```
/************************************************
 *
 *             Example Six
 *
 *  This example illustrates how different
 *  lighting and shading models can be
 *  implemented in shader programs
 *
 ************************************************/

#include <Windows.h>
#include <gl/glew.h>
#define GLFW_DLL
#define GLFW_INCLUDE_NONE
#include <GLFW/glfw3.h>
#define GLM_FORCE_RADIANS
#include <glm/glm.hpp>
#include <glm/gtc/matrix_transform.hpp>
#include <glm/gtc/type_ptr.hpp>
#include "shaders.h"
#include <stdio.h>
#include "tiny_obj_loader.h"
#include <iostream>

GLuint program;         // shader programs
GLuint objVAO;          // the data to be displayed
int triangles;          // number of triangles
int window;

char *vertexName;
char *fragmentName;

double theta, phi;
double r;

float angle = 0.0;
float counter = 0.0;

float red = 2.0;
float green = 0.0;
float blue = 0.0;

float cx, cy, cz;


glm::mat4 projection;   // projection matrix
float eyex, eyey, eyez; // eye position

/*
```

```
50   *  The init procedure creates the OpenGL data structures
51   *  that contain the triangle geometry, compiles our
52   *  shader program and links the shader programs to
53   *  the data.
54   */
55
56  void init() {
57      GLuint vbuffer;
58      GLuint ibuffer;
59      GLint vPosition;
60      GLint vNormal;
61      int vs;
62      int fs;
63      GLfloat *vertices;
64      GLfloat *normals;
65      GLuint *indices;
66      std::vector<tinyobj::shape_t> shapes;
67      std::vector<tinyobj::material_t> materials;
68      int nv;
69      int nn;
70      int ni;
71      int i;
72      float xmin, ymin, zmin;
73      float xmax, ymax, zmax;
74      char vname[256];
75      char fname[256];
76
77      glGenVertexArrays(1, &objVAO);
78      glBindVertexArray(objVAO);
79
80      /*  Load the obj file */
81      std::string err = tinyobj::LoadObj(shapes, materials, "vase.obj", 0);
82
83      if (!err.empty()) {
84          std::cerr << err << std::endl;
85          return;
86      }
87
88      /*  Retrieve the vertex coordinate data */
89      nv = shapes[0].mesh.positions.size();
90      vertices = new GLfloat[nv];
91      for(i=0; i<nv; i++) {
92          vertices[i] = shapes[0].mesh.positions[i];
93      }
94
95      /*
96       *  Find the range of the x, y and z
97       *  coordinates.
98       */
```

```
 99      xmin = ymin = zmin = 1000000.0;
100      xmax = ymax = zmax = -1000000.0;
101      for(i=0; i<nv/3; i++) {
102          if(vertices[3*i] < xmin)
103              xmin = vertices[3*i];
104          if(vertices[3*i] > xmax)
105              xmax = vertices[3*i];
106          if(vertices[3*i+1] < ymin)
107              ymin = vertices[3*i+1];
108          if(vertices[3*i+1] > ymax)
