|  |  |
| --- | --- |
|  | **LẬP TRÌNH WEBGL**  **VẼ CÁC ĐỐI TƯỢNG CƠ BẢN** |

**Họ và tên Sinh viên: Nguyễn Công Cường, Văn Phú Long**

**Mã Sinh viên: 102200013, 1022000**

**Nhóm học phần:**

* **Draw a Triangle**
* ***File WebGLDrawTriangle01.htm***

<!doctype html>

<html>

<body>

<canvas width = "570" height = "570" id = "my\_Canvas"></canvas>

<script>

/\*============== Creating a canvas ====================\*/

var canvas = document.getElementById('my\_Canvas');

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

/\*======== Defining and storing the geometry ===========\*/

var vertices = [

-0.5,0.5,0.0,

-0.5,-0.5,0.0,

0.5,-0.5,0.0,

];

indices = [0,1,2];

// Create an empty buffer object to store vertex buffer

var vertex\_buffer = gl.createBuffer();

// Bind appropriate array buffer to it

gl.bindBuffer(gl.ARRAY\_BUFFER, vertex\_buffer);

// Pass the vertex data to the buffer

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

// Unbind the buffer

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

// Create an empty buffer object to store Index buffer

var Index\_Buffer = gl.createBuffer();

// Bind appropriate array buffer to it

gl.bindBuffer(gl.ELEMENT\_ARRAY\_BUFFER, Index\_Buffer);

// Pass the vertex data to the buffer

gl.bufferData(gl.ELEMENT\_ARRAY\_BUFFER, new Uint16Array(indices), gl.STATIC\_DRAW);

// Unbind the buffer

gl.bindBuffer(gl.ELEMENT\_ARRAY\_BUFFER, null);

/\*================ Shaders ====================\*/

// Vertex shader source code

var vertCode =

'attribute vec3 coordinates;' +

'void main(void) {' +

' gl\_Position = vec4(coordinates, 1.0);' +

'}';

// Create a vertex shader object

var vertShader = gl.createShader(gl.VERTEX\_SHADER);

// Attach vertex shader source code

gl.shaderSource(vertShader, vertCode);

// Compile the vertex shader

gl.compileShader(vertShader);

//fragment shader source code

var fragCode =

'void main(void) {' +

' gl\_FragColor = vec4(0.0, 0.0, 0.0, 0.1);' +

'}';

// Create fragment shader object

var fragShader = gl.createShader(gl.FRAGMENT\_SHADER);

// Attach fragment shader source code

gl.shaderSource(fragShader, fragCode);

// Compile the fragmentt shader

gl.compileShader(fragShader);

// Create a shader program object to store

// the combined shader program

var shaderProgram = gl.createProgram();

// Attach a vertex shader

gl.attachShader(shaderProgram, vertShader);

// Attach a fragment shader

gl.attachShader(shaderProgram, fragShader);

// Link both the programs

gl.linkProgram(shaderProgram);

// Use the combined shader program object

gl.useProgram(shaderProgram);

/\*======= Associating shaders to buffer objects =======\*/

// Bind vertex buffer object

gl.bindBuffer(gl.ARRAY\_BUFFER, vertex\_buffer);

// Bind index buffer object

gl.bindBuffer(gl.ELEMENT\_ARRAY\_BUFFER, Index\_Buffer);

// Get the attribute location

var coord = gl.getAttribLocation(shaderProgram, "coordinates");

// Point an attribute to the currently bound VBO

gl.vertexAttribPointer(coord, 3, gl.FLOAT, false, 0, 0);

// Enable the attribute

gl.enableVertexAttribArray(coord);

/\*=========Drawing the triangle===========\*/

// Clear the canvas

gl.clearColor(0.5, 0.5, 0.5, 0.9);

// Enable the depth test

gl.enable(gl.DEPTH\_TEST);

// Clear the color buffer bit

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

// Set the view port

gl.viewport(0,0,canvas.width,canvas.height);

// Draw the triangle

gl.drawElements(gl.TRIANGLES, indices.length, gl.UNSIGNED\_SHORT,0);

</script>

</body>

</html>

* Kết quả thực hiện
* ***File WebGLDrawTriangle02.htm***

<!DOCTYPE html>

<html>

<head>

<title>Draw a triangle</title>

</head>

<body onload="init()">

<canvas id="myCanvas" width="640" height="480"></canvas>

</body>

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

attribute vec4 a\_Position;

void main() {

gl\_Position = a\_Position;

}

</script>

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

void main() {

gl\_FragColor = vec4(1.0, 0.0, 0.0, 1.0);

}

</script>

<script>

function init() {

// Get canvas object from the DOM

var canvas = document.getElementById("myCanvas");

// Init WebGL context

var gl = canvas.getContext("webgl");

if (!gl) {

console.log("Failed to get the rendering context for WebGL");

return;

}

// Init shaders

var vs = document.getElementById('shaderVs').innerHTML;

var fs = document.getElementById('shaderFs').innerHTML;

if (!initShaders(gl, vs, fs)) {

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;

}

// Clear canvas

gl.clearColor(0.0, 0.0, 0.0, 1.0);

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

// Draw

gl.drawArrays(gl.TRIANGLES, 0, n);

}

function initVertexBuffers(gl) {

// Vertices

var dim = 3;

var vertices = new Float32Array([

0, 0.5, 0, // Vertice #1

-0.5, -0.5, 0, // Vertice #2

0.5, -0.5, 0 // Vertice #3

]);

