Canvas

Canvas

- The canvas element provides scripts with a resolution-dependent bitmap canvas, which can be used for rendering graphs, game graphics, art, or other visual images on the fly.
- Two attributes to control the size of the element's bitmap: width and height.
- The HTML5 Canvas API supports the same two-dimensional drawing operations that most modern operating systems and frameworks support.
- To programmatically use a canvas, you have to
  - get its context
  - perform actions on the context
  - and finally apply those actions to the context.

Canvas

- Canvas definition

```
<canvas>
  Update your browser to enjoy canvas!
</canvas>
```

Alternative text if canvas not supported

- CSS can be applied to the canvas element itself to add borders, padding, margins, etc.
- Some CSS values are inherited by the contents of the canvas
  - fonts drawn into a canvas default to the settings of the canvas element itself.
- Some properties set on the context used in canvas operations follow the CSS syntax.
  - Colors and fonts, for example, use the same notation on the context that they use in any CSS document.

Canvas
Checking for support

```
try {
        document.createElement("canvas").getContext("2d");
} catch (e) {
        alert("HTML5 Canvas is not supported in your browser.“);
}
```

Canvas
Draw a Diagonal line

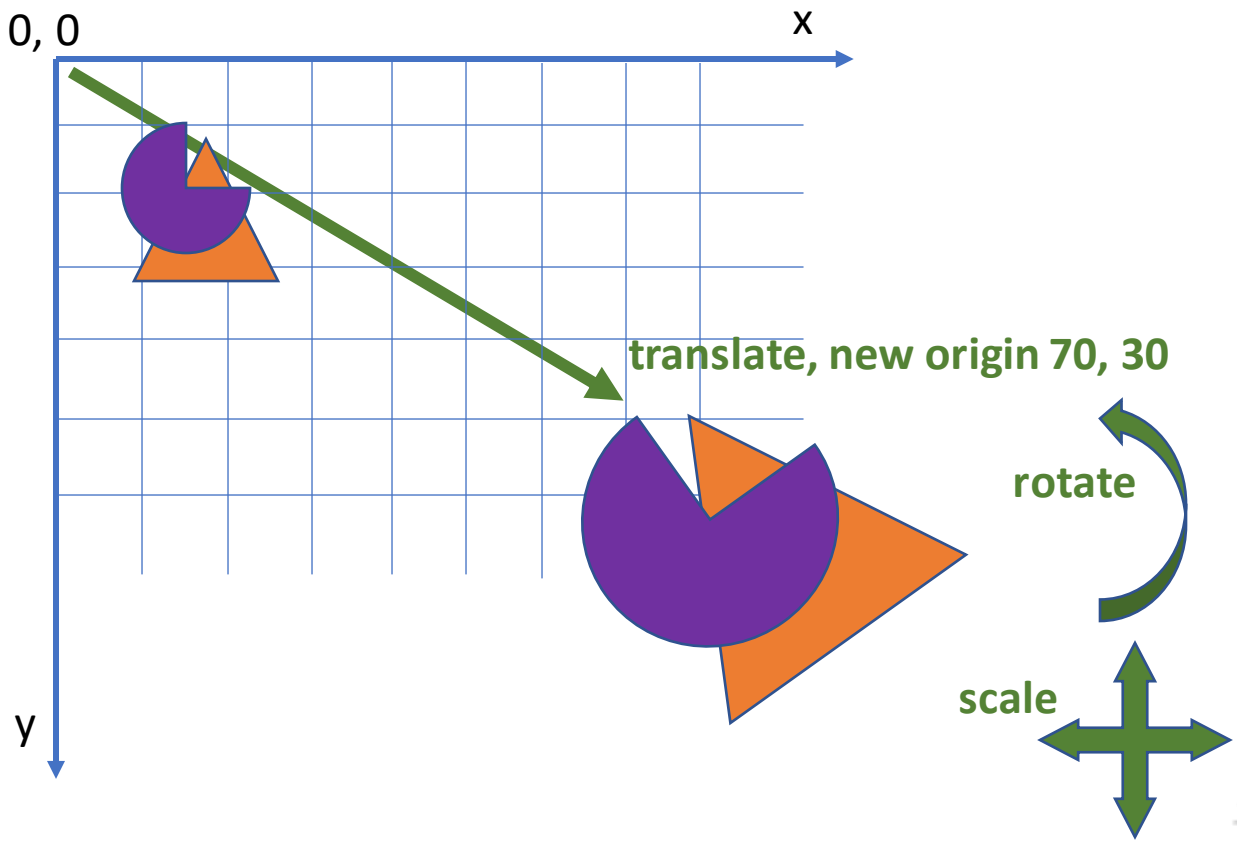
```
<!DOCTYPE html>
<html>
<meta charset="utf-8">
<title>Diagonal line example</title>
<canvas id="example1" style="border: 1px solid;"
      width="200" height="200">
</canvas>
<script>
function drawLine() {
  // Get a reference to the canvas
  const canvas = document.getElementById('example1');
  // Get a reference to the drawing context
  const context = canvas.getContext('2d');
```

