Back to basics – part 2

Recap

Skip to 4 – 3D Square → Coloured Cube (or review the next few pages... it's up to you).

0 – Shell

This does absolutely nothing... sweet Fanny Adams... diddly squat... beggar all... but provides the basic HMTL page layout for the workshop, as well as placeholders for vertex and fragment shaders, the canvas and assorted functions.

```
<!DOCTYPE html>
<meta content="text/html;charset=utf-8" http-equiv="Content-Type">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>Shell document for workshop - does absolutely nothing</title>
  <script id="vertex" type="x-shader">
  </script>

  <script id="fragment" type="x-shader">
  </script>

  <script type="text/javascript">
    function init() {
      initWebGL();
      initShaderProgram();
      getVariableLocations();
      initGeometry();
      initTexture();
      tick();
    }

    function initWebGL() {
    }

    function initShaderProgram() {
    }

    function getVariableLocations() {
    }

    function initTexture() {
    }

    function initGeometry() {
    }

    function draw() {
    }

    function animate() {
    }

    function tick() {
      requestAnimationFrame(tick);
      animate();
      draw();
    }
  </script>
</head>
<body onload="init()">
  <canvas id="mycanvas" width="500" height="500">
</body>
</html>
```

1 – Shell → Simple

Todo

1. Create a WebGL context.
2. Create and compile vertex and fragment shaders.
3. Create a program, attach and link the shaders.
4. Error check.
5. Use the shader program.
6. Clear the WebGL context colour to blue.

Notes: The two shaders are the simplest possible to give `function initShaderProgram()` something to work with. Shaders are "simply" strings passed to WenGL context for compilation. Here, they're held in the "x-shader" tags for convenience. "x-shader" has no special significance and isn't recognised by HTML.
`function initWebGL()` and `function initShaderProgram()` are pretty much common to any and all raw WebGL apps.
By convention, the WebGL context is called “gl”.

```
<script id="vertex" type="x-shader">
  void main() {
  }
</script>

<script id="fragment" type="x-shader">
  void main() {
  }
</script>
```

```
  function initWebGL() {
    var canvas = document.getElementById("mycanvas");
    gl = canvas.getContext("webgl");

    var names = ["webgl", "experimental-webgl", "webkit-3d", "moz-webgl"];
    for (var i = 0; i < names.length; ++i) {
      try {
        gl = canvas.getContext(names[i]);
      }
      catch (e) { }
      if (gl) break;
    }

    gl.viewportWidth = canvas.width;
    gl.viewportHeight = canvas.height;
  }
```

```
  function initShaderProgram() {
    var v = document.getElementById("vertex").firstChild.nodeValue;
    var f = document.getElementById("fragment").firstChild.nodeValue;

    var vs = gl.createShader(gl.VERTEX_SHADER);
    gl.shaderSource(vs, v);
    gl.compileShader(vs);

    var fs = gl.createShader(gl.FRAGMENT_SHADER);
    gl.shaderSource(fs, f);
    gl.compileShader(fs);

    shaderProgram = gl.createProgram();
    gl.attachShader(shaderProgram, vs);
    gl.attachShader(shaderProgram, fs);
    gl.linkProgram(shaderProgram);

    if (!gl.getShaderParameter(vs, gl.COMPILE_STATUS))
      console.log(gl.getShaderInfoLog(vs));

    if (!gl.getShaderParameter(fs, gl.COMPILE_STATUS))
      console.log(gl.getShaderInfoLog(fs));
```

```
    if (!gl.getProgramParameter(shaderProgram, gl.LINK_STATUS))
      console.log(gl.getProgramInfoLog(shaderProgram));

    gl.useProgram(shaderProgram);
  }
```

```
  function draw() {
    gl.clearColor(0.0, 1.0, 1.0, 1.0);
    gl.clear(gl.COLOR_BUFFER_BIT);
  }
```

2 – Simple → 2D Triangle

1. Add a vertex shader.
2. Add a fragment shader.
3. Get the variable locations within the shaders.
4. Set the geometry for the triangle.
5. Draw the geometry as triangles.

