```
// 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 light_ambient=[0.0,0.0,0.0];
var light_diffuse=[1.0,1.0,1.0];
var light_specular=[1.0,1.0,1.0];
var light_position=[100.0,100.0,100.0,1.0];

var material_ambient= [0.0,0.0,0.0];
var material_diffuse= [1.0,1.0,1.0];
var material_specular= [1.0,1.0,1.0];
var material_shininess= 50.0;

var sphere=null;

var perspectiveProjectionMatrix;

var modelMatrixUniform, viewMatrixUniform, projectionMatrixUniform;
var laUniform, ldUniform, lsUniform, lightPositionUniform;
var kaUniform, kdUniform, ksUniform, materialShininessUniform;
var LKeyPressedUniform;

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
```

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

```
102             canvas.msRequestFullscreen();
103         bFullscreen=true;
104     }
105     else // if already fullscreen
106     {
107         if(document.exitFullscreen)
108             document.exitFullscreen();
109         else if(document.mozCancelFullScreen)
110             document.mozCancelFullScreen();
111         else if(document.webkitExitFullscreen)
112             document.webkitExitFullscreen();
113         else if(document.msExitFullscreen)
114             document.msExitFullscreen();
115         bFullscreen=false;
116     }
117 }
118
119 function init()
120 {
121     // code
122     // get WebGL 2.0 context
123     gl = canvas.getContext("webgl2");
124     if(gl==null) // failed to get context
125     {
126         console.log("Failed to get the rendering context for WebGL");
127         return;
128     }
129     gl.viewportWidth = canvas.width;
130     gl.viewportHeight = canvas.height;
131
132     // vertex shader
133     var vertexShaderSourceCode=
134     "#version 300 es"+
135     "\n"+
136     "in vec4 vPosition;"+
137     "in vec3 vNormal;"+
138     "uniform mat4 u_model_matrix;"+
139     "uniform mat4 u_view_matrix;"+
140     "uniform mat4 u_projection_matrix;"+
141     "uniform mediump int u_LKeyPressed;"+
142     "uniform vec3 u_La;"+
143     "uniform vec3 u_Ld;"+
144     "uniform vec3 u_Ls;"+
145     "uniform vec4 u_light_position;"+
146     "uniform vec3 u_Ka;"+
147     "uniform vec3 u_Kd;"+
148     "uniform vec3 u_Ks;"+
149     "uniform float u_material_shininess;"+
150     "out vec3 phong_ads_color;"+
151     "void main(void)"+
152     "{"+
153     "if(u_LKeyPressed == 1)"+
```

```
154     "{"+
155     "vec4 eye_coordinates=u_view_matrix * u_model_matrix * vPosition;"+
156     "vec3 transformed_normals=normalize(mat3(u_view_matrix * u_model_matrix) *
          vNormal);"+
157     "vec3 light_direction = normalize(vec3(u_light_position) -
          eye_coordinates.xyz);"+
158     "float tn_dot_ld = max(dot(transformed_normals, light_direction),0.0);"+
159     "vec3 ambient = u_La * u_Ka;"+
160     "vec3 diffuse = u_Ld * u_Kd * tn_dot_ld;"+
161     "vec3 reflection_vector = reflect(-light_direction, transformed_normals);"+
162     "vec3 viewer_vector = normalize(-eye_coordinates.xyz);"+
163     "vec3 specular = u_Ls * u_Ks * pow(max(dot(reflection_vector, viewer_vector),
          0.0), u_material_shininess);"+
164     "phong_ads_color=ambient + diffuse + specular;"+
165     "}"+
166     "else"+
167     "{"+
168     "phong_ads_color = vec3(1.0, 1.0, 1.0);"+
169     "}"+
170     "gl_Position=u_projection_matrix * u_view_matrix * u_model_matrix *
          vPosition;"+
171     "}";
172
173     vertexShaderObject=gl.createShader(gl.VERTEX_SHADER);
174     gl.shaderSource(vertexShaderObject,vertexShaderSourceCode);
175     gl.compileShader(vertexShaderObject);
176     if(gl.getShaderParameter(vertexShaderObject,gl.COMPILE_STATUS)==false)
177     {
178         var error=gl.getShaderInfoLog(vertexShaderObject);
179         if(error.length > 0)
180         {
181             alert(error);
182             uninitialize();
183         }
184     }
185