Canvas
Draw a Diagonal line

```
  // Create a path
  context.beginPath(); // It's a new path
  context.moveTo(100, 100); // Move to starting point
  context.lineTo(130, 180); // Draw a straight line
  context.lineWidth = 10; // Line width
  context.strokeStyle = '#00ff00'; // Green color
  context.lineCap = 'round'; // Type of end-line:
  // butt (default), round, square
  // Stroke the path onto the canvas (it is now visible)
  context.stroke();
}
window.addEventListener("load", drawLine, true);
</script>
</html>
```

Canvas
Applying Transformations

- Recommendation for reusable code: draw at the origin (coordinate 0,0) and apply transformations (such as scale, translate, rotate) to obtain the desired appearance



Canvas
Draw a Diagonal line

```
function drawLine() {
  const canvas = document.getElementById('example2');
  const context = canvas.getContext('2d');
  // Save the current drawing state
  context.save();
  // Move the origin to the right, and down
  context.translate(100, 100);
  // Draw a line using the new the origin as a start
  context.beginPath();
  context.moveTo(0, 0);
  context.lineTo(30, 80);
  context.stroke();
  // Restore the old state
  context.restore();
}
```

Canvas
Draw a Diagonal line

Why context.save()?

- If you do not save the state, the modifications (translate, scale, rotate, etc) will continue to be applied to the context in future operations, and that might not be what you want.

- At the end, you restore the context to its original state, so that future canvas operations are performed without the translation/rotation/etc that was applied in this operation.

Canvas
Working with Paths

```
function drawTopPath(context) {
  // Draw the tree top
  context.beginPath();
  context.moveTo(-25, -50);
  context.lineTo(-35, -80);
  context.lineTo(-20, -100);
  context.lineTo(-5, -110);
  context.lineTo(20, -90);
  context.lineTo(40, -80);
  context.lineTo(40, -60);
  context.lineTo(15, -50);
  // Close the path
  context.closePath();
}
```

closePath(): similar to **lineTo()** but the destination is assumed to be the origination of the path

Canvas
Working with Paths

```
function drawLandscape() {
  const canvas = document.getElementById('example3');
  const context = canvas.getContext('2d');
  context.save();
  context.translate(100, 200);
  // Create the shape for our tree top
  drawTopPath(context);
  // Stroke the current path
  context.stroke();
  context.restore();
}
window.addEventListener("load", drawLandscape, true);
```