// Create a buffer object

var vertexBuffer = gl.createBuffer();

if (!vertexBuffer) {

console.log('Failed to create the buffer object');

return -1;

}

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

gl.bufferData(gl.ARRAY\_BUFFER, vertices, gl.STATIC\_DRAW);

// Assign the vertices in 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, dim, gl.FLOAT, false, 0, 0);

gl.enableVertexAttribArray(a\_Position);

// Return number of vertices

return vertices.length / dim;

}

function initShaders(gl, vs\_source, fs\_source) {

// Compile shaders

var vertexShader = makeShader(gl, vs\_source, gl.VERTEX\_SHADER);

var fragmentShader = makeShader(gl, fs\_source, gl.FRAGMENT\_SHADER);

// Create program

var glProgram = gl.createProgram();

// Attach and link shaders to the program

gl.attachShader(glProgram, vertexShader);

gl.attachShader(glProgram, fragmentShader);

gl.linkProgram(glProgram);

if (!gl.getProgramParameter(glProgram, gl.LINK\_STATUS)) {

alert("Unable to initialize the shader program");

return false;

}

// Use program

gl.useProgram(glProgram);

gl.program = glProgram;

return true;

}

function makeShader(gl, src, type) {

var shader = gl.createShader(type);

gl.shaderSource(shader, src);

gl.compileShader(shader);

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

alert("Error compiling shader: " + gl.getShaderInfoLog(shader));

return;

}

return shader;

}

</script>

</html>

* ***Bài tập***
* Tổ chức file Web trên thành 02 file riêng: file html và file .js
* **Draw Three Parallel Lines**
* ***File WebGLDrawThreeParallelLines.htm***

<!doctype html>

<html>

<body>

<canvas width = "300" height = "300" id = "my\_Canvas"></canvas>

<script>

/\*======= Creating a canvas =========\*/

var canvas = document.getElementById('my\_Canvas');

var gl = canvas.getContext('experimental-webgl');

/\*======= Defining and storing the geometry ======\*/

var vertices = [

-0.7,-0.1,0,

-0.3,0.6,0,

-0.3,-0.3,0,

0.2,0.6,0,

0.3,-0.3,0,

0.7,0.6,0

]

// Create an empty buffer object

var vertex\_buffer = gl.createBuffer();

// Bind appropriate array buffer to it

gl.bindBuffer(gl.ARRAY\_BUFFER, vertex\_buffer);

// Pass the vertex data to the buffer

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

// Unbind the buffer

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

/\*=================== Shaders ====================\*/

// Vertex shader source code

var vertCode =

'attribute vec3 coordinates;' +

'void main(void) {' +

' gl\_Position = vec4(coordinates, 1.0);' +

'}';

// Create a vertex shader object

var vertShader = gl.createShader(gl.VERTEX\_SHADER);

// Attach vertex shader source code

gl.shaderSource(vertShader, vertCode);

// Compile the vertex shader

gl.compileShader(vertShader);

// Fragment shader source code

var fragCode =

'void main(void) {' +

'gl\_FragColor = vec4(0.0, 0.0, 0.0, 0.1);' +

'}';

// Create fragment shader object

var fragShader = gl.createShader(gl.FRAGMENT\_SHADER);

// Attach fragment shader source code

gl.shaderSource(fragShader, fragCode);

// Compile the fragmentt shader

gl.compileShader(fragShader);

// Create a shader program object to store

// the combined shader program

var shaderProgram = gl.createProgram();

// Attach a vertex shader

gl.attachShader(shaderProgram, vertShader);

// Attach a fragment shader

gl.attachShader(shaderProgram, fragShader);

// Link both the programs

gl.linkProgram(shaderProgram);

// Use the combined shader program object

gl.useProgram(shaderProgram);

/\*======= Associating shaders to buffer objects ======\*/

// Bind vertex buffer object

gl.bindBuffer(gl.ARRAY\_BUFFER, vertex\_buffer);

// Get the attribute location

var coord = gl.getAttribLocation(shaderProgram, "coordinates");

// Point an attribute to the currently bound VBO

gl.vertexAttribPointer(coord, 3, gl.FLOAT, false, 0, 0);

// Enable the attribute

gl.enableVertexAttribArray(coord);

/\*============ Drawing the triangle =============\*/

// Clear the canvas

gl.clearColor(0.5, 0.5, 0.5, 0.9);

// Enable the depth test

gl.enable(gl.DEPTH\_TEST);

// Clear the color and depth buffer

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

// Set the view port

gl.viewport(0,0,canvas.width,canvas.height);

// Draw the triangle

gl.drawArrays(gl.LINES, 0, 6);

// POINTS, LINE\_STRIP, LINE\_LOOP, LINES,

// TRIANGLE\_STRIP,TRIANGLE\_FAN, TRIANGLES

</script>

</body>

</html>

* ***Kết quả thực hiện***
* ***Bài tập***
* Tổ chức file Web trên thành 02 file riêng: file html và file .js
* Vẽ các hình tam giác, chữ nhật, ngôi sao, … có tô màu
* **Draw a Quadrilateral**
* ***File WebGLDrawQuadrilateral.html***

<!doctype html>

<html>

<body>

<canvas width = "300" height = "300" id = "my\_Canvas"></canvas>

<script>

/\*============= Creating a canvas ==================\*/

var canvas = document.getElementById('my\_Canvas');

