Canvas and WebGL

Andrzej Pragacz

September 18, 2012

Brief history of canvas element

What can you do with canvas?

- draw straight lines, curves, circles, rects
- draw objects using paths
- draw images
- fill objects with gradients
- draw DOM objects
- transform objects
- save/restore canvas state
- compose (blend) objects in various ways

Getting context

assuming we have somewhere defined in the HTML code:

```
...
<canvas id="canvas" width="800" height="600">
</canvas>
...
```

we get the context:

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

And we're ready!

Drawing background rectangle

black:

```
ctx.fillStyle = "rgb(0,0,0)";
ctx.fillRect(0, 0, canvas.width, canvas.height);
```

with gradient:

```
var lingrad = ctx.createLinearGradient(0,0,
        canvas.width, canvas.height);
lingrad.addColorStop(0, '#00ABEB');
lingrad.addColorStop(0.5, '#fff');
lingrad.addColorStop(0.5, '#26C000');
lingrad.addColorStop(1, '#fff');
ctx.fillStyle = lingrad;
ctx.fillRect(0, 0, canvas.width, canvas.height);
```

also, radial gradient can be used

Drawing pentagon filled with semi-transparent color and shadow

```
var size = 20;
var sides  = 5;

ctx.shadowOffsetX = 4;
ctx.shadowOffsetY = 4;
ctx.shadowBlur = 4;
ctx.shadowColor = "rgba(0,0,0,0.5)";

ctx.fillStyle = "rgba(255,0,0,0.5)";
ctx.beginPath();
var angle = 0;
var x = Math.cos(angle);
var y = Math.sin(angle);
ctx.moveTo(size * x, size * y);
for (var i = 1; i <= this.sides; i++) {
    angle = 2 * i * Math.PI / sides;
    x = Math.cos(angle);
    y = Math.sin(angle);
    ctx.lineTo(size * x, size * y);
}
ctx.fill();
```

Drawing pentagon boundary

```
var size = 20;
var sides  = 5;

ctx.strokeStyle = "rgb(255,255,0)";
ctx.lineWidth = 6;
ctx.lineJoin = "bevel";

ctx.beginPath();
var angle = 0;
var x = Math.cos(angle);
var y = Math.sin(angle);
ctx.moveTo(size * x, size * y);
for (var i = 1; i <= this.sides; i++) {
    angle = 2 * i * Math.PI / sides;
    x = Math.cos(angle);
    y = Math.sin(angle);
    ctx.lineTo(size * x, size * y);
}
ctx.closePath();
ctx.stroke();
```

Transformations

the state of context can be saved (with all properties, like fillColor, globalAlpha etc.) this is useful when we change the coordinates system

```
ctx.save()
ctx.translate(x, y);
ctx.rotate(angle);
ctx.scale(scaleFactor,scaleFactor);
```

and then restored back:

```
ctx.restore();
```

Drawing images

```
image = Image();
image.src = 'img/megusta.png';
...
// assuming image is loaded
...
// drawing image (semi-transparent)
ctx.globalAlpha = 0.8;
ctx.drawImage(image, x, y);

// scaling
ctx.drawImage(image, x, y, width, height);

// cropping + scaling
drawImage(image, sx, sy, sWidth, sHeight, dx, dy, dWidth, dHeight)
```

canvas objects can also be drawn!

Tips

- Pre-render similar primitives or repeating objects on an off-screen canvas.
- Batch canvas calls together (for example, draw a poly-line instead of multiple separate lines).
- Avoid floating point coordinates and use integers instead.
- Avoid unnecessary canvas state changes.
- Render screen differences only, not the whole new state.
- Use multiple layered canvases for complex scenes.
- Avoid shadowBlur.
- With animations, use requestAnimationFrame.
- Test performance with JSPerf.

WebGL

- is based on OpenGL ES 2.0, which is based on OpenGL 2.0
- provides an API for 3D graphics
- uses the HTML5 canvas element

Compatibility

- Availability depends on graphics card driver (with opengl 2.0 support) and support from the browser (google chrome, firefox)
- Security issues - Cross-Origin Resource Sharing
- Mozilla Firefox - 4.0+ (CORS - 8.0+)
- Google Chrome - 9+ (CORS - 13.0+)
- Safari 5.1+ (disabled by default)
- Opera - 11+ (disabled by default)
- Internet Explorer - nope (but there are plugins, like The Chrome Frame and IEWebGL)
- Sometimes the support from graphics driver (if any) is not enough - on linux software emulation like Mesa library can be helpful

WebGL philosophy

- instead of glFunctionDoingSomething() we have gl.functionDoingSomething()
- instead of glBegin(), glVertex*() calls we rather use vertex buffers and let the shaders do all the work

Initializing

```
//assuming we have the canvas object
var gl = null;
try {
    gl = canvas.getContext("experimental-webgl");
    gl.viewportWidth = canvas.width;
    gl.viewportHeight = canvas.height;
} catch (e) {
}
if (!gl) {
    alert("Could not initialise WebGL, sorry :-(");
}
```

... but that's not all!

