

# 01.triangle.html

## HTML Outline

```
<!DOCTYPE html>
<html>
<head>
  <title>Triangle</title>
</head>
<body onload="init()">
  <canvas id="mycanvas" width="600" height="600"></canvas>
  <script type="text/javascript" src="rAF.js"></script>

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

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

  <script type="text/javascript">
</script>
</body>
</html>
```

The template HTML document.

rAF.js contains the polyfill function requestAnimationFrame in case it isn't supported by your browser

The **canvas** element is used to hold a WebGL context (it could be added dynamically via JavaScript).

**<script id="vertex" type="x-shader">** and **<script id="fragment" type="x-shader">** have no special significance and in fact aren't recognised as a valid by JavaScript. They serve as a convenient place to put the GLSL shader code within the html document, this trick won't work with external .js files.

The main javascript will go in **<script type="text/javascript">**.

## JavaScript Outline

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

  function init() {
    initWebGL();
    initShaderProgram();
    initVariableLocations();
    initGeometry();
    tick();
  }

  function initWebGL() { }

  function initShaderProgram() { }

  function initVariableLocations() { }

  function initGeometry() { }

  function draw() { }

  function animate() { }

  function tick() {
    requestAnimationFrame(tick);
    animate();
    draw();
  }
</script>
```

The outline for the JavaScript – hopefully self-explanatory

# Vertex and Fragment Shaders

## Vertex Shader

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

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

## Fragment Shader

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

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

Almost the simplest possible shaders.

`precision highp float;` is a directive indicating the precision for floats (also `mediump` and `lowp`).

The vertex shader initialises a 4D vector using the 2D vector `aVertexPosition` which is then assigned to `gl_Position`.

The fragment shader assigns a two-dimensional vertex position to a four dimensional position vector (XYZW) which is assigned to `gl_Position`.

The fourth, W ordinate is added to form homogeneous coordinates to simplify multiplication and doesn't affect the results of any calculations. By convention  $W = 0$  is used for direction vectors,  $W = 1$  for position vectors.

Data is passed into the shaders via

```
attribute vec2 aVertexPosition;
```

and

```
uniform vec4 uColor;
```

GLSL also has `const` and `varying` modifiers:

**attribute** Global variables that may change per vertex. Can only be used in vertex shaders. For the shader this is a read-only variable.

**uniform** Global variables that may change per primitive. Can be used in both vertex and fragment shaders. For the shaders this is a read-only variable.

**const** The declaration is of a compile time constant.

**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.

`gl_Position` and `gl_FragColor` are built-in GLSL variables:

**gl\_Position** the clip-space output position of the current vertex.

**gl\_FragColor** the output colour (RGBA) of the current pixel.

## Initialise WebGL Context

```
function initWebGL() {
  canvas = document.getElementById("mycanvas");

  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;
  }
}
```

Get the canvas element, get a WebGL context (try various names, just in case, older browsers used different names). `gl` is used by convention – WebGL specific commands then look like their OpenGL counterparts.

## Initialise the Shader Program

```
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);
}
```

`document.getElementById("").firstChild.nodeValue;` retrieves the text for the vertex and fragment shaders.

Then the code creates a shader, assigns the appropriate text, compiles it, attaches it to the shader program, links the program, checks its status, if everything is ok it then uses the shader program. Yow, what is this the '80s?

## Get the Variable Locations from the Shader Program

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

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

To do anything meaningful with the shader we need to send data to it.

`gl.getUniformLocation` gets the specified **uniform** location.

`gl.getAttributeLocation` gets the specified **attribute** location.

## Initialise the Shape Geometry

```
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;
}
```

GLSL is strongly typed so we need an array of floats to hold the vertices,

`.itemSize` is 2, as we have 2D vertices (with x and y co-ordinates only), for 3D vertices, this would be 3 (x, y and z).

`.numItems` is number of vertices in the buffer, in this case  $6 \div 2 = 3$ .

## Draw the Geometry

```
function draw() {
  gl.vertexAttribPointer(shaderProgram.aVertexPosition, cubeVertexPositionBuffer.itemSize, gl.FLOAT, false,
  0, 0);
  gl.uniform4fv(shaderProgram.uColor, [0.0, 1.0, 0.0, 1.0]);

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

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

`gl.vertexAttribPointer` specifies the location and data format of the array of vertex attributes at `shaderProgram.aVertexPosition` to use when rendering.

`cubeVertexPositionBuffer.itemSize` specifies the number of components per attribute and must be 1, 2, 3, or 4.

`gl.FLOAT` specifies the data type of each component.

`false` indicates that values are not to be normalised when they are accessed -- mapped to the range [-1,1] (for signed values) or [0,1] (for unsigned values).

`0` specifies the offset in bytes between the beginning of consecutive vertex attributes, allowing vertices and attributes to be packed into a single array or stored in separate arrays, and `0` specifies the offset in bytes of the first component of the first vertex attribute in the array.

`gl.clearColor` sets the colour to use when the colour buffer is cleared.

`gl.clear` clears the specified buffer, in this case, the colour buffer, i.e. it clears the screen.

`gl.uniform4fv` assigns an array of 4 floats to the specified uniform.

`gl.drawArrays` draws the array of vertex data as triangles.

## Animate

```
function animate() {
  // reserved for future use... derp
}
```

Nothing to see here move along

## Complete code

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <title>Triangle</title>
5 </head>
6 <body onload="init()">
7   <canvas id="mycanvas" width="600" height="600"></canvas>
8   <script type="text/javascript" src="rAF.js"></script>
9
10  <script id="vertex" type="x-shader">
11    attribute vec2 aVertexPosition;
12
13    void main() {
14      gl_Position = vec4(aVertexPosition, 0.0, 1.0);
15    }
16 </script>
17
18  <script id="fragment" type="x-shader">
19    precision highp float;
20    uniform vec4 uColor;
21
22    void main() {
23      gl_FragColor = uColor;
24    }
25 </script>
26
27  <script type="text/javascript">
28    var shaderProgram;
29    var cubeVertexPositionBuffer;
30    var gl;
31
32    function init() {
33      initWebGL();
34      initShaderProgram();
35      initVariableLocations();
36      initGeometry();
37      tick();
38    }
39
40    function initWebGL() {
41      canvas = document.getElementById("mycanvas");
42
43      var names = ["webgl", "experimental-webgl", "webkit-3d", "moz-webgl"];
44      for (var i = 0; i < names.length; ++i) {
45        try {
46          gl = canvas.getContext(names[i]);
47        }
48        catch (e) { }
49        if (gl) break;
50      }
51    }
52
53    function initShaderProgram() {
54      var v = document.getElementById("vertex").firstChild.nodeValue;
55      var f = document.getElementById("fragment").firstChild.nodeValue;
56
57      var vs = gl.createShader(gl.VERTEX_SHADER);
58      gl.shaderSource(vs, v);
59      gl.compileShader(vs);
60
61      var fs = gl.createShader(gl.FRAGMENT_SHADER);
62      gl.shaderSource(fs, f);
63      gl.compileShader(fs);
64
65      shaderProgram = gl.createProgram();
66      gl.attachShader(shaderProgram, vs);
67      gl.attachShader(shaderProgram, fs);
68      gl.linkProgram(shaderProgram);
69
70      if (!gl.getShaderParameter(vs, gl.COMPILE_STATUS))
71        console.log(gl.getShaderInfoLog(vs));
72
```

```

73     if (!gl.getShaderParameter(fs, gl.COMPILE_STATUS))
74         console.log(gl.getShaderInfoLog(fs));
75
76     if (!gl.getProgramParameter(shaderProgram, gl.LINK_STATUS))
77         console.log(gl.getProgramInfoLog(shaderProgram));
78
79     gl.useProgram(shaderProgram);
80 }
81
82 function initVariableLocations() {
83     shaderProgram.uColor = gl.getUniformLocation(shaderProgram, "uColor");
84
85     shaderProgram.aVertexPosition = gl.getAttribLocation(shaderProgram, "aVertexPosition");
86     gl.enableVertexAttribPointer(shaderProgram.aVertexPosition);
87 }
88
89 function initGeometry() {
90     var vertices = new Float32Array([-0.5, 0.5,
91                                     0.5, -0.5,
92                                     -0.5, -0.5]);
93
94     cubeVertexPositionBuffer = gl.createBuffer();
95     gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexPositionBuffer);
96     gl.bufferData(gl.ARRAY_BUFFER, vertices, gl.STATIC_DRAW);
97
98     cubeVertexPositionBuffer.itemSize = 2;
99     cubeVertexPositionBuffer.numItems = vertices.length / cubeVertexPositionBuffer.itemSize;
100 }
101
102 function draw() {
103     gl.vertexAttribPointer(shaderProgram.aVertexPosition, cubeVertexPositionBuffer.itemSize,
104                             gl.FLOAT, false, 0, 0);
105     gl.uniform4fv(shaderProgram.uColor, [0.0, 1.0, 0.0, 1.0]);
106
107     gl.clearColor(0, 0.5, 0, 1);
108     gl.clear(gl.COLOR_BUFFER_BIT);
109
110     gl.drawArrays(gl.TRIANGLES, 0, cubeVertexPositionBuffer.numItems);
111 }
112
113 function animate() { }
114
115 function tick() {
116     requestAnimationFrame(tick);
117     animate();
118     draw();
119 }
120 </script>
121 </body>
122 </html>

```

## 01.triangle.datgui.html

### Triangle with datGUI (A Brief Diversion)

This demo uses datGUI to change the colour of the triangle, and change the position of one of the vertices.

WebGL renders into a  $2 \times 2 \times 2$  volume  $(-1, -1, -1) - (1, 1, 1)$ . So anything outside that volume is clipped. Using the datGUI interface it's possible to move a vertex outside of the clipping volume, clipping the triangle. (Moving outside the clipping volume in the xy plane doesn't seem that unusual, moving the vertex outside the z limits hopefully gives a clearer impression of the clipping volume.)

### Task - Add another Triangle to Make a Square