Notes: WebGL is 2D/3D agnostic, no default 3D projection. If we want 3D we have to tell the shader how to do it.

attribute – Global variables that may change per vertex, that are passed from the OpenGL application to vertex shaders. This qualifier can only be used in vertex shaders. For the shader this is a read-only variable.

uniform – Global variables that may change per primitive, that are passed from the OpenGL application to the shaders. This qualifier can be used in both vertex and fragment shaders. For the shaders this is a read-only variable.

```
  <script id="vertex" type="x-shader">
    attribute vec2 aVertexPosition;

    void main() {
      gl_Position = vec4(aVertexPosition, 0.0, 1.0);
    }
  </script>
```

```
  <script id="fragment" type="x-shader">
    precision highp float;
    uniform vec4 uColor;

    void main() {
      gl_FragColor = uColor;
    }
  </script>
```

```
<script type="text/javascript">
  var shaderProgram;
  var cubeVertexPositionBuffer;
```

```
  function getVariableLocations() {
    shaderProgram.uColor = gl.getUniformLocation(shaderProgram, "uColor");

    shaderProgram.aVertexPosition = gl.getAttribLocation(shaderProgram, "aVertexPosition");
    gl.enableVertexAttribArray(shaderProgram.aVertexPosition);
  }
```

```
  function initGeometry() {
    var vertices = new Float32Array([-0.5, 0.5, 0.5, -0.5, -0.5, -0.5]);

    cubeVertexPositionBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexPositionBuffer);
    gl.bufferData(gl.ARRAY_BUFFER, vertices, gl.STATIC_DRAW);

    cubeVertexPositionBuffer.itemSize = 2;
    cubeVertexPositionBuffer.numItems = vertices.length / cubeVertexPositionBuffer.itemSize;
  }
```

```
  function draw() {
    gl.uniform4fv(shaderProgram.uColor, [0.0, 1.0, 0.0, 1.0]);

    gl.vertexAttribPointer(shaderProgram.aVertexPosition, cubeVertexPositionBuffer.itemSize, gl.FLOAT,
false, 0, 0);

    gl.clearColor(0, 0.5, 0, 1);
    gl.clear(gl.COLOR_BUFFER_BIT);

    gl.drawArrays(gl.TRIANGLES, 0, cubeVertexPositionBuffer.numItems);
  }
```

3 – Triangle → 3D square

To do

1. Change vec2 to vec3
2. Add perspective projection and model matrices
3. Change assignment to gl_Position
4. Make triangle 3d (add z ordinate)
5. Add co-ordinates for second triangle to form a triangle (initGeometry)
6. Increase itemsize (initGeometry)

Notes: `uMVMatrix` -- model and view matrices combined (position, orientation and scale).
`uPMatrix` -- projection matrix.
`uMVMatrix` and `uPMatrix` are hard coded for simplicity.
A third triangle has been added to show depth (use gl.LINE_LOOP in function draw() to see it).

```
<script id="vertex" type="x-shader">
  mat4 uMVMatrix = mat4 (1.0, 0.0, 0.0, 0.0,
                         0.0, 1.0, 0.0, 0.0,
                         0.0, 0.0, 1.0, 0.0,
                         0.0, 0.0, -3.333, 1.0 );

  mat4 uPMatrix = mat4 (2.41421, 0.0, 0.0, 0.0,
                        0.0, 2.41421, 0.0, 0.0,
                        0.0, 0.0, -1.002002, -1.0,
                        0.0, 0.0, -0.2002002, 0.0 );

  attribute vec3 aVertexPosition;

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

