**Unit IV**

**HTML5 – CANVAS**

**HTML5 – CANVAS: The Rendering Context, Browser Support, HTML5 Canvas Examples, Canvas - Drawing Rectangles, Canvas - Drawing Paths, Canvas - Drawing Lines, Canvas - Drawing Bezier Curves, Canvas - Drawing Quadratic Curves, Canvas - Using Images, Canvas - Create Gradients**

**4.1 Introduction**

SVG is vector based, so it does not work well for images with lots of fine details and textures like photographs. SVG is best suited for logos, icons and other graphics that use simpler color and shapes.

The HTML <canvas> element is used to draw graphics on a web page. It is used to draw graphicsvia scripting (usually JavaScript). Canvas has several methods for drawing paths, boxes, circles, text, and adding images. The <canvas> element is only a container for graphics. We must use a script to actually draw the graphics.

Canvas can draw colorful text, with or without animation.Canvas objects can move. Using canvas, from simple bouncing balls to complex animations are possible. Canvas can respond to JavaScript events. It can respond to any user action such as key press, mouse clicks, button clicks and finger movement.

<canvas>element gives us an easy and powerful way to draw graphics using JavaScript. It can be used to draw graphs, make photo compositions or do simple (and not so simple) animations.

The **<canvas>** element defines a **bitmapped** area in an HTML page.The **Canvas API** allows JavaScript to **draw graphics** on the canvas.The Canvas API can draw shapes, lines, curves, boxes, text, and images, with colors, rotations, transparencies, and other pixel manipulations.We can add a canvas element anywhere in an HTML page with the <canvas> tag.

The HTML canvas is a two-dimensional grid. The upper-left corner of the canvas has the coordinates (0,0).

**HTML SVG Vs HTML Canvas**

* SVG is a language for describing 2D graphics in XML whereas Canvas draws 2D graphicswith JavaScript.
* If attributes of an SVG object are changed, the browser can automatically re-render the shape whereas Canvas is rendered pixel by pixel. In canvas, once the graphic is drawn, it is forgotten by the browser.
* SVG is resolution independent whereas CANVAS is resolution-dependent.
* SVG is XML based, which means that every element is available within the SVG DOM. We can attach JavaScript event handlers for an element.

**Difference between HTML SVG and HTML Canvas**

|  |  |
| --- | --- |
| **SVG** | **Canvas** |
| SVG is vector based | Canvas is raster based |
| Resolution dependent | Resolution independent |
| Poor text rendering capabilities | Best suited for applications with large rendering areas (Google Maps) |
| SVG can be modified through script and CSS | Canvas can be modified through script only |
| Comparatively slow | Comparatively faster. so, better for quick animations |
| supportsevent handlers | Supports event handlers |

**Referring canvas in JavaScript**

**<canvas id = "mycanvas" width = "100" height = "100">**

**</canvas>**

We can easily find that <canvas> element in the DOM (Document Object Model) using *getElementById()* method as follows −

**var canvas = document.getElementById("mycanvas");**

**4.2 The Rendering Context**

The <canvas> is initially blank, and to display something, a script first needs to access the rendering context and draw on it.

The canvas element has a DOM method called **getContext**, used to obtain the rendering context and its drawing functions. This function takes one parameter, the type of context**2d**.

Following is the code to get required context along with a check if your browser supports <canvas> element −

***const c = document.getElementById("myCanvas");***

***const ctx = c.getContext("2d");***

## Properties and Methods of Canvas

## Drawing Methods

There are only 3 methods to draw directly on the canvas:

|  |  |
| --- | --- |
| **Method** | **Description** |
| [fillRect()](https://www.w3schools.com/tags/canvas_fillrect.asp) | Draws a "filled" rectangle |
| [strokeRect()](https://www.w3schools.com/tags/canvas_strokerect.asp) | Draws a rectangle (with no fill) |
| [clearRect()](https://www.w3schools.com/tags/canvas_clearrect.asp) | Clears specified pixels within a rectangle |