186     // fragment shader
187     var fragmentShaderSourceCode=
188     "#version 300 es"+
189     "\n"+
190     "precision highp float;"+
191     "in vec3 phong_ads_color;"+
192     "out vec4 FragColor;"+
193     "void main(void)"+
194     "{"+
195     "FragColor = vec4(phong_ads_color, 1.0);"+
196     "}";
197
198     fragmentShaderObject=gl.createShader(gl.FRAGMENT_SHADER);
199     gl.shaderSource(fragmentShaderObject,fragmentShaderSourceCode);
200     gl.compileShader(fragmentShaderObject);
201     if(gl.getShaderParameter(fragmentShaderObject,gl.COMPILE_STATUS)==false)
```

```
202     {
203         var error=gl.getShaderInfoLog(fragmentShaderObject);
204         if(error.length > 0)
205         {
206             alert(error);
207             uninitialize();
208         }
209     }
210
211     // shader program
212     shaderProgramObject=gl.createProgram();
213     gl.attachShader(shaderProgramObject,vertexShaderObject);
214     gl.attachShader(shaderProgramObject,fragmentShaderObject);
215
216     // pre-link binding of shader program object with vertex shader attributes
217     gl.bindAttribLocation
          (shaderProgramObject,WebGLMacros.VDG_ATTRIBUTE_VERTEX,"vPosition");
218     gl.bindAttribLocation
          (shaderProgramObject,WebGLMacros.VDG_ATTRIBUTE_NORMAL,"vNormal");
219
220     // linking
221     gl.linkProgram(shaderProgramObject);
222     if (!gl.getProgramParameter(shaderProgramObject, gl.LINK_STATUS))
223     {
224         var error=gl.getProgramInfoLog(shaderProgramObject);
225         if(error.length > 0)
226         {
227             alert(error);
228             uninitialize();
229         }
230     }
231
232     // get Model Matrix uniform location
233     modelMatrixUniform=gl.getUniformLocation
          (shaderProgramObject,"u_model_matrix");
234     // get View Matrix uniform location
235     viewMatrixUniform=gl.getUniformLocation(shaderProgramObject,"u_view_matrix");
236     // get Projection Matrix uniform location
237     projectionMatrixUniform=gl.getUniformLocation
          (shaderProgramObject,"u_projection_matrix");
238
239     // get single tap detecting uniform
240     LKeyPressedUniform=gl.getUniformLocation
          (shaderProgramObject,"u_LKeyPressed");
241
242     // ambient color intensity of light
243     laUniform=gl.getUniformLocation(shaderProgramObject,"u_La");
244     // diffuse color intensity of light
245     ldUniform=gl.getUniformLocation(shaderProgramObject,"u_Ld");
246     // specular color intensity of light
247     lsUniform=gl.getUniformLocation(shaderProgramObject,"u_Ls");
248     // position of light
```

```
249     lightPositionUniform=gl.getUniformLocation
          (shaderProgramObject,"u_light_position");
250
251     // ambient reflective color intensity of material
252     kaUniform=gl.getUniformLocation(shaderProgramObject,"u_Ka");
253     // diffuse reflective color intensity of material
254     kdUniform=gl.getUniformLocation(shaderProgramObject,"u_Kd");
255     // specular reflective color intensity of material
256     ksUniform=gl.getUniformLocation(shaderProgramObject,"u_Ks");
257     // shininess of material ( value is conventionally between 1 to 200 )
258     materialShininessUniform=gl.getUniformLocation
          (shaderProgramObject,"u_material_shininess");
259
260     // *** vertices, colors, shader attribs, vbo, vao initializations ***
261     sphere=new Mesh();
262     makeSphere(sphere,2.0,30,30);
263
264     // Depth test will always be enabled
265     gl.enable(gl.DEPTH_TEST);
266
267     // depth test to do
268     gl.depthFunc(gl.LEQUAL);
269
270     // We will always cull back faces for better performance
271     gl.enable(gl.CULL_FACE);
272
273     // set clear color
274     gl.clearColor(0.0, 0.0, 0.0, 1.0); // black
275
276     // initialize projection matrix
277     perspectiveProjectionMatrix=mat4.create();