```
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;
}
```

Add three more points to the array vertices to make a square.

**Optional extras (for extra credit (not really) try the following:**

In the function draw(), the function gl.DrawArrays uses the gl.TRIANGLES parameter to draw *each set* of coordinates as a triangle.

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

Try different parameters instead of gl.TRIANGLES:

- gl.POINTS
- gl.LINES, gl.LINE\_LOOP

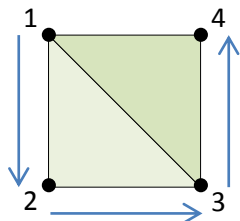
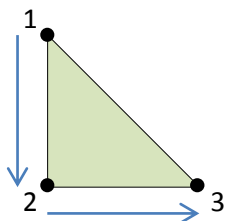
Note that we are drawing two adjacent triangles.

- Offset the vertices to separate them.

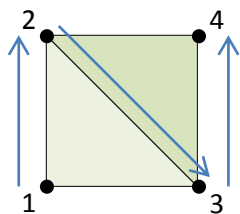
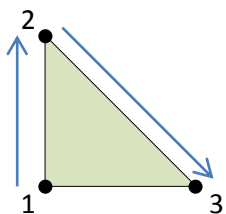


We could draw the square as a triangle fan or strip, reducing the number of vertices defined to four.

- Change the gl.DrawArrays parameter to **gl.TRIANGLE\_FAN** and alter the vertex list to work with it.



- Change the gl.DrawArrays parameter to **gl.TRIANGLE\_STRIP** and alter the vertex list to work with it.





## 02.square.boilerplate.html

### Boiler plate

Some code is always needed, changing little or not at all.

Replace repetitious code with boiler plate; hopefully making the code a bit clearer.

```
function init() {  
  var canvas = document.getElementById("mycanvas");  
  gl = WebGLUtils.setupWebGL(canvas);  
  
  shaderProgram = initShaderProgram("vertex", "fragment");  
  initVariableLocations();  
  initGeometry();  
  tick();  
}
```

Most of the changes are in the init() function.

There are multiple versions of webgl-utils.js online, some have a few extra functions some are completely different.

There are also a number of boilerplate libraries available their usefulness varies.

We will work with this code as a basis from now on.

## Complete code

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <title>Square (with boiler plate)</title>
5  </head>
6  <body onload="init()">
7    <canvas id="mycanvas" width="600" height="600"></canvas>
8
9    <script type="text/javascript" src="rAF.js"></script>
10   <script type="text/javascript" src="webgl-utils.js"></script>
11   <script type="text/javascript" src="utils.js"></script>
12
13   <script id="vertex" type="x-shader">
14     attribute vec2 aVertexPosition;
15
16     void main() {
17       gl_Position = vec4(aVertexPosition, 0.0, 1.0);
18     }
19   </script>
20
21   <script id="fragment" type="x-shader">
22     precision highp float;
23     uniform vec4 uColor;
24
25     void main() {
26       gl_FragColor = uColor;
27     }
28   </script>
29
30   <script type="text/javascript">
31     "use strict";
32     var shaderProgram;
33     var cubeVertexPositionBuffer;
34     var gl;
35
36     function init() {
37       var canvas = document.getElementById("mycanvas");
38       gl = WebGLUtils.setupWebGL(canvas);
39
40       shaderProgram = initShaderProgram("vertex", "fragment");
41       initVariableLocations();
42       initGeometry();
43       tick();
44     }
45
46     function initVariableLocations() {
47       shaderProgram.uColor = gl.getUniformLocation(shaderProgram, "uColor");
48
49       shaderProgram.aVertexPosition = gl.getAttribLocation(shaderProgram, "aVertexPosition");
50     }
51
52     function initGeometry() {
53       var vertices = new Float32Array([-0.5, 0.5,
54                                       0.5, -0.5,
55                                       -0.5, -0.5,
56                                       0.5, -0.5,
57                                       -0.5, 0.5,
58                                       0.5, 0.5]);
59
60       cubeVertexPositionBuffer = gl.createBuffer();
61       gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexPositionBuffer);
62       gl.bufferData(gl.ARRAY_BUFFER, vertices, gl.STATIC_DRAW);
63
64       cubeVertexPositionBuffer.itemSize = 2;
65       cubeVertexPositionBuffer.numItems = vertices.length / cubeVertexPositionBuffer.itemSize;
66     }
67
68     function draw() {
69       gl.enableVertexAttribArray(shaderProgram.aVertexPosition);
70       gl.vertexAttribPointer(shaderProgram.aVertexPosition, cubeVertexPositionBuffer.itemSize,
71                             gl.FLOAT, false, 0, 0);
72
```

```
73
74     gl.uniform4fv(shaderProgram.uColor, [0.0, 1.0, 0.0, 1.0]);
75     gl.clearColor(0, 0.5, 0, 1);
76     gl.clear(gl.COLOR_BUFFER_BIT);
77
78     gl.drawArrays(gl.TRIANGLES, 0, cubeVertexPositionBuffer.numItems);
79 }
80
81 function animate() { }
82
83 function tick() {
84     requestAnimationFrame(tick);
85     animate();
86     draw();
87 }
88 </script>
89 </body>
90 </html>
```

## 03.square.rotatable.html

### Rotatable Square

Let's make things a bit more entertaining.

Pointer.js is my own concoction and should also add touch support on your local device.

glMatrix, as the name suggests, is a matrix library for JavaScript. This isn't the latest version, but I find it a bit easier to use.

Add:

```
var updateVelocity = new UpdateVelocity();
function UpdateVelocity() {
  var oldPos = null;
  return function (pointer) {
    if (oldPos === null) {
      oldPos = { x: pointer.X, y: pointer.Y };
      return [0, 0];
    } else {
      var deltaX = oldPos.x - pointer.X;
      var deltaY = oldPos.y - pointer.Y;
      var fudgefactor = 2;
      var rvelx = deltaX / fudgefactor;
      var rvely = deltaY / fudgefactor;

      oldPos = { x: pointer.X, y: pointer.Y };
      return { x: rvelx, y: rvely };
    }
  }
}
```

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

  mat4.identity(newRotationMatrix);
  mat4.rotate(newRotationMatrix, degToRad(rvel.x), [0, 1, 0]);
  mat4.rotate(newRotationMatrix, degToRad(rvel.y), [1, 0, 0]);
  mat4.multiply(newRotationMatrix, rotationMatrix, rotationMatrix);

  rvel.x = rvel.x / 1.08;
  if (Math.abs(rvel.x) < 0.001) rvel.x = 0;
  rvel.y = rvel.y / 1.1;
  if (Math.abs(rvel.y) < 0.001) rvel.y = 0;
}
```

```
function tick() {
  requestAnimationFrame(tick);
  var velocity = updateVelocity(Pointer);
  if (Pointer.L) {
    rvel.x = -velocity.x;
    rvel.y = -velocity.y;
  }
  animate();
  draw();
}
```

## Complete code

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <title>Rotatable Square</title>
5 </head>
6 <body onload="init()">
7 <canvas id="mycanvas" width="600" height="600"></canvas>
8
9 <script type="text/javascript" src="rAF.js"></script>
10 <script type="text/javascript" src="webgl-utils.js"></script>
11 <script type="text/javascript" src="utils.js"></script>
12 <script type="text/javascript" src="glMatrix-0.9.5.min.js"></script>
13 <script type="text/javascript" src="Pointer.js"></script>
14
15 <script id="vertex" type="x-shader">
16     attribute vec2 aVertexPosition;
17     uniform mat4 uModelViewMatrix;
18
19     void main() {
20         gl_Position = uModelViewMatrix * vec4(aVertexPosition, 0.0, 1.0);
21     }
22 </script>
23
24 <script id="fragment" type="x-shader">
25     precision highp float;
26     uniform vec4 uColor;
27
28     void main() {
29         gl_FragColor = uColor;
30     }
31 </script>
32
33 <script type="text/javascript">
34     "use strict";
35     var shaderProgram;
36     var cubeVertexPositionBuffer;
37     var gl;
38
39     var modelViewMatrix = mat4.create();
40
41     var rotationMatrix = mat4.create();
42     mat4.identity(rotationMatrix);
43     var rvel = { x: 0, y: 0 };
44
45     function init() {
46         var canvas = document.getElementById("mycanvas");
47         gl = WebGLUtils.setupWebGL(canvas);
48
49         Pointer.initInputHandlers(canvas);
50
51         shaderProgram = initShaderProgram("vertex", "fragment");
52         initVariableLocations();
53         initGeometry();
54         tick();
55     }
56
57     function initVariableLocations() {
58         shaderProgram.uColor = gl.getUniformLocation(shaderProgram, "uColor");
59         shaderProgram.uModelViewMatrix = gl.getUniformLocation(shaderProgram, "uModelViewMatrix");
60
61         shaderProgram.aVertexPosition = gl.getAttribLocation(shaderProgram, "aVertexPosition");
62     }
63
64     function initGeometry() {
65         var vertices = new Float32Array([-0.5, 0.5,
66                                         0.5, -0.5,
67                                         -0.5, -0.5,
68                                         0.5, -0.5,
69                                         -0.5, 0.5,
70                                         0.5, 0.5]);
71
72         cubeVertexPositionBuffer = gl.createBuffer();
```

```

73     gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexPositionBuffer);
74     gl.bufferData(gl.ARRAY_BUFFER, vertices, gl.STATIC_DRAW);
75
76     cubeVertexPositionBuffer.itemSize = 2;
77     cubeVertexPositionBuffer.numItems = vertices.length / cubeVertexPositionBuffer.itemSize;
78 }
79
80 function draw() {
81     gl.enableVertexAttribArray(shaderProgram.aVertexPosition);
82     gl.vertexAttribPointer(shaderProgram.aVertexPosition, cubeVertexPositionBuffer.itemSize,
83                             gl.FLOAT, false, 0, 0);
84
85     mat4.identity(modelViewMatrix);
86     mat4.multiply(modelViewMatrix, rotationMatrix);
87     gl.uniformMatrix4fv(shaderProgram.uModelViewMatrix, false, modelViewMatrix);
88     gl.uniform4fv(shaderProgram.uColor, [0.0, 1.0, 0.0, 1.0]);
89
90     gl.clearColor(0, 0.5, 0, 1);
91     gl.clear(gl.COLOR_BUFFER_BIT);
92
93     gl.drawArrays(gl.TRIANGLES, 0, cubeVertexPositionBuffer.numItems);
94 }
95
96 var updateVelocity = new UpdateVelocity();
97 function UpdateVelocity() {
98     var oldPos = null;
99     return function (pointer) {
100         if (oldPos === null) {
101             oldPos = { x: pointer.X, y: pointer.Y };
102             return [0, 0];
103         } else {
104             var deltaX = oldPos.x - pointer.X;
105             var deltaY = oldPos.y - pointer.Y;
106             var fudgefactor = 2;
107             var rvelx = deltaX / fudgefactor;
108             var rvely = deltaY / fudgefactor;
109
110             oldPos = { x: pointer.X, y: pointer.Y };
111             return { x: rvelx, y: rvely };
112         }
113     }
114 }
115
116 function animate() {
117     var newRotationMatrix = mat4.create();
118
119     mat4.identity(newRotationMatrix);
120     mat4.rotate(newRotationMatrix, degToRad(rvel.x), [0, 1, 0]);
121     mat4.rotate(newRotationMatrix, degToRad(rvel.y), [1, 0, 0]);
122     mat4.multiply(newRotationMatrix, rotationMatrix, rotationMatrix);
123
124     rvel.x = rvel.x / 1.08;
125     if (Math.abs(rvel.x) < 0.001) rvel.x = 0;
126     rvel.y = rvel.y / 1.1;
127     if (Math.abs(rvel.y) < 0.001) rvel.y = 0;
128 }
129
130 function tick() {
131     requestAnimationFrame(tick);
132     var velocity = updateVelocity(Pointer);
133     if (Pointer.L) {
134         rvel.x = -velocity.x;
135         rvel.y = -velocity.y;
136     }
137     animate();
138     draw();
139 }
140 </script>
141 </body>
142 </html>

