CHƯƠNG 3: Vẽ và biến đổi tam giác

1. Vẽ hình chữ nhật màu vàng

```
// HelloQuad.js (c) 2012 matsuda
// Vertex shader program
var VSHADER_SOURCE =
  'attribute vec4 a_Position;\n' +
  'void main() {\n' +
  '  gl_Position = a_Position;\n' +
  '}\n';
// Fragment shader program
var FSHADER_SOURCE =
  'void main() {\n' +
  '  gl_FragColor = vec4(1.0, 01.0, 0.0, 1.0);\n' +
  '}\n';
function main() {
  // Retrieve <canvas> element
  var canvas = document.getElementById('webgl');
  // Get the rendering context for WebGL
  var gl = getWebGLContext(canvas);
  if (!gl) {
    console.log('Failed to get the rendering context for WebGL');
    return;
  }
  // Initialize shaders
  if (!initShaders(gl, VSHADER_SOURCE, FSHADER_SOURCE)) {
    console.log('Failed to intialize shaders.');
    return;}
  // Write the positions of vertices to a vertex shader
  var n = initVertexBuffers(gl);
  if (n < 0) {
    console.log('Failed to set the positions of the vertices');
    return;
  }
  // Specify the color for clearing <canvas>
  gl.clearColor(0, 0, 0, 1);
  // Clear <canvas>
  gl.clear(gl.COLOR_BUFFER_BIT);
  // Draw the rectangle
  gl.drawArrays(gl.TRIANGLE_STRIP, 0, n);
}
function initVertexBuffers(gl) {
  var vertices = new Float32Array([
    -0.5, 0.25,   -0.5, -0.25,   0.5, 0.25,    0.5, -0.25
  ]);
  var n = 4; // The number of vertices
  // Create a buffer object
  var vertexBuffer = gl.createBuffer();
  if (!vertexBuffer) {
    console.log('Failed to create the buffer object');
    return -1;
  }
  // Bind the buffer object to target
  gl.bindBuffer(gl.ARRAY_BUFFER, vertexBuffer);
  // Write date into the buffer object
  gl.bufferData(gl.ARRAY_BUFFER, vertices, gl.STATIC_DRAW);
  var a_Position = gl.getAttribLocation(gl.program, 'a_Position');
  if (a_Position < 0) {
    console.log('Failed to get the storage location of a_Position');
    return -1;
  }
  // Assign the buffer object to a_Position variable
  gl.vertexAttribPointer(a_Position, 2, gl.FLOAT, false, 0, 0);
  // Enable the assignment to a_Position variable
  gl.enableVertexAttribArray(a_Position);
  return n;
```

}

2. *Vẽ tam giác*

```
var VSHADER_SOURCE =
    'attribute vec4 a_Position;\n' +
    'void main() {\n' +
    '  gl_Position = a_Position;\n' +
    '}\n';
// Fragment shader program
var FSHADER_SOURCE =
    'void main() {\n' +
    '  gl_FragColor = vec4(1.0, 0.0, 0.0, 1.0);\n' +
    '}\n';
function main() {
    // Retrieve <canvas> element
    var canvas = document.getElementById('webgl');
    // Get the rendering context for WebGL
    var gl = getWebGLContext(canvas);
    if (!gl) {
        console.log('Failed to get the rendering context for WebGL');
        return;
    }
    // Initialize shaders
    if (!initShaders(gl, VSHADER_SOURCE, FSHADER_SOURCE)) {
        console.log('Failed to intialize shaders.');
        return;
    }
    // Write the positions of vertices to a vertex shader
    var n = initVertexBuffers(gl);
    if (n < 0) {
        console.log('Failed to set the positions of the vertices');
        return;
    }
    // Specify the color for clearing <canvas>
    gl.clearColor(0, 0, 0, 1);
    // Clear <canvas>
    gl.clear(gl.COLOR_BUFFER_BIT);
    // Draw the rectangle
    gl.drawArrays(gl.TRIANGLES, 0, n);
}
function initVertexBuffers(gl) {
    var vertices = new Float32Array([
        0, 0.5, -0.5, -0.5, 0.5, -0.5
    ]);
    var n = 3; // The number of vertices
    // Create a buffer object
    var vertexBuffer = gl.createBuffer();
    if (!vertexBuffer) {
        console.log('Failed to create the buffer object');
        return -1;
    }
    // Bind the buffer object to target
    gl.bindBuffer(gl.ARRAY_BUFFER, vertexBuffer);
    // Write date into the buffer object
    gl.bufferData(gl.ARRAY_BUFFER, vertices, gl.STATIC_DRAW);
    var a_Position = gl.getAttribLocation(gl.program, 'a_Position');
    if (a_Position < 0) {
        console.log('Failed to get the storage location of a_Position');
        return -1;
    }
    // Assign the buffer object to a_Position variable
    gl.vertexAttribPointer(a_Position, 2, gl.FLOAT, false, 0, 0);
    // Enable the assignment to a_Position variable
    gl.enableVertexAttribArray(a_Position);
    return n;
}
```

