90)

context.quadraticCurveTo(60, 10, 90, 90);

// destination point tangent

context.lineTo(60, 10);

// start point tangent

context.moveTo(10, 90);

context.lineTo(60, 10);

context.closePath();

context.stroke();

</script>

A picture containing shape

Description automatically generated

Sometimes we want the start point tangent and the destination point to have different control points. That is also possible to achieve using the bezierCurveTo() method.

<canvas width="500px" height="500px"></canvas>

<script>

let canvas = document.querySelector("canvas");

let context = canvas.getContext("2d");

context.beginPath();

// start point = (10, 90)

context.moveTo(10, 90);

// start control point = (60,10); destination control point = (30,80); destination point = (90,90)

context.bezierCurveTo(60, 10, 30, 80, 90, 90);

// destination point tangent

context.lineTo(30, 80);

// start point tangent

context.moveTo(10, 90);

context.lineTo(60, 10);

context.closePath();

context.stroke();

</script>

A picture containing shape

Description automatically generated

**TEXTS**

Texts might also be useful when we are creating graphs. We can draw texts using either fillText and strokeText. The latter will only render the outline of the texts instead of filling it.

<canvas width="1500px" height="500px"></canvas>

<script>

let canvas = document.querySelector("canvas");

let context = canvas.getContext("2d");

context.font = "28px Georgia";

context.fillText("Lorem ipsum dolor sit amet, consectetur adipiscing elit.", 10, 50);

context.strokeText("Lorem ipsum dolor sit amet, consectetur adipiscing elit.", 10, 100);

</script>

Text

Description automatically generated

The last two parameters indicate the position of the text, but unlike drawing shapes, it defines the coordinate of the start of the text’s baseline. The baseline is the line that the text “stands” on.

**TRANSFORMATIONS**

There are primarily three types of transformations, translation (translate()), scale (scale()) and rotation (rotate()). Remember that these methods need to be put **before** the graph you wish to transform.

Translation will move the graph from one position to another:

<canvas width="1500px" height="1500px"></canvas>

<script>

let canvas = document.querySelector("canvas");

let context = canvas.getContext("2d");

// Move whatever graph created after to the right for 50px and downward for 100px

context.translate(50, 100);

// Create a graph

context.beginPath();

context.fillStyle = "red";

context.moveTo(200, 10);

context.lineTo(250, 100);

context.lineTo(150, 100);

context.lineTo(200, 10);

context.fill();

</script>

The scale will make the original graph bigger or smaller:

<script>

let canvas = document.querySelector("canvas");

let context = canvas.getContext("2d");

// Make the graph 2 times wider (along x-axis) 0.5 time shorter (along y-axis)

context.scale(2, 1/2);

// Create a graph

...

</script>

And finally, we can rotate the graph along an axis:

<canvas width="1500px" height="1500px"></canvas>

<script>

let canvas = document.querySelector("canvas");

let context = canvas.getContext("2d");

// Rotate the graph clockwise for 18 degrees. Notice that the rotate() method takes radian instead of degree.

context.rotate(0.1 \* Math.PI);

// Create a graph

...

</script>

**BITMAP GRAPHICS**

In computer graphics, there is something called vector graphic and bitmap graphic. All the graphs we’ve been talking about so far are vector graphics. Their primary difference is that the bitmap graphics are formed by pixels while the vector graphics are not. Instead, they are formed by paths, with a direction and a magnitude (length), like a vector.

However, it is necessary for us sometimes to insert some bitmap graphics in our vector graphic design. We can do that by using the drawImage() method.

<canvas width="1500px" height="1500px"></canvas>

<script>

let canvas = document.querySelector("canvas");

let context = canvas.getContext("2d");

let img = document.createElement("img");

img.src = "cat.jpg";

img.addEventListener("load", () => {

context.drawImage(img, 10, 10, 360, 240);

});

</script>

In this example, the image will be drawn at the coordinate (10, 10), with the size 360px \* 240px.

We need to add the event listener because the image will load after the canvas without it, so we have to make the canvas wait for the image to load first.