```

You may notice the square doesn't display perspective when rotated. Let's add a quick and dirty perspective matrix.

```
<script id="vertex" type="x-shader">
  attribute vec2 aVertexPosition;
  uniform mat4 uModelViewMatrix;

  const mat4 uProjectionMatrix = 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);

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

We need to move the square away from us otherwise it will effectively sit at the viewing position and not be visible.

```
function draw() {
  gl.enableVertexAttribArray(shaderProgram.aVertexPosition);
  gl.vertexAttribPointer(shaderProgram.aVertexPosition, cubeVertexPositionBuffer.itemSize,
                        gl.FLOAT, false, 0, 0);

  mat4.identity(modelViewMatrix);
  mat4.translate(modelViewMatrix, [0.0, 0.0, -4.0]);
  mat4.multiply(modelViewMatrix, rotationMatrix);

  gl.uniformMatrix4fv(shaderProgram.uModelViewMatrix, false, modelViewMatrix);
  gl.uniform4fv(shaderProgram.uColor, [0.0, 1.0, 0.0, 1.0]);

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

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

## 04.rotatable.cube.html

### Rotatable Cube – Single Colour

Let's move into the third dimension and make the square a cube.

Replace the vertex data for the square with data for a cube.

```
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;
}
```

**Note:** we're dealing with 3D data so we also need to set itemSize to 3.

OK, it looks like pants. This is because the code is still set to draw the data as triangles.

We need to use indices to "join up" the vertices in each face.

Add:

```
var shaderProgram;
var cubeVertexPositionBuffer;
var cubeVertexIndexBuffer;
```

Rename `initGeometry()` as `initVertices()`.

Add:

```
function initGeometry() {
  initVertices();
  initFaceIndices();
}
```



```
}
```

```
function initFaceIndices() {  
  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.enableVertexAttribArray(shaderProgram.aVertexPosition);  
  gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexPositionBuffer);  
  gl.vertexAttribPointer(shaderProgram.aVertexPosition, cubeVertexPositionBuffer.itemSize, gl.FLOAT, false, 0,  
  0);  
  
  gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, cubeVertexIndexBuffer);  
  
  mat4.identity(modelViewMatrix);  
  mat4.translate(modelViewMatrix, [0.0, 0.0, -3.0]);  
  mat4.multiply(modelViewMatrix, rotationMatrix);  
  
  gl.uniformMatrix4fv(shaderProgram.uModelViewMatrix, false, modelViewMatrix);  
  gl.uniform4fv(shaderProgram.uColor, [0.0, 1.0, 0.0, 1.0]);  
  
  gl.clearColor(0, 0.5, 0, 1);  
  gl.clear(gl.COLOR_BUFFER_BIT);  
  
  gl.drawElements(gl.TRIANGLES, cubeVertexIndexBuffer.numItems, gl.UNSIGNED_SHORT, 0);  
  //gl.drawArrays(gl.TRIANGLES, 0, cubeVertexPositionBuffer.numItems);  
}
```

To finish up let's replace the fixed perspective matrix, to make it more flexible and a bit clearer what we're doing.

```
<script id="vertex" type="x-shader">  
  attribute vec2 aVertexPosition;  
  uniform mat4 uModelViewMatrix;  
  
  uniform mat4 uProjectionMatrix;  
  
  void main() {  
    gl_Position = uProjectionMatrix * uModelViewMatrix * vec4(aVertexPosition, 0.0, 1.0);  
  }  
</script>
```

```
function initVariableLocations() {  
  shaderProgram.uColor = gl.getUniformLocation(shaderProgram, "uColor");  
  shaderProgram.uModelViewMatrix = gl.getUniformLocation(shaderProgram, "uModelViewMatrix");  
  shaderProgram.uProjectionMatrix = gl.getUniformLocation(shaderProgram, "uProjectionMatrix");  
  
  shaderProgram.aVertexPosition = gl.getAttribLocation(shaderProgram, "aVertexPosition");  
}
```

```
function draw() {  
  mat4.perspective(45, gl.drawingBufferWidth / gl.drawingBufferHeight, 0.1, 100.0,  
  projectionMatrix);  
  gl.uniformMatrix4fv(shaderProgram.uProjectionMatrix, false, projectionMatrix);  
  
  gl.enableVertexAttribArray(shaderProgram.aVertexPosition);  
  gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexPositionBuffer);
```

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

gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, cubeVertexIndexBuffer);

mat4.identity(modelViewMatrix);
mat4.translate(modelViewMatrix, [0.0, 0.0, -3.0]);
mat4.multiply(modelViewMatrix, rotationMatrix);

gl.uniformMatrix4fv(shaderProgram.uModelViewMatrix, false, modelViewMatrix);
gl.uniform4fv(shaderProgram.uColor, [0.0, 1.0, 0.0, 1.0]);

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

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

## Complete code

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <title>Rotatable Cube (Green)</title>
5  </head>
6  <body onload="init()">
7    <canvas id="mycanvas" width="600" height="600"></canvas>
8
9    <script type="text/javascript" src="rAF.js"></script>
10   <script type="text/javascript" src="webgl-utils.js"></script>
11   <script type="text/javascript" src="utils.js"></script>
12   <script type="text/javascript" src="glMatrix-0.9.5.min.js"></script>
13   <script type="text/javascript" src="Pointer.js"></script>
14
15   <script id="vertex" type="x-shader">
16     attribute vec3 aVertexPosition;
17     uniform mat4 uModelViewMatrix;
18     uniform mat4 uProjectionMatrix;
19
20     void main() {
21       gl_Position = uProjectionMatrix * uModelViewMatrix * vec4(aVertexPosition, 1.0);
22     }
23   </script>
24
25   <script id="fragment" type="x-shader">
26     precision highp float;
27     uniform vec4 uColor;
28
29     void main() {
30       gl_FragColor = uColor;
31     }
32   </script>
33
34   <script type="text/javascript">
35     "use strict";
36     var shaderProgram;
37     var cubeVertexPositionBuffer;
38     var cubeVertexIndexBuffer;
39     var gl;
40
41     var modelViewMatrix = mat4.create();
42     var projectionMatrix = mat4.create();
43
44     var rotationMatrix = mat4.create();
45     mat4.identity(rotationMatrix);
46     var rvel = { x: 0, y: 0 };
47     var updateVelocity = new UpdateVelocity();
48
49     function init() {
50       var canvas = document.getElementById("mycanvas");
51       gl = WebGLUtils.setupWebGL(canvas);
52
53       Pointer.initInputHandlers(canvas);
54
55       shaderProgram = initShaderProgram("vertex", "fragment");
56       initVariableLocations();
57       initGeometry();
58       tick();
59     }
60
61     function initVariableLocations() {
62       shaderProgram.uColor = gl.getUniformLocation(shaderProgram, "uColor");
63       shaderProgram.uModelViewMatrix = gl.getUniformLocation(shaderProgram, "uModelViewMatrix");
64       shaderProgram.uProjectionMatrix = gl.getUniformLocation(shaderProgram, "uProjectionMatrix");
65
66       shaderProgram.aVertexPosition = gl.getAttribLocation(shaderProgram, "aVertexPosition");
67     }
68
69     function initGeometry() {
70       initVertices();
71       initFaceIndices();
72     }

```