## Path Methods

|  |  |
| --- | --- |
| **Method** | **Description** |
| [beginPath()](https://www.w3schools.com/tags/canvas_beginpath.asp) | Begins a new path or resets the current path |
| [closePath()](https://www.w3schools.com/tags/canvas_closepath.asp) | Adds a line to the path from the current point to the start |
| [isPointInPath()](https://www.w3schools.com/tags/canvas_ispointinpath.asp) | Returns true if the specified point is in the current path |
| [moveTo()](https://www.w3schools.com/tags/canvas_moveto.asp) | Moves the path to a point in the canvas (without drawing) |
| [lineTo()](https://www.w3schools.com/tags/canvas_lineto.asp) | Adds a line to the the path |
| [fill()](https://www.w3schools.com/tags/canvas_fill.asp) | Fills the current path |
| [rect()](https://www.w3schools.com/tags/canvas_rect.asp) | Adds a rectangle to the path |
| [stroke()](https://www.w3schools.com/tags/canvas_stroke.asp) | Draws the current path |
| **Circles and Curves** | |
| [bezierCurveTo()](https://www.w3schools.com/tags/canvas_beziercurveto.asp) | Adds a cubic Bézier curve to the path |
| [arc()](https://www.w3schools.com/tags/canvas_arc.asp) | Adds an arc/curve (circle, or parts of a circle) to the path |
| [arcTo()](https://www.w3schools.com/tags/canvas_arcto.asp) | Adds an arc/curve between two tangents to the path |
| [quadraticCurveTo()](https://www.w3schools.com/tags/canvas_quadraticcurveto.asp) | Adds a quadratic Bézier curve to the path |

## Text Methods

|  |  |
| --- | --- |
| **Method/Prop** | **Description** |
| [direction](https://www.w3schools.com/tags/canvas_direction.asp) | Sets or returns the direction used to draw text |
| [fillText()](https://www.w3schools.com/tags/canvas_filltext.asp) | Draws "filled" text on the canvas |
| [font](https://www.w3schools.com/tags/canvas_font.asp) | Sets or returns the font properties for text content |
| [measureText()](https://www.w3schools.com/tags/canvas_measuretext.asp) | Returns an object that contains the width of the specified text |
| [strokeText()](https://www.w3schools.com/tags/canvas_stroketext.asp) | Draws text on the canvas |
| [textAlign](https://www.w3schools.com/tags/canvas_textalign.asp) | Sets or returns the alignment for text content |
| [textBaseline](https://www.w3schools.com/tags/canvas_textbaseline.asp) | Sets or returns the text baseline used when drawing text |

## Colors, Styles, and Shadows

|  |  |
| --- | --- |
| **Method/Property** | **Description** |
| [addColorStop()](https://www.w3schools.com/tags/canvas_addcolorstop.asp) | Specifies the colors and stop positions in a gradient object |
| [createLinearGradient()](https://www.w3schools.com/tags/canvas_createlineargradient.asp) | Creates a linear gradient (to use on canvas content) |
| [createPattern()](https://www.w3schools.com/tags/canvas_createpattern.asp) | Repeats a specified element in the specified direction |
| [createRadialGradient()](https://www.w3schools.com/tags/canvas_createradialgradient.asp) | Creates a radial/circular gradient (to use on canvas content) |
| [fillStyle](https://www.w3schools.com/tags/canvas_fillstyle.asp) | Sets or returns the color, gradient, or pattern used to fill the drawing |
| [lineCap](https://www.w3schools.com/tags/canvas_linecap.asp) | Sets or returns the style of the end caps for a line |
| [lineJoin](https://www.w3schools.com/tags/canvas_linejoin.asp) | Sets or returns the type of corner created, when two lines meet |
| [lineWidth](https://www.w3schools.com/tags/canvas_linewidth.asp) | Sets or returns the current line width |
| [shadowBlur](https://www.w3schools.com/tags/canvas_shadowblur.asp) | Sets or returns the blur level for shadows |
| [shadowColor](https://www.w3schools.com/tags/canvas_shadowcolor.asp) | Sets or returns the color to use for shadows |
| [shadowOffsetX](https://www.w3schools.com/tags/canvas_shadowoffsetx.asp) | Sets or returns the horizontal distance of the shadow from the shape |
| [shadowOffsetY](https://www.w3schools.com/tags/canvas_shadowoffsety.asp) | Sets or returns the vertical distance of the shadow from the shape |
| [strokeStyle](https://www.w3schools.com/tags/canvas_strokestyle.asp) | Sets or returns the color, gradient, or pattern used for strokes |