278 }
279
280 function resize()
281 {
282     // code
283     if(bFullscreen==true)
284     {
285         canvas.width=window.innerWidth;
286         canvas.height=window.innerHeight;
287     }
288     else
289     {
290         canvas.width=canvas_original_width;
291         canvas.height=canvas_original_height;
292     }
293
294     // set the viewport to match
295     gl.viewport(0, 0, canvas.width, canvas.height);
296
297     mat4.perspective(perspectiveProjectionMatrix, 45.0, parseFloat(canvas.width)/
          parseFloat(canvas.height), 0.1, 100.0);
```

```
298 }
299
300 function draw()
301 {
302     // code
303     gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
304
305     gl.useProgram(shaderProgramObject);
306
307     if(bLKeyPressed==true)
308     {
309         gl.uniform1i(LKeyPressedUniform, 1);
310
311         // setting light properties
312         gl.uniform3fv(laUniform, light_ambient); // ambient intensity of light
313         gl.uniform3fv(ldUniform, light_diffuse); // diffuse intensity of light
314         gl.uniform3fv(lsUniform, light_specular); // specular intensity of light
315         gl.uniform4fv(lightPositionUniform, light_position); // light position
316
317         // setting material properties
318         gl.uniform3fv(kaUniform, material_ambient); // ambient reflectivity of
              material
319         gl.uniform3fv(kdUniform, material_diffuse); // diffuse reflectivity of
              material
320         gl.uniform3fv(ksUniform, material_specular); // specular reflectivity of
              material
321         gl.uniform1f(materialShininessUniform, material_shininess); // material
              shininess
322     }
323     else
324     {
325         gl.uniform1i(LKeyPressedUniform, 0);
326     }
327
328     var modelMatrix=mat4.create();
329     var viewMatrix=mat4.create();
330
331     mat4.translate(modelMatrix, modelMatrix, [0.0,0.0,-6.0]);
332
333     gl.uniformMatrix4fv(modelMatrixUniform,false,modelMatrix);
334     gl.uniformMatrix4fv(viewMatrixUniform,false,viewMatrix);
335     gl.uniformMatrix4fv
        (projectionMatrixUniform,false,perspectiveProjectionMatrix);
336
337     sphere.draw();
338
339     gl.useProgram(null);
340
341     // animation loop
342     requestAnimationFrame(draw, canvas);
343 }
344
```

```
345 function uninitialize()
346 {
347     // code
348     if(sphere)
349     {
350         sphere.deallocate();
351         sphere=null;
352     }
353
354     if(shaderProgramObject)
355     {
356         if(fragmentShaderObject)
357         {
358             gl.detachShader(shaderProgramObject,fragmentShaderObject);
359             gl.deleteShader(fragmentShaderObject);
360             fragmentShaderObject=null;
361         }
362
363         if(vertexShaderObject)
364         {
365             gl.detachShader(shaderProgramObject,vertexShaderObject);
366             gl.deleteShader(vertexShaderObject);
367             vertexShaderObject=null;
368         }
369
370         gl.deleteProgram(shaderProgramObject);
371         shaderProgramObject=null;
372     }
373 }
374
375 function keyDown(event)
376 {
377     // code
378     switch(event.keyCode)
379     {
380         case 27: // Escape
381             // uninitialize
382             uninitialize();
383             // close our application's tab
384             window.close(); // may not work in Firefox but works in Safari and
                chrome
385             break;
386         case 76: // for 'L' or 'l'
387             if(bLKeyPressed==false)
388                 bLKeyPressed=true;
389             else
390                 bLKeyPressed=false;
391             break;
392         case 70: // for 'F' or 'f'
393             toggleFullScreen();
394             break;
395     }
```

```
396 }
397
398 function mouseDown()
399 {
400     // code
401 }
402
```