Computer Graphics Lab 08-

HTML canvas graphics introduction

# Reference:

1. <https://www.tutorialrepublic.com/html-tutorial/html5-canvas.php>
2. <https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API/Tutorial/Basic_animations>

# Tools

**Any text editor (like vs code) + Modern browser like chrome/firefox browser**

# Introduction

## **What is Canvas?**

# The HTML5 canvas element can be used to draw graphics on the webpage via JavaScript. The canvas was originally introduced by Apple for the Mac OS dashboard widgets and to power graphics in the Safari web browser. Later it was adopted by the Firefox, Google Chrome and Opera. Now the canvas is a part of the new HTML5 specification for next generation web technologies.

# By default the <canvas> element has 300px of width and 150px of height without any border and content. However, custom width and height can be defined using the CSS [height](https://www.tutorialrepublic.com/css-reference/css-height-property.php) and [width](https://www.tutorialrepublic.com/css-reference/css-width-property.php) property whereas the border can be applied using the CSS [border](https://www.tutorialrepublic.com/css-reference/css-border-property.php) property.

## **Understanding Canvas Coordinates**

# The canvas is a two-dimensional rectangular area. The coordinates of the top-left corner of the canvas are (0, 0) which is known as origin, and the coordinates of the bottom-right corner are (*canvas width*, *canvas height*). Here's a simple demonstration of canvas default coordinate system.

# 

# 

# Labs

## **Lab 8-1- Drawing Path and Shapes on Canvas**

In this section we're going to take a closer look at how to draw basic paths and shapes using the newly introduced HTML5 canvas element and JavaScript.

Here is the base template for drawing paths and shapes onto the 2D HTML5 canvas.

#### **Example**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8">

<title>Drawing on Canvas</title>

<script>

window.onload = function() {

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

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

// draw stuff here

};

</script>

</head>

<body>

<canvas id="myCanvas" width="300" height="200"></canvas>

</body>

</html>

All the lines except those from 7 to 11 are pretty straight forward. The anonymous function attached to the window.onload event will execute when the page load. Once the page is loaded, we can access the canvas element with document.getElementById() method. Later we have defined a 2D canvas context by passing 2d into the getContext() method of the canvas object.

## **Lab 8-2-Drawing a Line**

The most basic path you can draw on canvas is a straight line. The most essential methods used for this purpose are moveTo(), lineTo() and the stroke().

The moveTo() method defines the position of drawing cursor onto the canvas, whereas the lineTo() method used to define the coordinates of the line's end point, and finally the stroke() method is used to make the line visible. Let's try out an example:

#### **Example**

<script>

window.onload = function() {

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

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

context.moveTo(50, 150);

context.lineTo(250, 50);

context.stroke();

};

</script>

## **Lab 8-3-Drawing a Arc**

You can create arcs using the arc() method. The syntax of this method is as follow:

*context.arc(centerX, centerY, radius, startingAngle, endingAngle, counterclockwise);*

The JavaScript code in the following example will draw an arc on the canvas.

#### **Example**

<script>

window.onload = function() {

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

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

context.arc(150, 150, 80, 1.2 \* Math.PI, 1.8 \* Math.PI, false);

context.stroke();

};

</script>

## **Lab 8-4-Drawing a Rectangle**

You can create rectangle and square shapes using the rect() method. This method requires four parameters x, y position of the rectangle and its width and height.

The basic syntax of the rect() method can be given with:

context.rect(x, y, width, height);

The following JavaScript code will draw a rectangle shape centered on the canvas.

#### **Example**

<script>

window.onload = function() {

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

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

context.rect(50, 50, 200, 100);

context.stroke();

};

</script>

**Lab 8-5-Drawing a Circle**

There is no specific method for creating a circle like rectangle's rect() method. However, you can create a fully enclosed arc such as a circle using the arc() method.

The syntax for drawing a complete circle using the arc() method can be given with:

context.arc(centerX, centerY, radius, 0, 2 \* Math.PI, false);

The following example will draw a complete circle centered on the canvas.

#### **Example**

<script>

window.onload = function() {

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

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

context.arc(150, 100, 70, 0, 2 \* Math.PI, false);

context.stroke();

};

</script>

## **Lab 8-6-Applying Styles and Colors on Stroke**

The default color of the stroke is black and its thickness is one pixel. But, you can set the color and width of the stoke using the strokeStyle and lineWidth property respectivley.

