

```
1 // global variables
2 var canvas=null;
3 var gl=null; // webgl context
4 var bFullscreen=false;
5 var canvas_original_width;
6 var canvas_original_height;
7
8 const WebGLMacros= // when whole 'WebGLMacros' is 'const', all inside it are automatically 'const'
9 {
10 VDG_ATTRIBUTE_VERTEX:0,
11 VDG_ATTRIBUTE_COLOR:1,
12 VDG_ATTRIBUTE_NORMAL:2,
13 VDG_ATTRIBUTE_TEXTURE0:3,
14 };
15
16 var vertexShaderObject;
17 var fragmentShaderObject;
18 var shaderProgramObject;
19
20 var vao_cube;
21 var vbo_cube_position;
22 var vbo_cube_normal;
23
24 var perspectiveProjectionMatrix;
25
26 var modelViewMatrixUniform, projectionMatrixUniform;
27 var ldUniform, kdUniform, lightPositionUniform;
28 var LKeyPressedUniform;
29
30 var angleCube=0.0;
31
32 var bLKeyPressed=false;
33
34 // To start animation : To have requestAnimationFrame() to be called "cross-
    browser" compatible
35 var requestAnimationFrame =
36 window.requestAnimationFrame ||
37 window.webkitRequestAnimationFrame ||
38 window.mozRequestAnimationFrame ||
39 window.oRequestAnimationFrame ||
40 window.msRequestAnimationFrame;
41
42 // To stop animation : To have cancelAnimationFrame() to be called "cross-
    browser" compatible
43 var cancelAnimationFrame =
44 window.cancelAnimationFrame ||
45 window.webkitCancelRequestAnimationFrame || window.webkitCancelAnimationFrame ||
46 window.mozCancelRequestAnimationFrame || window.mozCancelAnimationFrame ||
47 window.oCancelRequestAnimationFrame || window.oCancelAnimationFrame ||
48 window.msCancelRequestAnimationFrame || window.msCancelAnimationFrame;
49
```

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

```
102     else if(document.mozCancelFullScreen)
103         document.mozCancelFullScreen();
104     else if(document.webkitExitFullscreen)
105         document.webkitExitFullscreen();
106     else if(document.msExitFullscreen)
107         document.msExitFullscreen();
108     bFullscreen=false;
109 }
110 }
111
112 function init()
113 {
114     // code
115     // get WebGL 2.0 context
116     gl = canvas.getContext("webgl2");
117     if(gl==null) // failed to get context
118     {
119         console.log("Failed to get the rendering context for WebGL");
120         return;
121     }
122     gl.viewportWidth = canvas.width;
123     gl.viewportHeight = canvas.height;
124
125     // vertex shader
126     var vertexShaderSourceCode=
127     "#version 300 es"+
128     "\n"+
129     "in vec4 vPosition;" +
130     "in vec3 vNormal;" +
131     "uniform mat4 u_model_view_matrix;" +
132     "uniform mat4 u_projection_matrix;" +
133     "uniform mediump int u_LKeyPressed;" +
134     "uniform vec3 u_Ld;" +
135     "uniform vec3 u_Kd;" +
136     "uniform vec4 u_light_position;" +
137     "out vec3 diffuse_light;" +
138     "void main(void)" +
139     "{" +
140     "if(u_LKeyPressed == 1)" +
141     "{" +
142     "vec4 eyeCoordinates=u_model_view_matrix * vPosition;" +
143     "vec3 tnorm = normalize(mat3(u_model_view_matrix) * vNormal);" +
144     "vec3 s = normalize(vec3(u_light_position - eyeCoordinates));" +
145     "diffuse_light = u_Ld * u_Kd * max( dot( s, tnorm ), 0.0 );" +
146     "}" +
147     "gl_Position=u_projection_matrix * u_model_view_matrix * vPosition;" +
148     "}";
149
150     vertexShaderObject=gl.createShader(gl.VERTEX_SHADER);
151     gl.shaderSource(vertexShaderObject,vertexShaderSourceCode);
152     gl.compileShader(vertexShaderObject);
153     if(gl.getShaderParameter(vertexShaderObject,gl.COMPILE_STATUS)==false)
```



```
154     {
155         var error=gl.getShaderInfoLog(vertexShaderObject);
156         if(error.length > 0)
157         {
158             alert(error);
159             uninitialized();
160         }
161     }
162
163     // fragment shader
164     var fragmentShaderSourceCode=
165     "#version 300 es"+
166     "\n"+
