```
// global variables
var canvas=null;
var gl=null; // webgl context
var bFullscreen=false;
var canvas_original_width;
var canvas_original_height;

const WebGLMacros= // when whole 'WebGLMacros' is 'const', all inside it are
  automatically 'const'
{
VDG_ATTRIBUTE_VERTEX:0,
VDG_ATTRIBUTE_COLOR:1,
VDG_ATTRIBUTE_NORMAL:2,
VDG_ATTRIBUTE_TEXTURE0:3,
};

var vertexShaderObject;
var fragmentShaderObject;
var shaderProgramObject;

var vao;
var vbo;
var mvpUniform;

var orthographicProjectionMatrix;

// To start animation : To have requestAnimationFrame() to be called "cross-
  browser" compatible
var requestAnimationFrame =
window.requestAnimationFrame ||
window.webkitRequestAnimationFrame ||
window.mozRequestAnimationFrame ||
window.oRequestAnimationFrame ||
window.msRequestAnimationFrame;

// To stop animation : To have cancelAnimationFrame() to be called "cross-
  browser" compatible
var cancelAnimationFrame =
window.cancelAnimationFrame ||
window.webkitCancelRequestAnimationFrame || window.webkitCancelAnimationFrame ||
window.mozCancelRequestAnimationFrame || window.mozCancelAnimationFrame ||
window.oCancelRequestAnimationFrame || window.oCancelAnimationFrame ||
window.msCancelRequestAnimationFrame || window.msCancelAnimationFrame;

// onload function
function main()
{
    // get <canvas> element
    canvas = document.getElementById("AMC");
    if(!canvas)
        console.log("Obtaining Canvas Failed\n");
    else
```

```
        console.log("Obtaining Canvas Succeeded\n");
    canvas_original_width=canvas.width;
    canvas_original_height=canvas.height;

    // register keyboard's keydown event handler
    window.addEventListener("keydown", keyDown, false);
    window.addEventListener("click", mouseDown, false);
    window.addEventListener("resize", resize, false);

    // initialize WebGL
    init();

    // start drawing here as warming-up
    resize();
    draw();
}

function toggleFullScreen()
{
    // code
    var fullscreen_element =
    document.fullscreenElement ||
    document.webkitFullscreenElement ||
    document.mozFullScreenElement ||
    document.msFullscreenElement ||
    null;

    // if not fullscreen
    if(fullscreen_element==null)
    {
        if(canvas.requestFullscreen)
            canvas.requestFullscreen();
        else if(canvas.mozRequestFullScreen)
            canvas.mozRequestFullScreen();
        else if(canvas.webkitRequestFullscreen)
            canvas.webkitRequestFullscreen();
        else if(canvas.msRequestFullscreen)
            canvas.msRequestFullscreen();
        bFullscreen=true;
    }
    else // if already fullscreen
    {
        if(document.exitFullscreen)
            document.exitFullscreen();
        else if(document.mozCancelFullScreen)
            document.mozCancelFullScreen();
        else if(document.webkitExitFullscreen)
            document.webkitExitFullscreen();
        else if(document.msExitFullscreen)
            document.msExitFullscreen();
        bFullscreen=false;
    }
```

```
}

function init()
{
    // code
    // get WebGL 2.0 context
    gl = canvas.getContext("webgl2");
    if(gl==null) // failed to get context
    {
        console.log("Failed to get the rendering context for WebGL");
        return;
    }
    gl.viewportWidth = canvas.width;
    gl.viewportHeight = canvas.height;

    // vertex shader
    var vertexShaderSourceCode=
    "#version 300 es"+
    "\n"+
    "in vec4 vPosition;"+
    "uniform mat4 u_mvp_matrix;"+
    "void main(void)"+
    "{"+
    "gl_Position = u_mvp_matrix * vPosition;"+
    "}";
    vertexShaderObject=gl.createShader(gl.VERTEX_SHADER);
    gl.shaderSource(vertexShaderObject,vertexShaderSourceCode);
    gl.compileShader(vertexShaderObject);
    if(gl.getShaderParameter(vertexShaderObject,gl.COMPILE_STATUS)==false)
    {
        var error=gl.getShaderInfoLog(vertexShaderObject);
        if(error.length > 0)
        {
            alert(error);
            uninitialize();
        }
    }

    // fragment shader
    var fragmentShaderSourceCode=
    "#version 300 es"+
    "\n"+
    "precision highp float;"+
    "out vec4 FragColor;"+
    "void main(void)"+
    "{"+
    "FragColor = vec4(1.0, 1.0, 1.0, 1.0);"+
    "}"
    fragmentShaderObject=gl.createShader(gl.FRAGMENT_SHADER);
    gl.shaderSource(fragmentShaderObject,fragmentShaderSourceCode);
    gl.compileShader(fragmentShaderObject);
    if(gl.getShaderParameter(fragmentShaderObject,gl.COMPILE_STATUS)==false)
    {
        var error=gl.getShaderInfoLog(fragmentShaderObject);
        if(error.length > 0)
        {
