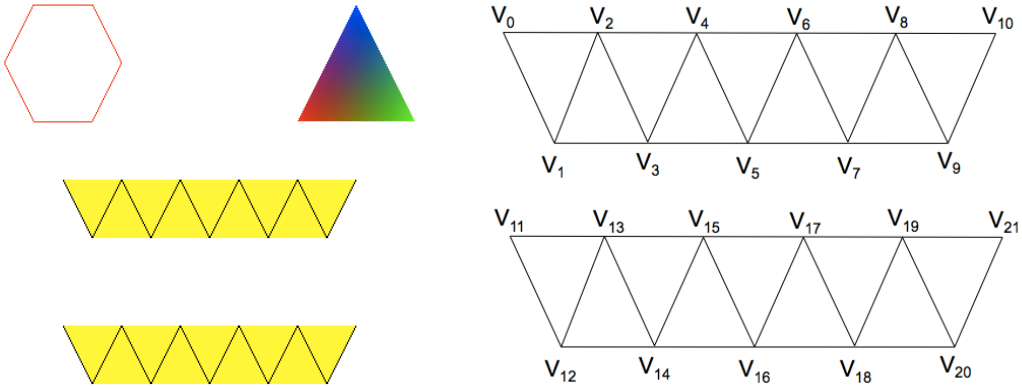
Tutorial 5 Drawing Methods

This tutorial introduces drawing **`gl.LINE_STRIP`**, **`gl.TRIANGLES`**, and **`gl.TRIANGLE_STRIP`** using vertex indices (**`gl.ELEMENT_ARRAY_BUFFER`**). The shapes to draw are shown on the left in the figure the below:



The hexagon is drawn as a line strip. Because it is a line primitive, the shape is not filled with any colour.

The triangle is drawn as a triangle array, as you did in the last tutorial.

The two triangular strips on the lower half of the canvas are drawn as **`gl.TRIANGLE_STRIP`** using `drawElements()`. Two buffers are used: one for the vertex coordinates, `stripVertexBuffer`, and one for the vertex indices, `stripElementBuffer`. Although shown as two separate strips, they are actually drawn as a single strip in one call to `gl.drawElements()` method (to save a function call to save CPU time, which may not be significant in this tutorial). To do so, degenerated triangles are required to link them together (see lecture notes for details). The vertices of the two strips are shown on the right in the figure above.

To help visualising the boundaries of the triangular strips, lines are also drawn from the vertex indices using the specified colour:

```
gl.vertexAttrib4f(shaderProgram.vertexColorAttribute,
                                  0.0, 0.0, 0.0, 1.0);
// Draw line for the upper strip using index 0-10
gl.drawArrays(gl.LINE_STRIP, 0, 11);
// Draw line for the lower strip using index 11-21
gl.drawArrays(gl.LINE_STRIP, 11, 11);
```

In `draw()` function, pay attention to the calling order of the following methods:

`gl.disableVertexAttribArray()`,
`gl.bindBuffer()`, and
`gl.vertexAttribPointer()`.

If a constant is used for an attribute, e.g., vertex colour attribute, we have to disable the attribute in the vertex shader by calling `gl.disableVertexAttribArray()`. If we need to use the attribute values stored in the attribute buffer afterwards, we must re-enable the attribute. The method call to `gl.bindBuffer()` make a buffer current (active). The call to the method `gl.vertexAttribPointer()` links the ***current*** buffer to the attribute in the vertex shader.

If have finished the last week's tutorial, you can amend your program to accommodate the newly added statements or blocks. Otherwise, download the UNFINISHED program (`unfinished program.doc`) from Moodle and start from there.