3. Vẽ đường viền tam giác

```
// HelloTriangle_LINE_LOOP.js (c) 2012 matsuda
// Vertex shader program
var VSHADER_SOURCE =
  'attribute vec4 a_Position;\n' +
  'void main() {\n' +
  '  gl_Position = a_Position;\n' +
  '}\n';
// Fragment shader program
var FSHADER_SOURCE =
  'void main() {\n' +
  '  gl_FragColor = vec4(1.0, 0.0, 0.0, 1.0);\n' +
  '}\n';
function main() {
  // Retrieve <canvas> element
  var canvas = document.getElementById('webgl');
  // Get the rendering context for WebGL
  var gl = getWebGLContext(canvas);
  if (!gl) {
    console.log('Failed to get the rendering context for WebGL');
    return;
  }
  // Initialize shaders
  if (!initShaders(gl, VSHADER_SOURCE, FSHADER_SOURCE)) {
    console.log('Failed to intialize shaders.');
    return;
  }
  // Write the positions of vertices to a vertex shader
  var n = initVertexBuffers(gl);
  if (n < 0) {
    console.log('Failed to set the positions of the vertices');
    return;
  }
  // Specify the color for clearing <canvas>
  gl.clearColor(0, 0, 0, 1);
  // Clear <canvas>
  gl.clear(gl.COLOR_BUFFER_BIT);
  // Draw the rectangle (Chọn 1 trong 3 tùy đề bài)
  gl.drawArrays(gl.LINES, 0, n);//Vẽ 1 đường =>  /
  gl.drawArrays(gl.LINE_STRIP, 0, n);//Vẽ 2 đường => /_
  gl.drawArrays(gl.LINE_LOOP, 0, n);//Vẽ 1 tam giác => /_\
}
function initVertexBuffers(gl) {
  var vertices = new Float32Array([
    0, 0.5,   -0.5, -0.5,   0.5, -0.5
  ]);
  var n = 3; // The number of vertices
  // Create a buffer object
  var vertexBuffer = gl.createBuffer();
  if (!vertexBuffer) {
    console.log('Failed to create the buffer object');
    return -1;
  }
  // Bind the buffer object to target
  gl.bindBuffer(gl.ARRAY_BUFFER, vertexBuffer);
  // Write date into the buffer object
  gl.bufferData(gl.ARRAY_BUFFER, vertices, gl.STATIC_DRAW);
  var a_Position = gl.getAttribLocation(gl.program, 'a_Position');
  if (a_Position < 0) {
    console.log('Failed to get the storage location of a_Position');
    return -1;
  }
  // Assign the buffer object to a_Position variable
  gl.vertexAttribPointer(a_Position, 2, gl.FLOAT, false, 0, 0);
  // Enable the assignment to a_Position variable
  gl.enableVertexAttribArray(a_Position);
  return n;
```

```
}
```

4. Vẽ 3 điểm tại 3 đỉnh tam giác

```
// MultiPoint.js (c) 2012 matsuda
// Vertex shader program
var VSHADER_SOURCE =
  'attribute vec4 a_Position;\n' +
  'void main() {\n' +
  '  gl_Position = a_Position;\n' +
  '  gl_PointSize = 10.0;\n' +
  '}\n';
// Fragment shader program
var FSHADER_SOURCE =
  'void main() {\n' +
  '  gl_FragColor = vec4(1.0, 0.0, 0.0, 1.0);\n' +
  '}\n';
function main() {
  // Retrieve <canvas> element
  var canvas = document.getElementById('webgl');
  // Get the rendering context for WebGL
  var gl = getWebGLContext(canvas);
  if (!gl) {
    console.log('Failed to get the rendering context for WebGL');
    return;
  }
  // Initialize shaders
  if (!initShaders(gl, VSHADER_SOURCE, FSHADER_SOURCE)) {
    console.log('Failed to intialize shaders.');
    return;
  }
  // Write the positions of vertices to a vertex shader
  var n = initVertexBuffers(gl);
  if (n < 0) {
    console.log('Failed to set the positions of the vertices');
    return;
  }
  // Specify the color for clearing <canvas>
  gl.clearColor(0, 0, 0, 1);
  // Clear <canvas>
  gl.clear(gl.COLOR_BUFFER_BIT);
  // Draw three points
  gl.drawArrays(gl.POINTS, 0, n);
}
function initVertexBuffers(gl) {
  var vertices = new Float32Array([
    0.0, 0.5,   -0.5, -0.5,   0.5, -0.5
  ]);
  var n = 3; // The number of vertices
  // Create a buffer object
  var vertexBuffer = gl.createBuffer();
  if (!vertexBuffer) {
    console.log('Failed to create the buffer object');
    return -1;
  }
  // Bind the buffer object to target
  gl.bindBuffer(gl.ARRAY_BUFFER, vertexBuffer);
  // Write date into the buffer object
  gl.bufferData(gl.ARRAY_BUFFER, vertices, gl.STATIC_DRAW);
  var a_Position = gl.getAttribLocation(gl.program, 'a_Position');
  if (a_Position < 0) {
    console.log('Failed to get the storage location of a_Position');
    return -1;
  }
  // Assign the buffer object to a_Position variable
  gl.vertexAttribPointer(a_Position, 2, gl.FLOAT, false, 0, 0);
  // Enable the assignment to a_Position variable
  gl.enableVertexAttribArray(a_Position);
  return n;
}
```

