Tutorial 1 HTML 5 Canvas

In this unit, we are going to use WebGL API to develop 3D graphics for web applications. This API is developed in and for JavaScript and use HTML 5 canvas to create and draw 3D graphics. Therefore, developing a WebGL application will involve programming in JavaScript and HTML. In this and the next tutorials, we will cover the necessary skills needed. You do not have to have extensive knowledge of JavaScript or HTML to succeed in this unit, but some experience of them will be helpful.

1. Brief into to HTML 5 (http://en.wikipedia.org)
The HTML5 is a markup language used for structuring and presenting content for World Wide Web and a core technology of the Internet. It is the fifth revision of the HTML standard (created in 1990 and standardized as HTML 4 as of 1997) and is a candidate recommendation of the WWW Consortium (W3C). Its core aims have been to improve the language with support for the latest multimedia while keeping it easily readable by humans and consistently understood by computers and devices (e.g., web browsers). HTML5 is a candidate for cross-platform mobile applications. Many features of HTML5 have been built with the consideration of being able to run on low-powered devices such as smartphones and tablets. HTML5 adds many new syntactic features. These include the new <video>, <audio> and <canvas> elements, as well as the integration of scalable vector graphics (SVG) content and MathML for mathematical formulas. These features are designed to make it easy to include and handle multimedia and graphical content on the web without having to resort to proprietary plugins and APIs. HTML5 is not yet an official standard, and no browsers have full HTML5 support. But all major browsers (Safari, Chrome, Firefox, Opera, Internet Explorer) continue to add new HTML5 features to their latest versions.

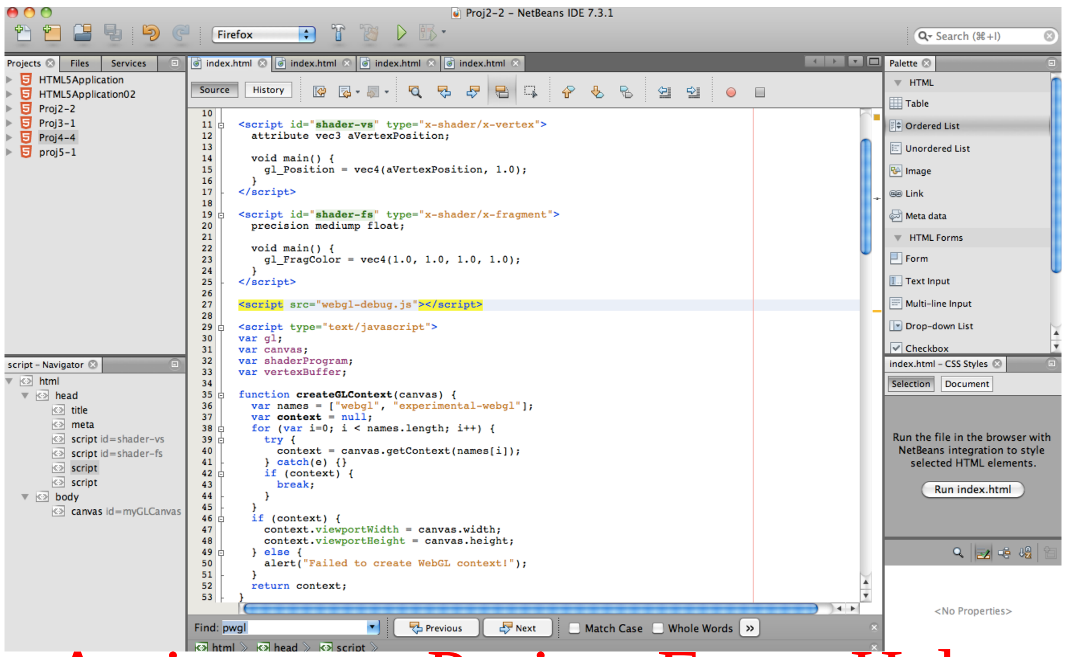
The HTML programming skills that we need in this unit is minimal: create a simple HTML document and the use of canvas, scripts, events handling and some Document Object Model (DOM) objects.

2. JavaScript
JavaScript was originally developed by Netscape, a browser competed with the internet explorer. It is an interpreted computer programming language. As part of web browsers, implementations allow client-side scripts to interact with the user, control the browser, and alter the document content that is displayed. It has also become common in server-side programming, game development and the creation of desktop applications. JavaScript is a scripting language with dynamic typing. JavaScript copies many names and naming conventions from Java.