```

73
74 function initVertices() {
75     var vertices = new Float32Array([
76         // Front face
77         -0.5, -0.5, 0.5,
78         0.5, -0.5, 0.5,
79         0.5, 0.5, 0.5,
80         -0.5, 0.5, 0.5,
81
82         // Back face
83         -0.5, -0.5, -0.5,
84         -0.5, 0.5, -0.5,
85         0.5, 0.5, -0.5,
86         0.5, -0.5, -0.5,
87
88         // Top face
89         -0.5, 0.5, -0.5,
90         -0.5, 0.5, 0.5,
91         0.5, 0.5, 0.5,
92         0.5, 0.5, -0.5,
93
94         // Bottom face
95         -0.5, -0.5, -0.5,
96         0.5, -0.5, -0.5,
97         0.5, -0.5, 0.5,
98         -0.5, -0.5, 0.5,
99
100        // Right face
101        0.5, -0.5, -0.5,
102        0.5, 0.5, -0.5,
103        0.5, 0.5, 0.5,
104        0.5, -0.5, 0.5,
105
106        // Left face
107        -0.5, -0.5, -0.5,
108        -0.5, -0.5, 0.5,
109        -0.5, 0.5, 0.5,
110        -0.5, 0.5, -0.5
111    ]);
112
113    cubeVertexPositionBuffer = gl.createBuffer();
114    gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexPositionBuffer);
115    gl.bufferData(gl.ARRAY_BUFFER, vertices, gl.STATIC_DRAW);
116
117    cubeVertexPositionBuffer.itemSize = 3;
118    cubeVertexPositionBuffer.numItems = vertices.length / cubeVertexPositionBuffer.itemSize;
119 }
120
121 function initFaceIndices() {
122     var indices = new Uint16Array([
123         0, 1, 2,    0, 2, 3,    // Front face
124         4, 5, 6,    4, 6, 7,    // Back face
125         8, 9, 10,   8, 10, 11,   // Top face
126         12, 13, 14, 12, 14, 15, // Bottom face
127         16, 17, 18, 16, 18, 19, // Right face
128         20, 21, 22, 20, 22, 23 // Left face
129     ]);
130    cubeVertexIndexBuffer = gl.createBuffer();
131    gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, cubeVertexIndexBuffer);
132    gl.bufferData(gl.ELEMENT_ARRAY_BUFFER, indices, gl.STATIC_DRAW);
133    cubeVertexIndexBuffer.itemSize = 1;
134    cubeVertexIndexBuffer.numItems = 36;
135 }
136
137 function draw() {
138     mat4.perspective(45, gl.drawingBufferWidth / gl.drawingBufferHeight, 0.1, 100.0,
139                    projectionMatrix);
140     gl.uniformMatrix4fv(shaderProgram.uProjectionMatrix, false, projectionMatrix);
141
142     gl.enableVertexAttribArray(shaderProgram.aVertexPosition);
143     gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexPositionBuffer);
144     gl.vertexAttribPointer(shaderProgram.aVertexPosition, cubeVertexPositionBuffer.itemSize,
145                            gl.FLOAT, false, 0, 0);
146

```

```

147     gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, cubeVertexIndexBuffer);
148
149     mat4.identity(modelViewMatrix);
150     mat4.translate(modelViewMatrix, [0.0, 0.0, -3.0]);
151     mat4.multiply(modelViewMatrix, rotationMatrix);
152
153     gl.uniformMatrix4fv(shaderProgram.uModelViewMatrix, false, modelViewMatrix);
154     gl.uniform4fv(shaderProgram.uColor, [0.0, 1.0, 0.0, 1.0]);
155
156     gl.clearColor(0, 0.5, 0, 1);
157     gl.clear(gl.COLOR_BUFFER_BIT);
158
159     gl.drawElements(gl.TRIANGLES, cubeVertexIndexBuffer.numItems, gl.UNSIGNED_SHORT, 0);
160 }
161
162 function UpdateVelocity() {
163     var oldPos = null;
164     return function (pointer) {
165         if (oldPos === null) {
166             oldPos = { x: pointer.X, y: pointer.Y };
167             return [0, 0];
168         } else {
169             var deltaX = oldPos.x - pointer.X;
170             var deltaY = oldPos.y - pointer.Y;
171             var fudgefactor = 2;
172             var rvelx = deltaX / fudgefactor;
173             var rvely = deltaY / fudgefactor;
174
175             oldPos = { x: pointer.X, y: pointer.Y };
176             return { x: rvelx, y: rvely };
177         }
178     }
179 }
180
181 function animate() {
182     var newRotationMatrix = mat4.create();
183
184     mat4.identity(newRotationMatrix);
185     mat4.rotate(newRotationMatrix, degToRad(rvel.x), [0, 1, 0]);
186     mat4.rotate(newRotationMatrix, degToRad(rvel.y), [1, 0, 0]);
187     mat4.multiply(newRotationMatrix, rotationMatrix, rotationMatrix);
188
189     rvel.x = rvel.x / 1.08;
190     if (Math.abs(rvel.x) < 0.001) rvel.x = 0;
191     rvel.y = rvel.y / 1.1;
192     if (Math.abs(rvel.y) < 0.001) rvel.y = 0;
193 }
194
195 function tick() {
196     requestAnimationFrame(tick);
197     var velocity = updateVelocity(Pointer);
198     if (Pointer.L) {
199         rvel.x = -velocity.x;
200         rvel.y = -velocity.y;
201     }
202     animate();
203     draw();
204 }
205 </script>
206 </body>
207 </html>

```

## 05.rotatable.cube.coloured.html

### Rotatable Cube Multi-Coloured

Let's add some colour to the cube.

```
<script id="vertex" type="x-shader">
  attribute vec3 aVertexPosition;
  uniform mat4 uModelViewMatrix;
  uniform mat4 uProjectionMatrix;

  varying vec4 vColor;
  attribute vec4 aVertexColor;

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

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

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

```
<script type="text/javascript">
  "use strict";
  var shaderProgram;
  var cubeVertexPositionBuffer;
  var cubeVertexIndexBuffer;
  var cubeVertexColorBuffer;
  var gl;
```

```
function initVariableLocations() {
  shaderProgram.vertexColorAttribute = gl.getAttribLocation(shaderProgram, "aVertexColor");

  shaderProgram.uModelViewMatrix = gl.getUniformLocation(shaderProgram, "uModelViewMatrix");
  shaderProgram.uProjectionMatrix = gl.getUniformLocation(shaderProgram, "uProjectionMatrix");

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

```
function initGeometry() {
  initVertices();
  initFaceIndices();
  initColours();
}
```

```
function initColours() {
  var 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);
    }
  }
  cubeVertexColorBuffer = gl.createBuffer();
```

```

gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexColorBuffer);
gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(unpackedColors), gl.STATIC_DRAW);
cubeVertexColorBuffer.itemSize = 4;
cubeVertexColorBuffer.numItems = 24;
}

```

We need to turn on depth testing. When all faces were the same colour this wasn't an issue, but now it would be obvious that something was amiss.

```

function draw() {
  mat4.perspective(45, gl.drawingBufferWidth / gl.drawingBufferHeight, 0.1, 100.0,
    projectionMatrix);
  gl.uniformMatrix4fv(shaderProgram.uProjectionMatrix, false, projectionMatrix);

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

  gl.enableVertexAttribArray(shaderProgram.vertexColorAttribute);
  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.enable(gl.DEPTH_TEST);

  mat4.identity(modelViewMatrix);
  mat4.translate(modelViewMatrix, [0.0, 0.0, -3.0]);
  mat4.multiply(modelViewMatrix, rotationMatrix);

  gl.uniformMatrix4fv(shaderProgram.uModelViewMatrix, false, modelViewMatrix);

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

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

```

Finally, a bit more tidying – some of the code in `function draw()` only needs to be called once as the data never changes. So we can factor that out. Note: if the model data were to change then it would have to rebound in `draw()`.

Add:

```

function init() {
  var canvas = document.getElementById("mycanvas");
  gl = WebGLUtils.setupWebGL(canvas);

  Pointer.initInputHandlers(canvas);

  shaderProgram = initShaderProgram("vertex", "fragment");
  initVariableLocations();
  initGeometry();
  drawOnce();
  tick();
}

```

Split `function draw()` into two:

```

function drawOnce() {
  mat4.perspective(45, gl.drawingBufferWidth / gl.drawingBufferHeight, 0.1, 100.0,
    projectionMatrix);
  gl.uniformMatrix4fv(shaderProgram.uProjectionMatrix, false, projectionMatrix);

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

  gl.enableVertexAttribArray(shaderProgram.vertexColorAttribute);
  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.enable(gl.DEPTH_TEST);
}
```

```
function draw() {
  mat4.identity(modelViewMatrix);
  mat4.translate(modelViewMatrix, [0.0, 0.0, -3.0]);
  mat4.multiply(modelViewMatrix, rotationMatrix);

  gl.uniformMatrix4fv(shaderProgram.uModelViewMatrix, false, modelViewMatrix);

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

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



## Complete code

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <title>Rotatable Cube (Multicoloured)</title>
5  </head>
6  <body onload="init()">
7    <canvas id="mycanvas" width="600" height="600"></canvas>
8
9    <script type="text/javascript" src="rAF.js"></script>
10   <script type="text/javascript" src="webgl-utils.js"></script>
11   <script type="text/javascript" src="utils.js"></script>
12   <script type="text/javascript" src="glMatrix-0.9.5.min.js"></script>
13   <script type="text/javascript" src="Pointer.js"></script>
14
15   <script id="vertex" type="x-shader">
16     attribute vec3 aVertexPosition;
17     uniform mat4 uModelViewMatrix;
18     uniform mat4 uProjectionMatrix;
19
20     varying vec4 vColor;
21     attribute vec4 aVertexColor;
22
23     void main() {
24       gl_Position = uProjectionMatrix * uModelViewMatrix * vec4(aVertexPosition, 1.0);
25       vColor = aVertexColor;
26     }
27   </script>
28
29   <script id="fragment" type="x-shader">
30     precision highp float;
31     varying vec4 vColor;
32
33     void main() {
34       gl_FragColor = vColor;
35     }
36   </script>
37
38   <script type="text/javascript">
39     "use strict";
40     var shaderProgram;
41     var cubeVertexPositionBuffer;
42     var cubeVertexIndexBuffer;
43     var cubeVertexColorBuffer;
44     var gl;
45
46     var modelViewMatrix = mat4.create();
47     var projectionMatrix = mat4.create();
48
49     var rotationMatrix = mat4.create();
50     mat4.identity(rotationMatrix);
51     var rvel = { x: 0, y: 0 };
52     var updateVelocity = new UpdateVelocity();
53
54     function init() {
55       var canvas = document.getElementById("mycanvas");
56       gl = WebGLUtils.setupWebGL(canvas);
57
58       Pointer.initInputHandlers(canvas);
59
60       shaderProgram = initShaderProgram("vertex", "fragment");
61       initVariableLocations();
62       initGeometry();
63       drawOnce();
64       tick();
65     }
66
67     function initVariableLocations() {
68       shaderProgram.vertexColorAttribute = gl.getAttribLocation(shaderProgram, "aVertexColor");
69
70       shaderProgram.uModelViewMatrix = gl.getUniformLocation(shaderProgram, "uModelViewMatrix");
71       shaderProgram.uProjectionMatrix = gl.getUniformLocation(shaderProgram, "uProjectionMatrix");
72
```