5. Xoay tam giác 90 ngược chiều kđh

```
// RotatedTriangle.js (c) 2012 matsuda
// Vertex shader program
var VSHADER_SOURCE =
  // x' = x cosβ - y sinβ
  // y' = x sinβ + y cosβ    Equation 3.3
  // z' = z
  'attribute vec4 a_Position;\n' +
  'uniform float u_CosB, u_SinB;\n' +
  'void main() {\n' +
  '  gl_Position.x = a_Position.x * u_CosB - a_Position.y * u_SinB;\n' +
  '  gl_Position.y = a_Position.x * u_SinB + a_Position.y * u_CosB;\n' +
  '  gl_Position.z = a_Position.z;\n' +
  '  gl_Position.w = 1.0;\n' +
  '}\n';
// Fragment shader program
var FSHADER_SOURCE =
  'void main() {\n' +
  '  gl_FragColor = vec4(1.0, 0.0, 0.0, 1.0);\n' +
  '}\n';
// The rotation angle
var ANGLE = 90.0;
function main() {
  // Retrieve <canvas> element
  var canvas = document.getElementById('webgl');
  // Get the rendering context for WebGL
  var gl = getWebGLContext(canvas);
  if (!gl) {
    console.log('Failed to get the rendering context for WebGL');
    return;
  }
  // Initialize shaders
  if (!initShaders(gl, VSHADER_SOURCE, FSHADER_SOURCE)) {
    console.log('Failed to intialize shaders.');
    return;
  }
  // Write the positions of vertices to a vertex shader
  var n = initVertexBuffers(gl);
  if (n < 0) {
    console.log('Failed to set the positions of the vertices');
    return;
  }
  // // Pass the data required to rotate the shape to the vertex shader
  var radian = Math.PI * ANGLE / 180.0; // Convert to radians
  var cosB = Math.cos(radian);
  var sinB = Math.sin(radian);
  var u_CosB = gl.getUniformLocation(gl.program, 'u_CosB');
  var u_SinB = gl.getUniformLocation(gl.program, 'u_SinB');
  if (!u_CosB || !u_SinB) {
    console.log('Failed to get the storage location of u_CosB or u_SinB');
    return;
  }
  gl.uniform1f(u_CosB, cosB);
  gl.uniform1f(u_SinB, sinB);
  // Specify the color for clearing <canvas>
  gl.clearColor(0, 0, 0, 1);
  // Clear <canvas>
  gl.clear(gl.COLOR_BUFFER_BIT);
  // Draw the rectangle
  gl.drawArrays(gl.TRIANGLES, 0, n);
}
function initVertexBuffers(gl) {
  var vertices = new Float32Array([
    0, 0.5,   -0.5, -0.5,   0.5, -0.5
  ]);
  var n = 3; // The number of vertices
  // Create a buffer object
  var vertexBuffer = gl.createBuffer();
  if (!vertexBuffer) {
    console.log('Failed to create the buffer object');
    return -1;
  }
  // Bind the buffer object to target
  gl.bindBuffer(gl.ARRAY_BUFFER, vertexBuffer);
  // Write date into the buffer object
  gl.bufferData(gl.ARRAY_BUFFER, vertices, gl.STATIC_DRAW);
  var a_Position = gl.getAttribLocation(gl.program, 'a_Position');
  if (a_Position < 0) {
    console.log('Failed to get the storage location of a_Position');
    return -1;
  }
  // Assign the buffer object to a_Position variable
  gl.vertexAttribPointer(a_Position, 2, gl.FLOAT, false, 0, 0);
  // Enable the assignment to a_Position variable
  gl.enableVertexAttribArray(a_Position);
  return n;
}
```

6. Co dãn theo trục Oy

```
// ScaledTriangle_Matrix.js (c) 2012 matsuda
// Vertex shader program
var VSHADER_SOURCE =
  'attribute vec4 a_Position;\n' +
  'uniform mat4 u_xformMatrix;\n' +
  'void main() {\n' +
  '  gl_Position = u_xformMatrix * a_Position;\n' +
  '}\n';
// Fragment shader program
var FSHADER_SOURCE =
  'void main() {\n' +
  '  gl_FragColor = vec4(1.0, 0.0, 0.0, 1.0);\n' +
  '}\n';
// The scaling factor
var Sx = 1.0, Sy = 1.5, Sz = 1.0;
function main() {
  // Retrieve <canvas> element
  var canvas = document.getElementById('webgl');
  // Get the rendering context for WebGL
  var gl = getWebGLContext(canvas);
  if (!gl) {
    console.log('Failed to get the rendering context for WebGL');
    return;
  }
  // Initialize shaders
  if (!initShaders(gl, VSHADER_SOURCE, FSHADER_SOURCE)) {
    console.log('Failed to intialize shaders.');
    return;
  }
  // Write the positions of vertices to a vertex shader
  var n = initVertexBuffers(gl);
  if (n < 0) {
    console.log('Failed to set the positions of the vertices');
    return;
  }
  // Note: WebGL is column major order
  var xformMatrix = new Float32Array([
      Sx,   0.0,  0.0,  0.0,
      0.0,  Sy,   0.0,  0.0,
      0.0,  0.0,  Sz,   0.0,
      0.0,  0.0,  0.0,  1.0
  ]);
  // Pass the rotation matrix to the vertex shader
  var u_xformMatrix = gl.getUniformLocation(gl.program, 'u_xformMatrix');
  if (!u_xformMatrix) {
    console.log('Failed to get the storage location of u_xformMatrix');
    return;
  }
  gl.uniformMatrix4fv(u_xformMatrix, false, xformMatrix);

  // Specify the color for clearing <canvas>
  gl.clearColor(0, 0, 0, 1);
  // Clear <canvas>
  gl.clear(gl.COLOR_BUFFER_BIT);
  // Draw the rectangle
  gl.drawArrays(gl.TRIANGLES, 0, n);
}
function initVertexBuffers(gl) {
  var vertices = new Float32Array([
    0, 0.5,   -0.5, -0.5,   0.5, -0.5
  ]);
  var n = 3; // The number of vertices
  // Create a buffer object
  var vertexBuffer = gl.createBuffer();
  if (!vertexBuffer) {
    console.log('Failed to create the buffer object');
    return false;
  }
  // Bind the buffer object to target
  gl.bindBuffer(gl.ARRAY_BUFFER, vertexBuffer);
  // Write date into the buffer object
  gl.bufferData(gl.ARRAY_BUFFER, vertices, gl.STATIC_DRAW);
  var a_Position = gl.getAttribLocation(gl.program, 'a_Position');
  if (a_Position < 0) {
    console.log('Failed to get the storage location of a_Position');
    return -1;
  }
  // Assign the buffer object to a_Position variable
  gl.vertexAttribPointer(a_Position, 2, gl.FLOAT, false, 0, 0);
  // Enable the assignment to a_Position variable
  gl.enableVertexAttribArray(a_Position);
  return n;
}
```

