# Fragment Shader

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

precision mediump float;

varying vec4 vColor;

void main(void)

{

gl\_FragColor = vColor;

}

</script>

# Vertex Shader

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

attribute vec3 aVertexPosition;

attribute vec4 aVertexColor;

uniform mat4 uMVMatrix;

uniform mat4 uPMatrix;

varying vec4 vColor;

void main(void)

{

gl\_Position = uPMatrix \* uMVMatrix \* vec4(aVertexPosition,

\_1.0);

vColor = aVertexColor;

}

</script>

# Initialize gl object

var gl;

function initGL(canvas)

{

try

{

gl = canvas.getContext("experimental-webgl");

gl.viewportWidth = canvas.width;

gl.viewportHeight = canvas.height;

}

catch (e)

{}

if (!gl)

{

alert("Could not initialise WebGL, sorry :-(");

}

}

# Get Shader

function getShader(gl, id) {

var shaderScript = document.getElementById(id);

if (!shaderScript) { return null; }

var str = "";

var k = shaderScript.firstChild;

while (k)

{

if (k.nodeType == 3)

{

str += k.textContent;

}

k = k.nextSibling;

}

var shader;

if (shaderScript.type == "x-shader/x-fragment")

{

shader = gl.createShader(gl.FRAGMENT\_SHADER);

}

else if (shaderScript.type == "x-shader/x-vertex")

{

shader = gl.createShader(gl.VERTEX\_SHADER);

}

else

{

return null;

}

gl.shaderSource(shader, str);

gl.compileShader(shader);

if (!gl.getShaderParameter(shader, gl.COMPILE\_STATUS))

{

alert(gl.getShaderInfoLog(shader));

return null;

}

return shader;

}

# Initialize Shaders

var shaderProgram;

function initShaders()

{

var fragmentShader = getShader(gl, "shader-fs");

var vertexShader = getShader(gl, "shader-vs");

shaderProgram = gl.createProgram();

gl.attachShader(shaderProgram, vertexShader);

gl.attachShader(shaderProgram, fragmentShader);

gl.linkProgram(shaderProgram);

if(!gl.getProgramParameter(shaderProgram, gl.LINK\_STATUS))

{

alert("Could not initialise shaders");

}

gl.useProgram(shaderProgram);

shaderProgram.vertexPositionAttribute =

\_ gl.getAttribLocation(shaderProgram, "aVertexPosition");

gl.enableVertexAttribArray(

\_ shaderProgram.vertexPositionAttribute);

shaderProgram.vertexColorAttribute =

\_ gl.getAttribLocation(shaderProgram, "aVertexColor");

gl.enableVertexAttribArray(

\_ shaderProgram.vertexColorAttribute);

shaderProgram.pMatrixUniform =

\_ gl.getUniformLocation(shaderProgram, "uPMatrix");

shaderProgram.mvMatrixUniform =

\_ gl.getUniformLocation(shaderProgram, "uMVMatrix");

}

# Setup Matrix Uniforms

var mvMatrix = mat4.create();

var pMatrix = mat4.create();

function setMatrixUniforms()

{

gl.uniformMatrix4fv(shaderProgram.pMatrixUniform, false,

\_ pMatrix);

gl.uniformMatrix4fv(shaderProgram.mvMatrixUniform, false,

\_ mvMatrix);

}

# Initialize Buffers

var triangleVertexPositionBuffer;

var triangleVertexColorBuffer;

var squareVertexPositionBuffer;

var squareVertexColorBuffer;

function initBuffers() {

triangleVertexPositionBuffer = gl.createBuffer();

gl.bindBuffer(gl.ARRAY\_BUFFER,

\_ triangleVertexPositionBuffer);

var vertices = [

0.0, 1.0, 0.0,

-1.0, -1.0, 0.0,

1.0, -1.0, 0.0

];

gl.bufferData(gl.ARRAY\_BUFFER, new Float32Array(vertices),

\_ gl.STATIC\_DRAW);

triangleVertexPositionBuffer.itemSize = 3;

triangleVertexPositionBuffer.numItems = 3;

triangleVertexColorBuffer = gl.createBuffer();

gl.bindBuffer(gl.ARRAY\_BUFFER, triangleVertexColorBuffer);

var colors = [

1.0, 0.0, 0.0, 1.0,

0.0, 1.0, 0.0, 1.0,

0.0, 0.0, 1.0, 1.0

];

gl.bufferData(gl.ARRAY\_BUFFER, new Float32Array(colors),

\_ gl.STATIC\_DRAW);

triangleVertexColorBuffer.itemSize = 4;

triangleVertexColorBuffer.numItems = 3;

squareVertexPositionBuffer = gl.createBuffer();

gl.bindBuffer(gl.ARRAY\_BUFFER, squareVertexPositionBuffer);

vertices = [

1.0, 1.0, 0.0,

-1.0, 1.0, 0.0,

1.0, -1.0, 0.0,

-1.0, -1.0, 0.0

];

gl.bufferData(gl.ARRAY\_BUFFER, new Float32Array(vertices),

\_ gl.STATIC\_DRAW);

squareVertexPositionBuffer.itemSize = 3;

squareVertexPositionBuffer.numItems = 4;

squareVertexColorBuffer = gl.createBuffer();

gl.bindBuffer(gl.ARRAY\_BUFFER, squareVertexColorBuffer);

colors = [];

for (var i=0; i < 4; i++)

{

colors = colors.concat([0.5, 0.5, 1.0, 1.0]);

}

gl.bufferData(gl.ARRAY\_BUFFER, new Float32Array(colors),

\_ gl.STATIC\_DRAW);

squareVertexColorBuffer.itemSize = 4;

squareVertexColorBuffer.numItems = 4;

}

# Draw Scene

function drawScene()

{

gl.viewport(0, 0, gl.viewportWidth, gl.viewportHeight);

gl.clear(gl.COLOR\_BUFFER\_BIT | gl.DEPTH\_BUFFER\_BIT);

mat4.perspective(45, gl.viewportWidth / gl.viewportHeight,

\_ 0.1, 100.0, pMatrix);

mat4.identity(mvMatrix);

mat4.translate(mvMatrix, [-1.5, 0.0, -7.0]);

gl.bindBuffer(gl.ARRAY\_BUFFER, triangleVertexPositionBuffer);

gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute,

\_ triangleVertexPositionBuffer.itemSize, gl.FLOAT, false, 0, 0);

gl.bindBuffer(gl.ARRAY\_BUFFER, triangleVertexColorBuffer);

gl.vertexAttribPointer(shaderProgram.vertexColorAttribute,

triangleVertexColorBuffer.itemSize, gl.FLOAT, false, 0, 0);

setMatrixUniforms();

gl.drawArrays(gl.TRIANGLES, 0,

\_ triangleVertexPositionBuffer.numItems);

mat4.translate(mvMatrix, [3.0, 0.0, 0.0]);

gl.bindBuffer(gl.ARRAY\_BUFFER, squareVertexPositionBuffer);

gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute,

\_ squareVertexPositionBuffer.itemSize, gl.FLOAT, false, 0, 0);

gl.bindBuffer(gl.ARRAY\_BUFFER, squareVertexColorBuffer);

gl.vertexAttribPointer(shaderProgram.vertexColorAttribute,

\_ squareVertexColorBuffer.itemSize, gl.FLOAT, false, 0, 0);

setMatrixUniforms();

gl.drawArrays(gl.TRIANGLE\_STRIP, 0,

\_ squareVertexPositionBuffer.numItems);

}