Tutorial 4 Draw using Colours

We explore the use of colour data in this tutorial. In last tutorial, we drew a triangle of constant colour, which means the entire triangle has been drawn using a single colour. The shaders we had used look like:

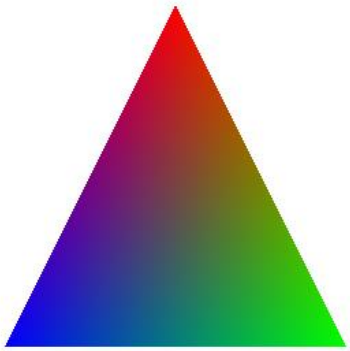
```
<script id="shader-vs" type="x-shader/x-vertex">
attribute vec3 aVertexPosition;
void main() {
   gl_Position = vec4(aVertexPosition, 1.0);
}
</script>


<script id="shader-fs" type="x-shader/x-fragment">
precision mediump float;
void main() {
   gl_FragColor = vec4(1.0, 1.0, 1.0, 1.0);
}
</script>
```

Here, the fragment colour is white and is the same for every fragment:

```
   gl_FragColor = vec4(1.0, 1.0, 1.0, 1.0);
```

When different colours for the vertices are required, e.g., one vertex is red, one is green and one blue, as in this tutorial, we need to treat the colour data in the same way as the vertex coordinate data. That is, we need to set up a buffer for them, and bind the buffer to **`gl.ARRAY_BUFFER`** and link it to the vertex shader.



In this tutorial, the vertex colours are specified in a buffer called **`triangleVertexColorBuffer`** in the **`setupBuffers()`** function. In the vertex shader, an attribute **`aVertexColor`** is defined to receive the colour data from the bound buffer. The received colour data of a vertex is then assigned to a variable of **`varying`** type, **`vColor`**,

```
varying vec4 vColor;
```

The type "varying" is a type of OpenGL ES Shading Language that is used for variables that send values from the vertex shader to the fragment shader. Notice that **`vColor`** is also declared in the fragment shader.

From the given vertex colours of the triangle, WebGL pipeline gets the colours of the other fragments of the triangle by *linearly interpolating* the vertex colours.

Exercise 1: Amend last week's program to accommodate the new features. Pay attention to the statements/functions shown in bold case.

```
<!DOCTYPE HTML>
<html lang="en">
<head>
<title>Tutorial 4 Lines, Triangle and Triangle Strip</title>
<meta charset="utf-8">

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


<script id="shader-vs" type="x-shader/x-vertex">
  attribute vec3 aVertexPosition;
  attribute vec4 aVertexColor;
  varying vec4 vColor;

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



<script id="shader-fs" type="x-shader/x-fragment">
  precision mediump float;
  varying vec4 vColor;

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

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

function createGLContext(canvas) {
      // see last tutorial
}

function loadShaderFromDOM(id) {
      // see last tutorial
}



function setupShaders() {
  vertexShader = loadShaderFromDOM("shader-vs");
  fragmentShader = loadShaderFromDOM("shader-fs");
  shaderProgram = gl.createProgram();
  gl.attachShader(shaderProgram, vertexShader);
  gl.attachShader(shaderProgram, fragmentShader);
  gl.linkProgram(shaderProgram);

  if (!gl.getProgramParameter(shaderProgram, gl.LINK_STATUS)) {
    alert("Failed to setup shaders");
  }

  gl.useProgram(shaderProgram);
```

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

  // get the pointer to "aVertexColor" variable in the vertex shader.
  shaderProgram.vertexColorAttribute = gl.getAttribLocation(shaderProgram,
                              "aVertexColor");
  // Enable the vertexColorAttribute,(i.e., aVertexColor, in the vertex
  // shader so that we draw the triangle use the per-vertex color. The same
  // as did with the vertex coordinates.
  gl.enableVertexAttribArray(shaderProgram.vertexColorAttribute);

}

function setupBuffers() {
    //triangle vertices
  triangleVertexBuffer = gl.createBuffer();
  gl.bindBuffer(gl.ARRAY_BUFFER, triangleVertexBuffer);
  var triangleVertices = [
        0.0,  0.5,  0.0,  // V0
        0.5, -0.5,  0.0,  // V1
       -0.5, -0.5,  0.0,  // V2
                          ];

  gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(triangleVertices),
                              gl.STATIC_DRAW);
  triangleVertexBuffer.itemSize = 3;
  triangleVertexBuffer.numberOfItems = 3;

  // Triangle vertex colours
  triangleVertexColorBuffer = gl.createBuffer();
  gl.bindBuffer(gl.ARRAY_BUFFER, triangleVertexColorBuffer);
  var colors = [
            1.0, 0.0, 0.0, 1.0, //v0
            0.0, 1.0, 0.0, 1.0, //v1
            0.0, 0.0, 1.0, 1.0  //v2
                               ];

  gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(colors), gl.STATIC_DRAW);
  triangleVertexColorBuffer.itemSize = 4;
  triangleVertexColorBuffer.numberOfItems = 3;
}

function draw() {
  gl.viewport(0, 0, gl.viewportWidth, gl.viewportHeight);
  gl.clear(gl.COLOR_BUFFER_BIT);


  // Make vertex buffer "triangleVertexBuffer" the current buffer
  gl.bindBuffer(gl.ARRAY_BUFFER, triangleVertexBuffer);

  // Link the current buffer to the attribute "aVertexPosition" in
  // the vertex shader
  gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute,
                              triangleVertexBuffer.itemSize, gl.FLOAT,
                              false, 0, 0);