7. Dịch tam giác lên góc trên bên phải

```
// TranslatedTriangle.js (c) 2012 matsuda
// Vertex shader program
var VSHADER_SOURCE =
  'attribute vec4 a_Position;\n' +
  'uniform vec4 u_Translation;\n' +
  'void main() {\n' +
  '  gl_Position = a_Position + u_Translation;\n' +
  '}\n';
// Fragment shader program
var FSHADER_SOURCE =
  'void main() {\n' +
  '  gl_FragColor = vec4(1.0, 0.0, 0.0, 1.0);\n' +
  '}\n';
// The translation distance for x, y, and z direction
var Tx = 0.5, Ty = 0.5, Tz = 0.0;
function main() {
  // Retrieve <canvas> element
  var canvas = document.getElementById('webgl');
  // Get the rendering context for WebGL
  var gl = getWebGLContext(canvas);
  if (!gl) {
    console.log('Failed to get the rendering context for WebGL');
    return;
  }
  // Initialize shaders
  if (!initShaders(gl, VSHADER_SOURCE, FSHADER_SOURCE)) {
    console.log('Failed to intialize shaders.');
    return;
  }
  // Write the positions of vertices to a vertex shader
  var n = initVertexBuffers(gl);
  if (n < 0) {
    console.log('Failed to set the positions of the vertices');
    return;
  }
  // Pass the translation distance to the vertex shader
  var u_Translation = gl.getUniformLocation(gl.program, 'u_Translation');
  if (!u_Translation) {
    console.log('Failed to get the storage location of u_Translation');
    return;
  }
  gl.unform4f(u_Translation, Tx, Ty, Tz, 0.0);
  // Specify the color for clearing <canvas>
  gl.clearColor(0, 0, 0, 1);
  // Clear <canvas>
  gl.clear(gl.COLOR_BUFFER_BIT);
  // Draw the rectangle
  gl.drawArrays(gl.TRIANGLES, 0, n);
}
function initVertexBuffers(gl) {
  var vertices = new Float32Array([
    0, 0.5,   -0.5, -0.5,   0.5, -0.5
  ]);
  var n = 3; // The number of vertices
  // Create a buffer object
  var vertexBuffer = gl.createBuffer();
  if (!vertexBuffer) {
    console.log('Failed to create the buffer object');
    return -1;
  }
  // Bind the buffer object to target
  gl.bindBuffer(gl.ARRAY_BUFFER, vertexBuffer);
  // Write date into the buffer object
  gl.bufferData(gl.ARRAY_BUFFER, vertices, gl.STATIC_DRAW);
  // Assign the buffer object to the attribute variable
  var a_Position = gl.getAttribLocation(gl.program, 'a_Position');
  if (a_Position < 0) {
    console.log('Failed to get the storage location of a_Position');
    return -1;
  }
  gl.vertexAttribPointer(a_Position, 2, gl.FLOAT, false, 0, 0);
  // Enable the assignment to a_Position variable
  gl.enableVertexAttribArray(a_Position);
  return n;
}
```

CHƯƠNG 4: Biến đổi kết hợp và hoạt cảnh cơ bản

1 Tịnh tiến 1 khoảng 0.5 theo Ox và xoay 1 góc 60^0 ngược chiều kđh

```
var VSHADER_SOURCE =
  'attribute vec4 a_Position;\n' +
  'uniform mat4 u_ModelMatrix;\n' +
  'void main() {\n' +
  '  gl_Position = u_ModelMatrix * a_Position;\n' +
  '}\n';
var FSHADER_SOURCE =
  'void main() {\n' +
  '  gl_FragColor = vec4(1.0, 0.0, 0.0, 1.0);\n' +
  '}\n';

function main() {
  // Retrieve <canvas> element
  var canvas = document.getElementById('webgl');
  // Get the rendering context for WebGL
  var gl = getWebGLContext(canvas);
  if (!gl) {
    console.log('Failed to get the rendering context for WebGL');
    return;
  }
  // Initialize shaders
  if (!initShaders(gl, VSHADER_SOURCE, FSHADER_SOURCE)) {
    console.log('Failed to intialize shaders.');
    return;
  }
  // Write the positions of vertices to a vertex shader
  var n = initVertexBuffers(gl);
  if (n < 0) {
    console.log('Failed to set the positions of the vertices');
    return;
  }
  // Create Matrix4 object for model transformation
  var modelMatrix = new Matrix4();
  // Calculate a model matrix
  var ANGLE = 60.0; // The rotation angle
  var Tx = 0.5;     // Translation distance
  modelMatrix.setRotate(ANGLE, 0, 0, 1);  // Set rotation matrix
  modelMatrix.translate(Tx, 0, 0);        // Multiply modelMatrix by the calculated translation matrix
  // Pass the model matrix to the vertex shader
  var u_ModelMatrix = gl.getUniformLocation(gl.program, 'u_ModelMatrix');
  if (!u_ModelMatrix) {
    console.log('Failed to get the storage location of u_xformMatrix');
    return;
  }
  gl.uniformMatrix4fv(u_ModelMatrix, false, modelMatrix.elements);
  // Specify the color for clearing <canvas>
  gl.clearColor(0, 0, 0, 1);
  // Clear <canvas>
  gl.clear(gl.COLOR_BUFFER_BIT);
  // Draw the rectangle
  gl.drawArrays(gl.TRIANGLES, 0, n);
}
function initVertexBuffers(gl) {
  var vertices = new Float32Array([
    0, 0.3,   -0.3, -0.3,   0.3, -0.3
  ]);
  var n = 3; // The number of vertices
  // Create a buffer object
  var vertexBuffer = gl.createBuffer();
  if (!vertexBuffer) {
    console.log('Failed to create the buffer object');
    return false;
  }
  // Bind the buffer object to target
  gl.bindBuffer(gl.ARRAY_BUFFER, vertexBuffer);
  // Write date into the buffer object
  gl.bufferData(gl.ARRAY_BUFFER, vertices, gl.STATIC_DRAW);
  var a_Position = gl.getAttribLocation(gl.program, 'a_Position');
  if (a_Position < 0) {
    console.log('Failed to get the storage location of a_Position');
    return -1;
  }
  // Assign the buffer object to a_Position variable
  gl.vertexAttribPointer(a_Position, 2, gl.FLOAT, false, 0, 0);
  // Enable the assignment to a_Position variable
  gl.enableVertexAttribArray(a_Position);
  return n;
}
```