109              ymax = vertices[3*i+1];
110          if(vertices[3*i+2] < zmin)
111              zmin = vertices[3*i+2];
112          if(vertices[3*i+2] > zmax)
113              zmax = vertices[3*i+2];
114      }
115
116      /* compute center and print range */
117      cx = (xmin+xmax)/2.0f;
118      cy = (ymin+ymax)/2.0f;
119      cz = (zmin+zmax)/2.0f;
120      printf("X range: %f %f\n",xmin,xmax);
121      printf("Y range: %f %f\n",ymin,ymax);
122      printf("Z range: %f %f\n",zmin,zmax);
123      printf("center: %f %f %f\n",cx, cy,cz);
124
125      /*  Retrieve the vertex normals */
126      nn = shapes[0].mesh.normals.size();
127      normals = new GLfloat[nn];
128      for(i=0; i<nn; i++) {
129          normals[i] = shapes[0].mesh.normals[i];
130      }
131
132      /*  Retrieve the triangle indices */
133      ni = shapes[0].mesh.indices.size();
134      triangles = ni/3;
135      indices = new GLuint[ni];
136      for(i=0; i<ni; i++) {
137          indices[i] = shapes[0].mesh.indices[i];
138      }
139
140      /*
141       *  load the vertex coordinate data
142       */
143      glGenBuffers(1, &vbuffer);
144      glBindBuffer(GL_ARRAY_BUFFER, vbuffer);
145      glBufferData(GL_ARRAY_BUFFER, (nv+nn)*sizeof(GLfloat), NULL,
           GL_STATIC_DRAW);
146      glBufferSubData(GL_ARRAY_BUFFER, 0, nv*sizeof(GLfloat), vertices);
```

```
147     glBufferSubData(GL_ARRAY_BUFFER, nv*sizeof(GLfloat), nn*sizeof(GLfloat),
          normals);
148
149     /*
150      *  load the vertex indexes
151      */
152     glGenBuffers(1, &ibuffer);
153     glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, ibuffer);
154     glBufferData(GL_ELEMENT_ARRAY_BUFFER, ni*sizeof(GLuint), indices,
          GL_STATIC_DRAW);
155
156     /*
157      *  compile and build the shader program
158      */
159     sprintf(vname,"example6c.vs", vertexName);
160     sprintf(fname,"example6c.fs", fragmentName);
161     vs = buildShader(GL_VERTEX_SHADER, vname);
162     fs = buildShader(GL_FRAGMENT_SHADER, fname);
163     program = buildProgram(vs,fs,0);
164
165     /*
166      *  link the vertex coordinates to the vPosition
167      *  variable in the vertex program.  Do the same
168      *  for the normal vectors.
169      */
170     glUseProgram(program);
171     vPosition = glGetAttribLocation(program,"vPosition");
172     glVertexAttribPointer(vPosition, 3, GL_FLOAT, GL_FALSE, 0, 0);
173     glEnableVertexAttribArray(vPosition);
174     vNormal = glGetAttribLocation(program, "vNormal");
175     glVertexAttribPointer(vNormal, 3, GL_FLOAT, GL_FALSE, 0, (void*)
          (nv*sizeof(GL_FLOAT)));
176     glEnableVertexAttribArray(vNormal);
177
178 }
179
180 /*
181  *  Executed each time the window is resized,