            alert(error);
            uninitialize();
        }
    }

    // shader program
    shaderProgramObject=gl.createProgram();
    gl.attachShader(shaderProgramObject,vertexShaderObject);
    gl.attachShader(shaderProgramObject,fragmentShaderObject);

    // pre-link binding of shader program object with vertex shader attributes
    gl.bindAttribLocation
      (shaderProgramObject,WebGLMacros.VDG_ATTRIBUTE_VERTEX,"vPosition");

    // linking
    gl.linkProgram(shaderProgramObject);
    if (!gl.getProgramParameter(shaderProgramObject, gl.LINK_STATUS))
    {
        var error=gl.getProgramInfoLog(shaderProgramObject);
        if(error.length > 0)
        {
            alert(error);
            uninitialize();
        }
    }

    // get MVP uniform location
    mvpUniform=gl.getUniformLocation(shaderProgramObject,"u_mvp_matrix");

    // *** vertices, colors, shader attribs, vbo, vao initializations ***
    var triangleVertices=new Float32Array([
                                           0.0,   50.0, 0.0,   // appex
                                           -50.0, -50.0, 0.0, // left-bottom
                                           50.0, -50.0, 0.0   // right-bottom
                                           ]);

    vao=gl.createVertexArray();
    gl.bindVertexArray(vao);

    vbo = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER,vbo);
    gl.bufferData(gl.ARRAY_BUFFER,triangleVertices,gl.STATIC_DRAW);
    gl.vertexAttribPointer(WebGLMacros.VDG_ATTRIBUTE_VERTEX,
                           3, // 3 is for X,Y,Z co-ordinates in our
                     triangleVertices array
                           gl.FLOAT,
                           false,0,0);
    gl.enableVertexAttribArray(WebGLMacros.VDG_ATTRIBUTE_VERTEX);
    gl.bindBuffer(gl.ARRAY_BUFFER,null);
    gl.bindVertexArray(null);

    // set clear color
    gl.clearColor(0.0, 0.0, 1.0, 1.0); // blue

    // initialize projection matrix
    orthographicProjectionMatrix=mat4.create();
}

function resize()
{
    // code
    if(bFullscreen==true)
    {
        canvas.width=window.innerWidth;
        canvas.height=window.innerHeight;
    }
    else
    {
        canvas.width=canvas_original_width;
        canvas.height=canvas_original_height;
    }

    // set the viewport to match
    gl.viewport(0, 0, canvas.width, canvas.height);

    // Orthographic Projection => left,right,bottom,top,near,far
    if (canvas.width <= canvas.height)
        mat4.ortho(orthographicProjectionMatrix, -100.0, 100.0, (-100.0 *
          (canvas.height / canvas.width)), (100.0 * (canvas.height /
          canvas.width)), -100.0, 100.0);
    else
        mat4.ortho(orthographicProjectionMatrix, (-100.0 * (canvas.width /
          canvas.height)), (100.0 * (canvas.width / canvas.height)), -100.0,
          100.0, -100.0, 100.0);
}

function draw()
{
    // code
    gl.clear(gl.COLOR_BUFFER_BIT);

    gl.useProgram(shaderProgramObject);

    var modelViewMatrix=mat4.create();
    var modelViewProjectionMatrix=mat4.create();
    mat4.multiply
      (modelViewProjectionMatrix,orthographicProjectionMatrix,modelViewMatrix);
    gl.uniformMatrix4fv(mvpUniform,false,modelViewProjectionMatrix);

    gl.bindVertexArray(vao);

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

    gl.bindVertexArray(null);

    gl.useProgram(null);

    // animation loop
    requestAnimationFrame(draw, canvas);
}

function uninitialize()
{
    // code
    if(vao)
    {
        gl.deleteVertexArray(vao);
        vao=null;
    }

    if(vbo)
    {
        gl.deleteBuffer(vbo);
        vbo=null;
    }

    if(shaderProgramObject)
    {
        if(fragmentShaderObject)
        {
            gl.detachShader(shaderProgramObject,fragmentShaderObject);
            gl.deleteShader(fragmentShaderObject);
            fragmentShaderObject=null;
        }

        if(vertexShaderObject)
        {
            gl.detachShader(shaderProgramObject,vertexShaderObject);
            gl.deleteShader(vertexShaderObject);
            vertexShaderObject=null;
        }

        gl.deleteProgram(shaderProgramObject);
        shaderProgramObject=null;
    }
}

function keyDown(event)
{
    // code
    switch(event.keyCode)
    {
        case 27: // Escape
            // uninitialize
            uninitialize();
            // close our application's tab
            window.close(); // may not work in Firefox but works in Safari and chrome
            break;
        case 70: // for 'F' or 'f'
            toggleFullScreen();
            break;
    }
}

function mouseDown()
{
    // code
}
```