2 Xoay tam giác 1 góc 90^0 ngược chiều kđh sử dụng Matrix4

```
// RotatedTriangle_Matrix4.js (c) 2012 matsuda
// Vertex shader program
var VSHADER_SOURCE =
  'attribute vec4 a_Position;\n' +
  'uniform mat4 u_xformMatrix;\n' +
  'void main() {\n' +
  '  gl_Position = u_xformMatrix * a_Position;\n' +
  '}\n';
// Fragment shader program
var FSHADER_SOURCE =
  'void main() {\n' +
  '  gl_FragColor = vec4(1.0, 0.0, 0.0, 1.0);\n' +
  '}\n';
function main() {
  // Retrieve <canvas> element
  var canvas = document.getElementById('webgl');
```

```
// Get the rendering context for WebGL
var gl = getWebGLContext(canvas);
if (!gl) {
  console.log('Failed to get the rendering context for WebGL');
  return;
}
// Initialize shaders
if (!initShaders(gl, VSHADER_SOURCE, FSHADER_SOURCE)) {
  console.log('Failed to intialize shaders.');
  return;
}
// Write the positions of vertices to a vertex shader
var n = initVertexBuffers(gl);
if (n < 0) {
  console.log('Failed to set the positions of the vertices');
  return;
}
// Create Matrix4 object for the rotation matrix
var xformMatrix = new Matrix4();
// Set the rotation matrix
var ANGLE = 90.0; // The rotation angle
xformMatrix.setRotate(ANGLE, 0, 0, 1);
// Pass the rotation matrix to the vertex shader
var u_xformMatrix = gl.getUniformLocation(gl.program, 'u_xformMatrix');
if (!u_xformMatrix) {
  console.log('Failed to get the storage location of u_xformMatrix');
  return;
}
gl.uniformMatrix4fv(u_xformMatrix, false, xformMatrix.elements);
// Specify the color for clearing <canvas>
gl.clearColor(0, 0, 0, 1);
// Clear <canvas>
gl.clear(gl.COLOR_BUFFER_BIT);
```

```
  // Draw the rectangle
  gl.drawArrays(gl.TRIANGLES, 0, n);
}
function initVertexBuffers(gl) {
  var vertices = new Float32Array([
    0, 0.5,   -0.5, -0.5,   0.5, -0.5
  ]);
  var n = 3; // The number of vertices
  // Create a buffer object
  var vertexBuffer = gl.createBuffer();
  if (!vertexBuffer) {
    console.log('Failed to create the buffer object');
    return false;
  }
  // Bind the buffer object to target
  gl.bindBuffer(gl.ARRAY_BUFFER, vertexBuffer);
  // Write date into the buffer object
  gl.bufferData(gl.ARRAY_BUFFER, vertices, gl.STATIC_DRAW);
  var a_Position = gl.getAttribLocation(gl.program, 'a_Position');
  if (a_Position < 0) {
    console.log('Failed to get the storage location of a_Position');
    return -1;
  }
  // Assign the buffer object to a_Position variable
  gl.vertexAttribPointer(a_Position, 2, gl.FLOAT, false, 0, 0);
  // Enable the assignment to a_Position variable
  gl.enableVertexAttribArray(a_Position);
  return n;
}
```

3 Quay tam giác quanh tâm

```
// RotatingTranslatedTriangle.js (c) 2012 matsuda
// Vertex shader program
```

```
var VSHADER_SOURCE =
  'attribute vec4 a_Position;\n' +
  'uniform mat4 u_ModelMatrix;\n' +
  'void main() {\n' +
  '  gl_Position = u_ModelMatrix * a_Position;\n' +
  '}\n';
// Fragment shader program
var FSHADER_SOURCE =
  'void main() {\n' +
  '  gl_FragColor = vec4(1.0, 0.0, 0.0, 1.0);\n' +
  '}\n';
// Rotation angle (degrees/second)
var ANGLE_STEP = 45.0;
function main() {
  // Retrieve <canvas> element
  var canvas = document.getElementById('webgl');
  // Get the rendering context for WebGL
  var gl = getWebGLContext(canvas);
  if (!gl) {
    console.log('Failed to get the rendering context for WebGL');
    return;
  }
  // Initialize shaders
  if (!initShaders(gl, VSHADER_SOURCE, FSHADER_SOURCE)) {
    console.log('Failed to intialize shaders.');
    return;
  }
  // Write the positions of vertices to a vertex shader
  var n = initVertexBuffers(gl);
  if (n < 0) {
    console.log('Failed to set the positions of the vertices');
    return;
  }
  // Specify the color for clearing <canvas>
  gl.clearColor(0, 0, 0, 1);
  // Get storage location of u_ModelMatrix
  var u_ModelMatrix = gl.getUniformLocation(gl.program, 'u_ModelMatrix');
  if (!u_ModelMatrix) {
    console.log('Failed to get the storage location of u_ModelMatrix');
    return;
  }
  // Current rotation angle
  var currentAngle = 0.0;
  // Model matrix
  var modelMatrix = new Matrix4();
  // Start drawing
  var tick = function() {
    currentAngle = animate(currentAngle);  // Update the rotation angle
    draw(gl, n, currentAngle, modelMatrix, u_ModelMatrix);   // Draw the triangle
    requestAnimationFrame(tick, canvas); // Request that the browser ?calls tick
  };
  tick();
}
function initVertexBuffers(gl) {
  var vertices = new Float32Array ([
    0, 0.5,   -0.5, -0.5,   0.5, -0.5
  ]);
  var n = 3;   // The number of vertices
  // Create a buffer object
  var vertexBuffer = gl.createBuffer();
  if (!vertexBuffer) {
    console.log('Failed to create the buffer object');
    return -1;
  }
  // Bind the buffer object to target
  gl.bindBuffer(gl.ARRAY_BUFFER, vertexBuffer);
  // Write date into the buffer object
  gl.bufferData(gl.ARRAY_BUFFER, vertices, gl.STATIC_DRAW);
  // Assign the buffer object to a_Position variable
  var a_Position = gl.getAttribLocation(gl.program, 'a_Position');
  if(a_Position < 0) {
    console.log('Failed to get the storage location of a_Position');
    return -1;
  }
  gl.vertexAttribPointer(a_Position, 2, gl.FLOAT, false, 0, 0);
  // Enable the assignment to a_Position variable
  gl.enableVertexAttribArray(a_Position);
  return n;
}
function draw(gl, n, currentAngle, modelMatrix, u_ModelMatrix) {
  // Set the rotation matrix
  modelMatrix.setRotate(currentAngle, 0, 0, 1); // Rotation angle, rotation axis (0, 0, 1)
 // modelMatrix.translate(X, 0, 0); //Xoay cách tâm 1 khoảng X
  // Pass the rotation matrix to the vertex shader
  gl.uniformMatrix4fv(u_ModelMatrix, false, modelMatrix.elements);
  // Clear <canvas>
  gl.clear(gl.COLOR_BUFFER_BIT);
  // Draw the rectangle
  gl.drawArrays(gl.TRIANGLES, 0, n);
}
// Last time that this function was called
var g_last = Date.now();
function animate(angle) {
  // Calculate the elapsed time
  var now = Date.now();
  var elapsed = now - g_last;
  g_last = now;
  // Update the current rotation angle (adjusted by the elapsed time)
  var newAngle = angle + (ANGLE_STEP * elapsed) / 1000.0;
  return newAngle %= 360;
}
```

