Exploring Canvas

In its simplest form <canvas> can be used to draw graphics via scripting (usually JavaScript). This can, for instance, be used to draw graphs, combine photos, or create simple (and not so simple) animations.

This post will only discuss the 2D drawing canvas mode. You can also use the canvas element to host 3D WebGL models, but that's a whole other set of articles, so we'll skip it for now.

Getting started

In our HTML document we need to add a `canvas` element where we will draw our content. The `id` attribute is the name that we'll use in the script to reference the `canvas` element.

Height and width are not required but it's always better to write them out

```
<canvas id="tutorial" width="600" height="600">
  <p>Your browser doesn't support Canvas</p>
</canvas>
```

We also need a script that will generate whatever we want to do in the canvas. In this example we'll generate a square and an arrow. How we generate them is not as important, we'll discuss it later.

```
// Procedural Canvas generation
const canvas =
  document.getElementById('tutorial');
if (canvas.getCont'tutorial'onst ctx = canvas.getContext('2d');

'2d'// This draws a rectangle
  ctx.strokeRect(100, 100, 125, 125);

  ctx.beginPath();
  ctx.moveTo(200, 50);
```

```
  ctx.lineTo(100, 75);
  ctx.lineTo(100, 25);
  ctx.fill();
}
```

Understanding the Canvas gird

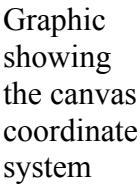

Figure 1: Canvas coordinates space. Taken from MDN

To better understand the material that we'll cover next, we need to talk about the canvas coordinate space.

Normally 1 unit in the grid corresponds to 1 pixel on the canvas. The origin of this grid is positioned in the ***top left*** corner at coordinate (0, 0) with all elements placed relative to this origin.

So the position of the top left corner of the blue square becomes x pixels from the left and y pixels from the top, at coordinate (x,y).

Later in this tutorial we'll see how we can translate the origin to a different position, rotate the grid and even scale it, but for now we'll stick to the default.

Drawing primitives

Canvas supports two primitives for drawing: rectangle and path. All other forms are derived from these two primitives.

Rectangles

Rectangles draw either rectangles or squares,

`fillRect(x, y, width, height)`
: Draws a filled rectangle.

`strokeRect(x, y, width, height)`
: Draws a rectangular outline.

`clearRect(x, y, width, height)`
: Clears the specified rectangular area, making it fully transparent.

Paths

Now let's look at paths. A path is a list of points, conected by segments of lines that can be of different shapes, curved or not, of different width and of different color. A path, or even a subpath, can be closed. To make shapes using paths takes some extra steps:

- First, you create the path
- Then you use drawing commands to draw into the path
- Once the path has been created, you can stroke or fill the path to render it.

Here are the functions used to perform these steps:

`beginPath()`
: Creates a new path. Once created, future drawing commands are directed into the path and used to build the path up.

Path methods
: Methods to set different paths for objects.

`closePath()`
: Adds a straight line to the path, going to the start of the current sub-path.

`stroke()`
: Draws the shape by stroking its outline.

`fill()`
: Draws a solid shape by filling the path's content area.

The first step to create a path is to call the `beginPath()`. Internally, paths are stored as a list of sub-paths (lines, arcs, etc) which together form a shape. Every time this method is called, the list is reset and we can start drawing new shapes.

The second step is calling the methods that actually specify the paths to be

drawn. We'll see these shortly.

The third, and an optional step, is to call closePath(). This method tries to close the shape by drawing a straight line from the current point to the start. If the shape has already been closed or there's only one point in the list, this function does nothing.

Notes

When the current path is empty, such as immediately after calling beginPath(), or on a newly created canvas, the first path construction command is always treated as a moveTo(), regardless of what it actually is. For that reason, you will almost always want to specifically set your starting position after resetting a path.

Note: When you call fill(), any open shapes are closed automatically, so you don't have to call closePath(). This is not the case when you call stroke().

Styles and colors

`fillStyle = color`
Sets the style used when filling shapes

`fillStroke = color`
Sets the style for shapes' outlines

Note: When you set the strokeStyle and/or fillStyle property, the new value becomes the default for all shapes being drawn from then on. If you want to use a different color, you must reassign `fillStyle` or `strokeStyle`.

You can also control transparency either globally, using `globalAlpha` which takes a value between 0 and 1 and will affect all shapes drawn after it's set up, or on per-element basis using `strokeStyle` and `fillStyle` with RGBA or another color format that supports transparency.

```
// Assings default transparency value
```

```
ctx.globalAlpha = 0.2;

// Assigning transparent colors
// to stroke and fill style

ctx.strokeStyle = 'rgba(255, 0, 0, 0.5)';
ctx.fillStyle = 'rgba(255, 0, 0, 0.5)';
'rgba(255, 0, 0, 0.5)'
```

Images

Importing images into a canvas is a two step process:

1. Get a reference to a supported image type. It is also possible to use images by providing a URL.
2. Draw the image on the canvas using the `drawImage()` function.

Supported image types

`HTMLImageElement`
: <img> elements or images made using the `Image()` constructor, as well as any element

`HTMLImageElement`
: These are images embedded using the <image> SVG element

`HTMLVideoElement`
: Using an HTML <video> element as your image source grabs the current frame from the video and uses it as an image

`HTMLCanvasElement`
: You can use another <canvas> element as your image source

Using an existing image in an external URL, we can use the following HTML

```
<canvas id='demo'>
  <h2>Your browser doesn't support Canvas</h2>
</canvas>
```

And the following JavaScript to insert an image into the Canvas element.

```
// Capture the canvas element
const canvas = document.getElementById('demo');
'demo'// Sets up a 2d context for the canvas ctx = canvas.getContext('2d'
// Ima'2d'// Image constrouctor and src assignment = new Image();
img.src =
'https://s3-us-west-2.amazonaws.com/0168.JPG';
// Draw th'https://s3-us-west-2.amazonaws.com/0168.JPG'// Draw the image
ctx.drawImage(img, 0,0);
```

We can then do further drawing on top of the image we just added using the primitives discussed earlier.

Idea

Using an image in a canvas element lends itself pretty well to having an outline and then draw on top of the outlines.

Where to next?

There are many other things you can do with canvas. Some ideas are:

- Image blending and transformations
- Mouse and pointer interaction
- Animations
- Image compositing and clipping

Canvass gives you a huge playground to play in. Play responsibly :)

Links, tutorials and resources

- Canvas Tutorial (MDN)

- HTML5 Canvas Tutorials Home
- Flavioo Copes HTML Canvas API Tutorial
- http://slicker.me/javascript/image_transition_effect.htm