```
function initGeometry() {
  var vertices = new Float32Array([
  //|------ 0 ------| |------ 1 ------| |------ 2 ------|
  //|  x    y    z  | |  x    y    z  | |  x    y    z  |
      0.5, 0.5, 0.5,    0.5,-0.5, 0.5,   -0.5,-0.5, 0.5,  // triangle 1
      0.5, 0.5, 0.5,   -0.5, 0.5, 0.5,   -0.5,-0.5, 0.5,  // triangle 2
      0.5, 0.5,-0.5,    0.5,-0.5,-0.5,   -0.5,-0.5,-0.5,  // triangle 3
    ]);

    cubeVertexPositionBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexPositionBuffer);
    gl.bufferData(gl.ARRAY_BUFFER, vertices, gl.STATIC_DRAW);

    cubeVertexPositionBuffer.itemSize = 3;
```

4 – 3D Square → Coloured Cube

We got about this far…

To do

1. Add colour to faces – add attribute and varying for colour to vertex shader
2. Make cube Define vertices, faces
   2.1. Add faces
   2.2. Connect faces
   2.3. Preview with lines

Notes: added getVariableLocations to help compartmentalise the code
drawElements replaces drawArrays in function draw(), uses indexed arrays
Varying - used for interpolated data between a vertex shader and a fragment shader. Available for writing in the vertex shader, and read-only in a fragment shader.

```
<script id="vertex" type="x-shader">
  ...
  attribute vec3 aVertexPosition;
  attribute vec4 aVertexColor;

  varying vec4 vColor;

  void main() {
    gl_Position = uPMatrix * uMVMatrix * vec4(aVertexPosition, 1.0);
    vColor = aVertexColor;
  }
</script>

<script id="fragment" type="x-shader">
    uniform vec4 uColor;
    varying vec4 vColor;
    ...
    gl_FragColor = vColor;
  }
</script>

<script type="text/javascript">
  var shaderProgram;
  var cubeVertexPositionBuffer;
  var cubeVertexColorBuffer;
  var cubeVertexIndexBuffer;

  function getVariableLocations() {
    shaderProgram.uColor = gl.getUniformLocation(shaderProgram, "uColor");
    shaderProgram.vColor = gl.getUniformLocation(shaderProgram, "vColor");

    shaderProgram.vertexPositionAttribute = gl.getAttribLocation(shaderProgram, "aVertexPosition");
    gl.enableVertexAttribArray(shaderProgram.vertexPositionAttribute);

    shaderProgram.vertexColorAttribute = gl.getAttribLocation(shaderProgram, "aVertexColor");
    gl.enableVertexAttribArray(shaderProgram.vertexColorAttribute);
  }

  function initGeometry() {
    var vertices = new Float32Array([
            // Front face
            -0.5, -0.5,  0.5,
             0.5, -0.5,  0.5,
             0.5,  0.5,  0.5,
            -0.5,  0.5,  0.5,

            // Back face
            -0.5, -0.5, -0.5,
            -0.5,  0.5, -0.5,
             0.5,  0.5, -0.5,
             0.5, -0.5, -0.5,
```

```
            // Top face
            -0.5,  0.5, -0.5,
            -0.5,  0.5,  0.5,
             0.5,  0.5,  0.5,
             0.5,  0.5, -0.5,

            // Bottom face
            -0.5, -0.5, -0.5,
             0.5, -0.5, -0.5,
             0.5, -0.5,  0.5,
            -0.5, -0.5,  0.5,

            // Right face
             0.5, -0.5, -0.5,
             0.5,  0.5, -0.5,
             0.5,  0.5,  0.5,
             0.5, -0.5,  0.5,