167     "precision highp float;"+
168     "in vec3 diffuse_light;"+
169     "out vec4 FragColor;"+
170     "uniform int u_LKeyPressed;"+
171     "void main(void)"+
172     "{"+
173     "vec4 color;"+
174     "if (u_LKeyPressed == 1)"+
175     "{"+
176     "color = vec4(diffuse_light,1.0);"+
177     "}"+"+
178     "else"+
179     "{"+
180     "color = vec4(1.0, 1.0, 1.0, 1.0);"+
181     "}"+"+
182     "FragColor = color;"+
183     "}";
184
185     fragmentShaderObject=gl.createShader(gl.FRAGMENT_SHADER);
186     gl.shaderSource(fragmentShaderObject,fragmentShaderSourceCode);
187     gl.compileShader(fragmentShaderObject);
188     if(gl.getShaderParameter(fragmentShaderObject,gl.COMPILE_STATUS)==false)
189     {
190         var error=gl.getShaderInfoLog(fragmentShaderObject);
191         if(error.length > 0)
192         {
193             alert(error);
194             uninitialized();
195         }
196     }
197
198     // shader program
199     shaderProgramObject=gl.createProgram();
200     gl.attachShader(shaderProgramObject,vertexShaderObject);
201     gl.attachShader(shaderProgramObject,fragmentShaderObject);
202
203     // pre-link binding of shader program object with vertex shader attributes
204     gl.bindAttribLocation
        (shaderProgramObject,WebGLMacros.VDG_ATTRIBUTE_VERTEX,"vPosition");
```

```
205     gl.bindAttribLocation
        (shaderProgramObject, WebGLMacros.VDG_ATTRIBUTE_NORMAL, "vNormal");
206
207     // linking
208     gl.linkProgram(shaderProgramObject);
209     if (!gl.getProgramParameter(shaderProgramObject, gl.LINK_STATUS))
210     {
211         var error=gl.getProgramInfoLog(shaderProgramObject);
212         if(error.length > 0)
213         {
214             alert(error);
215             uninitialized();
216         }
217     }
218
219     // get Model View Matrix uniform location
220     modelViewMatrixUniform=gl.getUniformLocation
        (shaderProgramObject, "u_model_view_matrix");
221     // get Projection Matrix uniform location
222     projectionMatrixUniform=gl.getUniformLocation
        (shaderProgramObject, "u_projection_matrix");
223
224     // get single tap detecting uniform
225     LKeyPressedUniform=gl.getUniformLocation
        (shaderProgramObject, "u_LKeyPressed");
226
227     // diffuse color intensity of light
228     ldUniform=gl.getUniformLocation(shaderProgramObject, "u_Ld");
229     // diffuse reflective color intensity of material
230     kdUniform=gl.getUniformLocation(shaderProgramObject, "u_Kd");
231     // position of light
232     lightPositionUniform=gl.getUniformLocation
        (shaderProgramObject, "u_light_position");
233
234     // *** vertices, colors, shader attribs, vbo, vao initializations ***
235     var cubeVertices=new Float32Array([
236         // top surface
237         1.0, 1.0, -1.0, // top-right of top
238         -1.0, 1.0, -1.0, // top-left of top
239         -1.0, 1.0, 1.0, // bottom-left of top
240         1.0, 1.0, 1.0, // bottom-right of top
241
242         // bottom surface
243         1.0, -1.0, 1.0, // top-right of bottom
244         -1.0, -1.0, 1.0, // top-left of bottom
245         -1.0, -1.0, -1.0, // bottom-left of bottom
246         1.0, -1.0, -1.0, // bottom-right of bottom
247
248         // front surface
249         1.0, 1.0, 1.0, // top-right of front
250         -1.0, 1.0, 1.0, // top-left of front
251         -1.0, -1.0, 1.0, // bottom-left of front
```

```
252         1.0,-1.0, 1.0, // bottom-right of front
253
254         // back surface
255         1.0,-1.0,-1.0, // top-right of back
256         -1.0,-1.0,-1.0, // top-left of back
257         -1.0, 1.0,-1.0, // bottom-left of back
258         1.0, 1.0,-1.0, // bottom-right of back
259
260         // left surface
261         -1.0, 1.0, 1.0, // top-right of left
262         -1.0, 1.0,-1.0, // top-left of left