182  *  usually once at the start of the program.
183  */
184 void framebufferSizeCallback(GLFWwindow *window, int w, int h) {
185     // Prevent a divide by zero, when window is too short (you cant make a
          window of zero width).
186     if (h == 0) h = 1;
187     float ratio = 1.0f * w / h;
188
189     glfwMakeContextCurrent(window);
190     glViewport(0, 0, w, h);
191
```

```
192     projection = glm::perspective(0.7f, ratio, 1.0f, 800.0f);
193
194 }
195
196 /*
197  *  This procedure is called each time the screen needs
198  *  to be redisplayed
199  */
200 void display() {
201     glm::mat4 view;
202     int modelViewLoc;
203     int projectionLoc;
204     int normalLoc;
205     int eyeLoc;
206     int LpositionLoc;
207
208     view = glm::lookAt(glm::vec3(eyex, eyey, eyez),
209         glm::vec3(cx, cy, cz),
210         glm::vec3(0.0f, 1.0f, 0.0f));
211
212     glm::mat3 normal = glm::transpose(glm::inverse(glm::mat3(view)));
213
214     glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
215     glUseProgram(program);
216     modelViewLoc = glGetUniformLocation(program, "modelView");
217     glUniformMatrix4fv(modelViewLoc, 1, 0, glm::value_ptr(view));
218     projectionLoc = glGetUniformLocation(program, "projection");
219     glUniformMatrix4fv(projectionLoc, 1, 0, glm::value_ptr(projection));
220     normalLoc = glGetUniformLocation(program, "normalMat");
221     glUniformMatrix3fv(normalLoc, 1, 0, glm::value_ptr(normal));
222
223     eyeLoc = glGetUniformLocation(program, "eye");
224     glUniform3f(eyeLoc, eyex, eyey, eyez);
225
226     LpositionLoc = glGetUniformLocation(program, "Lposition");
227     float lx = cx + 250.0 * cos(angle);
228     float lz = cz + 250.0 * sin(angle);
229     glUniform3f(LpositionLoc, lx, cy, lz);
230
231     glBindVertexArray(objVAO);
232     glDrawElements(GL_TRIANGLES, 3 * triangles, GL_UNSIGNED_INT, NULL);
233
234
235     if (counter >= 0.0 && counter <= 1.99) {
236         red = red - 0.01;
237         green = green + 0.01;
238
239         counter = counter + 0.01;
240     }
```

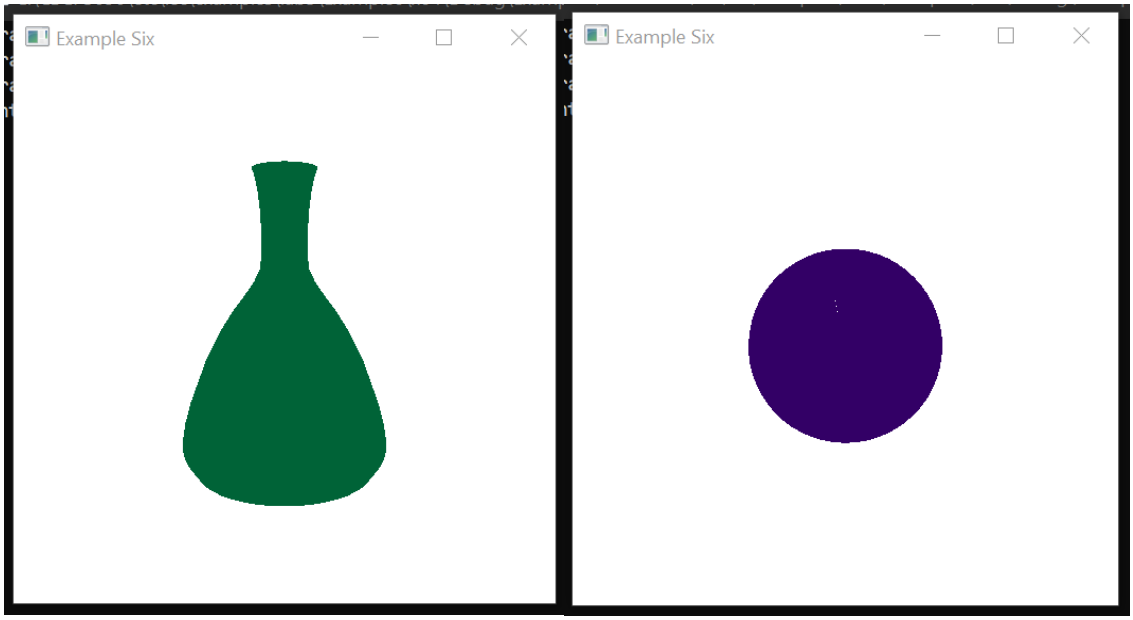
```
241     else if (counter >= 1.99 && counter <= 3.99) {
242         green = green - 0.01;
243         blue = blue + 0.01;
244 
245         counter = counter + 0.01;
246     }
247     else if (counter >= 3.99 && counter <= 5.99) {
248         red = red + 0.01;
249         blue = blue - 0.01;
250 
251         counter = counter + 0.01;
252     }
253 
254     if (counter >= 5.99) counter = 0.0;
255 
256     int colourLoc;
257     colourLoc = glGetUniformLocation(program, "colour");
258     glUniform4f(colourLoc, red, green, blue, 1.0);
259 
260 }
261 
262 /*
263  *  Called each time a key is pressed on
264  *  the keyboard.