  // Make color buffer "triangleVertexColorBuffer" the current buffer
  gl.bindBuffer(gl.ARRAY_BUFFER, triangleVertexColorBuffer);
  // Link the current buffer to the attribute "aVertexColor" in
  // the vertex shader
  gl.vertexAttribPointer(shaderProgram.vertexColorAttribute,
                              triangleVertexColorBuffer.itemSize, gl.FLOAT,
                              false, 0, 0);
  gl.drawArrays(gl.TRIANGLES, 0, triangleVertexBuffer.numberOfItems);
}
```

```
function startup() {
  canvas = document.getElementById("myGLCanvas");
  gl = WebGLDebugUtils.makeDebugContext(createGLContext(canvas));
  setupShaders();
  setupBuffers();
  gl.clearColor(1.0, 1.0, 1.0, 1.0);

  draw();
}
</script>

</head>
<body onload="startup();">
  <canvas id="myGLCanvas" width="500" height="500"></canvas>
</body>
</html>
```

Exercise 2: Draw the coloured triangle using interleaved data.

In the previous program, we have used two separate buffers for vertex data and colour data. This approach is often referred to as ***structure of arrays***. A different approach is to store all vertex data, i.e., coordinates, colour, normal, etc., in one single array in a WebGL buffer object. This means that the different types of vertex data are interleaved in the same array. This is often referred to as ***array of structures***.

From a performance point of view, the second approach, i.e, interleaving the data as an array of structures, is better. This is because a better memory locality can be achieved for the vertex data. It is likely that when a vertex shader needs the position for a certain vertex, it will also need the colour, normal and texture coordinates for the same vertex at almost the same time. By keeping these coordinates close together in the memory, it is more likely that when one is read into the pre-transform vertex cache, the others will also be read in the same block and will be available in the pre-transform vertex cache.

The following implementation of the colour triangle interleaves the vertex coordinates and colour data into a single array. As vertex coordinates and colour data are using different data types (vertex coordinates is 32-bit float whereas colour is 8-bit unsigned integer)

Replace the `setupBuffer()` and `draw()` functions in the previous program with the following:

```
function setupBuffers() {
  triangleVertexBuffer = gl.createBuffer();
  gl.bindBuffer(gl.ARRAY_BUFFER, triangleVertexBuffer);
  // The vertex coordinates and colours are interleaved
  var triangleVertices = [
      //( x     y     z )  (r     g    b    a )
      // ------------------------------------
        0.0,  0.5,  0.0,   255,   0,   0,  255, // V0
        0.5, -0.5,  0.0,     0, 250,   6,  255, // V1
       -0.5, -0.5,  0.0,     0,   0, 255,  255  // V2
                                                    ];

  var nbrOfVertices = 3;//total number of vertices

  // Calculate how many bytes that are needed for one vertex element
  // that consists of (x,y,z) + (r,g,b,a)
  var vertexSizeInBytes = 3 * Float32Array.BYTES_PER_ELEMENT +
                              4 * Uint8Array.BYTES_PER_ELEMENT;
  var vertexSizeInFloats = vertexSizeInBytes /
                              Float32Array.BYTES_PER_ELEMENT;

  // Allocate the buffer
  var buffer = new ArrayBuffer(nbrOfVertices * vertexSizeInBytes);

  // Map the buffer to a Float32Array view to access the position
  var positionView = new Float32Array(buffer);

  // Map the same buffer to a Uint8Array to access the color
  var colorView = new Uint8Array(buffer);

  // Populate the ArrayBuffer from the JavaScript Array
  var positionOffsetInFloats = 0;
  var colorOffsetInBytes = 12;

  var k = 0; // index to JavaScript Array
  for (var i = 0; i < nbrOfVertices; i++) {
    positionView[positionOffsetInFloats] = triangleVertices[k];     // x
```

```
    positionView[1+positionOffsetInFloats] = triangleVertices[k+1]; // y
    positionView[2+positionOffsetInFloats] = triangleVertices[k+2]; // z

    colorView[colorOffsetInBytes] = triangleVertices[k+3];          // R
    colorView[1+colorOffsetInBytes] = triangleVertices[k+4];        // G
    colorView[2+colorOffsetInBytes] = triangleVertices[k+5];        // B
    colorView[3+colorOffsetInBytes] = triangleVertices[k+6];        // A

    positionOffsetInFloats +=vertexSizeInFloats;
    colorOffsetInBytes +=vertexSizeInBytes;

    k +=7;
  }

  gl.bufferData(gl.ARRAY_BUFFER, buffer, gl.STATIC_DRAW);
  triangleVertexBuffer.positionSize = 3;
  triangleVertexBuffer.colorSize = 4;
  triangleVertexBuffer.numberOfItems = 3;
}

function draw() {
  gl.viewport(0, 0, gl.viewportWidth, gl.viewportHeight);
  gl.clear(gl.COLOR_BUFFER_BIT);

  // Bind the buffer containing both position and color
  gl.bindBuffer(gl.ARRAY_BUFFER, triangleVertexBuffer);

  // Describe how the positions are organized in the vertex array
  gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute,
                         triangleVertexBuffer.positionSize,
                         gl.FLOAT, false, 16, 0);

  // Describe how colors are organized in the vertex array
  gl.vertexAttribPointer(shaderProgram.vertexColorAttribute,
                         triangleVertexBuffer.colorSize,
                         gl.UNSIGNED_BYTE, true, 16, 12);

  // Draw the triangle
  gl.drawArrays(gl.TRIANGLES, 0, triangleVertexBuffer.numberOfItems);
}
```