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

```
<!DOCTYPE HTML>
<html lang="en">
<head>
<title>Tutorial 5 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">
  //vertex shader

</script>



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

</script>

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

var hexagonVertexBuffer;
var stripVertexBuffer;
var stripElementBuffer;

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

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



function setupShaders() {

      // from last tutorial
      ...


  gl.useProgram(shaderProgram);

  shaderProgram.vertexPositionAttribute =
                  gl.getAttribLocation(shaderProgram, "aVertexPosition");
  shaderProgram.vertexColorAttribute = gl.getAttribLocation(shaderProgram,
                              "aVertexColor");
  gl.enableVertexAttribArray(shaderProgram.vertexPositionAttribute);
  // For the triangle we want to use per-vertex color so
  // the vertexColorAttribute, aVertexColor, in the vertex shader
  // is enabled.
```

```
  // You must enable this attribute here or in draw method before the
  //triangle is drawn
  gl.enableVertexAttribArray(shaderProgram.vertexColorAttribute);

}

function setupBuffers() {

  // update the triangle vertex position data used last week
  // because we have changed its size and location
  triangleVertexBuffer = gl.createBuffer();
  gl.bindBuffer(gl.ARRAY_BUFFER, triangleVertexBuffer);
  var triangleVertices = [
        0.3,  0.4,  0.0, //v0
        0.7,  0.4,  0.0, //v1
        0.5,  0.8,  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;


  // Add new items: the followings are newly added items

  //hexagon vertices
  hexagonVertexBuffer = gl.createBuffer();
  gl.bindBuffer(gl.ARRAY_BUFFER, hexagonVertexBuffer);
  var hexagonVertices = [
        -0.3,  0.6,  0.0, //v0
        -0.4,  0.8,  0.0, //v1
        -0.6,  0.8,  0.0, //v2
        -0.7,  0.6,  0.0, //v3
        -0.6,  0.4,  0.0, //v4
        -0.4,  0.4,  0.0, //v5
        -0.3,  0.6,  0.0, //v6
                              ];

  gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(hexagonVertices),
                              gl.STATIC_DRAW);
  hexagonVertexBuffer.itemSize = 3;
  hexagonVertexBuffer.numberOfItems = 7;


  //Triangle strip vertices.
  stripVertexBuffer = gl.createBuffer();
  gl.bindBuffer(gl.ARRAY_BUFFER, stripVertexBuffer);
```

```
  var stripVertices = [
        -0.5,  0.2,  0.0, //v0
        -0.4,  0.0,  0.0, //v1
        -0.3,  0.2,  0.0, //v2
        -0.2,  0.0,  0.0, //v3
        -0.1,  0.2,  0.0, //v4
         0.0,  0.0,  0.0, //v5
         0.1,  0.2,  0.0, //v6
         0.2,  0.0,  0.0, //v7
         0.3,  0.2,  0.0, //v8
         0.4,  0.0,  0.0, //v9
         0.5,  0.2,  0.0, //v10

         // Second strip
        -0.5, -0.3,  0.0, //v11
        -0.4, -0.5,  0.0, //v12
        -0.3, -0.3,  0.0, //v13
        -0.2, -0.5,  0.0, //v14
        -0.1, -0.3,  0.0, //v15
         0.0, -0.5,  0.0, //v16
         0.1, -0.3,  0.0, //v17
         0.2, -0.5,  0.0, //v18
         0.3, -0.3,  0.0, //v19
         0.4, -0.5,  0.0, //v20
         0.5, -0.3,  0.0  //v21
                              ];

  gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(stripVertices),
                               gl.STATIC_DRAW);
  stripVertexBuffer.itemSize = 3;
  stripVertexBuffer.numberOfItems = 22;

  // Strip vertex indices
  stripElementBuffer = gl.createBuffer();
  gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, stripElementBuffer);

  var indices = [
      // put correct indices here. Use degenerated triangles to link the
      // strips together
      ];

  gl.bufferData(gl.ELEMENT_ARRAY_BUFFER, new Uint16Array(indices),
                               gl.STATIC_DRAW);
  stripElementBuffer.numberOfItems = 25;
}

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

  // Draw triangle. No change is made to the last week's code here
  // For the triangle we want to use per-vertex color so
  // the vertexColorAttribute, aVertexColor, in the vertex shader
  // is enabled
  // gl.enableVertexAttribArray(shaderProgram.vertexColorAttribute);

  // 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);



// Draw the newly added items

// Draw the hexagon. We draw the hexagon with constant colour, i.e.,
// a constant colour (1.0, 0.0, 0.0, 1.0) is used for all vertices of the
// hexagon. To do so, we need to disable the vertex colour attribute
// (i.e., stop using the data buffer that is assign to variable (
// aVertexColor)

gl.disableVertexAttribArray(shaderProgram.vertexColorAttribute);

// Instead, a constant colour is specified when aVertexColor is disabled
gl.vertexAttrib4f(shaderProgram.vertexColorAttribute, 1.0, 0.0, 0.0, 1.0);

// Make vertex buffer "hexagonVertexBuffer" the current buffer
gl.bindBuffer(gl.ARRAY_BUFFER, hexagonVertexBuffer);
// Link the current buffer to the attribute "aVertexPosition" in
// the vertex shader
gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute,
                           hexagonVertexBuffer.itemSize, gl.FLOAT,
                           false, 0, 0);
//Draw line strip
gl.drawArrays(gl.LINE_STRIP, 0, hexagonVertexBuffer.numberOfItems);




// draw triangle-strip
// Again, we draw the triangle strips with a constant colour
// (1.0, 1.0, 0.0, 1.0)

gl.bindBuffer(gl.ARRAY_BUFFER, stripVertexBuffer);
gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute,
                           stripVertexBuffer.itemSize, gl.FLOAT,
                           false, 0, 0);
// Specify the constant colour to fill the triangle strip
gl.vertexAttrib4f(shaderProgram.vertexColorAttribute, 1.0, 1.0, 0.0, 1.0);
// The triangle strip will be drawn from its vertex index. We first
// make the index buffer the current buffer by binding it
gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, stripElementBuffer);
gl.drawElements(gl.TRIANGLE_STRIP, stripElementBuffer.numberOfItems,
                           gl.UNSIGNED_SHORT, 0);

// Draw border lines of the triangles so that we see the triangles that
// build up the triangle-strip. But we use a different constant colour for
// the line
gl.vertexAttrib4f(shaderProgram.vertexColorAttribute, 0.0, 0.0, 0.0, 1.0);
```

```
  // Draw line for the upper strip using index 0-10
  gl.drawArrays(gl.LINE_STRIP, 0, 11);
  // Draw line for the lower strip using index 11-21
  gl.drawArrays(gl.LINE_STRIP, 11, 11);
}

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>
```