## Transformations Methods

|  |  |
| --- | --- |
| **Method** | **Description** |
| [scale()](https://www.w3schools.com/tags/canvas_scale.asp) | Scales the current drawing bigger or smaller |
| [rotate()](https://www.w3schools.com/tags/canvas_rotate.asp) | Rotates the current drawing |
| [translate()](https://www.w3schools.com/tags/canvas_translate.asp) | Remaps the (0,0) position on the canvas |
| [transform()](https://www.w3schools.com/tags/canvas_transform.asp) | Replaces the current transformation matrix for the drawing |
| [setTransform()](https://www.w3schools.com/tags/canvas_settransform.asp) | Resets the current transform to the identity matrix. Then runs [transform()](https://www.w3schools.com/tags/canvas_transform.asp) |

## Image Drawing

|  |  |
| --- | --- |
| **Method** | **Description** |
| [drawImage()](https://www.w3schools.com/tags/canvas_drawimage.asp) | Draws an image, canvas, or video onto the canvas |

## 4.3 Browser Support

canvas is supported in all modern browsers such as

* Chrome
* Edge
* Firefox
* Safari
* Opera
* IE 9-11

**4.4 HTML5 Canvas Examples**

[Live Demo](http://tpcg.io/lfhwVW)

**Example 4.1 Simple canvas demo**

<html>

<head>

<style>

#mycanvas{border:1px solid red;}

</style>

</head>

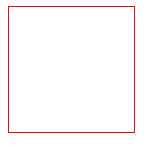
<body>

<canvas id = "mycanvas" width = "100" height = "100"></canvas>

</body>

</html>

output



**Note**

The <canvas> element must have an id attribute so it can be referred to by JavaScript. The width and height attribute is necessary to define the size of the canvas. In addition we can define all the core HTML5 attributes like id, name and class, etc. We can have multiple <canvas> elements on one HTML page. By default, the <canvas> element has no border and no content.

**Example 4.2 Simple canvas demo**

<!DOCTYPE html>

<html>

<body>

<h1>HTML5 Canvas</h1>

<h2>The fillRect() Method</h2>

<canvas id="myCanvas" width="300" height="150" style="border:1px solid grey"></canvas>

<script>

const c = document.getElementById("myCanvas");

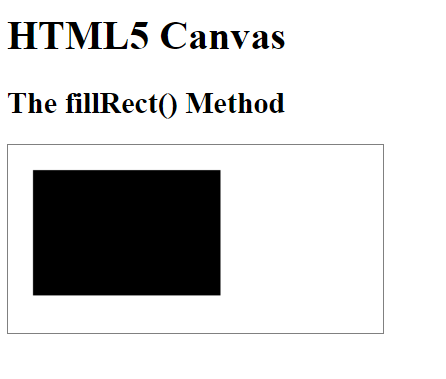
const ctx = c.getContext("2d");

ctx.fillRect(20, 20, 150, 100);

</script>

</body>

</html>



**Example 4.3 Canvas Text Example**

<!DOCTYPE html>

<html>

<body>

<canvas id="myCanvas" width="600" height="100" style="border:4px solid slateblue;">

<script>

var c = document.getElementById("myCanvas");

var ctx = c.getContext("2d");

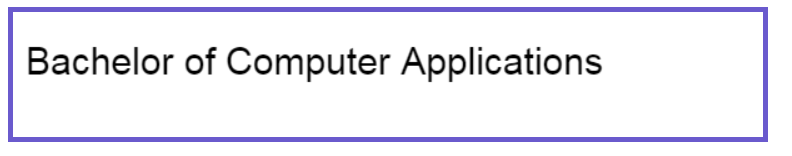
ctx.font = "30px Arial";

ctx.fillText("Bachelor of Computer Applications",10,50);

</script>

</body>

</html>



**4.5 Canvas - Drawing Rectangles**

 In HTML5 canvas, fillRect() method is used to draw rectangle. It takes the general format,

***fillRect(x, y, width, height)***

where

x--> x coordinate

y--> y coordinate

5width -->width of the rectangle

height -->height of the rectangle

**Example 4.4 Canvas Rectangle Example**

<!DOCTYPE html>

<html>

<body>

<h1>HTML5 Canvas</h1>

<canvas id="myCanvas" width="200" height="100" style="border:1px solid grey;"></canvas>