```

73     shaderProgram.aVertexPosition = gl.getAttribLocation(shaderProgram, "aVertexPosition");
74 }
75
76 function initGeometry() {
77     initVertices();
78     initFaceIndices();
79     initColours();
80 }
81
82 function initVertices() {
83     var vertices = new Float32Array([
84         // Front face
85         -0.5, -0.5, 0.5,
86         0.5, -0.5, 0.5,
87         0.5, 0.5, 0.5,
88         -0.5, 0.5, 0.5,
89
90         // Back face
91         -0.5, -0.5, -0.5,
92         -0.5, 0.5, -0.5,
93         0.5, 0.5, -0.5,
94         0.5, -0.5, -0.5,
95
96         // Top face
97         -0.5, 0.5, -0.5,
98         -0.5, 0.5, 0.5,
99         0.5, 0.5, 0.5,
100        0.5, 0.5, -0.5,
101
102        // Bottom face
103        -0.5, -0.5, -0.5,
104        0.5, -0.5, -0.5,
105        0.5, -0.5, 0.5,
106        -0.5, -0.5, 0.5,
107
108        // Right face
109        0.5, -0.5, -0.5,
110        0.5, 0.5, -0.5,
111        0.5, 0.5, 0.5,
112        0.5, -0.5, 0.5,
113
114        // Left face
115        -0.5, -0.5, -0.5,
116        -0.5, -0.5, 0.5,
117        -0.5, 0.5, 0.5,
118        -0.5, 0.5, -0.5
119    ]);
120
121    cubeVertexPositionBuffer = gl.createBuffer();
122    gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexPositionBuffer);
123    gl.bufferData(gl.ARRAY_BUFFER, vertices, gl.STATIC_DRAW);
124
125    cubeVertexPositionBuffer.itemSize = 3;
126    cubeVertexPositionBuffer.numItems = vertices.length / cubeVertexPositionBuffer.itemSize;
127 }
128
129 function initFaceIndices() {
130     var indices = new Uint16Array([
131         0, 1, 2,    0, 2, 3,    // Front face
132         4, 5, 6,    4, 6, 7,    // Back face
133         8, 9, 10,   8, 10, 11,   // Top face
134         12, 13, 14, 12, 14, 15, // Bottom face
135         16, 17, 18, 16, 18, 19, // Right face
136         20, 21, 22, 20, 22, 23 // Left face
137    ]);
138    cubeVertexIndexBuffer = gl.createBuffer();
139    gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, cubeVertexIndexBuffer);
140    gl.bufferData(gl.ELEMENT_ARRAY_BUFFER, indices, gl.STATIC_DRAW);
141    cubeVertexIndexBuffer.itemSize = 1;
142    cubeVertexIndexBuffer.numItems = 36;
143 }
144
145 function initColours() {
146     var colors = [

```

```

147     [0.0, 1.0, 1.0, 1.0], // Front face
148     [1.0, 1.0, 0.0, 1.0], // Back face
149     [0.0, 1.0, 0.0, 1.0], // Top face
150     [1.0, 0.0, 1.0, 1.0], // Bottom face
151     [1.0, 0.0, 0.0, 1.0], // Right face
152     [0.0, 0.0, 1.0, 1.0] // Left face
153 ];
154 var unpackedColors = [];
155 for (var i in colors) {
156     var color = colors[i];
157     for (var j = 0; j < 4; j++) {
158         unpackedColors = unpackedColors.concat(color);
159     }
160 }
161 cubeVertexColorBuffer = gl.createBuffer();
162 gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexColorBuffer);
163 gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(unpackedColors), gl.STATIC_DRAW);
164 cubeVertexColorBuffer.itemSize = 4;
165 cubeVertexColorBuffer.numItems = 24;
166 }
167
168 function drawOnce() {
169     mat4.perspective(45, gl.drawingBufferWidth / gl.drawingBufferHeight, 0.1, 100.0,
170                     projectionMatrix);
171     gl.uniformMatrix4fv(shaderProgram.uProjectionMatrix, false, projectionMatrix);
172
173     gl.enableVertexAttribArray(shaderProgram.aVertexPosition);
174     gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexPositionBuffer);
175     gl.vertexAttribPointer(shaderProgram.aVertexPosition, cubeVertexPositionBuffer.itemSize,
176                           gl.FLOAT, false, 0, 0);
177
178     gl.enableVertexAttribArray(shaderProgram.vertexColorAttribute);
179     gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexColorBuffer);
180     gl.vertexAttribPointer(shaderProgram.vertexColorAttribute, cubeVertexColorBuffer.itemSize,
181                           gl.FLOAT, false, 0, 0);
182
183     gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, cubeVertexIndexBuffer);
184
185     gl.enable(gl.DEPTH_TEST);
186 }
187
188 function draw() {
189     mat4.identity(modelViewMatrix);
190     mat4.translate(modelViewMatrix, [0.0, 0.0, -3.0]);
191     mat4.multiply(modelViewMatrix, rotationMatrix);
192
193     gl.uniformMatrix4fv(shaderProgram.uModelViewMatrix, false, modelViewMatrix);
194
195     gl.clearColor(0, 0.5, 0, 1);
196     gl.clear(gl.COLOR_BUFFER_BIT);
197
198     gl.drawElements(gl.TRIANGLES, cubeVertexIndexBuffer.numItems, gl.UNSIGNED_SHORT, 0);
199 }
200
201 function UpdateVelocity() {
202     var oldPos = null;
203     return function (pointer) {
204         if (oldPos === null) {
205             oldPos = { x: pointer.X, y: pointer.Y };
206             return [0, 0];
207         } else {
208             var deltaX = oldPos.x - pointer.X;
209             var deltaY = oldPos.y - pointer.Y;
210             var fudgefactor = 2;
211             var rvelx = deltaX / fudgefactor;
212             var rvely = deltaY / fudgefactor;
213
214             oldPos = { x: pointer.X, y: pointer.Y };
215             return { x: rvelx, y: rvely };
216         }
217     }
218 }
219
220 function animate() {

```

```
221     var newRotationMatrix = mat4.create();
222
223     mat4.identity(newRotationMatrix);
224     mat4.rotate(newRotationMatrix, degToRad(rvel.x), [0, 1, 0]);
225     mat4.rotate(newRotationMatrix, degToRad(rvel.y), [1, 0, 0]);
226     mat4.multiply(newRotationMatrix, rotationMatrix, rotationMatrix);
227
228     rvel.x = rvel.x / 1.08;
229     if (Math.abs(rvel.x) < 0.001) rvel.x = 0;
230     rvel.y = rvel.y / 1.1;
231     if (Math.abs(rvel.y) < 0.001) rvel.y = 0;
232 }
233
234 function tick() {
235     requestAnimationFrame(tick);
236     var velocity = updateVelocity(Pointer);
237     if (Pointer.L) {
238         rvel.x = -velocity.x;
239         rvel.y = -velocity.y;
240     }
241     animate();
242     draw();
243 }
244 </script>
245 </body>
246 </html>
```

## 06.rotatable.dice.html

### Rotatable Dice 1 - Using Internal Data

Now we'll add the dice faces from local data. This avoids the need for a server, and shows how user defined data can be used to generate a texture.

Add:

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

dicemap.js contains an ASCII array representing the dice faces.

```
<script id="vertex" type="x-shader">
  attribute vec3 aVertexPosition;
  uniform mat4 uModelViewMatrix;
  uniform mat4 uProjectionMatrix;

  attribute vec2 aTextureCoord;
  varying vec2 vTextureCoord;

  varying vec4 vColor;
  attribute vec4 aVertexColor;

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

Extra credit: if you're feeling funky, change the code to keep the background colour.

