```
// 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
```

```
50          console.log("Obtaining Canvas Succeeded\n");
51      canvas_original_width=canvas.width;
52      canvas_original_height=canvas.height;
53
54      // register keyboard's keydown event handler
55      window.addEventListener("keydown", keyDown, false);
56      window.addEventListener("click", mouseDown, false);
57      window.addEventListener("resize", resize, false);
58
59      // initialize WebGL
60      init();
61
62      // start drawing here as warming-up
63      resize();
64      draw();
65  }
66
67  function toggleFullScreen()
68  {
69      // code
70      var fullscreen_element =
71      document.fullscreenElement ||
72      document.webkitFullscreenElement ||
73      document.mozFullScreenElement ||
74      document.msFullscreenElement ||
75      null;
76
77      // if not fullscreen
78      if(fullscreen_element==null)
79      {
80          if(canvas.requestFullscreen)
81              canvas.requestFullscreen();
82          else if(canvas.mozRequestFullScreen)
83              canvas.mozRequestFullScreen();
84          else if(canvas.webkitRequestFullscreen)
85              canvas.webkitRequestFullscreen();
86          else if(canvas.msRequestFullscreen)
87              canvas.msRequestFullscreen();
88          bFullscreen=true;
89      }
90      else // if already fullscreen
91      {
92          if(document.exitFullscreen)
93              document.exitFullscreen();
94          else if(document.mozCancelFullScreen)
95              document.mozCancelFullScreen();
96          else if(document.webkitExitFullscreen)
97              document.webkitExitFullscreen();
98          else if(document.msExitFullscreen)
99              document.msExitFullscreen();
100         bFullscreen=false;
101     }
```

```
102 }
103
104 function init()
105 {
106     // code
107     // get WebGL 2.0 context
108     gl = canvas.getContext("webgl2");
109     if(gl==null) // failed to get context
110     {
111         console.log("Failed to get the rendering context for WebGL");
112         return;
113     }
114     gl.viewportWidth = canvas.width;
115     gl.viewportHeight = canvas.height;
116
117     // vertex shader
118     var vertexShaderSourceCode=
119     "#version 300 es"+
120     "\n"+
121     "in vec4 vPosition;"+
122     "uniform mat4 u_mvp_matrix;"+
123     "void main(void)"+
124     "{"+
125     "gl_Position = u_mvp_matrix * vPosition;"+
126     "}";
127     vertexShaderObject=gl.createShader(gl.VERTEX_SHADER);
128     gl.shaderSource(vertexShaderObject,vertexShaderSourceCode);
129     gl.compileShader(vertexShaderObject);
130     if(gl.getShaderParameter(vertexShaderObject,gl.COMPILE_STATUS)==false)
131     {
132         var error=gl.getShaderInfoLog(vertexShaderObject);
133         if(error.length > 0)
134         {
135             alert(error);
136             uninitialize();
137         }
138     }
139
140     // fragment shader
141     var fragmentShaderSourceCode=
142     "#version 300 es"+
143     "\n"+
144     "precision highp float;"+
145     "out vec4 FragColor;"+
146     "void main(void)"+
147     "{"+
148     "FragColor = vec4(1.0, 1.0, 1.0, 1.0);"+
149     "}"
150     fragmentShaderObject=gl.createShader(gl.FRAGMENT_SHADER);
151     gl.shaderSource(fragmentShaderObject,fragmentShaderSourceCode);
152     gl.compileShader(fragmentShaderObject);
153     if(gl.getShaderParameter(fragmentShaderObject,gl.COMPILE_STATUS)==false)
```

```
154     {
155         var error=gl.getShaderInfoLog(fragmentShaderObject);
156         if(error.length > 0)
157         {
158             alert(error);
159             uninitialize();
160         }
161     }
162
163     // shader program
164     shaderProgramObject=gl.createProgram();
165     gl.attachShader(shaderProgramObject,vertexShaderObject);
166     gl.attachShader(shaderProgramObject,fragmentShaderObject);
167
168     // pre-link binding of shader program object with vertex shader attributes
169     gl.bindAttribLocation
          (shaderProgramObject,WebGLMacros.VDG_ATTRIBUTE_VERTEX,"vPosition");
170
171     // linking
172     gl.linkProgram(shaderProgramObject);
173     if (!gl.getProgramParameter(shaderProgramObject, gl.LINK_STATUS))
174     {
175         var error=gl.getProgramInfoLog(shaderProgramObject);
176         if(error.length > 0)
177         {
178             alert(error);
179             uninitialize();