265  */
266 static void key_callback(GLFWwindow* window, int key, int scancode, int
      action, int mods)
267 {
268     if (key == GLFW_KEY_ESCAPE && action == GLFW_PRESS)
        glfwSetWindowShouldClose(window, GLFW_TRUE);
269 
270     if (key == GLFW_KEY_A && action == GLFW_PRESS) phi -= 0.1;
271     if (key == GLFW_KEY_D && action == GLFW_PRESS) phi += 0.1;
272     if (key == GLFW_KEY_W && action == GLFW_PRESS) theta += 0.1;
273     if (key == GLFW_KEY_S && action == GLFW_PRESS) theta -= 0.1;
274 
275     eyex = (float)(r*sin(theta)*cos(phi));
276     eyey = (float)(r*sin(theta)*sin(phi));
277     eyez = (float)(r*cos(theta));
278 
279 }
280 
281 void error_callback(int error, const char* description)
282 {
283     fprintf(stderr, "Error: %s\n", description);
284 }
285 
286 int main(int argc, char **argv) {
287     GLFWwindow *window;
```

```
288
289     if(argc > 1)
290         vertexName = argv[1];
291     else
292         vertexName = "a";
293
294     if(argc > 2)
295         fragmentName = argv[2];
296     else
297         fragmentName = "a";
298
299     // start by setting error callback in case something goes wrong
300     glfwSetErrorCallback(error_callback);
301
302     // initialize glfw
303     if (!glfwInit()) {
304         fprintf(stderr, "can't initialize GLFW\n");
305     }
306
307     // create the window used by our application
308     window = glfwCreateWindow(512, 512, "Example Six", NULL, NULL);
309
310     if (!window)
311     {
312         glfwTerminate();
313         exit(EXIT_FAILURE);
314     }
315
316     // establish framebuffer size change and input callbacks
317     glfwSetFramebufferSizeCallback(window, framebufferSizeCallback);
318     glfwSetKeyCallback(window, key_callback);
319
320     //initialize glew
321     glfwMakeContextCurrent(window);
322     GLenum error = glewInit();
323     if(error != GLEW_OK) {
324         printf("Error starting GLEW: %s\n",glewGetErrorString(error));
325         exit(0);
326     }
327
328     eyex = 0.0;
329     eyey = 0.0;
330     eyez = 500.0;
331
332     theta = 0.5;
333     phi = 1.5;
334     r = 500.0;
335
336     init();
```

```
337
338     glEnable(GL_DEPTH_TEST);
339     glClearColor(1.0,1.0,1.0,1.0);
340     glViewport(0, 0, 512, 512);
341
342     projection = glm::perspective(0.7f, 1.0f, 1.0f, 800.0f);
343     glfwSwapInterval(1);
344
345     // GLFW main loop, display model, swapbuffer and check for input
346     while (!glfwWindowShouldClose(window)) {
347         display();
348         glfwSwapBuffers(window);
349         glfwPollEvents();
350     }
351
352     glfwTerminate();
353
354 }
```

```
/*
 *  Simple vertex shader for example six
 */

#version 330 core

in vec4 vPosition;
in vec3 vNormal;
uniform mat4 modelView;
uniform mat4 projection;
uniform mat3 normalMat;
out vec3 normal;
out vec4 position;

void main() {

    gl_Position = projection * modelView * vPosition;
    position = vPosition;
    normal = vNormal;
}
```

```
/*
 *  Simple fragment sharde for laboratory two
 */

#version 330 core

in vec3 normal;
in vec4 position;
uniform vec3 eye;
uniform vec3 Lposition;
uniform vec4 Lcolour;
uniform vec4 colour;

void main() {
    vec3 N;
    vec3 H;
    float diffuse;
    float specular;
    float n = 100.0;
    vec3 L;

    N = normalize(normal);
    L = Lposition - position.xyz;
    H = normalize(L + eye);
    L = normalize(L);

    diffuse = dot(N,L);

    if(diffuse < 0.0) {
        diffuse = 0.0;
        specular = 0.0;
    } else {
        specular = pow(max(0.0, dot(N,H)),n);
    }

    gl_FragColor = min(0.3*colour + diffuse*colour*Lcolour + Lcolour*specular, vec4(1.0));
    gl_FragColor.a = colour.a;
}
```


Example Six
Example Six