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

/\*========== Defining and storing the geometry ==========\*/

var vertices = [

-0.5,0.5,0.0,

-0.5,-0.5,0.0,

0.5,-0.5,0.0,

0.5,0.5,0.0

];

var colors = [0,0,1, 1,0,0, 0,1,0, 1,0,1,];

indices = [3,2,1,3,1,0];

// Create an empty buffer object and store vertex data

var vertex\_buffer = gl.createBuffer();

gl.bindBuffer(gl.ARRAY\_BUFFER, vertex\_buffer);

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

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

// Create an empty buffer object and store Index data

var Index\_Buffer = gl.createBuffer();

gl.bindBuffer(gl.ELEMENT\_ARRAY\_BUFFER, Index\_Buffer);

gl.bufferData(gl.ELEMENT\_ARRAY\_BUFFER, new Uint16Array(indices), gl.STATIC\_DRAW);

gl.bindBuffer(gl.ELEMENT\_ARRAY\_BUFFER, null);

// Create an empty buffer object and store color data

var color\_buffer = gl.createBuffer ();

gl.bindBuffer(gl.ARRAY\_BUFFER, color\_buffer);

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

/\*======================= Shaders =======================\*/

// vertex shader source code

var vertCode = 'attribute vec3 coordinates;'+

'attribute vec3 color;'+

'varying vec3 vColor;'+

'void main(void) {' +

' gl\_Position = vec4(coordinates, 1.0);' +

'vColor = color;'+

'}';

// Create a vertex shader object

var vertShader = gl.createShader(gl.VERTEX\_SHADER);

// Attach vertex shader source code

gl.shaderSource(vertShader, vertCode);

// Compile the vertex shader

gl.compileShader(vertShader);

// fragment shader source code

var fragCode = 'precision mediump float;'+

'varying vec3 vColor;'+

'void main(void) {'+

'gl\_FragColor = vec4(vColor, 1.);'+

'}';

// Create fragment shader object

var fragShader = gl.createShader(gl.FRAGMENT\_SHADER);

// Attach fragment shader source code

gl.shaderSource(fragShader, fragCode);

// Compile the fragmentt shader

gl.compileShader(fragShader);

// Create a shader program object to

// store the combined shader program

var shaderProgram = gl.createProgram();

// Attach a vertex shader

gl.attachShader(shaderProgram, vertShader);

// Attach a fragment shader

gl.attachShader(shaderProgram, fragShader);

// Link both the programs

gl.linkProgram(shaderProgram);

// Use the combined shader program object

gl.useProgram(shaderProgram);

/\* ======== Associating shaders to buffer objects =======\*/

// Bind vertex buffer object

gl.bindBuffer(gl.ARRAY\_BUFFER, vertex\_buffer);

// Bind index buffer object

gl.bindBuffer(gl.ELEMENT\_ARRAY\_BUFFER, Index\_Buffer);

// Get the attribute location

var coord = gl.getAttribLocation(shaderProgram, "coordinates");

// point an attribute to the currently bound VBO

gl.vertexAttribPointer(coord, 3, gl.FLOAT, false, 0, 0);

// Enable the attribute

gl.enableVertexAttribArray(coord);

// bind the color buffer

gl.bindBuffer(gl.ARRAY\_BUFFER, color\_buffer);

// get the attribute location

var color = gl.getAttribLocation(shaderProgram, "color");

// point attribute to the volor buffer object

gl.vertexAttribPointer(color, 3, gl.FLOAT, false,0,0) ;

// enable the color attribute

gl.enableVertexAttribArray(color);

/\*============Drawing the Quad====================\*/

// Clear the canvas

gl.clearColor(0.5, 0.5, 0.5, 0.9);

// Enable the depth test

gl.enable(gl.DEPTH\_TEST);

// Clear the color buffer bit

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

// Set the view port

gl.viewport(0,0,canvas.width,canvas.height);

//Draw the triangle

gl.drawElements(gl.TRIANGLES, indices.length, gl.UNSIGNED\_SHORT,0);

</script>

</body>

</html>

* **Translate**
* ***File WebGLTranslate.html***

<!doctype html>

<html>

<body>

<canvas width = "300" height = "300" id = "my\_Canvas"></canvas>

<script>

/\*=================Creating a canvas=========================\*/

var canvas = document.getElementById('my\_Canvas');

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

/\*===========Defining and storing the geometry==============\*/

var vertices = [

-0.5,0.5,0.0,

-0.5,-0.5,0.0,

0.5,-0.5,0.0,

];

//Create an empty buffer object and store vertex data

var vertex\_buffer = gl.createBuffer();

//Create a new buffer

gl.bindBuffer(gl.ARRAY\_BUFFER, vertex\_buffer);

//bind it to the current buffer

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

// Pass the buffer data

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

/\*========================Shaders============================\*/

//vertex shader source code

var vertCode =

'attribute vec4 coordinates;' +

'uniform vec4 translation;'+

'void main(void) {' +

' gl\_Position = coordinates + translation;' +

'}';

//Create a vertex shader program object and compile it

var vertShader = gl.createShader(gl.VERTEX\_SHADER);

gl.shaderSource(vertShader, vertCode);

gl.compileShader(vertShader);

//fragment shader source code

var fragCode =

'void main(void) {' +

' gl\_FragColor = vec4(0.0, 0.0, 0.0, 0.1);' +

'}';

//Create a fragment shader program object and compile it

var fragShader = gl.createShader(gl.FRAGMENT\_SHADER);

gl.shaderSource(fragShader, fragCode);

gl.compileShader(fragShader);