263         -1.0,-1.0,-1.0, // bottom-left of left
264         -1.0,-1.0, 1.0, // bottom-right of left
265
266         // right surface
267         1.0, 1.0,-1.0, // top-right of right
268         1.0, 1.0, 1.0, // top-left of right
269         1.0,-1.0, 1.0, // bottom-left of right
270         1.0,-1.0,-1.0, // bottom-right of right
271     ]);
272
273     var cubeNormals=new Float32Array([
274         // top
275         0.0, 1.0, 0.0,
276         0.0, 1.0, 0.0,
277         0.0, 1.0, 0.0,
278         0.0, 1.0, 0.0,
279
280         // bottom
281         0.0, -1.0, 0.0,
282         0.0, -1.0, 0.0,
283         0.0, -1.0, 0.0,
284         0.0, -1.0, 0.0,
285
286         // front
287         0.0, 0.0, 1.0,
288         0.0, 0.0, 1.0,
289         0.0, 0.0, 1.0,
290         0.0, 0.0, 1.0,
291
292         // back
293         0.0, 0.0, -1.0,
294         0.0, 0.0, -1.0,
295         0.0, 0.0, -1.0,
296         0.0, 0.0, -1.0,
297
298         // left
299         -1.0, 0.0, 0.0,
300         -1.0, 0.0, 0.0,
301         -1.0, 0.0, 0.0,
302         -1.0, 0.0, 0.0,
303
```



```
304                                     // right
305                                     1.0, 0.0, 0.0,
306                                     1.0, 0.0, 0.0,
307                                     1.0, 0.0, 0.0,
308                                     1.0, 0.0, 0.0
309                                     ]);
310
311     vao_cube=gl.createVertexArray();
312     gl.bindVertexArray(vao_cube);
313
314     vbo_cube_position = gl.createBuffer();
315     gl.bindBuffer(gl.ARRAY_BUFFER,vbo_cube_position);
316     gl.bufferData(gl.ARRAY_BUFFER,cubeVertices,gl.STATIC_DRAW);
317     gl.vertexAttribPointer(WebGLMacros.VDG_ATTRIBUTE_VERTEX,
318                             3, // 3 is for X,Y,Z co-ordinates in our Vertices
                                array
319                             gl.FLOAT,
320                             false,0,0);
321     gl.enableVertexAttribArray(WebGLMacros.VDG_ATTRIBUTE_VERTEX);
322     gl.bindBuffer(gl.ARRAY_BUFFER,null);
323
324     vbo_cube_normal = gl.createBuffer();
325     gl.bindBuffer(gl.ARRAY_BUFFER,vbo_cube_normal);
326     gl.bufferData(gl.ARRAY_BUFFER,cubeNormals,gl.STATIC_DRAW);
327     gl.vertexAttribPointer(WebGLMacros.VDG_ATTRIBUTE_NORMAL,
328                             3, // 3 is for X,Y,Z co-ordinates in our Normals array
                                array
329                             gl.FLOAT,
330                             false,0,0);
331     gl.enableVertexAttribArray(WebGLMacros.VDG_ATTRIBUTE_NORMAL);
332     gl.bindBuffer(gl.ARRAY_BUFFER,null);
333
334     gl.bindVertexArray(null);
335
336     // set clear color
337     gl.clearColor(0.0, 0.0, 0.0, 1.0); // black
338
339     // Depth test will always be enabled
340     gl.enable(gl.DEPTH_TEST);
341
342     // depth test to do
343     gl.depthFunc(gl.LEQUAL);
344
345     // We will always cull back faces for better performance
346     gl.enable(gl.CULL_FACE);
347
348     // initialize projection matrix
349     perspectiveProjectionMatrix=mat4.create();
350 }
351
352 function resize()
353 {
354     // code
```

```
355     if(bFullscreen==true)
356     {
357         canvas.width=window.innerWidth;
358         canvas.height=window.innerHeight;
359     }
360     else
361     {
362         canvas.width=canvas_original_width;
363         canvas.height=canvas_original_height;
364     }
365
366     // set the viewport to match
367     gl.viewport(0, 0, canvas.width, canvas.height);
368
369     mat4.perspective(perspectiveProjectionMatrix, 45.0, parseFloat(canvas.width)/
        parseFloat(canvas.height), 0.1, 100.0);
370 }
371
372 function draw()
373 {
374     // code
375     gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
376
377     gl.useProgram(shaderProgramObject);
378
