```
// 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_cube;
var vbo_cube_position;
var vbo_cube_normal;

var perspectiveProjectionMatrix;

var modelViewMatrixUniform, projectionMatrixUniform;
var ldUniform, kdUniform, lightPositionUniform;
var LKeyPressedUniform;

var angleCube=0.0;

var bLKeyPressed=false;

// 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;"+
    "in vec3 vNormal;"+
    "uniform mat4 u_model_view_matrix;"+
    "uniform mat4 u_projection_matrix;"+
    "uniform mediump int u_LKeyPressed;"+
    "uniform vec3 u_Ld;"+
    "uniform vec3 u_Kd;"+
    "uniform vec4 u_light_position;"+
    "out vec3 diffuse_light;"+
    "void main(void)"+
    "{"+
    "if(u_LKeyPressed == 1)"+
    "{"+
    "vec4 eyeCoordinates=u_model_view_matrix * vPosition;"+
    "vec3 tnorm =  normalize(mat3(u_model_view_matrix) * vNormal);"+
    "vec3 s = normalize(vec3(u_light_position - eyeCoordinates));"+
    "diffuse_light = u_Ld * u_Kd * max( dot( s, tnorm ), 0.0 );"+
    "}"+
    "gl_Position=u_projection_matrix * u_model_view_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;"+
    "in vec3 diffuse_light;"+
    "out vec4 FragColor;"+
    "uniform int u_LKeyPressed;"+
    "void main(void)"+
    "{"+
    "vec4 color;"+
    "if (u_LKeyPressed == 1)"+
    "{"+
    "color = vec4(diffuse_light,1.0);"+
    "}"+
    "else"+
    "{"+
    "color = vec4(1.0, 1.0, 1.0, 1.0);"+
    "}"+
    "FragColor = color;"+
    "}";

    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");
    gl.bindAttribLocation
      (shaderProgramObject,WebGLMacros.VDG_ATTRIBUTE_NORMAL,"vNormal");

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

    // get Model View Matrix uniform location
    modelViewMatrixUniform=gl.getUniformLocation
      (shaderProgramObject,"u_model_view_matrix");
    // get Projection Matrix uniform location
    projectionMatrixUniform=gl.getUniformLocation
      (shaderProgramObject,"u_projection_matrix");

    // get single tap detecting uniform
    LKeyPressedUniform=gl.getUniformLocation
      (shaderProgramObject,"u_LKeyPressed");

    // diffuse color intensity of light
    ldUniform=gl.getUniformLocation(shaderProgramObject,"u_Ld");
    // diffuse reflective color intensity of material
    kdUniform=gl.getUniformLocation(shaderProgramObject,"u_Kd");
    // position of light
    lightPositionUniform=gl.getUniformLocation
      (shaderProgramObject,"u_light_position");

    // *** vertices, colors, shader attribs, vbo, vao initializations ***
    var cubeVertices=new Float32Array([
                                   // top surface
                                   1.0, 1.0,-1.0,  // top-right of top
                                   -1.0, 1.0,-1.0, // top-left of top
                                   -1.0, 1.0, 1.0, // bottom-left of top
                                   1.0, 1.0, 1.0,  // bottom-right of top

                                   // bottom surface
                                   1.0,-1.0, 1.0,  // top-right of bottom
                                   -1.0,-1.0, 1.0, // top-left of bottom
                                   -1.0,-1.0,-1.0, // bottom-left of bottom
                                   1.0,-1.0,-1.0,  // bottom-right of bottom

                                   // front surface
                                   1.0, 1.0, 1.0,  // top-right of front
                                   -1.0, 1.0, 1.0, // top-left of front
                                   -1.0,-1.0, 1.0, // bottom-left of front
```

```
                                        1.0,-1.0, 1.0,  // bottom-right of front

                                        // back surface
                                        1.0,-1.0,-1.0,  // top-right of back
                                        -1.0,-1.0,-1.0, // top-left of back
                                        -1.0, 1.0,-1.0, // bottom-left of back
                                        1.0, 1.0,-1.0,  // bottom-right of back

                                        // left surface
                                        -1.0, 1.0, 1.0, // top-right of left
                                        -1.0, 1.0,-1.0, // top-left of left
                                        -1.0,-1.0,-1.0, // bottom-left of left
                                        -1.0,-1.0, 1.0, // bottom-right of left

                                        // right surface
                                        1.0, 1.0,-1.0,  // top-right of right
                                        1.0, 1.0, 1.0,  // top-left of right
                                        1.0,-1.0, 1.0,  // bottom-left of right
                                        1.0,-1.0,-1.0,  // bottom-right of right
                                        ]);

        var cubeNormals=new Float32Array([
                                       // top
                                       0.0, 1.0, 0.0,