//Create and use combiened shader program

var shaderProgram = gl.createProgram();

gl.attachShader(shaderProgram, vertShader);

gl.attachShader(shaderProgram, fragShader);

gl.linkProgram(shaderProgram);

gl.useProgram(shaderProgram);

/\* ===========Associating shaders to buffer objects============\*/

gl.bindBuffer(gl.ARRAY\_BUFFER, vertex\_buffer);

var coordinatesVar = gl.getAttribLocation(shaderProgram, "coordinates");

gl.vertexAttribPointer(coordinatesVar, 3, gl.FLOAT, false, 0, 0);

gl.enableVertexAttribArray(coordinatesVar);

/\* ==========translation======================================\*/

var Tx = 0.5, Ty = 0.5, Tz = 0.0;

var translation = gl.getUniformLocation(shaderProgram, 'translation');

gl.uniform4f(translation, Tx, Ty, Tz, 0.0);

/\*=================Drawing the riangle and transforming it========================\*/

gl.clearColor(0.5, 0.5, 0.5, 0.9);

gl.enable(gl.DEPTH\_TEST);

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

gl.viewport(0,0,canvas.width,canvas.height);

gl.drawArrays(gl.TRIANGLES, 0, 3);

</script>

</body>

</html>

* **Scaling**
* ***File WebGLScale.html***

<!doctype html>

<html>

<body>

<canvas width = "300" height = "300" id = "my\_Canvas"></canvas>

<script>

/\*=================Creating a canvas=========================\*/

var canvas = document.getElementById('my\_Canvas');

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

/\*===========Defining and storing the geometry==============\*/

var vertices = [

-0.5,0.5,0.0,

-0.5,-0.5,0.0,

0.5,-0.5,0.0,

];

//Create an empty buffer object and store vertex data

var vertex\_buffer = gl.createBuffer();

gl.bindBuffer(gl.ARRAY\_BUFFER, vertex\_buffer);

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

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

/\*========================Shaders============================\*/

//Vertex shader source code

var vertCode =

'attribute vec4 coordinates;' +

'uniform mat4 u\_xformMatrix;' +

'void main(void) {' +

' gl\_Position = u\_xformMatrix \* coordinates;' +

'}';

//Create a vertex shader program object and compile it

var vertShader = gl.createShader(gl.VERTEX\_SHADER);

gl.shaderSource(vertShader, vertCode);

gl.compileShader(vertShader);

//fragment shader source code

var fragCode =

'void main(void) {' +

' gl\_FragColor = vec4(0.0, 0.0, 0.0, 0.1);' +

'}';

//Create a fragment shader program object and compile it

var fragShader = gl.createShader(gl.FRAGMENT\_SHADER);

gl.shaderSource(fragShader, fragCode);

gl.compileShader(fragShader);

//Create and use combiened shader program

var shaderProgram = gl.createProgram();

gl.attachShader(shaderProgram, vertShader);

gl.attachShader(shaderProgram, fragShader);

gl.linkProgram(shaderProgram);

gl.useProgram(shaderProgram);

/\*===================scaling==========================\*/

var Sx = 1.0, Sy = 1.5, Sz = 1.0;

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

]);

var u\_xformMatrix = gl.getUniformLocation(shaderProgram, 'u\_xformMatrix');

gl.uniformMatrix4fv(u\_xformMatrix, false, xformMatrix);

/\* ===========Associating shaders to buffer objects============\*/

gl.bindBuffer(gl.ARRAY\_BUFFER, vertex\_buffer);

var coordinatesVar = gl.getAttribLocation(shaderProgram, "coordinates");

gl.vertexAttribPointer(coordinatesVar, 3, gl.FLOAT, false, 0, 0);

gl.enableVertexAttribArray(coordinatesVar);

/\*=================Drawing the Quad========================\*/

gl.clearColor(0.5, 0.5, 0.5, 0.9);

gl.enable(gl.DEPTH\_TEST);

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

gl.viewport(0,0,canvas.width,canvas.height);

gl.drawArrays(gl.TRIANGLES, 0, 3);

</script>

</body>

</html>

* **Rotation**
* ***File WebGLRotation.html***

<!doctype html>

<html>

<body>

<canvas width = "400" height = "400" id = "my\_Canvas"></canvas>

<script>

/\*=================Creating a canvas=========================\*/

var canvas = document.getElementById('my\_Canvas');

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

/\*===========Defining and storing the geometry==============\*/

var vertices = [ -1,-1,-1, 1,-1,-1, 1, 1,-1 ];

var colors = [ 1,1,1, 1,1,1, 1,1,1 ];

var indices = [ 0,1,2 ];

//Create and store data into vertex buffer

var vertex\_buffer = gl.createBuffer ();

gl.bindBuffer(gl.ARRAY\_BUFFER, vertex\_buffer);

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

//Create and store data into color buffer

var color\_buffer = gl.createBuffer ();

gl.bindBuffer(gl.ARRAY\_BUFFER, color\_buffer);

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

//Create and store data into index buffer

var index\_buffer = gl.createBuffer ();

gl.bindBuffer(gl.ELEMENT\_ARRAY\_BUFFER, index\_buffer);

gl.bufferData(gl.ELEMENT\_ARRAY\_BUFFER, new Uint16Array(indices), gl.STATIC\_DRAW);