            // Left face
            -0.5, -0.5, -0.5,
            -0.5, -0.5,  0.5,
            -0.5,  0.5,  0.5,
            -0.5,  0.5, -0.5
      ]);
    cubeVertexPositionBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexPositionBuffer);
    gl.bufferData(gl.ARRAY_BUFFER, vertices, gl.STATIC_DRAW);
    cubeVertexPositionBuffer.itemSize = 3;
    cubeVertexPositionBuffer.numItems = vertices.length / cubeVertexPositionBuffer.itemSize;

    cubeVertexColorBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexColorBuffer);
    colors = [
        [0.0, 1.0, 1.0, 1.0], // Front face
        [1.0, 1.0, 0.0, 1.0], // Back face
        [0.0, 1.0, 0.0, 1.0], // Top face
        [1.0, 0.0, 1.0, 1.0], // Bottom face
        [1.0, 0.0, 0.0, 1.0], // Right face
        [0.0, 0.0, 1.0, 1.0]  // Left face
    ];
    var unpackedColors = [];
    for (var i in colors) {
        var color = colors[i];
        for (var j=0; j < 4; j++) {
            unpackedColors = unpackedColors.concat(color);
        }
    }
    gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(unpackedColors), gl.STATIC_DRAW);
    cubeVertexColorBuffer.itemSize = 4;
    cubeVertexColorBuffer.numItems = 24;

    var indices = new Uint16Array([
        0, 1, 2,      0, 2, 3,    // Front face
        4, 5, 6,      4, 6, 7,    // Back face
        8, 9, 10,     8, 10, 11,  // Top face
        12, 13, 14,   12, 14, 15, // Bottom face
        16, 17, 18,   16, 18, 19, // Right face
        20, 21, 22,   20, 22, 23  // Left face
    ]);
    cubeVertexIndexBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, cubeVertexIndexBuffer);
    gl.bufferData(gl.ELEMENT_ARRAY_BUFFER, indices, gl.STATIC_DRAW);
    cubeVertexIndexBuffer.itemSize = 1;
    cubeVertexIndexBuffer.numItems = 36;
  }

  function draw() {
    gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexPositionBuffer);
    gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute, cubeVertexPositionBuffer.itemSize,
gl.FLOAT, false, 0, 0);

    gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexColorBuffer);
```

```
    gl.vertexAttribPointer(shaderProgram.vertexColorAttribute, cubeVertexColorBuffer.itemSize, gl.FLOAT,
false, 0, 0);

    gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, cubeVertexIndexBuffer);

    gl.clearColor(0, 0.5, 0, 1);
    gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
    gl.enable(gl.DEPTH_TEST);

    gl.drawElements(gl.TRIANGLES, cubeVertexIndexBuffer.numItems, gl.UNSIGNED_SHORT, 0);
  }
```

5 – Coloured Cube → Rotated Cube

To do

1. Include glMatrix library
2. Change `uMVMatrix` and `uPMatrix` to attributes
3. Add rotation matrix

```
<script type="text/javascript" src="glMatrix-0.9.5.min.js"></script>
```

```
<script id="vertex" type="x-shader">
  uniform mat4 uMVMatrix;
  uniform mat4 uPMatrix;

  attribute vec3 aVertexPosition;

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

```
<script type="text/javascript">
  ...
  var mvMatrix = mat4.create();
  var mvMatrixStack = [];
  var pMatrix = mat4.create();
  var rotationMatrix = mat4.create();
  mat4.identity(rotationMatrix);
  ...

  function getVariableLocations() {
    ...
    shaderProgram.pMatrixUniform = gl.getUniformLocation(shaderProgram, "uPMatrix");
    shaderProgram.mvMatrixUniform = gl.getUniformLocation(shaderProgram, "uMVMatrix");
    ...
  }
```

```
  function draw() {
    mat4.perspective(45, gl.viewportWidth / gl.viewportHeight, 0.1, 100.0, pMatrix);
    mat4.identity(mvMatrix);
    mat4.translate(mvMatrix, [0.0, 0.0, -4.0]);
    mat4.rotate(mvMatrix, 3, [1, 1, 1]);
    mat4.multiply(mvMatrix, rotationMatrix);

    gl.uniformMatrix4fv(shaderProgram.pMatrixUniform, false, pMatrix);
    gl.uniformMatrix4fv(shaderProgram.mvMatrixUniform, false, mvMatrix);
```