Canvas
Working with Styles

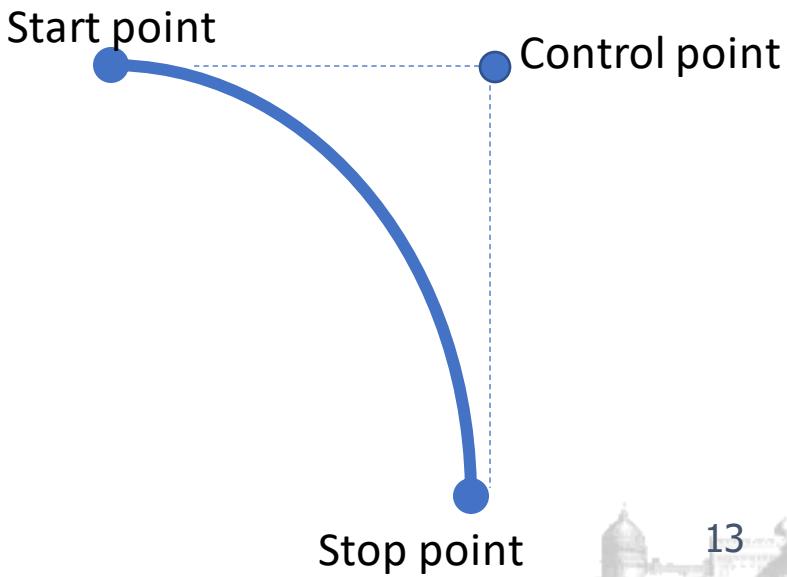
```
// Set line width
context.lineWidth = 4;
// Round corners
context.lineJoin = 'round';
// Color light green
context.strokeStyle = '#77BB00';
// Fill color darker green
context.fillStyle = '#339900';
context.fill();
// Change fill color to brown
context.fillStyle = '#884400';
// Tree trunk: filled rectangle
context.fillRect(-8, -50, 16, 50);
```

Canvas
Drawing Curves

context.quadraticCurveTo(xC, yC, xS, yS);

(xC,yC) – coordinates of the control point. The control point sits to the side of the curve. By adjusting the location of the control point, you can adjust the curvature of the path.

(xS,yS) – stop point.



Canvas
Drawing Curves

```
function createRiver(context) {
  // Draw the river in blue, wide line
  context.beginPath();
  context.moveTo(0, 0);
  context.quadraticCurveTo(150, -180, 460, -140);
  context.strokeStyle = '#5599FF';
  context.lineWidth = 30;
}
```

Canvas
Inserting Images into a Canvas

- Images can be stamped, stretched, modified with transformations.
- The image has to be loaded completely before you attempt to render it.

Solution

```
// Load the wood image
const wood = new Image();
wood.src = "wood.png";

// Once the image is loaded, draw it on the canvas
wood.onload = function () {
  drawLandscape();
}
```

Canvas
Inserting Images into a Canvas

Drawing an image on a canvas

```
// Draw the wood pattern image
// as the trunk
context.drawImage(wood, -8, -50, 16, 50);
```

x *y* *width* *height*

This option will scale the image to fit into the 16 × 50 pixel space that we decided for the trunk

Canvas
Using Gradients

- Gradients allow you to apply a gradual algorithmic sampling of colors as either a stroke or fill style
- Creating gradients requires a three-step process:
  1. Create the gradient object itself.
  2. Apply color stops to the gradient object, each color stop marks a change in color along the transition.
  3. Set the gradient as either a *fillStyle* or a *strokeStyle* on the context.
- If you supply points A and B as the arguments to the creation of a gradient, the color will be transitioned for any stroke or fill that moves in the direction of point A to point B.

Canvas
Using Gradients

```
function drawTree(context) {
// Draw the tree top
context.lineWidth = 1;
context.beginPath();
context.moveTo(-25, -50);
context.lineTo(-35, -80);
context.lineTo(-20, -100);
context.lineTo(-5, -110);
context.lineTo(20, -90);
context.lineTo(40, -80);
context.lineTo(40, -60);
context.lineTo(15, -50);
// Close the path
context.closePath();
// Fill color darker green
context.fillStyle = '#339900';
context.fill();
```

Canvas
Using Gradients

```
// The top of the tree:
// gradient from up left to down right becomes darker
// using a black color with changing transparency
const topShadow =
    context.createLinearGradient(-35, -100, 40, -50);
// The beginning of the shadow gradient is
// black, fully transparent
topShadow.addColorStop(0, 'rgba(0, 0, 0, 0.0)');
// The end of the shadow is black not completely transparent
topShadow.addColorStop(0.8, 'rgba(0, 0, 0, 0.7)');
// Draw the shadow
context.fillStyle = topShadow;
context.fill();
```