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

  varying vec2 vTextureCoord;
  uniform sampler2D uSampler;

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

```
<script type="text/javascript">
  "use strict";
  var shaderProgram;
  var cubeVertexPositionBuffer;
  var cubeVertexIndexBuffer;
  var cubeVertexColorBuffer;
  var cubeVertexTextureCoordBuffer;
  var cubeTexture;
  var gl;
```

```
function init() {
  var canvas = document.getElementById("mycanvas");
  gl = WebGLUtils.setupWebGL(canvas);

  Pointer.initInputHandlers(canvas);

  shaderProgram = initShaderProgram("vertex", "fragment");
  initVariableLocations();
  initGeometry();
  initTexture();
  drawOnce();
  tick();
}
```

```
function initVariableLocations() {
```

```

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

shaderProgram.uModelViewMatrix = gl.getUniformLocation(shaderProgram, "uModelViewMatrix");
shaderProgram.uProjectionMatrix = gl.getUniformLocation(shaderProgram, "uProjectionMatrix");

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

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

```

```

function initGeometry() {
  initVertices();
  initFaceIndices();
  initColours();
  initTextureCoordinates();
}

```

For each “x” character stored in the “diceMap” source array, we write values to the “pixel” array. The “pixel” array is then used to create a texture.

```

function initTexture() {
  var pixels = new Uint8Array(diceMap.length * 3);
  var k = 0;
  for (var 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);
}

```

Although there are six faces the code saves space by reusing parts of the texture, leading to some magic numbers... sorry. It looks fiddly, and it is, I made at least one mistake.

```

function initTextureCoordinates() {
  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 drawOnce() {
    mat4.perspective(45, gl.drawingBufferWidth / gl.drawingBufferHeight, 0.1, 100.0,
        projectionMatrix);
    gl.uniformMatrix4fv(shaderProgram.uProjectionMatrix, false, projectionMatrix);

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

    gl.enableVertexAttribArray(shaderProgram.vertexColorAttribute);
    gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexColorBuffer);
    gl.vertexAttribPointer(shaderProgram.vertexColorAttribute, cubeVertexColorBuffer.itemSize,
        gl.FLOAT, false, 0, 0);

    gl.enableVertexAttribArray(shaderProgram.textureCoordAttribute);
    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);

    gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, cubeVertexIndexBuffer);

    gl.enable(gl.DEPTH_TEST);
}

```

## Complete code

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <title>Rotatable Dice</title>
5  </head>
6  <body onload="init()">
7    <canvas id="mycanvas" width="600" height="600"></canvas>
8
9    <script type="text/javascript" src="rAF.js"></script>
10   <script type="text/javascript" src="webgl-utils.js"></script>
11   <script type="text/javascript" src="utils.js"></script>
12   <script type="text/javascript" src="glMatrix-0.9.5.min.js"></script>
13   <script type="text/javascript" src="Pointer.js"></script>
14   <script type="text/javascript" src="dicemap.js"></script>
15
16   <script id="vertex" type="x-shader">
17     attribute vec3 aVertexPosition;
18     uniform mat4 uModelViewMatrix;
19     uniform mat4 uProjectionMatrix;
20     attribute vec2 aTextureCoord;
21
22     varying vec2 vTextureCoord;
23
24     varying vec4 vColor;
25     attribute vec4 aVertexColor;
26
27     void main() {
28       gl_Position = uProjectionMatrix* uModelViewMatrix * vec4(aVertexPosition, 1.0);
29       vColor = aVertexColor;
30       vTextureCoord = aTextureCoord;
31     }
32   </script>
33
34   <script id="fragment" type="x-shader">
35     precision highp float;
36     varying vec4 vColor;
37
38     varying vec2 vTextureCoord;
39     uniform sampler2D uSampler;
40
41     void main() {
42       gl_FragColor = texture2D(uSampler, vec2(vTextureCoord.s, vTextureCoord.t));
43     }
44   </script>
45
46   <script type="text/javascript">
47     "use strict";
48     var shaderProgram;
49     var cubeVertexPositionBuffer;
50     var cubeVertexIndexBuffer;
51     var cubeVertexColorBuffer;
52     var cubeVertexTextureCoordBuffer;
53     var cubeTexture;
54     var gl;
55
56     var modelViewMatrix = mat4.create();
57     var projectionMatrix = mat4.create();
58
59     var rotationMatrix = mat4.create();
60     mat4.identity(rotationMatrix);
61     var rvel = { x: 0, y: 0 };
62     var updateVelocity = new UpdateVelocity();
63
64     function init() {
65       var canvas = document.getElementById("mycanvas");
66       gl = WebGLUtils.setupWebGL(canvas);
67
68       Pointer.initInputHandlers(canvas);
69
70       shaderProgram = initShaderProgram("vertex", "fragment");
71       initVariableLocations();
72       initGeometry();
```



```

73     initTexture();
74     drawOnce();
75     tick();
76 }
77
78 function initVariableLocations() {
79     shaderProgram.vertexColorAttribute = gl.getAttribLocation(shaderProgram, "aVertexColor");
80
81     shaderProgram.uModelViewMatrix = gl.getUniformLocation(shaderProgram, "uModelViewMatrix");
82     shaderProgram.uProjectionMatrix = gl.getUniformLocation(shaderProgram, "uProjectionMatrix");
83
84     shaderProgram.aVertexPosition = gl.getAttribLocation(shaderProgram, "aVertexPosition");
85
86     shaderProgram.samplerUniform = gl.getUniformLocation(shaderProgram, "uSampler");
87     shaderProgram.textureCoordAttribute = gl.getAttribLocation(shaderProgram, "aTextureCoord");
88 }
89
90 function initGeometry() {
91     initVertices();
92     initFaceIndices();
93     initColours();
94     initTextureCoordinates();
95 }
96
97 function initVertices() {
98     var vertices = new Float32Array([
99         // Front face
100        -0.5, -0.5, 0.5,
101         0.5, -0.5, 0.5,
102         0.5, 0.5, 0.5,
103        -0.5, 0.5, 0.5,
104
105        // Back face
106        -0.5, -0.5, -0.5,
107        -0.5, 0.5, -0.5,
108         0.5, 0.5, -0.5,
109         0.5, -0.5, -0.5,
110
111        // Top face
112        -0.5, 0.5, -0.5,
113        -0.5, 0.5, 0.5,
114         0.5, 0.5, 0.5,
115         0.5, 0.5, -0.5,
116
117        // Bottom face
118        -0.5, -0.5, -0.5,
119         0.5, -0.5, -0.5,
120         0.5, -0.5, 0.5,
121        -0.5, -0.5, 0.5,
122
123        // Right face
124         0.5, -0.5, -0.5,
125         0.5, 0.5, -0.5,
126         0.5, 0.5, 0.5,
127         0.5, -0.5, 0.5,
128
129        // Left face
130        -0.5, -0.5, -0.5,
131        -0.5, -0.5, 0.5,
132        -0.5, 0.5, 0.5,
133        -0.5, 0.5, -0.5
134    ]);
135
136    cubeVertexPositionBuffer = gl.createBuffer();
137    gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexPositionBuffer);
138    gl.bufferData(gl.ARRAY_BUFFER, vertices, gl.STATIC_DRAW);
139
140    cubeVertexPositionBuffer.itemSize = 3;
141    cubeVertexPositionBuffer.numItems = vertices.length / cubeVertexPositionBuffer.itemSize;
142 }
143
144 function initFaceIndices() {
145     var indices = new Uint16Array([
146         0, 1, 2,     0, 2, 3,     // Front face

```

```

147     4, 5, 6,     4, 6, 7,     // Back face
148     8, 9, 10,   8, 10, 11,  // Top face
149     12, 13, 14, 12, 14, 15, // Bottom face
150     16, 17, 18, 16, 18, 19, // Right face
151     20, 21, 22, 20, 22, 23  // Left face
152 ];
153 cubeVertexIndexBuffer = gl.createBuffer();
154 gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, cubeVertexIndexBuffer);
155 gl.bufferData(gl.ELEMENT_ARRAY_BUFFER, indices, gl.STATIC_DRAW);
156 cubeVertexIndexBuffer.itemSize = 1;
157 cubeVertexIndexBuffer.numItems = 36;
158 }
159
160 function initColours() {
161     var colors = [
162         [0.0, 1.0, 1.0, 1.0], // Front face
163         [1.0, 1.0, 0.0, 1.0], // Back face
164         [0.0, 1.0, 0.0, 1.0], // Top face
165         [1.0, 0.0, 1.0, 1.0], // Bottom face
166         [1.0, 0.0, 0.0, 1.0], // Right face
167         [0.0, 0.0, 1.0, 1.0] // Left face
168     ];
169     var unpackedColors = [];
170     for (var i in colors) {
171         var color = colors[i];
172         for (var j = 0; j < 4; j++) {
173             unpackedColors = unpackedColors.concat(color);
174         }
175     }
176     cubeVertexColorBuffer = gl.createBuffer();
177     gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexColorBuffer);
178     gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(unpackedColors), gl.STATIC_DRAW);
179     cubeVertexColorBuffer.itemSize = 4;
180     cubeVertexColorBuffer.numItems = 24;
181 }
182
183 function initTexture() {
184     var pixels = new Uint8Array(diceMap.length * 3);
185     var k = 0;
186     for (var i = 0; i < diceMap.length; i++) {
187         if (diceMap[i] == 'x') {
188             /* Dark blue */
189             pixels[k++] = 0;
190             pixels[k++] = 0;
191             pixels[k++] = 127;
192         } else {
193             /* Off-white */
194             pixels[k++] = 255;
195             pixels[k++] = 255;
196             pixels[k++] = 240;
197         }
198     }
199
200     cubeTexture = gl.createTexture();
201     gl.bindTexture(gl.TEXTURE_2D, cubeTexture);
202     gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_S, gl.CLAMP_TO_EDGE);
203     gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_T, gl.CLAMP_TO_EDGE);
204     gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.LINEAR);
205     gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.LINEAR);
206     gl.pixelStorei(gl.UNPACK_ALIGNMENT, 1);
207     gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGB, 64, 304, 0, gl.RGB, gl.UNSIGNED_BYTE, pixels);
208 }
209
210 function initTextureCoordinates() {
211     cubeVertexTextureCoordBuffer = gl.createBuffer();
212     gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexTextureCoordBuffer);
213     var textureCoords = [
214         // Front face
215         0.0, 0.0,
216         1.0, 0.0,
217         1.0, 63 / 304,
218         0.0, 63 / 304,
219
220         // Back face

```