180         }
181     }
182
183     // get MVP uniform location
184     mvpUniform=gl.getUniformLocation(shaderProgramObject,"u_mvp_matrix");
185
186     // *** vertices, colors, shader attribs, vbo, vao initializations ***
187     var triangleVertices=new Float32Array([
188                                             0.0,  50.0, 0.0,   // appex
189                                             -50.0, -50.0, 0.0, // left-bottom
190                                             50.0, -50.0, 0.0   // right-bottom
191                                             ]);
192
193     vao=gl.createVertexArray();
194     gl.bindVertexArray(vao);
195
196     vbo = gl.createBuffer();
197     gl.bindBuffer(gl.ARRAY_BUFFER,vbo);
198     gl.bufferData(gl.ARRAY_BUFFER,triangleVertices,gl.STATIC_DRAW);
199     gl.vertexAttribPointer(WebGLMacros.VDG_ATTRIBUTE_VERTEX,
200                            3, // 3 is for X,Y,Z co-ordinates in our
                       triangleVertices array
201                            gl.FLOAT,
202                            false,0,0);
203     gl.enableVertexAttribArray(WebGLMacros.VDG_ATTRIBUTE_VERTEX);
```

```
204     gl.bindBuffer(gl.ARRAY_BUFFER,null);
205     gl.bindVertexArray(null);
206
207     // set clear color
208     gl.clearColor(0.0, 0.0, 1.0, 1.0); // blue
209
210     // initialize projection matrix
211     orthographicProjectionMatrix=mat4.create();
212 }
213
214 function resize()
215 {
216     // code
217     if(bFullscreen==true)
218     {
219         canvas.width=window.innerWidth;
220         canvas.height=window.innerHeight;
221     }
222     else
223     {
224         canvas.width=canvas_original_width;
225         canvas.height=canvas_original_height;
226     }
227
228     // set the viewport to match
229     gl.viewport(0, 0, canvas.width, canvas.height);
230
231     // Orthographic Projection => left,right,bottom,top,near,far
232     if (canvas.width <= canvas.height)
233         mat4.ortho(orthographicProjectionMatrix, -100.0, 100.0, (-100.0 *
           (canvas.height / canvas.width)), (100.0 * (canvas.height /
           canvas.width)), -100.0, 100.0);
234     else
235         mat4.ortho(orthographicProjectionMatrix, (-100.0 * (canvas.width /
           canvas.height)), (100.0 * (canvas.width / canvas.height)), -100.0,
           100.0, -100.0, 100.0);
236 }
237
238 function draw()
239 {
240     // code
241     gl.clear(gl.COLOR_BUFFER_BIT);
242
243     gl.useProgram(shaderProgramObject);
244
245     var modelViewMatrix=mat4.create();
246     var modelViewProjectionMatrix=mat4.create();
247     mat4.multiply
         (modelViewProjectionMatrix,orthographicProjectionMatrix,modelViewMatrix);
248     gl.uniformMatrix4fv(mvpUniform,false,modelViewProjectionMatrix);
249
250     gl.bindVertexArray(vao);
```

```
251
252     gl.drawArrays(gl.TRIANGLES,0,3);
253
254     gl.bindVertexArray(null);
255
256     gl.useProgram(null);
257
258     // animation loop
259     requestAnimationFrame(draw, canvas);
260 }
261
262 function uninitialize()
263 {
264     // code
265     if(vao)
266     {
267         gl.deleteVertexArray(vao);
268         vao=null;
269     }
270
271     if(vbo)
272     {
273         gl.deleteBuffer(vbo);
274         vbo=null;
275     }
276
277     if(shaderProgramObject)
278     {
279         if(fragmentShaderObject)
280         {
281             gl.detachShader(shaderProgramObject,fragmentShaderObject);
282             gl.deleteShader(fragmentShaderObject);
283             fragmentShaderObject=null;
284         }
285
286         if(vertexShaderObject)
287         {
288             gl.detachShader(shaderProgramObject,vertexShaderObject);
289             gl.deleteShader(vertexShaderObject);
290             vertexShaderObject=null;
291         }
292
293         gl.deleteProgram(shaderProgramObject);
294         shaderProgramObject=null;
295     }
296 }
297
298 function keyDown(event)
299 {
300     // code
301     switch(event.keyCode)
302     {
```

```
303         case 27: // Escape
304             // uninitialize
305             uninitialize();
306             // close our application's tab
307             window.close(); // may not work in Firefox but works in Safari and
                  chrome
308             break;
309         case 70: // for 'F' or 'f'
310             toggleFullScreen();
311             break;
312     }
313 }
314
315 function mouseDown()
316 {
317     // code
318 }
319
```