6 – Rotated Cube → Rotatable Cube

To do

1. Mouse input
2. Touch input
3. Update cube orientation

Notes: A lot of additional non-WebGL code to get touch and mouse input... soz ☹

```
<script type="text/javascript">
  ...
  var mouseDown = false;
  var lastMouseX = null;
  var lastMouseY = null;
  var rvelX = 0;
  var rvelY = 0;

  function degToRad(degrees) {
    return degrees * Math.PI / 180;
  }

  function handleMouseDown(event) {
    mouseDown = true;
    lastMouseX = event.clientX;
    lastMouseY = event.clientY;
  }

  function handleMouseUp(event) {
    mouseDown = false;
  }

  function handleMouseMove(event) {
    if (mouseDown)
      updateXY(event.clientX, event.clientY);
  }

  function handleTouchDown(event) {
    mouseDown = true;
    lastMouseX = event.targetTouches[0].pageX;
    lastMouseY = event.targetTouches[0].pageY;
  }

  function handleTouchMove(event) {
    event.preventDefault();
    //event.stopPropagation();

    if (mouseDown)
      updateXY(event.targetTouches[0].pageX, event.targetTouches[0].pageY);
  }

  function updateXY(newX, newY) {
    var fudgefactor = 2;
    var deltaX = newX - lastMouseX;
    var deltaY = newY - lastMouseY;
    lastMouseX = newX
    lastMouseY = newY;

    rvelX = deltaX / fudgefactor;
    rvelY = deltaY / fudgefactor;
  }
```

```
  function init() {
    var canvas = document.getElementById("mycanvas");
    canvas.onmousedown = handleMouseDown;
    document.onmouseup = handleMouseUp;
    document.onmousemove = handleMouseMove;
    canvas.addEventListener('touchstart', handleTouchDown, false);
    canvas.addEventListener('touchmove', handleTouchMove, true);
    canvas.addEventListener('touchend', handleMouseUp, false);
    ...
```

```
function animate() {
  var newRotationMatrix = mat4.create();

  mat4.identity(newRotationMatrix);
  mat4.rotate(newRotationMatrix, degToRad(rvelX), [0, 1, 0]);
  mat4.rotate(newRotationMatrix, degToRad(rvelY), [1, 0, 0]);
  mat4.multiply(newRotationMatrix, rotationMatrix, rotationMatrix);

  rvelX = rvelX / 1.08;
  if(Math.abs(rvelX) < 0.001) rvelX = 0;
  rvelY = rvelY / 1.1;
  if(Math.abs(rvelY) < 0.001) rvelY = 0;
}
```

```
function draw() {
  mat4.perspective(45, gl.viewportWidth / gl.viewportHeight, 0.1, 100.0, pMatrix);
  mat4.identity(mvMatrix);
  mat4.translate(mvMatrix, [0.0, 0.0, -4.0]);
  mat4.rotate(mvMatrix, 3, [1, 1, 1]);
  mat4.multiply(mvMatrix, rotationMatrix);
  ...
```

7 – Rotatable Cube → Dice (texture from array)