The following example will draw an orange color line having 5 pixels width.

#### **Example 1**

<script>

window.onload = function() {

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

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

context.lineWidth = 5;

context.strokeStyle = "orange";

context.moveTo(50, 150);

context.lineTo(250, 50);

context.stroke();

};

</script>

You can also set the cap style for the lines using the lineCap property. There are three styles available for the line caps — butt, round, and square. Here's an example:

#### **Example 2**

<script>

window.onload = function() {

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

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

context.lineWidth = 10;

context.strokeStyle = "orange";

context.lineCap = "round";

context.arc(150, 150, 80, 1.2 \* Math.PI, 1.8 \* Math.PI, false);

context.stroke();

};

</script>

## **Lab 8-7-Filling Colors inside Canvas Shapes**

You can also fill color inside the canvas shapes using the fillStyle() method.

The following example will show you how to fill a solid color inside a rectangle shape.

#### **Example 1**

<script>

window.onload = function() {

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

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

context.rect(50, 50, 200, 100);

context.fillStyle = "#FB8B89";

context.fill();

context.lineWidth = 5;

context.strokeStyle = "black";

context.stroke();

};

</script>

**Tip:** While styling the shapes on canvas, it is recommended to use the fill() method before the stroke() method in order to render the stroke correctly.

Similarly, you can use the fillStyle() method to fill solid color inside a circle too.

#### **Example 2**

<script>

window.onload = function() {

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

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

context.arc(150, 100, 70, 0, 2 \* Math.PI, false);

context.fillStyle = "#FB8B89";

context.fill();

context.lineWidth = 5;

context.strokeStyle = "black";

context.stroke();

};

</script>

## **Lab 8-8-Filling Gradient Colors inside Canvas Shapes**

You can also fill gradient color inside the canvas shapes. A gradient is just a smooth visual transition from one color to another. There are two types of gradient available — *linear* and *radial*.

The basic syntax for creating a linear gradient can be given with:

var grd = context.createLinearGradient(startX, startY, endX, endY);

The following example uses the createLinearGradient() method to fill a linear gradient color inside a rectangle. Let's try it out to understand how it basically works:

#### **Example 1**

<script>

window.onload = function() {

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

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

context.rect(50, 50, 200, 100);

var grd = context.createLinearGradient(0, 0, canvas.width, canvas.height);

grd.addColorStop(0, '#8ED6FF');

grd.addColorStop(1, '#004CB3');

context.fillStyle = grd;

context.fill();

context.stroke();

};

</script>

Similarly, you can fill canvas shapes with radial gradient using the createRadialGradient() method. The basic syntax for creating a radial gradient can be given with:

var grd = context.createRadialGradient(startX, startY, startRadius, endX, endY, endRadius);

The following example uses the createRadialGradient() method to fill a radial gradient color inside a circle. Let's try it out to understand how it actually works:

#### **Example 2**

<script>

window.onload = function() {

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

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

context.arc(150, 100, 70, 0, 2 \* Math.PI, false);

var grd = context.createRadialGradient(150, 100, 10, 160, 110, 100);

grd.addColorStop(0, '#8ED6FF');

grd.addColorStop(1, '#004CB3');

context.fillStyle = grd;

context.fill();

context.stroke();

};

</script>

## **Lab 8-9-Drawing Text on Canvas**

You can also draw text onto canvas. These texts can contain any Unicode characters. The following example will draw a simple greeting message "Hello World!" onto a canvas.

#### **Example 1**

<script>

window.onload = function() {

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

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

context.font = "bold 32px Arial";

context.fillText("Hello World!", 50, 100);

};

</script>

You can additionally set the color and alignment of the text on the canvas, like this:

#### **Example 2**

<script>

window.onload = function() {

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

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

context.font = "bold 32px Arial";

context.textAlign = "center";

context.textBaseline = "middle";

context.fillStyle = "orange";

context.fillText("Hello World!", 150, 100);

};

</script>

You can also apply stroke on text using the strokeText() method. This method will color the perimeter of the text instead of filling it. However if you want to set both the fill and stroke on canvas text you can use both the fillText() and the strokeText() methods together.