/\*==========================Shaders=========================\*/

var vertCode = 'attribute vec3 position;'+

'uniform mat4 Pmatrix;'+

'uniform mat4 Vmatrix;'+

'uniform mat4 Mmatrix;'+

'attribute vec3 color;'+//the color of the point

'varying vec3 vColor;'+

'void main(void) { '+//pre-built function

'gl\_Position = Pmatrix\*Vmatrix\*Mmatrix\*vec4(position, 1.);'+

'vColor = color;'+

'}';

var fragCode = 'precision mediump float;'+

'varying vec3 vColor;'+

'void main(void) {'+

'gl\_FragColor = vec4(vColor, 1.);'+

'}';

var vertShader = gl.createShader(gl.VERTEX\_SHADER);

gl.shaderSource(vertShader, vertCode);

gl.compileShader(vertShader);

var fragShader = gl.createShader(gl.FRAGMENT\_SHADER);

gl.shaderSource(fragShader, fragCode);

gl.compileShader(fragShader);

var shaderProgram = gl.createProgram();

gl.attachShader(shaderProgram, vertShader);

gl.attachShader(shaderProgram, fragShader);

gl.linkProgram(shaderProgram);

/\*===========associating attributes to vertex shader ============\*/

var Pmatrix = gl.getUniformLocation(shaderProgram, "Pmatrix");

var Vmatrix = gl.getUniformLocation(shaderProgram, "Vmatrix");

var Mmatrix = gl.getUniformLocation(shaderProgram, "Mmatrix");

gl.bindBuffer(gl.ARRAY\_BUFFER, vertex\_buffer);

var position = gl.getAttribLocation(shaderProgram, "position");

gl.vertexAttribPointer(position, 3, gl.FLOAT, false,0,0) ; //position

gl.enableVertexAttribArray(position);

gl.bindBuffer(gl.ARRAY\_BUFFER, color\_buffer);

var color = gl.getAttribLocation(shaderProgram, "color");

gl.vertexAttribPointer(color, 3, gl.FLOAT, false,0,0) ; //color

gl.enableVertexAttribArray(color);

gl.useProgram(shaderProgram);

/\*========================= MATRIX ========================= \*/

function get\_projection(angle, a, zMin, zMax) {

var ang = Math.tan((angle\*.5)\*Math.PI/180);//angle\*.5

return [

0.5/ang, 0 , 0, 0,

0, 0.5\*a/ang, 0, 0,

0, 0, -(zMax+zMin)/(zMax-zMin), -1,

0, 0, (-2\*zMax\*zMin)/(zMax-zMin), 0

];

}

var proj\_matrix = get\_projection(40, canvas.width/canvas.height, 1, 100);

var mov\_matrix = [1,0,0,0, 0,1,0,0, 0,0,1,0, 0,0,0,1];

var view\_matrix = [1,0,0,0, 0,1,0,0, 0,0,1,0, 0,0,0,1];

//translating z

view\_matrix[14] = view\_matrix[14]-6; //zoom

/\*=======================rotation========================\*/

function rotateZ(m, angle) {

var c = Math.cos(angle);

var s = Math.sin(angle);

var mv0 = m[0], mv4 = m[4], mv8 = m[8];

m[0] = c\*m[0]-s\*m[1];

m[4] = c\*m[4]-s\*m[5];

m[8] = c\*m[8]-s\*m[9];

m[1] = c\*m[1]+s\*mv0;

m[5] = c\*m[5]+s\*mv4;

m[9] = c\*m[9]+s\*mv8;

}

/\*=================Drawing===========================\*/

var time\_old = 0;

var animate = function(time) {

var dt = time-time\_old;

rotateZ(mov\_matrix, dt\*0.002);

time\_old = time;

gl.enable(gl.DEPTH\_TEST);

gl.depthFunc(gl.LEQUAL);

gl.clearColor(0.5, 0.5, 0.5, 0.9);

gl.clearDepth(1.0);

gl.viewport(0.0, 0.0, canvas.width, canvas.height);

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

gl.uniformMatrix4fv(Pmatrix, false, proj\_matrix);

gl.uniformMatrix4fv(Vmatrix, false, view\_matrix);

gl.uniformMatrix4fv(Mmatrix, false, mov\_matrix);

gl.bindBuffer(gl.ELEMENT\_ARRAY\_BUFFER, index\_buffer);

gl.drawElements(gl.TRIANGLES, indices.length, gl.UNSIGNED\_SHORT, 0);

window.requestAnimationFrame(animate);

}

animate(0);

</script>

</body>

</html>

* **Rotating 3D cube**
* ***File WebGLRotation3DCube.html***

<!doctype html>

<html>

<body>

<canvas width = "570" height = "570" id = "my\_Canvas"></canvas>

<script>

/\*============= Creating a canvas =================\*/

var canvas = document.getElementById('my\_Canvas');

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

/\*============ Defining and storing the geometry =========\*/

var vertices = [

-1,-1,-1, 1,-1,-1, 1, 1,-1, -1, 1,-1,

-1,-1, 1, 1,-1, 1, 1, 1, 1, -1, 1, 1,

-1,-1,-1, -1, 1,-1, -1, 1, 1, -1,-1, 1,

1,-1,-1, 1, 1,-1, 1, 1, 1, 1,-1, 1,

-1,-1,-1, -1,-1, 1, 1,-1, 1, 1,-1,-1,

-1, 1,-1, -1, 1, 1, 1, 1, 1, 1, 1,-1,

];

var colors = [

5,3,7, 5,3,7, 5,3,7, 5,3,7,

1,1,3, 1,1,3, 1,1,3, 1,1,3,

0,0,1, 0,0,1, 0,0,1, 0,0,1,

1,0,0, 1,0,0, 1,0,0, 1,0,0,

1,1,0, 1,1,0, 1,1,0, 1,1,0,

0,1,0, 0,1,0, 0,1,0, 0,1,0

];

var indices = [

0,1,2, 0,2,3, 4,5,6, 4,6,7,

8,9,10, 8,10,11, 12,13,14, 12,14,15,

16,17,18, 16,18,19, 20,21,22, 20,22,23

];

