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">`.

JaveScript 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.getAttribLocation(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.getAttribLocation` 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 ÷ 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

```
<!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">
    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;
    var gl;

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

    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;
      }
    }

    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 initVariableLocations() {
      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.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);
    }

    function animate() { }

    function tick() {
      requestAnimationFrame(tick);
      animate();
      draw();
    }
  </script>
</body>
</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 2×2×2 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.

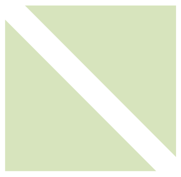
```
      gl.drawArrays(gl.TRIANGLES, 0, cubeVertexPositionBuffer.numItems);
```

Try different parameters instead of gl.TRIANGLES:

- gl.POINTS
- gl.LINES, gl.LINE_LOOP

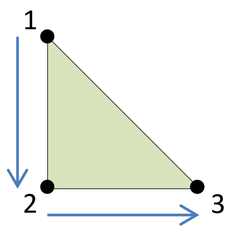
Note that we are drawing two adjacent triangles.

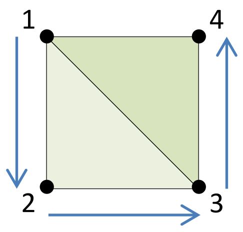
- Offset the vertices to separate them.



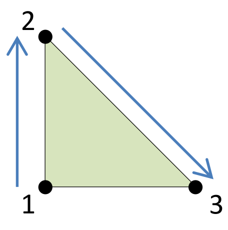
We could draw the square as a triangle fan or strip, reducing the number of vertices defined to four.

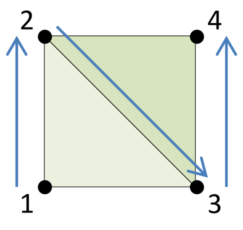
- Change the gl.DrawArrays parameter to **gl.TRIANGLE_FAN** and alter the vertex list to work with it.





- 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

```
<!DOCTYPE html>
<html>
<head>
  <title>Square (with boiler plate)</title
</head>
<body onload="init()">
  <canvas id="mycanvas" width="600" height="600"></canvas>

  <script type="text/javascript" src="rAF.js"></script>
  <script type="text/javascript" src="webgl-utils.js"></script>
  <script type="text/javascript" src="utils.js"></script>

  <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">
    "use strict";
    var shaderProgram;
    var cubeVertexPositionBuffer;
    var gl;

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

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

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

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

    function initGeometry() {
      var vertices = new Float32Array([-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 = 2;
      cubeVertexPositionBuffer.numItems = vertices.length / cubeVertexPositionBuffer.itemSize;
    }

    function draw() {
      gl.enableVertexAttribArray(shaderProgram.aVertexPosition);
      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);
      }

      function animate() { }

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

```
<!DOCTYPE html>
<html>
<head>
  <title>Rotatable Square</title
</head>
<body onload="init()">
  <canvas id="mycanvas" width="600" height="600"></canvas>

  <script type="text/javascript" src="rAF.js"></script>
  <script type="text/javascript" src="webgl-utils.js"></script>
  <script type="text/javascript" src="utils.js"></script>
  <script type="text/javascript" src="glMatrix-0.9.5.min.js"></script>
  <script type="text/javascript" src="Pointer.js"></script>

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

    void main() {
      gl_Position = uModelViewMatrix * 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">
    "use strict";
    var shaderProgram;
    var cubeVertexPositionBuffer;
    var gl;

    var modelViewMatrix = mat4.create();

    var rotationMatrix = mat4.create();
    mat4.identity(rotationMatrix);
    var rvel = { x: 0, y: 0 };

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

      Pointer.initInputHandlers(canvas);

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

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

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

    function initGeometry() {
      var vertices = new Float32Array([-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 = 2;
      cubeVertexPositionBuffer.numItems = vertices.length / cubeVertexPositionBuffer.itemSize;
    }

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

      mat4.identity(modelViewMatrix);
      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);
    }

    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();
    }
  </script>
</body>
</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

```
<!DOCTYPE html>
<html>
<head>
  <title>Rotatable Cube (Green)</title>
</head>
<body onload="init()">
  <canvas id="mycanvas" width="600" height="600"></canvas>

