```
// 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 vec4 u_light_position;"+
143     "out vec3 transformed_normals;"+
144     "out vec3 light_direction;"+
145     "out vec3 viewer_vector;"+
146     "void main(void)"+
147     "{"+
148     "if(u_LKeyPressed == 1)"+
149     "{"+
150     "vec4 eye_coordinates=u_view_matrix * u_model_matrix * vPosition;"+
151     "transformed_normals=mat3(u_view_matrix * u_model_matrix) * vNormal;"+
152     "light_direction = vec3(u_light_position) - eye_coordinates.xyz;"+
153     "viewer_vector = -eye_coordinates.xyz;"+
```

```
154     "}"+
155     "gl_Position=u_projection_matrix * u_view_matrix * u_model_matrix *
          vPosition;"+
156     "}";
157
158     vertexShaderObject=gl.createShader(gl.VERTEX_SHADER);
159     gl.shaderSource(vertexShaderObject,vertexShaderSourceCode);
160     gl.compileShader(vertexShaderObject);
161     if(gl.getShaderParameter(vertexShaderObject,gl.COMPILE_STATUS)==false)
162     {
163         var error=gl.getShaderInfoLog(vertexShaderObject);
164         if(error.length > 0)
165         {
166             alert(error);
167             uninitialize();
168         }
169     }
170
171     // fragment shader
172     var fragmentShaderSourceCode=
173     "#version 300 es"+
174     "\n"+
175     "precision highp float;"+
176     "in vec3 transformed_normals;"+
177     "in vec3 light_direction;"+
178     "in vec3 viewer_vector;"+
179     "out vec4 FragColor;"+
180     "uniform vec3 u_La;"+
181     "uniform vec3 u_Ld;"+
182     "uniform vec3 u_Ls;"+
183     "uniform vec3 u_Ka;"+
184     "uniform vec3 u_Kd;"+
185     "uniform vec3 u_Ks;"+
186     "uniform float u_material_shininess;"+
187     "uniform int u_LKeyPressed;"+
188     "void main(void)"+
189     "{"+
190     "vec3 phong_ads_color;"+
191     "if(u_LKeyPressed == 1)"+
192     "{"+
193     "vec3 normalized_transformed_normals=normalize(transformed_normals);"+
194     "vec3 normalized_light_direction=normalize(light_direction);"+
195     "vec3 normalized_viewer_vector=normalize(viewer_vector);"+
196     "vec3 ambient = u_La * u_Ka;"+
197     "float tn_dot_ld = max(dot(normalized_transformed_normals,
          normalized_light_direction),0.0);"+
198     "vec3 diffuse = u_Ld * u_Kd * tn_dot_ld;"+
199     "vec3 reflection_vector = reflect(-normalized_light_direction,
          normalized_transformed_normals);"+
200     "vec3 specular = u_Ls * u_Ks * pow(max(dot(reflection_vector,
          normalized_viewer_vector), 0.0), u_material_shininess);"+
201     "phong_ads_color=ambient + diffuse + specular;"+
```

```
202     "}"+
203     "else"+
204     "{"+
205     "phong_ads_color = vec3(1.0, 1.0, 1.0);"+
206     "}"+
207     "FragColor = vec4(phong_ads_color, 1.0);"+
208     "}"
209
210     fragmentShaderObject=gl.createShader(gl.FRAGMENT_SHADER);
211     gl.shaderSource(fragmentShaderObject,fragmentShaderSourceCode);
212     gl.compileShader(fragmentShaderObject);
213     if(gl.getShaderParameter(fragmentShaderObject,gl.COMPILE_STATUS)==false)
214     {
215         var error=gl.getShaderInfoLog(fragmentShaderObject);
216         if(error.length > 0)
217         {
218             alert(error);
219             uninitialize();
220         }
221     }
222
223     // shader program
224     shaderProgramObject=gl.createProgram();
225     gl.attachShader(shaderProgramObject,vertexShaderObject);
226     gl.attachShader(shaderProgramObject,fragmentShaderObject);
227
228     // pre-link binding of shader program object with vertex shader attributes
229     gl.bindAttribLocation
          (shaderProgramObject,WebGLMacros.VDG_ATTRIBUTE_VERTEX,"vPosition");
230     gl.bindAttribLocation
          (shaderProgramObject,WebGLMacros.VDG_ATTRIBUTE_NORMAL,"vNormal");
231
232     // linking
233     gl.linkProgram(shaderProgramObject);
234     if (!gl.getProgramParameter(shaderProgramObject, gl.LINK_STATUS))
235     {