Initializing cont.

```
gl.viewport(0, 0, gl.viewportWidth, gl.viewportHeight);
gl.clearColor(0.0, 0.0, 0.0, 1.0);
gl.enable(gl.DEPTH_TEST);
var mvMatrix = mat4.create();
var pMatrix = mat4.create();
//mat4 is from glMatrix library
//setting projection matrix to perspective with fov=45
mat4.perspective(45, gl.viewportWidth / gl.viewportHeight,
        0.1, 100.0, pMatrix);
//setting modelview matrix to identity
mat4.identity(mvMatrix);
```

Shaders

vertex shader

```
attribute vec3 aVertexPosition;

uniform mat4 uMVMatrix;
uniform mat4 uPMatrix;

void main(void) {
    gl_Position = uPMatrix * uMVMatrix
            * vec4(aVertexPosition, 1.0);
}
```

fragment shader

```
precision mediump float;

void main(void) {
    gl_FragColor = vec4(1.0, 1.0, 1.0, 1.0);
}
```

Loading shaders

```
//shaderCode contains the code of vertex shader
var vertexShader = gl.createShader(gl.VERTEX_SHADER);
gl.shaderSource(vertexShader, shaderCode);
gl.compileShader(vertexShader);

if (!gl.getShaderParameter(vertexShader, gl.COMPILE_STATUS)) {
    alert(gl.getShaderInfoLog(vertexShader));
}
```

```
//shaderCode contains the code of fragment shader
var fragmentShader = gl.createShader(gl.FRAGMENT_SHADER);
gl.shaderSource(fragmentShader, shaderCode);
gl.compileShader(fragmentShader);

if (!gl.getShaderParameter(fragmentShader, gl.COMPILE_STATUS)) {
    alert(gl.getShaderInfoLog(fragmentShader));
}
```

Shader program

```
var shaderProgram = gl.createProgram();
gl.attachShader(shaderProgram, vertexShader);
gl.attachShader(shaderProgram, fragmentShader);
gl.linkProgram(shaderProgram);

if (!gl.getProgramParameter(shaderProgram, gl.LINK_STATUS)) {
    alert("Could not initialise shaders");
}

gl.useProgram(shaderProgram);

shaderProgram.vertexPositionAttribute =
    gl.getAttribLocation(shaderProgram, "aVertexPosition");
gl.enableVertexAttribArray(shaderProgram.vertexPositionAttribute);
shaderProgram.pMatrixUniform =
    gl.getUniformLocation(shaderProgram, "uPMatrix");
shaderProgram.mvMatrixUniform =
    gl.getUniformLocation(shaderProgram, "uMVMatrix");
```

Creating vertex buffer

```
var triangleVertexPositionBuffer = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, triangleVertexPositionBuffer);
var vertices = [
     0.0,  1.0,  0.0,
    -1.0, -1.0,  0.0,
     1.0, -1.0,  0.0
];
//binding data to current buffer
gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertices),
      gl.STATIC_DRAW);
```

Finally, drawing the scene

```
gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);

mat4.identity(mvMatrix);

gl.uniformMatrix4fv(shaderProgram.pMatrixUniform, false,
        pMatrix);
gl.uniformMatrix4fv(shaderProgram.mvMatrixUniform, false,
        mvMatrix);
gl.bindBuffer(gl.ARRAY_BUFFER, triangleVertexPositionBuffer);
gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute,
        triangleVertexPositionBuffer.itemSize, gl.FLOAT,
        false, 0, 0);
gl.drawArrays(gl.TRIANGLES, 0,
        triangleVertexPositionBuffer.numItems);
```

Libraries

libCanvas is powerful and lightweight canvas framework
Processing.js is a port of the Processing visualization language
EaselJS is a library with a Flash-like API
PlotKit is a charting and graphing library
Rekapi is an animation keyframing API for Canvas
PhiloGL is a WebGL framework for data visualization, creative coding and game development.
JavaScript InfoVis Toolkit creates interactive 2D Canvas data visualizations for the Web.