<script>

const canvas = document.getElementById("myCanvas");

const ctx = canvas.getContext("2d");

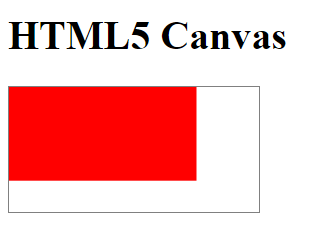
ctx.fillStyle = "#FF0000";

ctx.fillRect(0,0,150,75);

</script>

</body>

</html>

****

**4.6 Canvas - Drawing Paths**

The path is nothing but forming/drawing a basic shape constructed between two points, it can be defined as a list of points, connected by segments of lines, arcs, curves, etc. that can be used to make different shapes.

We can draw a path using the methods provided by the paths2D interface of HTML5 canvas.Path elements are various basic elements like curves, lines, and arcs used to form a path. Following are the methods provided by HTML5 Canvas API to draw various path elements

|  |  |
| --- | --- |
| Method | Description |
| moveTo() | We use a virtual pointer while drawing with the path. It is always located at a specified point which can be accessed using the moveTo(x, y) method |
| lineTo() | This method draws a line from the virtual pointer to the point given as parameters in the lineTo() method. Before drawing a line, we must use the moveTo() function to send the cursor to the starting point from which the line should be drawn. |
| arcTo() | This method will draw an arc using the path. It takes two points and a radius as parameters. The arc is drawn from start point to end point with the curvature using radius. |
| quadraticCurveTo() | This function will draw a curve using one control point which is taken in reference to drawing the curve between the two points. |
| bezierCurveTo() | This function will draw a curve between two points using two control points which determines the structure of the curve. |

**Example 4.5 Canvas path Example**

<!DOCTYPE html>

<html>

<body>

<canvas id="myCanvas" width="200" height="100" style="border:1px solid #d3d3d3;">

<script>

var c = document.getElementById("myCanvas");

var ctx = c.getContext("2d");

ctx.moveTo(0,0);

ctx.lineTo(200,100);

ctx.stroke();

</script>

</body>

</html>

**Example 4.6 Canvas path Example**

<html>

<body>

<canvas id="myCanvas" width="200" height="100" style="border:1px solid #d3d3d3;">

<script>

var c = document.getElementById("myCanvas");

var ctx = c.getContext("2d");

ctx.beginPath();

ctx.arc(95,50,40,0,2\*Math.PI);

ctx.stroke();

</script>

</body>

</html>

**4.7 Canvas - Drawing Lines**

To draw a straight line on a canvas, we can use the following methods:

* moveTo(*x,y*) - defines the starting point of the line
* lineTo(*x,y*) - defines the ending point of the line

# To draw the line, we must use one of the "ink" methods, like stroke

|  |  |
| --- | --- |
| **Method** | **Description** |
| beginPath() | This method resets the current path |
| moveTo(x, y) | This method creates a new subpath with the given point. |
| closePath() | This method marks the current subpath as closed, and starts a new subpath with a point the same as the start and end of the newly closed subpath. |
| fill() | This method fills the subpaths with the current fill style. |
| stroke() | This method strokes the subpaths with the current stroke style. |
| lineTo(x, y) | This method adds the given point to the current subpath, connected to the previous one by a straight line. |

**Example 4.7 Canvas path**

<html>

<body>

<canvas id="myCanvas" width="220" height="220" style="border:3px solid #d3d3d3;">

<script>

var c = document.getElementById("myCanvas");

var ctx = c.getContext("2d");

// Filled triangle

ctx.beginPath();

ctx.moveTo(25,25);

ctx.lineTo(105,25);

ctx.lineTo(25,105);

ctx.fill();

// Stroked triangle

ctx.beginPath();

ctx.moveTo(125,125);

ctx.lineTo(125,45);

ctx.lineTo(45,125);

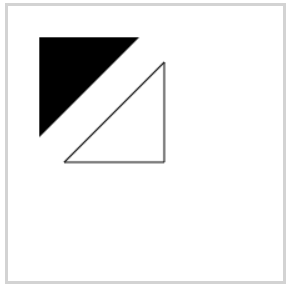
ctx.closePath();

ctx.stroke();