#### **Example 3**

<script>

window.onload = function() {

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

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

context.font = "bold 32px Arial";

context.textAlign = "center";

context.textBaseline = "middle";

context.strokeStyle = "orange";

context.strokeText("Hello World!", 150, 100);

};

</script>

## **Lab 8-10- Animation Basics[[1]](#footnote-0)**

Since we're using JavaScript to control [<canvas>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/canvas) elements, it's also very easy to make (interactive) animations. In this chapter we will take a look at how to do some basic animations.

Probably the biggest limitation is, that once a shape gets drawn, it stays that way. If we need to move it we have to redraw it and everything that was drawn before it. It takes a lot of time to redraw complex frames and the performance depends highly on the speed of the computer it's running on.

## [**Basic animation steps**](https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API/Tutorial/Basic_animations#basic_animation_steps)

These are the steps you need to take to draw a frame:

1. **Clear the canvas** Unless the shapes you'll be drawing fill the complete canvas (for instance a backdrop image), you need to clear any shapes that have been drawn previously. The easiest way to do this is using the [clearRect()](https://developer.mozilla.org/en-US/docs/Web/API/CanvasRenderingContext2D/clearRect) method.
2. **Save the canvas state** If you're changing any setting (such as styles, transformations, etc.) which affect the canvas state and you want to make sure the original state is used each time a frame is drawn, you need to save that original state.
3. **Draw animated shapes** The step where you do the actual frame rendering.
4. **Restore the canvas state** If you've saved the state, restore it before drawing a new frame.

## [**Controlling an animation**](https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API/Tutorial/Basic_animations#controlling_an_animation)

Shapes are drawn to the canvas by using the canvas methods directly or by calling custom functions. In normal circumstances, we only see these results appear on the canvas when the script finishes executing. For instance, it isn't possible to do an animation from within a for loop.

That means we need a way to execute our drawing functions over a period of time. There are two ways to control an animation like this.

### [**Scheduled updates**](https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API/Tutorial/Basic_animations#scheduled_updates)

First there's the [window.setInterval()](https://developer.mozilla.org/en-US/docs/Web/API/WindowOrWorkerGlobalScope/setInterval), [window.setTimeout()](https://developer.mozilla.org/en-US/docs/Web/API/WindowOrWorkerGlobalScope/setTimeout), and [window.requestAnimationFrame()](https://developer.mozilla.org/en-US/docs/Web/API/window/requestAnimationFrame) functions, which can be used to call a specific function over a set period of time.

[setInterval(function, delay)](https://developer.mozilla.org/en-US/docs/Web/API/WindowOrWorkerGlobalScope/setInterval)

Starts repeatedly executing the function specified by function every delay milliseconds.

[setTimeout(function, delay)](https://developer.mozilla.org/en-US/docs/Web/API/WindowOrWorkerGlobalScope/setTimeout)

Executes the function specified by function in delay milliseconds.

[requestAnimationFrame(callback)](https://developer.mozilla.org/en-US/docs/Web/API/window/requestAnimationFrame)

Tells the browser that you wish to perform an animation and requests that the browser call a specified function to update an animation before the next repaint.

If you don't want any user interaction you can use the setInterval() function which repeatedly executes the supplied code. If we wanted to make a game, we could use keyboard or mouse events to control the animation and use setTimeout(). By setting [EventListener](https://developer.mozilla.org/en-US/docs/Web/API/EventListener)s, we catch any user interaction and execute our animation functions.

In the examples below, we'll use the [window.requestAnimationFrame()](https://developer.mozilla.org/en-US/docs/Web/API/window/requestAnimationFrame) method to control the animation. The requestAnimationFrame method provides a smoother and more efficient way for animating by calling the animation frame when the system is ready to paint the frame. The number of callbacks is usually 60 times per second and may be reduced to a lower rate when running in background tabs. For more information about the animation loop, especially for games, see the article [Anatomy of a video game](https://developer.mozilla.org/en-US/docs/Games/Anatomy) in our [Game development zone](https://developer.mozilla.org/en-US/docs/Games).