// Create and store data into vertex buffer

var vertex\_buffer = gl.createBuffer ();

gl.bindBuffer(gl.ARRAY\_BUFFER, vertex\_buffer);

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

// Create and store data into color buffer

var color\_buffer = gl.createBuffer ();

gl.bindBuffer(gl.ARRAY\_BUFFER, color\_buffer);

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

// Create and store data into index buffer

var index\_buffer = gl.createBuffer ();

gl.bindBuffer(gl.ELEMENT\_ARRAY\_BUFFER, index\_buffer);

gl.bufferData(gl.ELEMENT\_ARRAY\_BUFFER, new Uint16Array(indices), gl.STATIC\_DRAW);

/\*=================== Shaders =========================\*/

var vertCode = 'attribute vec3 position;'+

'uniform mat4 Pmatrix;'+

'uniform mat4 Vmatrix;'+

'uniform mat4 Mmatrix;'+

'attribute vec3 color;'+//the color of the point

'varying vec3 vColor;'+

'void main(void) { '+//pre-built function

'gl\_Position = Pmatrix\*Vmatrix\*Mmatrix\*vec4(position, 1.);'+

'vColor = color;'+

'}';

var fragCode = 'precision mediump float;'+

'varying vec3 vColor;'+

'void main(void) {'+

'gl\_FragColor = vec4(vColor, 1.);'+

'}';

var vertShader = gl.createShader(gl.VERTEX\_SHADER);

gl.shaderSource(vertShader, vertCode);

gl.compileShader(vertShader);

var fragShader = gl.createShader(gl.FRAGMENT\_SHADER);

gl.shaderSource(fragShader, fragCode);

gl.compileShader(fragShader);

var shaderProgram = gl.createProgram();

gl.attachShader(shaderProgram, vertShader);

gl.attachShader(shaderProgram, fragShader);

gl.linkProgram(shaderProgram);

/\* ====== Associating attributes to vertex shader =====\*/

var Pmatrix = gl.getUniformLocation(shaderProgram, "Pmatrix");

var Vmatrix = gl.getUniformLocation(shaderProgram, "Vmatrix");

var Mmatrix = gl.getUniformLocation(shaderProgram, "Mmatrix");

gl.bindBuffer(gl.ARRAY\_BUFFER, vertex\_buffer);

var position = gl.getAttribLocation(shaderProgram, "position");

gl.vertexAttribPointer(position, 3, gl.FLOAT, false,0,0) ;

// Position

gl.enableVertexAttribArray(position);

gl.bindBuffer(gl.ARRAY\_BUFFER, color\_buffer);

var color = gl.getAttribLocation(shaderProgram, "color");

gl.vertexAttribPointer(color, 3, gl.FLOAT, false,0,0) ;

// Color

gl.enableVertexAttribArray(color);

gl.useProgram(shaderProgram);

/\*==================== MATRIX =====================\*/

function get\_projection(angle, a, zMin, zMax) {

var ang = Math.tan((angle\*.5)\*Math.PI/180);//angle\*.5

return [

0.5/ang, 0 , 0, 0,

0, 0.5\*a/ang, 0, 0,

0, 0, -(zMax+zMin)/(zMax-zMin), -1,

0, 0, (-2\*zMax\*zMin)/(zMax-zMin), 0

];

}

var proj\_matrix = get\_projection(40, canvas.width/canvas.height, 1, 100);

var mov\_matrix = [1,0,0,0, 0,1,0,0, 0,0,1,0, 0,0,0,1];

var view\_matrix = [1,0,0,0, 0,1,0,0, 0,0,1,0, 0,0,0,1];

// translating z

view\_matrix[14] = view\_matrix[14]-6;//zoom

/\*==================== Rotation ====================\*/

function rotateZ(m, angle) {

var c = Math.cos(angle);

var s = Math.sin(angle);

var mv0 = m[0], mv4 = m[4], mv8 = m[8];

m[0] = c\*m[0]-s\*m[1];

m[4] = c\*m[4]-s\*m[5];

m[8] = c\*m[8]-s\*m[9];

m[1]=c\*m[1]+s\*mv0;

m[5]=c\*m[5]+s\*mv4;

m[9]=c\*m[9]+s\*mv8;

}

function rotateX(m, angle) {

var c = Math.cos(angle);

var s = Math.sin(angle);

var mv1 = m[1], mv5 = m[5], mv9 = m[9];

m[1] = m[1]\*c-m[2]\*s;

m[5] = m[5]\*c-m[6]\*s;

m[9] = m[9]\*c-m[10]\*s;

m[2] = m[2]\*c+mv1\*s;

m[6] = m[6]\*c+mv5\*s;

m[10] = m[10]\*c+mv9\*s;

}

function rotateY(m, angle) {

var c = Math.cos(angle);

var s = Math.sin(angle);

var mv0 = m[0], mv4 = m[4], mv8 = m[8];

m[0] = c\*m[0]+s\*m[2];

m[4] = c\*m[4]+s\*m[6];

m[8] = c\*m[8]+s\*m[10];

m[2] = c\*m[2]-s\*mv0;

m[6] = c\*m[6]-s\*mv4;

m[10] = c\*m[10]-s\*mv8;