```

221     0.0, 241 / 304,
222     1.0, 241 / 304,
223     1.0, 1.0,
224     0.0, 1.0,
225
226     // Top face
227     0.0, 0.197,
228     1.0, 0.197,
229     1.0, 0.407,
230     0.0, 0.407,
231
232     // Bottom face
233     0.0, 0.332,
234     1.0, 0.332,
235     1.0, 0.539,
236     0.0, 0.539,
237
238     // Right face
239     0.0, 161 / 304,
240     1.0, 161 / 304,
241     1.0, 223 / 304,
242     0.0, 223 / 304,
243
244     // Left face
245     0.0, 201 / 304,
246     1.0, 201 / 304,
247     1.0, 263 / 304,
248     0.0, 263 / 304,
249 ];
250 gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(textureCoords), gl.STATIC_DRAW);
251 cubeVertexTextureCoordBuffer.itemSize = 2;
252 cubeVertexTextureCoordBuffer.numItems = 24;
253 }
254
255 function drawOnce() {
256     mat4.perspective(45, gl.drawingBufferWidth / gl.drawingBufferHeight, 0.1, 100.0,
257                     projectionMatrix);
258     gl.uniformMatrix4fv(shaderProgram.uProjectionMatrix, false, projectionMatrix);
259
260     gl.enableVertexAttribArray(shaderProgram.aVertexPosition);
261     gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexPositionBuffer);
262     gl.vertexAttribPointer(shaderProgram.aVertexPosition, cubeVertexPositionBuffer.itemSize,
263                           gl.FLOAT, false, 0, 0);
264
265     gl.enableVertexAttribArray(shaderProgram.vertexColorAttribute);
266     gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexColorBuffer);
267     gl.vertexAttribPointer(shaderProgram.vertexColorAttribute, cubeVertexColorBuffer.itemSize,
268                           gl.FLOAT, false, 0, 0);
269
270     gl.enableVertexAttribArray(shaderProgram.textureCoordAttribute);
271     gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexTextureCoordBuffer);
272     gl.vertexAttribPointer(shaderProgram.textureCoordAttribute,
273                           cubeVertexTextureCoordBuffer.itemSize, gl.FLOAT, false, 0, 0);
274
275     gl.activeTexture(gl.TEXTURE0);
276     gl.bindTexture(gl.TEXTURE_2D, cubeTexture);
277     gl.uniform1i(shaderProgram.samplerUniform, 0);
278
279     gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, cubeVertexIndexBuffer);
280
281     gl.enable(gl.DEPTH_TEST);
282 }
283
284 function draw() {
285     mat4.identity(modelViewMatrix);
286     mat4.translate(modelViewMatrix, [0.0, 0.0, -3.0]);
287     mat4.multiply(modelViewMatrix, rotationMatrix);
288
289     gl.uniformMatrix4fv(shaderProgram.uModelViewMatrix, false, modelViewMatrix);
290
291     gl.clearColor(0, 0.5, 0, 1);
292     gl.clear(gl.COLOR_BUFFER_BIT);
293
294     gl.drawElements(gl.TRIANGLES, cubeVertexIndexBuffer.numItems, gl.UNSIGNED_SHORT, 0);

```

```

295     }
296
297     function UpdateVelocity() {
298         var oldPos = null;
299         return function (pointer) {
300             if (oldPos === null) {
301                 oldPos = { x: pointer.X, y: pointer.Y };
302                 return [0, 0];
303             } else {
304                 var deltaX = oldPos.x - pointer.X;
305                 var deltaY = oldPos.y - pointer.Y;
306                 var fudgefactor = 2;
307                 var rvelx = deltaX / fudgefactor;
308                 var rvely = deltaY / fudgefactor;
309
310                 oldPos = { x: pointer.X, y: pointer.Y };
311                 return { x: rvelx, y: rvely };
312             }
313         }
314     }
315
316     function animate() {
317         var newRotationMatrix = mat4.create();
318
319         mat4.identity(newRotationMatrix);
320         mat4.rotate(newRotationMatrix, degToRad(rvel.x), [0, 1, 0]);
321         mat4.rotate(newRotationMatrix, degToRad(rvel.y), [1, 0, 0]);
322         mat4.multiply(newRotationMatrix, rotationMatrix, rotationMatrix);
323
324         rvel.x = rvel.x / 1.08;
325         if (Math.abs(rvel.x) < 0.001) rvel.x = 0;
326         rvel.y = rvel.y / 1.1;
327         if (Math.abs(rvel.y) < 0.001) rvel.y = 0;
328     }
329
330     function tick() {
331         requestAnimationFrame(tick);
332         var velocity = updateVelocity(Pointer);
333         if (Pointer.L) {
334             rvel.x = -velocity.x;
335             rvel.y = -velocity.y;
336         }
337         animate();
338         draw();
339     }
340 </script>
341 </body>
342 </html>

```

## 07.rotatable.dice.ext.texture.html

### Rotatable Dice 2 Using External Texture

Let's load the texture from an external file. You'll need a server for this or you'll fall foul of the CORS problem.

Remove:

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

```
function init() {
  var canvas = document.getElementById("mycanvas");
  gl = WebGLUtils.setupWebGL(canvas);

  Pointer.initInputHandlers(canvas);

  shaderProgram = initShaderProgram("vertex", "fragment");
  initVariableLocations();
  initGeometry();
  initTexture();
  //drawOnce();
  //tick();
}
```

As we'll be loading an image asynchronously we have to delay drawing until it's loaded.

Add:

```
function initTexture() {
  cubeTexture = gl.createTexture();
  var cubeImage = new Image();
  cubeImage.onload = function() { handleTextureLoaded(cubeImage, cubeTexture); }
  cubeImage.src = "1.png";
}
```

The binding, etc. is moved to a callback.

```
function handleTextureLoaded(image, texture) {
  gl.bindTexture(gl.TEXTURE_2D, texture);
  gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED_BYTE, image);
  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.generateMipmap(gl.TEXTURE_2D);
  gl.bindTexture(gl.TEXTURE_2D, null);
  drawOnce();
  tick();
}
```

The texture coordinates are a bit more sensible as we have six stacked images at regular intervals.

```
function initTextureCoordinates() {
  cubeVertexTextureCoordBuffer = gl.createBuffer();
  gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexTextureCoordBuffer);
  var textureCoords = [
    // Front
    0.0, 0.0,
    1.0, 0.0,
    1.0, 1.0,
    0.0, 1.0,
    // Back
    0.0, 0.0,
    1.0, 0.0,
    1.0, 1.0,
    0.0, 1.0,
    // Top
    0.0, 0.0,
    1.0, 0.0,
    1.0, 1.0,
    0.0, 1.0,
    // Bottom
    0.0, 0.0,
```

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

And that's your lot.

The more adventurous amongst you may like to experiment with cube maps.

## Complete code

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <title>Rotatable Dice</title>
5  </head>
6  <body onload="init()">
7    <canvas id="mycanvas" width="600" height="600"></canvas>
8
9    <script type="text/javascript" src="rAF.js"></script>
10   <script type="text/javascript" src="webgl-utils.js"></script>
11   <script type="text/javascript" src="utils.js"></script>
12   <script type="text/javascript" src="glMatrix-0.9.5.min.js"></script>
13   <script type="text/javascript" src="Pointer.js"></script>
14
15   <script id="vertex" type="x-shader">
16     attribute vec3 aVertexPosition;
17     uniform mat4 uModelViewMatrix;
18     uniform mat4 uProjectionMatrix;
19     attribute vec2 aTextureCoord;
20
21     varying vec2 vTextureCoord;
22
23     varying vec4 vColor;
24     attribute vec4 aVertexColor;
25
26     void main() {
27       gl_Position = uProjectionMatrix* uModelViewMatrix * vec4(aVertexPosition, 1.0);
28       vColor = aVertexColor;
29       vTextureCoord = aTextureCoord;
30     }
31   </script>
32
33   <script id="fragment" type="x-shader">
34     precision highp float;
35     varying vec4 vColor;
36
37     varying vec2 vTextureCoord;
38     uniform sampler2D uSampler;
39
40     void main() {
41       gl_FragColor = texture2D(uSampler, vec2(vTextureCoord.s, vTextureCoord.t));
42     }
43   </script>
44
45   <script type="text/javascript">
46     "use strict";
47     var shaderProgram;
48     var cubeVertexPositionBuffer;
49     var cubeVertexIndexBuffer;
50     var cubeVertexColorBuffer;
51     var cubeVertexTextureCoordBuffer;
52     var cubeTexture;
53     var gl;
54
55     var modelViewMatrix = mat4.create();
56     var projectionMatrix = mat4.create();
57
58     var rotationMatrix = mat4.create();
59     mat4.identity(rotationMatrix);
60     var rvel = { x: 0, y: 0 };
61     var updateVelocity = new UpdateVelocity();
62
63     function init() {
64       var canvas = document.getElementById("mycanvas");
65       gl = WebGLUtils.setupWebGL(canvas);
66
67       Pointer.initInputHandlers(canvas);
68
69       shaderProgram = initShaderProgram("vertex", "fragment");
70       initVariableLocations();
71       initGeometry();
72       initTexture();
```

```

73     //drawOnce();
74     //tick();
75 }
76
77 function initVariableLocations() {
78     shaderProgram.vertexColorAttribute = gl.getAttribLocation(shaderProgram, "aVertexColor");
79
80     shaderProgram.uModelViewMatrix = gl.getUniformLocation(shaderProgram, "uModelViewMatrix");
81     shaderProgram.uProjectionMatrix = gl.getUniformLocation(shaderProgram, "uProjectionMatrix");
82
83     shaderProgram.aVertexPosition = gl.getAttribLocation(shaderProgram, "aVertexPosition");
84
85     shaderProgram.samplerUniform = gl.getUniformLocation(shaderProgram, "uSampler");
86     shaderProgram.textureCoordAttribute = gl.getAttribLocation(shaderProgram, "aTextureCoord");
87 }
88
89 function initGeometry() {
90     initVertices();
91     initFaceIndices();
92     initColours();
93     initTextureCoordinates();
94 }
95
96 function initVertices() {
97     var vertices = new Float32Array([
98         // Front face
99         -0.5, -0.5, 0.5,
100         0.5, -0.5, 0.5,
101         0.5, 0.5, 0.5,
102         -0.5, 0.5, 0.5,
103
104         // Back face
105         -0.5, -0.5, -0.5,
106         -0.5, 0.5, -0.5,
107         0.5, 0.5, -0.5,
108         0.5, -0.5, -0.5,
109
110         // Top face
111         -0.5, 0.5, -0.5,
112         -0.5, 0.5, 0.5,
113         0.5, 0.5, 0.5,
114         0.5, 0.5, -0.5,
115
116         // Bottom face
117         -0.5, -0.5, -0.5,
118         0.5, -0.5, -0.5,
119         0.5, -0.5, 0.5,
120         -0.5, -0.5, 0.5,
121
122         // Right face
123         0.5, -0.5, -0.5,
124         0.5, 0.5, -0.5,
125         0.5, 0.5, 0.5,
126         0.5, -0.5, 0.5,
127
128         // Left face
129         -0.5, -0.5, -0.5,
130         -0.5, -0.5, 0.5,
131         -0.5, 0.5, 0.5,
132         -0.5, 0.5, -0.5
133     ]);
134
135     cubeVertexPositionBuffer = gl.createBuffer();
136     gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexPositionBuffer);
137     gl.bufferData(gl.ARRAY_BUFFER, vertices, gl.STATIC_DRAW);
138
139     cubeVertexPositionBuffer.itemSize = 3;
140     cubeVertexPositionBuffer.numItems = vertices.length / cubeVertexPositionBuffer.itemSize;
141 }
142
143 function initFaceIndices() {
144     var indices = new Uint16Array([
145         0, 1, 2,    0, 2, 3,    // Front face
146         4, 5, 6,    4, 6, 7,    // Back face