</script>

</body>

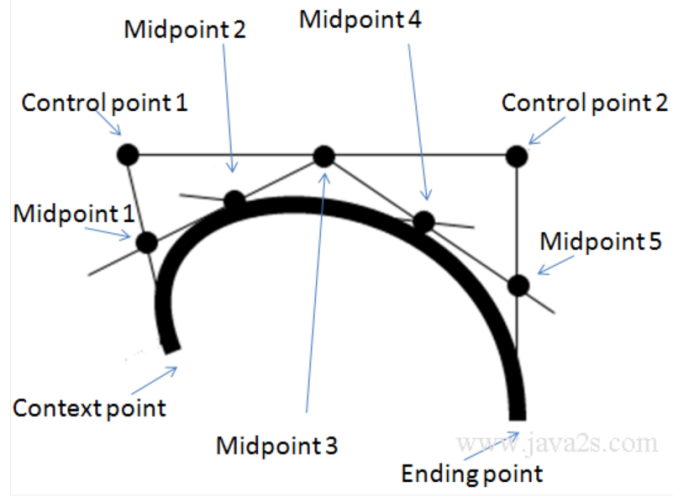
</html>



**4.8 Canvas - Drawing Bezier Curves**

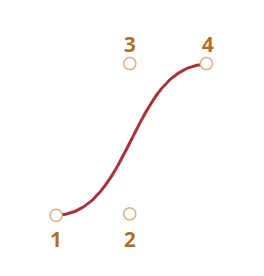
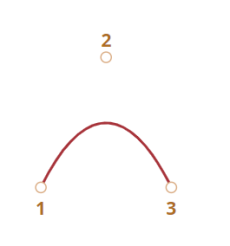
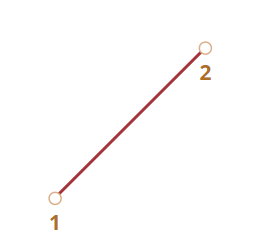
**Bezier Curve** is parametric curve defined by a set of control points. Two points are ends of the curve. Other points determine the shape of the curve.The shape of the curve is not disturbed easily because changing one of the control points does not change the whole shape of the curve. The Bezier Curve has the global control over the curve.Bezier curves are used in computer graphics to draw shapes, for CSS animation and in many other places.

Curves on HTML canvas can be drawn using arcs, but drawing a complex diagram using arcs is quite a tedious task. In the given circumstance, Bezier curve will be very useful in providing more flexibility in drawing curves. Bezier curves on HTML canvas are drawn using a start point, one or more control point/points and an endpoint.



The bezierCurveTo() method adds a curve to the path by using the control points that represent a cubic Bézier curve. We can Use the stroke() or fill() method to draw the path.

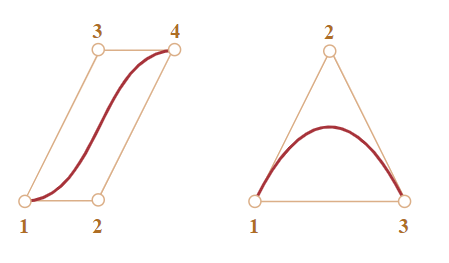
A [bezier curve](https://en.wikipedia.org/wiki/B%C3%A9zier_curve) is defined by control points.There may be 2, 3, 4 or more.For instance, two points curve:



2 points 3 points 4 points

In thesecurves,

1. **Points are not always on curve.**
2. **The curve order equals the number of points minus one**. For two points we have a linear curve (that’s a straight line), for three points – quadratic curve (parabolic), for four points – cubic curve.
3. **A curve is always inside the**[**convex hull**](https://en.wikipedia.org/wiki/Convex_hull)**of control points:**



Because of that last property, in computer graphics it’s possible to optimize intersection tests. Checking the intersection of convex hulls is much easier, because they are rectangles, triangles and so on (see the picture above), much simpler figures than the curve.