}

/\*================= Drawing ===========================\*/

var time\_old = 0;

var animate = function(time) {

var dt = time-time\_old;

rotateZ(mov\_matrix, dt\*0.005);//time

rotateY(mov\_matrix, dt\*0.002);

rotateX(mov\_matrix, dt\*0.003);

time\_old = time;

gl.enable(gl.DEPTH\_TEST);

gl.depthFunc(gl.LEQUAL);

gl.clearColor(0.5, 0.5, 0.5, 0.9);

gl.clearDepth(1.0);

gl.viewport(0.0, 0.0, canvas.width, canvas.height);

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

gl.uniformMatrix4fv(Pmatrix, false, proj\_matrix);

gl.uniformMatrix4fv(Vmatrix, false, view\_matrix);

gl.uniformMatrix4fv(Mmatrix, false, mov\_matrix);

gl.bindBuffer(gl.ELEMENT\_ARRAY\_BUFFER, index\_buffer);

gl.drawElements(gl.TRIANGLES, indices.length, gl.UNSIGNED\_SHORT, 0);

window.requestAnimationFrame(animate);

}

animate(0);

</script>

</body>

</html>

* **Interactive 3D cube**

Draw a 3D cube that can be rotated using mouse controls

* ***File WebGLRotation3DCubeMouse.html***

<!doctype html>

<html>

<body>

<canvas width = "570" height = "570" id = "my\_Canvas"></canvas>

<script>

/\*============= Creating a canvas ======================\*/

var canvas = document.getElementById('my\_Canvas');

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

/\*========== Defining and storing the geometry ==========\*/

var vertices = [

-1,-1,-1, 1,-1,-1, 1, 1,-1, -1, 1,-1,

-1,-1, 1, 1,-1, 1, 1, 1, 1, -1, 1, 1,

-1,-1,-1, -1, 1,-1, -1, 1, 1, -1,-1, 1,

1,-1,-1, 1, 1,-1, 1, 1, 1, 1,-1, 1,

-1,-1,-1, -1,-1, 1, 1,-1, 1, 1,-1,-1,

-1, 1,-1, -1, 1, 1, 1, 1, 1, 1, 1,-1,

];

var colors = [

5,3,7, 5,3,7, 5,3,7, 5,3,7,

1,1,3, 1,1,3, 1,1,3, 1,1,3,

0,0,1, 0,0,1, 0,0,1, 0,0,1,

1,0,0, 1,0,0, 1,0,0, 1,0,0,

1,1,0, 1,1,0, 1,1,0, 1,1,0,

0,1,0, 0,1,0, 0,1,0, 0,1,0

];

var indices = [

0,1,2, 0,2,3, 4,5,6, 4,6,7,

8,9,10, 8,10,11, 12,13,14, 12,14,15,

16,17,18, 16,18,19, 20,21,22, 20,22,23

];

// Create and store data into vertex buffer

var vertex\_buffer = gl.createBuffer ();

gl.bindBuffer(gl.ARRAY\_BUFFER, vertex\_buffer);

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

// Create and store data into color buffer

var color\_buffer = gl.createBuffer ();

gl.bindBuffer(gl.ARRAY\_BUFFER, color\_buffer);

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

// Create and store data into index buffer

var index\_buffer = gl.createBuffer ();

gl.bindBuffer(gl.ELEMENT\_ARRAY\_BUFFER, index\_buffer);

gl.bufferData(gl.ELEMENT\_ARRAY\_BUFFER, new Uint16Array(indices), gl.STATIC\_DRAW);

/\*=================== SHADERS =================== \*/

var vertCode = 'attribute vec3 position;'+

'uniform mat4 Pmatrix;'+

'uniform mat4 Vmatrix;'+

'uniform mat4 Mmatrix;'+

'attribute vec3 color;'+//the color of the point

'varying vec3 vColor;'+

'void main(void) { '+//pre-built function

'gl\_Position = Pmatrix\*Vmatrix\*Mmatrix\*vec4(position, 1.);'+

'vColor = color;'+

'}';

var fragCode = 'precision mediump float;'+

'varying vec3 vColor;'+

'void main(void) {'+

'gl\_FragColor = vec4(vColor, 1.);'+

'}';

var vertShader = gl.createShader(gl.VERTEX\_SHADER);

gl.shaderSource(vertShader, vertCode);

gl.compileShader(vertShader);

var fragShader = gl.createShader(gl.FRAGMENT\_SHADER);

gl.shaderSource(fragShader, fragCode);

gl.compileShader(fragShader);

var shaderprogram = gl.createProgram();

gl.attachShader(shaderprogram, vertShader);

gl.attachShader(shaderprogram, fragShader);

gl.linkProgram(shaderprogram);

/\*======== Associating attributes to vertex shader =====\*/

var \_Pmatrix = gl.getUniformLocation(shaderprogram, "Pmatrix");

var \_Vmatrix = gl.getUniformLocation(shaderprogram, "Vmatrix");

var \_Mmatrix = gl.getUniformLocation(shaderprogram, "Mmatrix");

gl.bindBuffer(gl.ARRAY\_BUFFER, vertex\_buffer);

var \_position = gl.getAttribLocation(shaderprogram, "position");

gl.vertexAttribPointer(\_position, 3, gl.FLOAT, false,0,0);

gl.enableVertexAttribArray(\_position);

gl.bindBuffer(gl.ARRAY\_BUFFER, color\_buffer);

var \_color = gl.getAttribLocation(shaderprogram, "color");

gl.vertexAttribPointer(\_color, 3, gl.FLOAT, false,0,0) ;

gl.enableVertexAttribArray(\_color);

gl.useProgram(shaderprogram);