Canvas
Using Gradients

```
// Create a 3 stop horizontal gradient
const trunkGradient =
    context.createLinearGradient(-8, -50, 16, -50);
// Left of trunk is brown
trunkGradient.addColorStop(0, '#663300');
// The middle-left of the trunk is light brown
trunkGradient.addColorStop(0.2, '#996600');
// The right of the trunk is black
trunkGradient.addColorStop(1, '#000000');
// Apply the gradient as the fill style, and draw the trunk
context.fillStyle = trunkGradient;
context.fillRect(-8, -50, 16, 50);
}
```

Canvas
Using Background Patterns

- The HTML5 Canvas API also includes an option to set an image as a repeatable pattern for either a path stroke or fill.

```
const water = new Image();
water.src = "water.jpg";
water.onload = function () {
  drawLandscape();
}
```

```
function createRiver(context) {
  context.beginPath();
  // The first curve bends up and right
  context.moveTo(0, 0);
  context.quadraticCurveTo(150, -180, 460, -140);
  // Draw the river with a pattern, wide line
  context.strokeStyle = context.createPattern(water, 'repeat');
  context.lineWidth = 30;
}
```

Canvas
Using Background Patterns

Repetition Patterns

Repeat	Value
repeat	(Default) The image is repeated in both directions
repeat-x	The image is repeated only in the X dimension
repeat-y	The image is repeated only in the Y dimension
no-repeat	The image is displayed once and not repeated

Canvas
Scaling Canvas Objects

```
context.save();
context.translate(100, 250);
drawTree(context);
context.stroke();
context.restore();

context.save();
context.translate(200, 180);
context.scale(1.2,1.2);
drawTree(context);
context.stroke();
context.restore();

context.save();
context.translate(50, 170);
context.scale(0.8,1.4);
drawTree(context);
context.stroke();
context.restore();
```

Canvas
Scaling Canvas Objects

Note: transforms such as **scale** and **rotate** operate from the origin.

- If you perform a rotate transform to a shape drawn off origin, a rotate transform will rotate the shape around the origin rather than rotating in place.
- Similarly, if you performed a scale operation to shapes before translating them to their proper position, all locations for path coordinates would also be multiplied by the scaling factor.
- Depending on the scale factor applied, this new location could even be off the canvas altogether, leaving you wondering why your scale operation just deleted the image.

Canvas
Scaling Canvas Objects

```
context.save();
// rotation angle is specified in radians
context.rotate(1.57);
context.drawImage(myImage, 0, 0, 100, 100);
context.restore();
```

Canvas
Scaling Canvas Objects

```
function drawShadow(context) {
context.save();
// Transform to obtain a slanted shape
context.transform(1, 0, 1, 1, 0, 0);
// Y size of shadow is 0.6 of the tree
context.scale(1, 0.6);
// Top of the tree shadow
context.lineWidth = 1;
context.beginPath();
context.moveTo(-25, -50);
context.lineTo(-35, -80);
context.lineTo(-20, -100);
context.lineTo(-5, -110);
context.lineTo(20, -90);
context.lineTo(40, -80);
context.lineTo(40, -60);
context.lineTo(15, -50);
// Close the path
context.closePath();
```

Canvas
Scaling Canvas Objects

```
// Fill color black transparent
context.fillStyle = 'rgba(0, 0, 0, 0.3)';
context.fill();
// Trunk shadow
context.fillRect(-8, -50, 16, 50);
context.restore();
}
```

Canvas
Scaling Canvas Objects

transform(a, b, c, d, e, f);

a) Horizontal scaling.
b) Horizontal skewing.
c) Vertical skewing.
d) Vertical scaling.
e) Horizontal moving.
f) Vertical moving.

Canvas
Using Canvas Text

Two function for drawing text onto the canvas:

- fillText (text, x, y, maxwidth)
- strokeText (text, x, y, maxwidth)

Both functions take the text as well as the location at which it should be placed.

Optionally, a maxwidth argument can be provided to constrain the size of the text by automatically shrinking the font to fit the given size.