```



```

147         8, 9, 10,    8, 10, 11, // Top face
148         12, 13, 14, 12, 14, 15, // Bottom face
149         16, 17, 18, 16, 18, 19, // Right face
150         20, 21, 22, 20, 22, 23 // Left face
151     });
152     cubeVertexIndexBuffer = gl.createBuffer();
153     gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, cubeVertexIndexBuffer);
154     gl.bufferData(gl.ELEMENT_ARRAY_BUFFER, indices, gl.STATIC_DRAW);
155     cubeVertexIndexBuffer.itemSize = 1;
156     cubeVertexIndexBuffer.numItems = 36;
157 }
158
159 function initColours() {
160     var colors = [
161         [0.0, 1.0, 1.0, 1.0], // Front face
162         [1.0, 1.0, 0.0, 1.0], // Back face
163         [0.0, 1.0, 0.0, 1.0], // Top face
164         [1.0, 0.0, 1.0, 1.0], // Bottom face
165         [1.0, 0.0, 0.0, 1.0], // Right face
166         [0.0, 0.0, 1.0, 1.0] // Left face
167     ];
168     var unpackedColors = [];
169     for (var i in colors) {
170         var color = colors[i];
171         for (var j = 0; j < 4; j++) {
172             unpackedColors = unpackedColors.concat(color);
173         }
174     }
175     cubeVertexColorBuffer = gl.createBuffer();
176     gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexColorBuffer);
177     gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(unpackedColors), gl.STATIC_DRAW);
178     cubeVertexColorBuffer.itemSize = 4;
179     cubeVertexColorBuffer.numItems = 24;
180 }
181
182 function initTexture() {
183     cubeTexture = gl.createTexture();
184     var cubeImage = new Image();
185     cubeImage.onload = function() { handleTextureLoaded(cubeImage, cubeTexture); }
186     cubeImage.src = "1.png";
187 }
188
189 function handleTextureLoaded(image, texture) {
190     gl.bindTexture(gl.TEXTURE_2D, texture);
191     gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED_BYTE, image);
192     gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_S, gl.CLAMP_TO_EDGE);
193     gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_T, gl.CLAMP_TO_EDGE);
194     gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.LINEAR);
195     gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.LINEAR);
196     gl.generateMipmap(gl.TEXTURE_2D);
197     gl.bindTexture(gl.TEXTURE_2D, null);
198     drawOnce();
199     tick();
200 }
201
202 function initTextureCoordinates() {
203     cubeVertexTextureCoordBuffer = gl.createBuffer();
204     gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexTextureCoordBuffer);
205     var textureCoords = [
206         // Front
207         0.0, 0.0,
208         1.0, 0.0,
209         1.0, 1.0,
210         0.0, 1.0,
211         // Back
212         0.0, 0.0,
213         1.0, 0.0,
214         1.0, 1.0,
215         0.0, 1.0,
216         // Top
217         0.0, 0.0,
218         1.0, 0.0,
219         1.0, 1.0,
220         0.0, 1.0,

```

```

221     // Bottom
222     0.0, 0.0,
223     1.0, 0.0,
224     1.0, 1.0,
225     0.0, 1.0,
226     // Right
227     0.0, 0.0,
228     1.0, 0.0,
229     1.0, 1.0,
230     0.0, 1.0,
231     // Left
232     0.0, 0.0,
233     1.0, 0.0,
234     1.0, 1.0,
235     0.0, 1.0
236 ];
237 gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(textureCoords), gl.STATIC_DRAW);
238 cubeVertexTextureCoordBuffer.itemSize = 2;
239 cubeVertexTextureCoordBuffer.numItems = 24;
240 }
241
242 function drawOnce() {
243     mat4.perspective(45, gl.drawingBufferWidth / gl.drawingBufferHeight, 0.1, 100.0,
244                     projectionMatrix);
245     gl.uniformMatrix4fv(shaderProgram.uProjectionMatrix, false, projectionMatrix);
246
247     gl.enableVertexAttribArray(shaderProgram.aVertexPosition);
248     gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexPositionBuffer);
249     gl.vertexAttribPointer(shaderProgram.aVertexPosition, cubeVertexPositionBuffer.itemSize,
250                            gl.FLOAT, false, 0, 0);
251
252     gl.enableVertexAttribArray(shaderProgram.vertexColorAttribute);
253     gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexColorBuffer);
254     gl.vertexAttribPointer(shaderProgram.vertexColorAttribute, cubeVertexColorBuffer.itemSize,
255                            gl.FLOAT, false, 0, 0);
256
257     gl.enableVertexAttribArray(shaderProgram.textureCoordAttribute);
258     gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexTextureCoordBuffer);
259     gl.vertexAttribPointer(shaderProgram.textureCoordAttribute,
260                            cubeVertexTextureCoordBuffer.itemSize, gl.FLOAT, false, 0, 0);
261
262     gl.activeTexture(gl.TEXTURE0);
263     gl.bindTexture(gl.TEXTURE_2D, cubeTexture);
264     gl.uniform1i(shaderProgram.samplerUniform, 0);
265
266     gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, cubeVertexIndexBuffer);
267
268     gl.enable(gl.DEPTH_TEST);
269 }
270
271 function draw() {
272     mat4.identity(modelViewMatrix);
273     mat4.translate(modelViewMatrix, [0.0, 0.0, -3.0]);
274     mat4.multiply(modelViewMatrix, rotationMatrix);
275
276     gl.uniformMatrix4fv(shaderProgram.uModelViewMatrix, false, modelViewMatrix);
277
278     gl.clearColor(0, 0.5, 0, 1);
279     gl.clear(gl.COLOR_BUFFER_BIT);
280
281     gl.drawElements(gl.TRIANGLES, cubeVertexIndexBuffer.numItems, gl.UNSIGNED_SHORT, 0);
282 }
283
284 function UpdateVelocity() {
285     var oldPos = null;
286     return function (pointer) {
287         if (oldPos === null) {
288             oldPos = { x: pointer.X, y: pointer.Y };
289             return [0, 0];
290         } else {
291             var deltaX = oldPos.x - pointer.X;
292             var deltaY = oldPos.y - pointer.Y;
293             var fudgefactor = 2;
294             var rvelx = deltaX / fudgefactor;

```

```

295     var rvely = deltaY / fudgefactor;
296
297     oldPos = { x: pointer.X, y: pointer.Y };
298     return { x: rvelx, y: rvely };
299   }
300 }
301 }
302
303 function animate() {
304   var newRotationMatrix = mat4.create();
305
306   mat4.identity(newRotationMatrix);
307   mat4.rotate(newRotationMatrix, degToRad(rvel.x), [0, 1, 0]);
308   mat4.rotate(newRotationMatrix, degToRad(rvel.y), [1, 0, 0]);
309   mat4.multiply(newRotationMatrix, rotationMatrix, rotationMatrix);
310
311   rvel.x = rvel.x / 1.08;
312   if (Math.abs(rvel.x) < 0.001) rvel.x = 0;
313   rvel.y = rvel.y / 1.1;
314   if (Math.abs(rvel.y) < 0.001) rvel.y = 0;
315 }
316
317 function tick() {
318   requestAnimationFrame(tick);
319   var velocity = updateVelocity(Pointer);
320   if (Pointer.L) {
321     rvel.x = -velocity.x;
322     rvel.y = -velocity.y;
323   }
324   animate();
325   draw();
326 }
327 </script>
328 </body>
329 </html>

```

There are workarounds but they are complicated.

“There is no simple way to pass per-face data to WebGL.

“In WebGL (and OpenGL ES 2) there are two kinds of data that can be passed to the shaders: vertex attributes and uniforms. Vertex attributes can have a different value for each vertex, and uniform data has a single value that only changes explicitly via one of the `uniform*()` calls.

“When a vertex attribute, say a color, is passed directly from the vertex shader to the fragment shader, it is automatically interpolated by the hardware so that the result is a smooth change of the value (color) over the surface of the resulting triangle. So to give a triangle a "flat" color, all of the vertices of the triangle need to be given the same color. Since the vertices will also be part of other triangles that will likely have different colors, the only option is to create a separate vertex entry for each different color of each face that uses the same point.

“If you were to use a uniform instead, the uniform would have to be changed as each face is drawn. This would mean you would have to separate the cube into 6 parts, and make 6 separate drawing calls. This is much less efficient. It is well accepted that the "less evil" solution is to have multiple vertex entries as described and use up some extra GPU memory to make the drawing faster.

“In other versions of OpenGL there is a function `glVertexAttribDivisor()` that lets you set a vertex attribute that only changes every `n` vertices. This can be used to achieve what you are trying to do. However it is unfortunately not available in WebGL.”