/\*==================== MATRIX ====================== \*/

function get\_projection(angle, a, zMin, zMax) {

var ang = Math.tan((angle\*.5)\*Math.PI/180);//angle\*.5

return [

0.5/ang, 0 , 0, 0,

0, 0.5\*a/ang, 0, 0,

0, 0, -(zMax+zMin)/(zMax-zMin), -1,

0, 0, (-2\*zMax\*zMin)/(zMax-zMin), 0

];

}

var proj\_matrix = get\_projection(40, canvas.width/canvas.height, 1, 100);

var mo\_matrix = [ 1,0,0,0, 0,1,0,0, 0,0,1,0, 0,0,0,1 ];

var view\_matrix = [ 1,0,0,0, 0,1,0,0, 0,0,1,0, 0,0,0,1 ];

view\_matrix[14] = view\_matrix[14]-6;

/\*================= Mouse events ======================\*/

var AMORTIZATION = 0.95;

var drag = false;

var old\_x, old\_y;

var dX = 0, dY = 0;

var mouseDown = function(e) {

drag = true;

old\_x = e.pageX, old\_y = e.pageY;

e.preventDefault();

return false;

};

var mouseUp = function(e){

drag = false;

};

var mouseMove = function(e) {

if (!drag) return false;

dX = (e.pageX-old\_x)\*2\*Math.PI/canvas.width,

dY = (e.pageY-old\_y)\*2\*Math.PI/canvas.height;

THETA+= dX;

PHI+=dY;

old\_x = e.pageX, old\_y = e.pageY;

e.preventDefault();

};

canvas.addEventListener("mousedown", mouseDown, false);

canvas.addEventListener("mouseup", mouseUp, false);

canvas.addEventListener("mouseout", mouseUp, false);

canvas.addEventListener("mousemove", mouseMove, false);

/\*=========================rotation================\*/

function rotateX(m, angle) {

var c = Math.cos(angle);

var s = Math.sin(angle);

var mv1 = m[1], mv5 = m[5], mv9 = m[9];

m[1] = m[1]\*c-m[2]\*s;

m[5] = m[5]\*c-m[6]\*s;

m[9] = m[9]\*c-m[10]\*s;

m[2] = m[2]\*c+mv1\*s;

m[6] = m[6]\*c+mv5\*s;

m[10] = m[10]\*c+mv9\*s;

}

function rotateY(m, angle) {

var c = Math.cos(angle);

var s = Math.sin(angle);

var mv0 = m[0], mv4 = m[4], mv8 = m[8];

m[0] = c\*m[0]+s\*m[2];

m[4] = c\*m[4]+s\*m[6];

m[8] = c\*m[8]+s\*m[10];

m[2] = c\*m[2]-s\*mv0;

m[6] = c\*m[6]-s\*mv4;

m[10] = c\*m[10]-s\*mv8;

}

/\*=================== Drawing =================== \*/

var THETA = 0,

PHI = 0;

var time\_old = 0;

var animate = function(time) {

var dt = time-time\_old;

if (!drag) {

dX \*= AMORTIZATION, dY\*=AMORTIZATION;

THETA+=dX, PHI+=dY;

}

//set model matrix to I4

mo\_matrix[0] = 1, mo\_matrix[1] = 0, mo\_matrix[2] = 0,

mo\_matrix[3] = 0,

mo\_matrix[4] = 0, mo\_matrix[5] = 1, mo\_matrix[6] = 0,

mo\_matrix[7] = 0,

mo\_matrix[8] = 0, mo\_matrix[9] = 0, mo\_matrix[10] = 1,

mo\_matrix[11] = 0,

mo\_matrix[12] = 0, mo\_matrix[13] = 0, mo\_matrix[14] = 0,

mo\_matrix[15] = 1;

rotateY(mo\_matrix, THETA);

rotateX(mo\_matrix, PHI);

time\_old = time;

gl.enable(gl.DEPTH\_TEST);

// gl.depthFunc(gl.LEQUAL);

gl.clearColor(0.5, 0.5, 0.5, 0.9);

gl.clearDepth(1.0);

gl.viewport(0.0, 0.0, canvas.width, canvas.height);

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

gl.uniformMatrix4fv(\_Pmatrix, false, proj\_matrix);

gl.uniformMatrix4fv(\_Vmatrix, false, view\_matrix);

gl.uniformMatrix4fv(\_Mmatrix, false, mo\_matrix);

gl.bindBuffer(gl.ELEMENT\_ARRAY\_BUFFER, index\_buffer);

gl.drawElements(gl.TRIANGLES, indices.length, gl.UNSIGNED\_SHORT, 0);

window.requestAnimationFrame(animate);

}

animate(0);

</script>

</body>

</html>

* **Bài tập**

Vẽ các hình sau trên WebGL:

* Hình tròn, hình elippse
* Hình khối tứ diện
* Đồ thị hình sin, cos
* D
* **Tham khảo**
* <https://www.tutorialspoint.com/webgl/webgl_interactive_cube.htm>

(Tài liệu lưu hành nội bộ)

-----------------------------------------------