Canvas
Using Canvas Text

Property	Values	Note
font	CSS font string	Example: italic Arial, sans-serif
textAlign	start, end, left, right, center	Defaults to start
textBaseline	top, hanging, middle, alphabetic, ideographic, bottom	Defaults to alphabetic

```
function makeTitle(context){
  // Draw some text on our canvas
  context.save();
  // Set fontface and size
  context.font = "40px Arial";
  // Use blue color
  context.fillStyle = '#003388';
  // Text is left aligned
  context.textAlign = 'left';
  context.fillText('I like trees', 100, 40);
  context.restore();
}
```

Canvas
Applying shadows

Property	Values	Note
shadowColor	Any	CSS color Can include an alpha component
shadowOffsetX	Pixel count	Positive values move shadow to the right, negative left
shadowOffsetY	Pixel count	Positive values move shadow down, negative up
shadowBlur	Gaussian blur	Higher values cause blurrier shadow edges

```
function makeTitle(context){
  context.save();
  // Arial font, 40px
  context.font = "40px Arial";
  context.fillStyle = '#003388';
  // Set some shadow on our text, black with 20% alpha
  context.shadowColor = 'rgba(0, 0, 0, 0.3)';
  // Move the shadow left 5px, up 5px
  context.shadowOffsetX = -5;
  context.shadowOffsetY = -5;
  // Blur the shadow
  context.shadowBlur = 3;
  // Text can be aligned when displayed
  context.textAlign = 'left';
  context.fillText('I like trees', 100, 40);
  context.restore();
}
```

Canvas Events

- There are many different application possibilities for using the Canvas API:
  - graphs, charts, image editing, and so on.
- Another example:
  - Areas on the map with high levels of activity are colored as hot (for example, red, yellow, or white).

Animation: an example

```
const XDIM = 1000;
const YDIM = 1000;
let stars = [];
let direction = 0;
let angle = 0;
const astronave = new Image();

function newStar(){
  let x = (Math.random()-0.5)*100;
  let y = (Math.random()-0.5)*100;
  stars.push([x, y]);
  stars = stars.filter(s => s);
}
function newPosition(){
  for(let i=0; i<stars.length; i++) {
    stars[i][0]=stars[i][0]*1.01; // change x
    stars[i][1]=stars[i][1]*1.01; // change y
  }
  stars = stars.filter(s => {return Math.abs(s[0]) < XDIM/2
                    && Math.abs(s[1]) < YDIM/2});
}
```

```
function drawStars(ctx){
  for(let i=0; i<stars.length; i++) {
    ctx.beginPath();
    let s = stars[i];
    let d = (s[0]*s[0]+s[1]*s[1])/(XDIM/2*XDIM/2);
    d = Math.min(d, 1);
    ctx.fillStyle = `rgba(255, 255, 255, ${d})`;
    ctx.arc(stars[i][0], stars[i][1], 2, 0, Math.PI*2);
    ctx.fill();
  }
}

function init() {
  astronave.src = 'astronave.png';
  const ctx = document.getElementById('canvas').getContext('2d');
  ctx.translate(XDIM/2, YDIM/2);
  window.requestAnimationFrame(draw);
  document.addEventListener('keydown', (event) =>
                {if(event.key === 'a') direction = -1;
                 else if(event.key=== 'd') direction = 1;});
  document.addEventListener('keyup', (event) => {direction = 0;});
}
```

```
function draw() {
  const ctx = document.getElementById('canvas').getContext('2d');
  ctx.fillStyle = "black";
  ctx.fillRect(-XDIM/2, -YDIM/2, XDIM, YDIM);
  newStar();
  newPosition();
  angle = angle + direction*0.02;
  ctx.save();
  ctx.rotate(angle);
  drawStars(ctx);
  ctx.restore();
  ctx.drawImage(astronave, -80, -60, 160, 120);
  window.requestAnimationFrame(draw);
}
</script>
</head>
<body onload="init()">
<canvas id="canvas" width="1000" height="1000"></canvas>
</body>
```