  <script type="text/javascript" src="rAF.js"></script>
  <script type="text/javascript" src="webgl-utils.js"></script>
  <script type="text/javascript" src="utils.js"></script>
  <script type="text/javascript" src="glMatrix-0.9.5.min.js"></script>
  <script type="text/javascript" src="Pointer.js"></script>

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

    void main() {
      gl_Position = uProjectionMatrix * uModelViewMatrix * vec4(aVertexPosition, 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">
    "use strict";
    var shaderProgram;
    var cubeVertexPositionBuffer;
    var cubeVertexIndexBuffer;
    var gl;

    var modelViewMatrix = mat4.create();
    var projectionMatrix = mat4.create();

    var rotationMatrix = mat4.create();
    mat4.identity(rotationMatrix);
    var rvel = { x: 0, y: 0 };
    var updateVelocity = new UpdateVelocity();

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

      Pointer.initInputHandlers(canvas);

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

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

```

      function initVertices() {
        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;
      }

      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() {
        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);
      }

      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();
      }
    </script>
  </body>
  </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

```html
<!DOCTYPE html>
<html>
<head>
  <title>Rotatable Cube (Multicoloured)</title>
</head>
<body onload="init()">
  <canvas id="mycanvas" width="600" height="600"></canvas>

  <script type="text/javascript" src="rAF.js"></script>
  <script type="text/javascript" src="webgl-utils.js"></script>
  <script type="text/javascript" src="utils.js"></script>
  <script type="text/javascript" src="glMatrix-0.9.5.min.js"></script>
  <script type="text/javascript" src="Pointer.js"></script>

  <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;

    var modelViewMatrix = mat4.create();
    var projectionMatrix = mat4.create();

    var rotationMatrix = mat4.create();
    mat4.identity(rotationMatrix);
    var rvel = { x: 0, y: 0 };
    var updateVelocity = new UpdateVelocity();

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

      Pointer.initInputHandlers(canvas);

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

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

      function initVertices() {
        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;
      }

      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 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;
      }

      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);
      }

      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();
      }
    </script>
  </body>
</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 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 be 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

```
<!DOCTYPE html>
<html>
<head>
  <title>Rotatable Dice</title>
</head>
<body onload="init()">
  <canvas id="mycanvas" width="600" height="600"></canvas>

  <script type="text/javascript" src="rAF.js"></script>
  <script type="text/javascript" src="webgl-utils.js"></script>
  <script type="text/javascript" src="utils.js"></script>
  <script type="text/javascript" src="glMatrix-0.9.5.min.js"></script>
  <script type="text/javascript" src="Pointer.js"></script>
  <script type="text/javascript" src="dicemap.js"></script>

  <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>

  <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;

    var modelViewMatrix = mat4.create();
    var projectionMatrix = mat4.create();

    var rotationMatrix = mat4.create();
    mat4.identity(rotationMatrix);
    var rvel = { x: 0, y: 0 };
    var updateVelocity = new UpdateVelocity();

    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();
        }

        function initVertices() {
          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;
        }

        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 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;
        }

        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);
        }

        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);
        }

        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);
```

```
      }

      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();
      }
    </script>
  </body>
</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 fowl 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

```
<!DOCTYPE html>
<html>
<head>
  <title>Rotatable Dice</title>
</head>
<body onload="init()">
  <canvas id="mycanvas" width="600" height="600"></canvas>

  <script type="text/javascript" src="rAF.js"></script>
  <script type="text/javascript" src="webgl-utils.js"></script>
  <script type="text/javascript" src="utils.js"></script>
  <script type="text/javascript" src="glMatrix-0.9.5.min.js"></script>
  <script type="text/javascript" src="Pointer.js"></script>

  <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>

  <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;

    var modelViewMatrix = mat4.create();
    var projectionMatrix = mat4.create();

    var rotationMatrix = mat4.create();
    mat4.identity(rotationMatrix);
    var rvel = { x: 0, y: 0 };
    var updateVelocity = new UpdateVelocity();

    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();
      }

      function initVertices() {
        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;
      }

      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 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;
      }

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

      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();
      }

      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;
      }

      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);
      }

      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);
      }

      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();
      }
    </script>
  </body>
</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.”