379     if(bLKeyPressed==true)
380     {
381         gl.uniform1i(LKeyPressedUniform, 1);
382
383         // setting light properties
384         gl.uniform3f(ldUniform, 1.0, 1.0, 1.0); // diffuse intensity of light
385         // setting material properties
386         gl.uniform3f(kdUniform, 0.5, 0.5, 0.5); // diffuse reflectivity of
            material
387         var lightPosition = [0.0, 0.0, 2.0, 1.0];
388         gl.uniform4fv(lightPositionUniform, lightPosition); // light position
389     }
390     else
391     {
392         gl.uniform1i(LKeyPressedUniform, 0);
393     }
394
395     var modelViewMatrix=mat4.create(); // itself creates identity matrix
396
397     mat4.translate(modelViewMatrix, modelViewMatrix, [0.0,0.0,-4.0]);
398
399     mat4.rotateX(modelViewMatrix ,modelViewMatrix, degToRad(angleCube));
400     mat4.rotateY(modelViewMatrix ,modelViewMatrix, degToRad(angleCube));
401     mat4.rotateZ(modelViewMatrix ,modelViewMatrix, degToRad(angleCube));
402
403     gl.uniformMatrix4fv(modelViewMatrixUniform,false,modelViewMatrix);
404     gl.uniformMatrix4fv
```



```
(projectionMatrixUniform,false,perspectiveProjectionMatrix);

405
406 gl.bindVertexArray(vao_cube);
407
408 // *** draw, either by glDrawTriangles() or glDrawArrays() or glDrawElements ↗
409 ()
410 // actually 2 triangles make 1 cube, so there should be 6 vertices,
411 // but as 2 tringles while making square meet each other at diagonal,
412 // 2 of 6 vertices are common to both triangles, and hence 6-2=4
413 gl.drawArrays(gl.TRIANGLE_FAN,0,4);
414 gl.drawArrays(gl.TRIANGLE_FAN,4,4);
415 gl.drawArrays(gl.TRIANGLE_FAN,8,4);
416 gl.drawArrays(gl.TRIANGLE_FAN,12,4);
417 gl.drawArrays(gl.TRIANGLE_FAN,16,4);
418 gl.drawArrays(gl.TRIANGLE_FAN,20,4);
419
420 gl.bindVertexArray(null);
421
422 gl.useProgram(null);
423
424 angleCube=angleCube+2.0;
425 if(angleCube>=360.0)
426     angleCube=angleCube-360.0;
427
428 // animation loop
429 requestAnimationFrame(draw, canvas);
430 }
431
432 function uninitialized()
433 {
434     // code
435     if(vao_cube)
436     {
437         gl.deleteVertexArray(vao_cube);
438         vao_cube=null;
439     }
440
441     if(vbo_cube_normal)
442     {
443         gl.deleteBuffer(vbo_cube_normal);
444         vbo_cube_normal=null;
445     }
446
447     if(vbo_cube_position)
448     {
449         gl.deleteBuffer(vbo_cube_position);
450         vbo_cube_position=null;
451     }
452
453     if(shaderProgramObject)
454     {
455         if(fragmentShaderObject)
```

```
455     {
456         gl.detachShader(shaderProgramObject, fragmentShaderObject);
457         gl.deleteShader(fragmentShaderObject);
458         fragmentShaderObject=null;
459     }
460
461     if(vertexShaderObject)
462     {
463         gl.detachShader(shaderProgramObject, vertexShaderObject);
464         gl.deleteShader(vertexShaderObject);
465         vertexShaderObject=null;
466     }
467
468     gl.deleteProgram(shaderProgramObject);
469     shaderProgramObject=null;
470 }
471 }
472
473 function keyDown(event)
474 {
475     // code
476     switch(event.keyCode)
477     {
478         case 27: // Escape
479             // uninitialized
480             uninitialized();
481             // close our application's tab
482             window.close(); // may not work in Firefox but works in Safari and chrome
483             break;
484         case 76: // for 'L' or 'l'
485             if(bLKeyPressed==false)
486                 bLKeyPressed=true;
487             else
488                 bLKeyPressed=false;
489             break;
490         case 70: // for 'F' or 'f'
491             toggleFullScreen();
492             break;
493     }
494 }
495
496 function mouseDown()
497 {
498     // code
499 }
500
501 function degToRad(degrees)
502 {
503     // code
504     return(degrees * Math.PI / 180);
505 }
```