                                       0.0, 1.0, 0.0,
                                       0.0, 1.0, 0.0,
                                       0.0, 1.0, 0.0,

                                       // bottom
                                       0.0, -1.0, 0.0,
                                       0.0, -1.0, 0.0,
                                       0.0, -1.0, 0.0,
                                       0.0, -1.0, 0.0,

                                       // front
                                       0.0, 0.0, 1.0,
                                       0.0, 0.0, 1.0,
                                       0.0, 0.0, 1.0,
                                       0.0, 0.0, 1.0,

                                       // back
                                       0.0, 0.0, -1.0,
                                       0.0, 0.0, -1.0,
                                       0.0, 0.0, -1.0,
                                       0.0, 0.0, -1.0,

                                       // left
                                       -1.0, 0.0, 0.0,
                                       -1.0, 0.0, 0.0,
                                       -1.0, 0.0, 0.0,
                                       -1.0, 0.0, 0.0,

                                        // right
                                        1.0, 0.0, 0.0,
                                        1.0, 0.0, 0.0,
                                        1.0, 0.0, 0.0,
                                        1.0, 0.0, 0.0
                                        ]);

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

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

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

    gl.bindVertexArray(null);

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

    // Depth test will always be enabled
    gl.enable(gl.DEPTH_TEST);

    // depth test to do
    gl.depthFunc(gl.LEQUAL);

    // We will always cull back faces for better performance
    gl.enable(gl.CULL_FACE);

    // initialize projection matrix
    perspectiveProjectionMatrix=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);

    mat4.perspective(perspectiveProjectionMatrix, 45.0, parseFloat(canvas.width)/
      parseFloat(canvas.height), 0.1, 100.0);
}

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

    gl.useProgram(shaderProgramObject);

    if(bLKeyPressed==true)
    {
        gl.uniform1i(LKeyPressedUniform, 1);

        // setting light properties
        gl.uniform3f(ldUniform, 1.0, 1.0, 1.0); // diffuse intensity of light
        // setting material properties
        gl.uniform3f(kdUniform, 0.5, 0.5, 0.5); // diffuse reflectivity of
          material
        var lightPosition = [0.0, 0.0, 2.0, 1.0];
        gl.uniform4fv(lightPositionUniform, lightPosition); // light position
    }
    else
    {
        gl.uniform1i(LKeyPressedUniform, 0);
    }

    var modelViewMatrix=mat4.create(); // itself creates identity matrix

    mat4.translate(modelViewMatrix, modelViewMatrix, [0.0,0.0,-4.0]);

    mat4.rotateX(modelViewMatrix ,modelViewMatrix, degToRad(angleCube));
    mat4.rotateY(modelViewMatrix ,modelViewMatrix, degToRad(angleCube));
    mat4.rotateZ(modelViewMatrix ,modelViewMatrix, degToRad(angleCube));

    gl.uniformMatrix4fv(modelViewMatrixUniform,false,modelViewMatrix);
    gl.uniformMatrix4fv
```

```
        (projectionMatrixUniform,false,perspectiveProjectionMatrix);

    gl.bindVertexArray(vao_cube);

    // *** draw, either by glDrawTriangles() or glDrawArrays() or glDrawElements
      ()
    // actually 2 triangles make 1 cube, so there should be 6 vertices,
    // but as 2 tringles while making square meet each other at diagonal,
    // 2 of 6 vertices are common to both triangles, and hence 6-2=4
    gl.drawArrays(gl.TRIANGLE_FAN,0,4);
    gl.drawArrays(gl.TRIANGLE_FAN,4,4);
    gl.drawArrays(gl.TRIANGLE_FAN,8,4);
    gl.drawArrays(gl.TRIANGLE_FAN,12,4);
    gl.drawArrays(gl.TRIANGLE_FAN,16,4);
    gl.drawArrays(gl.TRIANGLE_FAN,20,4);

    gl.bindVertexArray(null);

    gl.useProgram(null);

    angleCube=angleCube+2.0;
    if(angleCube>=360.0)
        angleCube=angleCube-360.0;

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

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

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

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

    if(shaderProgramObject)
    {
        if(fragmentShaderObject)
```

```javascript
        {
            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 76: // for 'L' or 'l'
            if(bLKeyPressed==false)
                bLKeyPressed=true;
            else
                bLKeyPressed=false;
            break;
        case 70: // for 'F' or 'f'
            toggleFullScreen();
            break;
    }
}

function mouseDown()
{
    // code
}

function degToRad(degrees)
{
    // code
    return(degrees * Math.PI / 180);
}
```