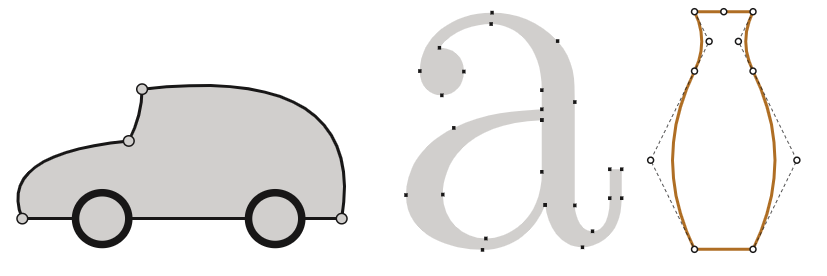
**The main value of Bezier curves for drawing – by moving the points the curve is changing in a particular way.**

If we are Trying to move control points using a mouse in the example below:

**As you can notice, the curve stretches along the tangential lines 1 → 2 and 3 → 4.**

After some practice it becomes obvious how to place points to get the needed curve. And by connecting several curves we can get practically anything.

Here are some examples:



**Bezier Curve** **methods**

|  |  |
| --- | --- |
| Method | Description |
| beginPath() | This method resets the current path. |
| moveTo(x, y) | This method creates a new subpath with the given point. |
| closePath() | This method marks the current subpath as closed, and starts a new subpath with a point the same as the start and end of the newly closed subpath. |
| fill() | This method fills the subpaths with the current fill style. |
| stroke() | This method strokes the subpaths with the current stroke style. |
| bezierCurveTo(cp1x, cp1y, cp2x, cp2y, x, y) | This method adds the given point to the current path, connected to the previous one by a cubic Bezier curve with the given control points. The x and y parameters in bezierCurveTo() method are the coordinates of the end point. cp1x and cp1y are the coordinates of the first control point, and cp2x and cp2y are the coordinates of the second control point. |

|  |  |
| --- | --- |
| cp1x | The x-coordinate of the first Bézier control point |
| cp1y | The y-coordinate of the first Bézier control point |
| cp2x | The x-coordinate of the second Bézier control point |
| cp2y | The y-coordinate of the second Bézier control point |
| X | The x-coordinate of the ending point |
| Y | The y-coordinate of the ending point |

**Example 4.8 Bezier Curve**

<!DOCTYPE html>

<html>

<body>

<canvas id="canvas" style='background-color:#EEE;' width='500px' height='200px'/>

<script >

var canvas= document.getElementById('canvas');

var ctx = canvas.getContext('2d');

ctx.beginPath();

ctx.moveTo(170, 80);

ctx.bezierCurveTo(13, 100, 130, 150, 230, 150);

ctx.bezierCurveTo(25, 180, 320, 180, 340, 150);

ctx.bezierCurveTo(420, 150, 420, 120, 390, 100);

ctx.bezierCurveTo(430, 40, 370, 30, 340, 50);

ctx.bezierCurveTo(320, 5, 250, 20, 250, 50);

ctx.bezierCurveTo(200, 5, 150, 20, 170, 80);

ctx.closePath();

ctx.lineWidth = 5;

ctx.fillStyle = '#8ED6FF';

ctx.fill();

ctx.strokeStyle = 'blue';

ctx.stroke();

</script>

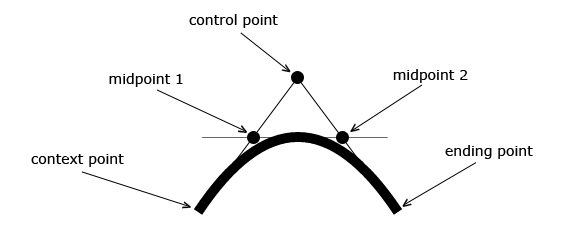
</body>

</html>



**4.9 Canvas - Drawing Quadratic Curves**

To create a quadratic curve with HTML5 Canvas, we can use the quadraticCurveTo() method. Quadratic curves are defined by the context point, a control point, and an ending point.  Quadratic curves can be styled with the linewidth, strokeStyle and lineCap (Shape of end point of line) properties.



This method adds a point to the current path by using the specified control points that represent the quadratic parametric curve. It takes the general syntax,

***context.quadraticCurveTo( cx, cy, x, y )***

where,

**cx ->** This parameter holds the x-coordinate of the quadratic control point.

**cy ->** This parameter holds the y-coordinate of the quadratic control point.

**X ->** This parameter specifies the x-coordinate of the ending point.

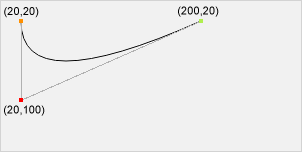
**Y->** This parameter specifies the y-coordinate of the ending point.