## [**An animated solar system**](https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API/Tutorial/Basic_animations#an_animated_solar_system)

This example animates a small model of our solar system.

var sun = new Image();

var moon = new Image();

var earth = new Image();

function init() {

sun.src = 'https://mdn.mozillademos.org/files/1456/Canvas\_sun.png';

moon.src = 'https://mdn.mozillademos.org/files/1443/Canvas\_moon.png';

earth.src = 'https://mdn.mozillademos.org/files/1429/Canvas\_earth.png';

window.requestAnimationFrame(draw);

}

function draw() {

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

ctx.globalCompositeOperation = 'destination-over';

ctx.clearRect(0, 0, 300, 300); // clear canvas

ctx.fillStyle = 'rgba(0, 0, 0, 0.4)';

ctx.strokeStyle = 'rgba(0, 153, 255, 0.4)';

ctx.save();

ctx.translate(150, 150);

// Earth

var time = new Date();

ctx.rotate(((2 \* Math.PI) / 60) \* time.getSeconds() + ((2 \* Math.PI) / 60000) \* time.getMilliseconds());

ctx.translate(105, 0);

ctx.fillRect(0, -12, 40, 24); // Shadow

ctx.drawImage(earth, -12, -12);

// Moon

ctx.save();

ctx.rotate(((2 \* Math.PI) / 6) \* time.getSeconds() + ((2 \* Math.PI) / 6000) \* time.getMilliseconds());

ctx.translate(0, 28.5);

ctx.drawImage(moon, -3.5, -3.5);

ctx.restore();

ctx.restore();

ctx.beginPath();

ctx.arc(150, 150, 105, 0, Math.PI \* 2, false); // Earth orbit

ctx.stroke();

ctx.drawImage(sun, 0, 0, 300, 300);

window.requestAnimationFrame(draw);

}

init();

## **Optional Lab 8-11-**[**A looping panorama**](https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API/Tutorial/Basic_animations#a_looping_panorama)

In this example, a panorama is scrolled left-to-right. We're using [an image of Yosemite National Park](https://commons.wikimedia.org/wiki/File:Capitan_Meadows,_Yosemite_National_Park.jpg) we took from Wikipedia, but you could use any image that's larger than the canvas.

var img = new Image();

// User Variables - customize these to change the image being scrolled, its

// direction, and the speed.

img.src = 'https://mdn.mozillademos.org/files/4553/Capitan\_Meadows,\_Yosemite\_National\_Park.jpg';

var CanvasXSize = 800;

var CanvasYSize = 200;

var speed = 30; // lower is faster

var scale = 1.05;

var y = -4.5; // vertical offset

// Main program

var dx = 0.75;

var imgW;

var imgH;

var x = 0;

var clearX;

var clearY;

var ctx;

img.onload = function() {

imgW = img.width \* scale;

imgH = img.height \* scale;

if (imgW > CanvasXSize) {

// image larger than canvas

x = CanvasXSize - imgW;

}

if (imgW > CanvasXSize) {

// image width larger than canvas

clearX = imgW;

} else {

clearX = CanvasXSize;

}

if (imgH > CanvasYSize) {

// image height larger than canvas

clearY = imgH;

} else {

clearY = CanvasYSize;

}

// get canvas context

ctx = document.getElementById('canvas').getContext('2d');

// set refresh rate

return setInterval(draw, speed);

}

function draw() {

ctx.clearRect(0, 0, clearX, clearY); // clear the canvas

// if image is <= Canvas Size

if (imgW <= CanvasXSize) {

// reset, start from beginning

if (x > CanvasXSize) {

x = -imgW + x;

}

// draw additional image1

if (x > 0) {

ctx.drawImage(img, -imgW + x, y, imgW, imgH);

}

// draw additional image2

if (x - imgW > 0) {

ctx.drawImage(img, -imgW \* 2 + x, y, imgW, imgH);

}

}

// image is > Canvas Size

else {

// reset, start from beginning

if (x > (CanvasXSize)) {

x = CanvasXSize - imgW;

}

// draw additional image

if (x > (CanvasXSize-imgW)) {

ctx.drawImage(img, x - imgW + 1, y, imgW, imgH);

}

}

// draw image

ctx.drawImage(img, x, y,imgW, imgH);

// amount to move

x += dx;

}

Below is the [<canvas>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/canvas) in which the image is scrolled. Note that the width and height specified here must match the values of the CanvasXZSize and CanvasYSize variables in the JavaScript code.

<canvas id="canvas" width="800" height="200"></canvas>

1. <https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API/Tutorial/Basic_animations> [↑](#footnote-ref-0)