1. Include texture array
2. Add texture attribute and sampler2D
3. define texture from array

Notes: Even more code, texture handling is involved... soz ☹☹☹

The texture data is taken from an array.

```
<script type="text/javascript" src="dicemap.js"></script>
```

```
<script id="vertex" type="x-shader">
  attribute vec3 aVertexPosition;
  //attribute vec4 aVertexColor;
  attribute vec2 aTextureCoord;

  uniform mat4 uMVMatrix;
  uniform mat4 uPMatrix;

  varying vec2 vTextureCoord;
  varying vec4 vColor;

  void main() {
    gl_Position = uPMatrix * uMVMatrix * vec4(aVertexPosition, 1.0);
    //vColor = aVertexColor;
    vTextureCoord = aTextureCoord;
  }
</script>
```

```
<script id="fragment" type="x-shader">
  precision highp float;
  varying vec4 vColor;

  varying vec2 vTextureCoord;
  uniform sampler2D uSampler;

  void main() {
    //gl_FragColor = vColor;
    gl_FragColor = texture2D(uSampler, vec2(vTextureCoord.s, vTextureCoord.t));
  }
</script>
```

```
<script type="text/javascript">
  var cubeVertexTextureCoordBuffer;
  var cubeTexture;
```

```
  function getVariableLocations() {
    ...
    shaderProgram.aVertexPosition = gl.getAttribLocation(shaderProgram, "aVertexPosition");
    gl.enableVertexAttribArray(shaderProgram.aVertexPosition);

    shaderProgram.samplerUniform = gl.getUniformLocation(shaderProgram, "uSampler");
    shaderProgram.textureCoordAttribute = gl.getAttribLocation(shaderProgram, "aTextureCoord");
    gl.enableVertexAttribArray(shaderProgram.textureCoordAttribute);
```

```
    function initTexture() {
      var pixels = new Uint8Array(diceMap.length * 3);
      var k = 0;
      for (i = 0; i < diceMap.length; i++) {
        if (diceMap[i] == 'x') {
          /* Dark blue */
          pixels[k++] = 0;
          pixels[k++] = 0;
          pixels[k++] = 127;
        } else {
          /* Off-white */
          pixels[k++] = 255;
          pixels[k++] = 255;
          pixels[k++] = 240;
```

```
        }
      }
      cubeTexture = gl.createTexture();
      gl.bindTexture(gl.TEXTURE_2D, cubeTexture);
      gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_S, gl.CLAMP_TO_EDGE);
      gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_T, gl.CLAMP_TO_EDGE);
      gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.LINEAR);
      gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.LINEAR);
      gl.pixelStorei(gl.UNPACK_ALIGNMENT, 1);
      gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGB, 64, 304, 0, gl.RGB, gl.UNSIGNED_BYTE, pixels);
    }
```

```
  function initGeometry() {
    ...
   cubeVertexTextureCoordBuffer = gl.createBuffer();
   gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexTextureCoordBuffer);
   var textureCoords = [
     // Front face
     0.0, 0.0,
     1.0, 0.0,
     1.0, 63 / 304,
     0.0, 63 / 304,

     // Back face
     0.0, 241 / 304,
     1.0, 241 / 304,
     1.0, 1.0,
     0.0, 1.0,

     // Top face
     0.0, 0.197,
     1.0, 0.197,
     1.0, 0.407,
     0.0, 0.407,

     // Bottom face
     0.0, 0.332,
     1.0, 0.332,
     1.0, 0.539,
     0.0, 0.539,

     // Right face
     0.0, 161 / 304,
     1.0, 161 / 304,
     1.0, 223 / 304,
     0.0, 223 / 304,

     // Left face
     0.0, 201 / 304,
     1.0, 201 / 304,
     1.0, 263 / 304,
     0.0, 263 / 304,
    ];
    gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(textureCoords), gl.STATIC_DRAW);
    cubeVertexTextureCoordBuffer.itemSize = 2;
    cubeVertexTextureCoordBuffer.numItems = 24;
```

```
  function draw() {
    ...
    gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexTextureCoordBuffer);
    gl.vertexAttribPointer(shaderProgram.textureCoordAttribute, cubeVertexTextureCoordBuffer.itemSize,
gl.FLOAT, false, 0, 0);

    gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexTextureCoordBuffer);
    gl.vertexAttribPointer(shaderProgram.textureCoordAttribute, cubeVertexTextureCoordBuffer.itemSize,
gl.FLOAT, false, 0, 0);

    gl.activeTexture(gl.TEXTURE0);
    gl.bindTexture(gl.TEXTURE_2D, cubeTexture);
    gl.uniform1i(shaderProgram.samplerUniform, 0);
```