4 Xoay 1 góc 60^0 ngược chiều kđh và tịnh tiến 1 đoạn 0.5 theo Ox

```
// TranslatedRotatedTriangle.js (c) 2012 matsuda
// Vertex shader program
var VSHADER_SOURCE =
  'attribute vec4 a_Position;\n' +
  'uniform mat4 u_ModelMatrix;\n' +
  'void main() {\n' +
  '  gl_Position = u_ModelMatrix * a_Position;\n' +
  '}\n';
// Fragment shader program
var FSHADER_SOURCE =
  'void main() {\n' +
  '  gl_FragColor = vec4(1.0, 0.0, 0.0, 1.0);\n' +
  '}\n';
function main() {
  // Retrieve <canvas> element
  var canvas = document.getElementById('webgl');
  // Get the rendering context for WebGL
  var gl = getWebGLContext(canvas);
  if (!gl) {
    console.log('Failed to get the rendering context for WebGL');
    return;
  }

  // Initialize shaders
  if (!initShaders(gl, VSHADER_SOURCE, FSHADER_SOURCE)) {
    console.log('Failed to intialize shaders.');
    return;
  }
  // Write the positions of vertices to a vertex shader
  var n = initVertexBuffers(gl);
  if (n < 0) {
    console.log('Failed to set the positions of the vertices');
    return;
  }
  // Create Matrix4 object for model transformation
  var modelMatrix = new Matrix4();
  // Calculate a model matrix
  var ANGLE = 60.0; // The rotation angle
  var Tx = 0.5;     // Translation distance
  modelMatrix.setTranslate(Tx, 0, 0);  // Set translation matrix
  modelMatrix.rotate(ANGLE, 0, 0, 1);  // Multiply modelMatrix by the calculated rotation matrix
  // Pass the model matrix to the vertex shader
  var u_ModelMatrix = gl.getUniformLocation(gl.program, 'u_ModelMatrix');
  if (!u_ModelMatrix) {
    console.log('Failed to get the storage location of u_xformMatrix');
    return;
  }
  gl.uniformMatrix4fv(u_ModelMatrix, false, modelMatrix.elements);

  // Specify the color for clearing <canvas>
  gl.clearColor(0, 0, 0, 1);
  // Clear <canvas>
  gl.clear(gl.COLOR_BUFFER_BIT);
  // Draw the rectangle
  gl.drawArrays(gl.TRIANGLES, 0, n);
}
function initVertexBuffers(gl) {
  var vertices = new Float32Array([
    0, 0.3,   -0.3, -0.3,   0.3, -0.3
  ]);
  var n = 3; // The number of vertices
  // Create a buffer object
  var vertexBuffer = gl.createBuffer();
  if (!vertexBuffer) {
    console.log('Failed to create the buffer object');
    return false;
  }
  // Bind the buffer object to target
  gl.bindBuffer(gl.ARRAY_BUFFER, vertexBuffer);
  // Write date into the buffer object
  gl.bufferData(gl.ARRAY_BUFFER, vertices, gl.STATIC_DRAW);
  var a_Position = gl.getAttribLocation(gl.program, 'a_Position');
  if (a_Position < 0) {
    console.log('Failed to get the storage location of a_Position');
    return -1;
  }
  // Assign the buffer object to a_Position variable
  gl.vertexAttribPointer(a_Position, 2, gl.FLOAT, false, 0, 0);
  // Enable the assignment to a_Position variable
  gl.enableVertexAttribArray(a_Position);
  return n;
}
```