We require the following methods to draw quadratic curves on the canvas −

|  |  |
| --- | --- |
| **Method** | **Description** |
| beginPath() | This method resets the current path. |
| moveTo(x, y) | This method creates a new subpath with the given point. |
| closePath() | This method marks the current subpath as closed, and starts a new subpath with a point the same as the start and end of the newly closed subpath. |
| fill() | This method fills the subpaths with the current fill style. |
| stroke() | This method strokes the subpaths with the current stroke style. |
| quadraticCurveTo(cpx, cpy, x, y) | This method adds the given point to the current path, connected to the previous one by a quadratic Bezier curve with the given control point. |

The x and y parameters in quadraticCurveTo() method are the coordinates of the end point. cpx and cpy are the coordinates of the control point.The quadraticCurveTo() method adds a curve to the current path by using the control points that represent a quadratic Bézier curve. We can use the stroke() or fill() method to draw the path.

A quadratic Bézier curve requires two points. The first point is a control point that is used in the quadratic Bézier calculation and the second point is the ending point for the curve. The starting point for the curve is the last point in the current path. If a path does not exist, use the [beginPath()](https://www.w3schools.com/jsref/canvas_beginpath.asp) and [moveTo()](https://www.w3schools.com/jsref/canvas_moveto.asp)methods to define a starting point.



Start point: moveTo(**20, 20**)

Control point: quadraticCurveTo(**20, 100**, 200, 20)

End point: quadraticCurveTo(20, 100,**200, 20**)

**Example 4.9QuadraticCurve**

<!DOCTYPE html>

<html>

<body>

<canvas id="canvas" width='500px' height='200px'/>

<script >

var canvas= document.getElementById('canvas');

var ctx = canvas.getContext('2d');

ctx.beginPath();

ctx.moveTo(75,25);

ctx.quadraticCurveTo(25,25,25,62.5);

ctx.quadraticCurveTo(25,100,50,100);

ctx.quadraticCurveTo(50,120,30,125);

ctx.quadraticCurveTo(60,120,65,100);

ctx.quadraticCurveTo(125,100,125,62.5);

ctx.quadraticCurveTo(125,25,75,25);

ctx.stroke();

</script>

</body>

</html>

****

**Example 4.10QuadraticCurve and Bezier curve**

<!DOCTYPE html>

<html>

<head>

<title>Drawing a fish using Bezier Curve </title>

</head>

<body>

<canvas id="myCanvas" width="400" height="200"style="border:solid 4px green">

<script>

var c = document.getElementById("myCanvas");

var ctx = c.getContext("2d");

/\* Start a new Path \*/

ctx.beginPath();

ctx.lineWidth=3;

/\* Upper curve of the fish, from mouth to tail \*/

ctx.moveTo(60, 120);

ctx.bezierCurveTo(90, 30, 200, 130, 310, 55);

/\* Lower curve of the fish, from mouth to tail \*/

ctx.moveTo(60, 120);

ctx.bezierCurveTo(90, 170, 200, 110, 310, 160);

/\* Upper half of tail \*/

ctx.moveTo(310, 55);

ctx.quadraticCurveTo(320, 80, 280, 110);

/\* lower half of tail \*/

ctx.moveTo(310, 160);

ctx.quadraticCurveTo(320, 120, 280, 110);

/\* Eye of the fish \*/

ctx.moveTo(100, 100);

ctx.arc(100, 100, 5, 0, 2\*Math.PI);

/\* Mouth of the fish \*/

ctx.moveTo(60, 120);

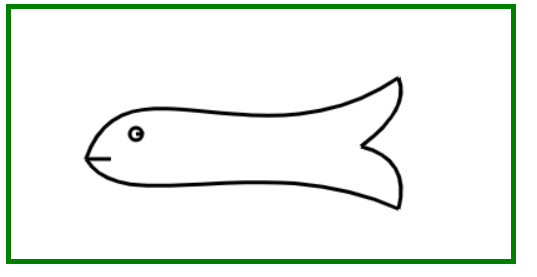
ctx.lineTo(80, 120);

ctx.stroke();

</script>

</body>

</html>



**Difference between Bezier and Quadratic Curve**

A cubic Bezier curve has 2 control points, whereas the quadratic Bezier curve only has 1 control point.