We will not be able to introduce the full features of the language. Our introduction of JavaScript will be focussing on the key elements that are important to us: important data types and constructs and their behaviour, e.g., variables, arrays, and functions.

3. Tools:
Any text-editing tool will suit for creating HTML and JavaScript documents, e.g., Notepad or Textpad. More sophisticated and professional tools are available if you have access to them, e.g., CoffeeCup HTML editor and NetBeans IDE (free). Such tools are handy when you

debug complicated applications and organise large projects. For this unit, any of the above tools will do.



```
<script id="shader-vs" type="x-shader/x-vertex">
  attribute vec3 aVertexPosition;

  void main() {
    gl_Position = vec4(aVertexPosition, 1.0);
  }
</script>

<script id="shader-fs" type="x-shader/x-fragment">
  precision mediump float;

  void main() {
    gl_FragColor = vec4(1.0, 1.0, 1.0, 1.0);
  }
</script>

<script src="webgl-debug.js"></script>

<script type="text/javascript">
var gl;
var canvas;
var shaderProgram;
var vertexBuffer;

function createGLContext(canvas) {
  var names = ["webgl", "experimental-webgl"];
  var context = null;
  for (var i=0; i < names.length; i++) {
    try {
      context = canvas.getContext(names[i]);
    } catch(e) {}
    if (context) {
      break;
    }
  }
  if (context) {
    context.viewportWidth = canvas.width;
    context.viewportHeight = canvas.height;
  } else {
    alert("Failed to create WebGL context!");
  }
  return context;
}
```

Screenshot of NetBeans IDE for Mac.

Debugging tools

It is unlikely that your document will contain no bugs when you load them to the browsers for the first time. Without using a proper tool, debugging is largely a guesswork that requires experience. Debugging tools come with browsers, although sometimes you may need to download and install them as plugins/add-ons separately.

4. Create a simple HTML document

HTML is a markup language, which consists of a set of markup **tags**. The tags describe document content. A HTML documents contain HTML tags and plain text. Below is a simple HTML5 document, shows the structure of a typical HTML document:

```
<!DOCTYPE html>
<html>

<head>
   <meta charset="UTF-8">
   <title>Title of the document</title>
</head>

<body>
 <h1>My First Heading</h1>
 <p>My first paragraph.</p>
</body>

</html>
```

where

- The DOCTYPE declaration defines the document type, which is HTML
- The text between <html> and </html> describes the web page

- The text between <body> and </body> is the visible page content
- The text between <h1> and </h1> is displayed as a heading (<h1>…<h6> indicate the sizes of the font)
- The text between <p> and </p> is displayed as a paragraph

5. HTML Canvas

A canvas is a rectangular area on an HTML page, where 2D or 3D graphics can be drawn. The canvas is a two-dimensional grid. Its coordinate system has the origin (0, 0) at the upper-left corner of the canvas, x axis is horizontal and y axis is vertical (in pixels). However, the <canvas> element has no drawing abilities of its own (it is only a container for graphics) - you must use a script to actually draw the graphics. The getContext() method returns an object that provides methods and properties for drawing on the canvas.

A canvas is specified with the <canvas> tag. We always specify an **id** attribute (to be referred to in a script), and a width and height attribute to define the size of the canvas. By default, the <canvas> element has no border and no content. A HTML document can have multiple canvases. The following snippet define a canvas of width 200, height 100 with a border of solid line of 1 pixel in width and black in colour:

```
<canvas id="myCanvas" width="200" height="100"
style="border:1px solid #000000;">
</canvas>
```

Draw onto the Canvas with JavaScript

All drawing on the canvas must be done inside a JavaScript

```
<script type="text/javascript">
  var c=document.getElementById("myCanvas");
var ctx=c.getContext("2d");
ctx.fillStyle="#FF0000";
ctx.fillRect(0,0,150,75);
</script>
```

where

- `var c=document.getElementById("myCanvas")` retrieves the canvas object
- `var ctx=c.getContext("2d")` retrieves the built-in HTML 5 object, 2D context, with many properties and methods for drawing paths, boxes, circles, text, images, and more. When draw 3D graphics, you must retrieve "3D" context, like `getContext("webgl")`.
- `ctx.fillStyle="#FF0000";` sets the colour to draw. The default `fillStyle` is #000000 (black).
  `ctx.fillRect(0,0,150,75);` draws a rectangle filled with the current fill style.