CHƯƠNG 5

1: ColoredTriangle.js (c) 2012 matsuda

```
//Vẽ 3 màu tại 3 đỉnh của tam giác
// Vertex shader program
var VSHADER_SOURCE =
  'attribute vec4 a_Position;\n' +
  'attribute vec4 a_Color;\n' +
  'varying vec4 v_Color;\n' +
  'void main() {\n' +
  '  gl_Position = a_Position;\n' +
  '  v_Color = a_Color;\n' +
  '}\n';

// Fragment shader program
var FSHADER_SOURCE =
  '#ifdef GL_ES\n' +
  'precision mediump float;\n' +
  '#endif GL_ES\n' +
  'varying vec4 v_Color;\n' +
  'void main() {\n' +
  '  gl_FragColor = v_Color;\n' +
  '}\n';

function main() {
  // Retrieve <canvas> element
  var canvas = document.getElementById('webgl');

  // Get the rendering context for WebGL
  var gl = getWebGLContext(canvas);
  if (!gl) {
    console.log('Failed to get the rendering context for WebGL');
    return;
  }
  // Initialize shaders
  if (!initShaders(gl, VSHADER_SOURCE, FSHADER_SOURCE)) {
    console.log('Failed to intialize shaders.');
    return;
  }

  //
  var n = initVertexBuffers(gl);
  if (n < 0) {
    console.log('Failed to set the vertex information');
    return;
  }

  // Specify the color for clearing <canvas>
  gl.clearColor(0.0, 0.0, 0.0, 1.0);

  // Clear <canvas>
  gl.clear(gl.COLOR_BUFFER_BIT);

  // Draw the rectangle
  gl.drawArrays(gl.TRIANGLES, 0, n);
}

function initVertexBuffers(gl) {
  var verticesColors = new Float32Array([
    // Vertex coordinates and color
     0.0,  0.5,  1.0,  0.0,  0.0,
    -0.5, -0.5,  0.0,  1.0,  0.0,
     0.5, -0.5,  0.0,  0.0,  1.0,
  ]);
  var n = 3;
```

```
// Create a buffer object
var vertexColorBuffer = gl.createBuffer();
if (!vertexColorBuffer) {
  console.log('Failed to create the buffer object');
  return false;
}

// Bind the buffer object to target
gl.bindBuffer(gl.ARRAY_BUFFER, vertexColorBuffer);
gl.bufferData(gl.ARRAY_BUFFER, verticesColors, gl.STATIC_DRAW);

var FSIZE = verticesColors.BYTES_PER_ELEMENT;
//Get the storage location of a_Position, assign and enable buffer
var a_Position = gl.getAttribLocation(gl.program, 'a_Position');
if (a_Position < 0) {
  console.log('Failed to get the storage location of a_Position');
  return -1;
}
gl.vertexAttribPointer(a_Position, 2, gl.FLOAT, false, FSIZE * 5, 0);
gl.enableVertexAttribArray(a_Position);  // Enable the assignment of the buffer object

// Get the storage location of a_Position, assign buffer and enable
var a_Color = gl.getAttribLocation(gl.program, 'a_Color');
if(a_Color < 0) {
  console.log('Failed to get the storage location of a_Color');
  return -1;
}
gl.vertexAttribPointer(a_Color, 3, gl.FLOAT, false, FSIZE * 5, FSIZE * 2);
gl.enableVertexAttribArray(a_Color);  // Enable the assignment of the buffer object

// Unbind the buffer object
gl.bindBuffer(gl.ARRAY_BUFFER, null);
```

```
  return n;
}
```

CHƯƠNG 7

1: LookAtTriangles.js (c) 2012 matsuda

```
// Vertex shader program
var VSHADER_SOURCE =
  'attribute vec4 a_Position;\n' +
  'attribute vec4 a_Color;\n' +
  'uniform mat4 u_ViewMatrix;\n' +
  'varying vec4 v_Color;\n' +
  'void main() {\n' +
  '  gl_Position = u_ViewMatrix * a_Position;\n' +
  '  v_Color = a_Color;\n' +
  '}\n';

// Fragment shader program
var FSHADER_SOURCE =
  '#ifdef GL_ES\n' +
  'precision mediump float;\n' +
  '#endif\n' +
  'varying vec4 v_Color;\n' +
  'void main() {\n' +
  '  gl_FragColor = v_Color;\n' +
  '}\n';

function main() {
  // Retrieve <canvas> element
  var canvas = document.getElementById('webgl');

  // Get the rendering context for WebGL
  var gl = getWebGLContext(canvas);
  if (!gl) {
```

```
  console.log('Failed to get the rendering context for WebGL');
  return;
}

// Initialize shaders
if (!initShaders(gl, VSHADER_SOURCE, FSHADER_SOURCE)) {
  console.log('Failed to intialize shaders.');
  return;
}

// Set the vertex coordinates and color (the blue triangle is in the front)
var n = initVertexBuffers(gl);
if (n < 0) {
  console.log('Failed to set the vertex information');
  return;
}

// Specify the color for clearing <canvas>
gl.clearColor(0, 0, 0, 1);

// Get the storage location of u_ViewMatrix
var u_ViewMatrix = gl.getUniformLocation(gl.program, 'u_ViewMatrix');
if (!u_ViewMatrix) {
  console.log('Failed to get the storage locations of u_ViewMatrix');
  return;
}

// Set the matrix to be used for to set the camera view
var viewMatrix = new Matrix4();
viewMatrix.setLookAt(0.20, 0.25, 0.25, 0, 0, 0, 0, 1, 0);

// Set the view matrix
gl.uniformMatrix4fv(u_ViewMatrix, false, viewMatrix.elements);
```

```
  // Clear <canvas>
  gl.clear(gl.COLOR_BUFFER_BIT);

  // Draw the rectangle
  gl.drawArrays(gl.TRIANGLES, 0, n);
}

function initVertexBuffers(gl) {
  var verticesColors = new Float32Array([
    // Vertex coordinates and color(RGBA)
     0.0,  0.5,  -0.4,  0.4,  1.0,  0.4, // The back green one
    -0.5, -0.5,  -0.4,  0.4,  1.0,  0.4,
     0.5, -0.5,  -0.4,  1.0,  0.4,  0.4,

     0.5,  0.4,  -0.2,  1.0,  0.4,  0.4, // The middle yellow one
    -0.5,  0.4,  -0.2,  1.0,  1.0,  0.4,
     0.0, -0.6,  -0.2,  1.0,  1.0,  0.4,

     0.0,  0.5,   0.0,  0.4,  0.4,  1.0,  // The front blue one
    -0.5, -0.5,   0.0,  0.4,  0.4,  1.0,
     0.5, -0.5,   0.0,  1.0,  0.4,  0.4,
  ]);
  var n = 9;

  // Create a buffer object
  var vertexColorbuffer = gl.createBuffer();
  if (!vertexColorbuffer) {
    console.log('Failed to create the buffer object');
    return -1;
  }

  // Write the vertex coordinates and color to the buffer object
  gl.bindBuffer(gl.ARRAY_BUFFER, vertexColorbuffer);
  gl.bufferData(gl.ARRAY_BUFFER, verticesColors, gl.STATIC_DRAW);

  var FSIZE = verticesColors.BYTES_PER_ELEMENT;
  // Assign the buffer object to a_Position and enable the assignment
  var a_Position = gl.getAttribLocation(gl.program, 'a_Position');
  if(a_Position < 0) {
    console.log('Failed to get the storage location of a_Position');
    return -1;
  }
  gl.vertexAttribPointer(a_Position, 3, gl.FLOAT, false, FSIZE * 6, 0);
  gl.enableVertexAttribArray(a_Position);

  // Assign the buffer object to a_Color and enable the assignment
  var a_Color = gl.getAttribLocation(gl.program, 'a_Color');
  if(a_Color < 0) {
    console.log('Failed to get the storage location of a_Color');
    return -1;
  }
  gl.vertexAttribPointer(a_Color, 3, gl.FLOAT, false, FSIZE * 6, FSIZE * 3);
  gl.enableVertexAttribArray(a_Color);

  // Unbind the buffer object
  gl.bindBuffer(gl.ARRAY_BUFFER, null);

  return n;
}
```