236         var error=gl.getProgramInfoLog(shaderProgramObject);
237         if(error.length > 0)
238         {
239             alert(error);
240             uninitialize();
241         }
242     }
243
244     // get Model Matrix uniform location
245     modelMatrixUniform=gl.getUniformLocation
          (shaderProgramObject,"u_model_matrix");
246     // get View Matrix uniform location
247     viewMatrixUniform=gl.getUniformLocation(shaderProgramObject,"u_view_matrix");
248     // get Projection Matrix uniform location
249     projectionMatrixUniform=gl.getUniformLocation
          (shaderProgramObject,"u_projection_matrix");
```

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

```
299     }
300     else
301     {
302         canvas.width=canvas_original_width;
303         canvas.height=canvas_original_height;
304     }
305
306     // set the viewport to match
307     gl.viewport(0, 0, canvas.width, canvas.height);
308
309     mat4.perspective(perspectiveProjectionMatrix, 45.0, parseFloat(canvas.width)/
          parseFloat(canvas.height), 0.1, 100.0);
310 }
311
312 function draw()
313 {
314     // code
315     gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
316
317     gl.useProgram(shaderProgramObject);
318
319     if(bLKeyPressed==true)
320     {
321         gl.uniform1i(LKeyPressedUniform, 1);
322
323         // setting light properties
324         gl.uniform3fv(laUniform, light_ambient); // ambient intensity of light
325         gl.uniform3fv(ldUniform, light_diffuse); // diffuse intensity of light
326         gl.uniform3fv(lsUniform, light_specular); // specular intensity of light
327         gl.uniform4fv(lightPositionUniform, light_position); // light position
328
329         // setting material properties
330         gl.uniform3fv(kaUniform, material_ambient); // ambient reflectivity of
              material
331         gl.uniform3fv(kdUniform, material_diffuse); // diffuse reflectivity of
              material
332         gl.uniform3fv(ksUniform, material_specular); // specular reflectivity of
              material
333         gl.uniform1f(materialShininessUniform, material_shininess); // material
              shininess
334     }
335     else
336     {
337         gl.uniform1i(LKeyPressedUniform, 0);
338     }
339
340     var modelMatrix=mat4.create();
341     var viewMatrix=mat4.create();
342
343     mat4.translate(modelMatrix, modelMatrix, [0.0,0.0,-6.0]);
344
345     gl.uniformMatrix4fv(modelMatrixUniform,false,modelMatrix);
```

```
346     gl.uniformMatrix4fv(viewMatrixUniform,false,viewMatrix);
347     gl.uniformMatrix4fv
          (projectionMatrixUniform,false,perspectiveProjectionMatrix);
348
349     sphere.draw();
350
351     gl.useProgram(null);
352
353     // animation loop
354     requestAnimationFrame(draw, canvas);
355 }
356
357 function uninitialize()
358 {
359     // code
360     if(sphere)
361     {
362         sphere.deallocate();
363         sphere=null;
364     }
365
366     if(shaderProgramObject)
367     {
368         if(fragmentShaderObject)
369         {
370             gl.detachShader(shaderProgramObject,fragmentShaderObject);
371             gl.deleteShader(fragmentShaderObject);
372             fragmentShaderObject=null;
373         }
374
375         if(vertexShaderObject)
376         {
377             gl.detachShader(shaderProgramObject,vertexShaderObject);
378             gl.deleteShader(vertexShaderObject);
379             vertexShaderObject=null;
380         }
381
382         gl.deleteProgram(shaderProgramObject);
383         shaderProgramObject=null;
384     }
385 }
386
387 function keyDown(event)
388 {
389     // code
390     switch(event.keyCode)
391     {
392         case 27: // Escape
393             // uninitialize
394             uninitialize();
395             // close our application's tab
396             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
}

```