**4.10Canvas - Using Images**

To draw an image on a canvas, drawImage() method is used. It takes the general format:

**drawImage(*image,x,y*)**

where

image --> Image to be drawn

x -> x coordinate of the image

y->y coordinate of the image

|  |  |
| --- | --- |
| **Method** | **Description** |
| beginPath() | This method resets the current path. |
| moveTo(x, y) | This method creates a new subpath with the given point. |
| closePath() | This method marks the current subpath as closed, and starts a new subpath with a point the same as the start and end of the newly closed subpath. |
| fill() | This method fills the subpaths with the current fill style. |
| stroke() | This method strokes the subpaths with the current stroke style. |
| drawImage(image, dx, dy) | This method draws the given image onto the canvas. Here *image* is a reference to an image or canvas object. x and y form the coordinate on the target canvas where our image should be placed. |

**Example 4.11Canvas Image**

<!DOCTYPE html>

<html>

<body>

<img id="epche" src="EPCHE.png" alt="Higher Education Block" width="220" height="277">

<script>

var c = document.getElementById("myCanvas");

var ctx = c.getContext("2d");

var img = document.getElementById("epche");

ctx.drawImage(img,10,10);

</script>

</body>

</html>



**4.11 Canvas - Create Gradients**

Gradients can be used to fill rectangles, circles, lines, text, etc. Shapes on the canvas are not limited to solid colors.There are two different types of gradients.

* createLinearGradient(*x,y,x1,y1*) - creates a linear gradient
* createRadialGradient(*x,y,r,x1,y1,r1*) - creates a radial/circular gradient

Once we have a gradient object, we must add two or more color stops.

The addColorStop() method specifies the color stops, and its position along the gradient. Gradient positions can be anywhere between 0 to 1.

To use the gradient, set the fillStyle or strokeStyle property to the gradient, then draw the shape (rectangle, text, or a line).

HTML5 canvas allows us to fill and stroke shapes using linear and radial gradients using the following methods −

|  |  |
| --- | --- |
| Method | Description |
| addColorStop(offset, color) | This method adds a color stop with the given color to the gradient at the given offset. Here 0.0 is the offset at one end of the gradient, 1.0 is the offset at the other end. |
| createLinearGradient(x0, y0, x1, y1) | This method returns a CanvasGradient object that represents a linear gradient that paints along the line given by the coordinates represented by the arguments. The four arguments represent the starting point (x1,y1) and end point (x2,y2) of the gradient. |
| createRadialGradient(x0, y0, r0, x1, y1, r1) | This method returns a CanvasGradient object that represents a radial gradient that paints along the cone given by the circles represented by the arguments. The first three arguments define a circle with coordinates (x1,y1) and radius r1 and the second a circle with coordinates (x2,y2) and radius r2. |

**Example 4.12Canvas Linear Gradient**

<!DOCTYPE html>

<html>

<body>

<h1>HTML5 Canvas</h1>

<canvas id="myCanvas" width="2000" height="1000" style="border:1px solid grey;"></canvas>

<script>

const c = document.getElementById("myCanvas");

const ctx = c.getContext("2d");

// Create gradient

const grd = ctx.createLinearGradient(0,100,200,0);

grd.addColorStop(0,"slateblue");

grd.addColorStop(1,"green");

// Fill with gradient

ctx.fillStyle = grd;

ctx.fillRect(10,10,1500,800);

</script>

</body>

</html>



**Example 4.13Canvas radial Gradient**

<!DOCTYPE html>

<html>

<body>

<h1>HTML5 Canvas</h1>

<canvas id="myCanvas" width="1000" height="1000" style="border:1px solid grey;"></canvas>

<script>

const c = document.getElementById("myCanvas");

const ctx = c.getContext("2d");

// Create gradient

const grd = ctx.createRadialGradient(75,50,50,90,60,1000);

grd.addColorStop(0,"slateblue");

grd.addColorStop(1,"green");

// Fill with gradient

ctx.fillStyle = grd;

ctx.fillRect(12,12,1500,1800);

</script>

</body>

</html>