2: HelloCube.js (c) 2012 matsuda

```
// Vertex shader program
var VSHADER_SOURCE =
  'attribute vec4 a_Position;\n' +
  'attribute vec4 a_Color;\n' +
  'uniform mat4 u_MvpMatrix;\n' +
  'varying vec4 v_Color;\n' +
  'void main() {\n' +
  '  gl_Position = u_MvpMatrix * a_Position;\n' +
  '  v_Color = a_Color;\n' +
  '}\n';

// Fragment shader program
var FSHADER_SOURCE =
  '#ifdef GL_ES\n' +
  'precision mediump float;\n' +
  '#endif\n' +
  'varying vec4 v_Color;\n' +
  'void main() {\n' +
  '  gl_FragColor = v_Color;\n' +
  '}\n';

function main() {
  // Retrieve <canvas> element
  var canvas = document.getElementById('webgl');

  // Get the rendering context for WebGL
  var gl = getWebGLContext(canvas);
  if (!gl) {
    console.log('Failed to get the rendering context for WebGL');
    return;
  }

  // Initialize shaders
  if (!initShaders(gl, VSHADER_SOURCE, FSHADER_SOURCE)) {
    console.log('Failed to intialize shaders.');
    return;
  }
  // Set the vertex coordinates and color
  var n = initVertexBuffers(gl);
  if (n < 0) {
    console.log('Failed to set the vertex information');
    return;
  }

  // Set clear color and enable hidden surface removal
  gl.clearColor(0.0, 0.0, 0.0, 1.0);
  gl.enable(gl.DEPTH_TEST);

  // Get the storage location of u_MvpMatrix
  var u_MvpMatrix = gl.getUniformLocation(gl.program, 'u_MvpMatrix');
  if (!u_MvpMatrix) {
    console.log('Failed to get the storage location of u_MvpMatrix');
    return;
  }

  // Set the eye point and the viewing volume
  var mvpMatrix = new Matrix4();
  mvpMatrix.setPerspective(30, 1, 1, 100);
  mvpMatrix.lookAt(3, 3, 7, 0, 0, 0, 0, 1, 0);

  // Pass the model view projection matrix to u_MvpMatrix
  gl.uniformMatrix4fv(u_MvpMatrix, false, mvpMatrix.elements);

  // Clear color and depth buffer
  gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);

  // Draw the cube
  gl.drawElements(gl.TRIANGLES, n, gl.UNSIGNED_BYTE, 0);
}
```

```
function initVertexBuffers(gl) {
  // Create a cube
  //    v6----- v5
  //   /|      /|
  //  v1------v0|
  //  | |     | |
  //  | |v7---|-|v4
  //  |/      |/
  //  v2------v3
  var verticesColors = new Float32Array([
    // Vertex coordinates and color
     1.0,  1.0,  1.0,     1.0,  1.0,  1.0,  // v0 White
    -1.0,  1.0,  1.0,     1.0,  0.0,  1.0,  // v1 Magenta
    -1.0, -1.0,  1.0,     1.0,  0.0,  0.0,  // v2 Red
     1.0, -1.0,  1.0,     1.0,  1.0,  0.0,  // v3 Yellow
     1.0, -1.0, -1.0,     0.0,  1.0,  0.0,  // v4 Green
     1.0,  1.0, -1.0,     0.0,  1.0,  1.0,  // v5 Cyan
    -1.0,  1.0, -1.0,     0.0,  0.0,  1.0,  // v6 Blue
    -1.0, -1.0, -1.0,     0.0,  0.0,  0.0   // v7 Black
  ]);

  // Indices of the vertices
  var indices = new Uint8Array([
    0, 1, 2,   0, 2, 3,    // front
    0, 3, 4,   0, 4, 5,    // right
    0, 5, 6,   0, 6, 1,    // up
    1, 6, 7,   1, 7, 2,    // left
    7, 4, 3,   7, 3, 2,    // down
    4, 7, 6,   4, 6, 5     // back
 ]);

  // Create a buffer object
  var vertexColorBuffer = gl.createBuffer();
  var indexBuffer = gl.createBuffer();
  if (!vertexColorBuffer || !indexBuffer) {
    return -1;
  }

  // Write the vertex coordinates and color to the buffer object
  gl.bindBuffer(gl.ARRAY_BUFFER, vertexColorBuffer);
  gl.bufferData(gl.ARRAY_BUFFER, verticesColors, gl.STATIC_DRAW);

  var FSIZE = verticesColors.BYTES_PER_ELEMENT;
  // Assign the buffer object to a_Position and enable the assignment
  var a_Position = gl.getAttribLocation(gl.program, 'a_Position');
  if(a_Position < 0) {
    console.log('Failed to get the storage location of a_Position');
    return -1;
  }
  gl.vertexAttribPointer(a_Position, 3, gl.FLOAT, false, FSIZE * 6, 0);
  gl.enableVertexAttribArray(a_Position);
  // Assign the buffer object to a_Color and enable the assignment
  var a_Color = gl.getAttribLocation(gl.program, 'a_Color');
  if(a_Color < 0) {
    console.log('Failed to get the storage location of a_Color');
    return -1;
  }
  gl.vertexAttribPointer(a_Color, 3, gl.FLOAT, false, FSIZE * 6, FSIZE * 3);
  gl.enableVertexAttribArray(a_Color);

  // Write the indices to the buffer object
  gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, indexBuffer);
  gl.bufferData(gl.ELEMENT_ARRAY_BUFFER, indices, gl.STATIC_DRAW);
  return indices.length;
}
```