Draw paths

To draw straight lines on a canvas, we will use the following methods:

- `moveTo(`*x,y*`)` defines the **starting** point of the line
- `lineTo(`*x,y*`)` defines the **ending** point of the line
- `stroke ()` specifies that the drawing method will be "pen stroke" instead of "fill".

```
<script type="text/javascript">
var c=document.getElementById("myCanvas");
var ctx=c.getContext("2d");
ctx.moveTo(0,0);
ctx.lineTo(200,100);
ctx.stroke();
</script>
```

Draw a Circle
To draw a circle on a canvas, we will use the method arc(x, y, r, start_angle, stop_angle)), where the angles are in radians.

```
 var c=document.getElementById("myCanvas");
 var ctx=c.getContext("2d");
 ctx.beginPath(); //begins a path
 ctx.arc(95, 50, 40, 0, 2*Math.PI);
 ctx.stroke(); // or use ctx.fill() to draw a solid circle.
```

Draw text
We can draw text using the methods fillText(text, x, y) or strokeText(text, x, y) with the specified font, where (x,y) is the position of the **lower left corner** of the text:

```
 var c=document.getElementById("myCanvas");
 var ctx=c.getContext("2d");
 ctx.font="30px Arial"; //30 pixels in height
 ctx.fillText("Hello World", 10, 35);
 ctx.rotate(20*Math.PI/180);//rotate the canvas by 20 degrees
 ctx.strokeText("Hello World", 10, 35);
```

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

- createLinearGradient(*x,y,x1,y1*) - Creates a linear gradient with starting point (x,y) and ending point (x1,y1)
- createRadialGradient(*x,y,r, x1,y1,r1*) - Creates a radial/circular gradient with (x,y,r) defining the starting circle and (x1,y1,r1) defining the ending circle
- Once we have a gradient object, we must add two or more colour stops. The addColorStop() method specifies the color stops, and its position along the gradient. Gradient positions can be anywhere between 0 to 1.

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

// Create a gradient use one of the following method
var grd=ctx.createLinearGradient(0, 0, 200, 0);//linear gradient
//var grd=ctx.createRadialGradient(75, 50, 5, 90, 60, 100);//radial
gradient

grd.addColorStop(0,"red");
grd.addColorStop(1,"white");

// Fill with gradient
ctx.fillStyle=grd;
ctx.fillRect(10,10,150,80);
```

Draw images
To draw an image on a canvas, we will use the method drawImage(*image,x,y*), where (x,y), is the position of the up-left corner of the image on the canvas.

```
 <canvas id="myCanvas" width="240" height="297" style="border:1px solid
 #000000;">
```

```
</canvas>
<script type="text/javascript">
var c=document.getElementById("myCanvas");
var ctx=c.getContext("2d");
var image = new Image();
// When the image has loaded, draw it to the canvas
image.onload = function()
    {
        ctx.drawImage(image,0,0);
    }
    image.src = "scream.png";
</script>
```

Notice that, to draw an image on canvas, we have to load the image as an HTML object first, as does the highlighted code shows. Here we use the `onload` event – we draw the image only when it is loaded to the page.
Of course, you don't have to use a canvas to load images to an HTML page. Compare the above with the following code snippet, where we first load the image as an html object and then reference it in JavaScript.

```
<img id="scream" src="scream.png" alt="The Scream">

<canvas id="myCanvas" width="240" height="297" style="border:1px solid
#000000;">
</canvas>
<script type="text/javascript">
var c=document.getElementById("myCanvas");
var ctx=c.getContext("2d");
var image=document.getElementById("scream");
ctx.drawImage(image, 0, 0);
</script>
```

6. WebGL Examples

There are many good examples of WebGL applications on the web. The followings show the potential and capability of WebGL.

1. The Brainviewer at Harvard university:
   http://www.nmr.mgh.harvard.edu/~rudolph/webgl/brain_viewer/brain_viewer.html
2. Animated Jellyfish: http://aleksandarrodic.com/p/jellyfish/
3. WebGL Aquarium:
   http://webglsamples.googlecode.com/hg/aquarium/aquarium.html
4. Water and Sphere:
   http://webglsamples.googlecode.com/hg/aquarium/aquarium.html

7. Information

Information such as WebGL specification, development tools and example can be found at http://www.khronos.org/webgl/wiki/Main_Page