```
/*********************************************
 *
 *             Example 8
 *
 *  This example uses the vase model plus
 *  a texture map loaded from a file to
 *  illustrate the basic process of texture
 *  mapping.
 *
 *********************************************/

#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>
#include <FreeImage.h>

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

double theta, phi;
double r;

float cx, cy, cz;

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

#define checkImageWidth 64
#define checkImageHeight 64
GLubyte checkImage[checkImageWidth][checkImageHeight][4];
GLuint texName;         // texture name

struct textureStruct {
    int height;
    int width;
    int bytes;
```

```
50      unsigned char *data;
51  };
52
53  textureStruct* loadImage(char *filename) {
54      int i,j;
55      FIBITMAP *bitmap;
56      BYTE *bits;
57      int width;
58      int height;
59      int bytes;
60      unsigned char *data;
61      textureStruct *result;
62      int k;
63
64      result = new textureStruct();
65
66      bitmap = FreeImage_Load(FIF_JPEG, filename, JPEG_DEFAULT);
67      height = FreeImage_GetHeight(bitmap);
68      width = FreeImage_GetWidth(bitmap);
69      bytes = FreeImage_GetBPP(bitmap)/8;
70      printf("image size: %d %d %d\n",width, height, bytes);
71      data = new unsigned char[width*height*bytes];
72      result->height = height;
73      result->width = width;
74      result->bytes = bytes;
75      result->data = data;
76
77      k = 0;
78      for(j=0; j<height; j++) {
79          bits = FreeImage_GetScanLine(bitmap,j);
80              for(i=0; i<width; i++) {
81                  data[k++] = bits[FI_RGBA_RED];
82                  data[k++] = bits[FI_RGBA_GREEN];
83                  data[k++] = bits[FI_RGBA_BLUE];
84                  bits += 3;
85              }
86      }
87      FreeImage_Unload(bitmap);
88      return(result);
89  }
90
91
92  void makeCheckImage(void)
93  {
94      int i, j, c;
95
96      for (i = 0; i < checkImageWidth; i++) {
97          for (j = 0; j < checkImageHeight; j++) {
98              c = ((((i&0x8)==0)^((j&0x8)==0)))*255;
```

```
 99              checkImage[i][j][0] = (GLubyte) c;
100              checkImage[i][j][1] = (GLubyte) c;
101              checkImage[i][j][2] = (GLubyte) c;
102              checkImage[i][j][3] = (GLubyte) 255;
103          }
104      }
105  }
106
107
108  /*
109   *  The init procedure creates the OpenGL data structures
110   *  that contain the triangle geometry, compiles our
111   *  shader program and links the shader programs to
112   *  the data.
113   */
114
115  void init() {
116      GLuint vbuffer;
117      GLint vPosition;
118      GLint vNormal;
119      GLint vTexcoord;
120      int vs;
121      int fs;
122      GLfloat *vertices;
123      GLfloat *normals;
124      GLfloat *texcoords;
125      GLuint *indices;
126      std::vector<tinyobj::shape_t> shapes;
127      std::vector<tinyobj::material_t> materials;
128      int nv;
129      int nn;
130      int nt;
131      int ni;
132      int i;
133      float xmin, ymin, zmin;
134      float xmax, ymax, zmax;
135      textureStruct *texture;
136
137      glGenVertexArrays(1, &objVAO);
138      glBindVertexArray(objVAO);
139
140      /*  Load the obj file */
141
142      std::string err = tinyobj::LoadObj(shapes, materials, "vase.obj", 0);
143
144      if (!err.empty()) {
145          std::cerr << err << std::endl;
146          return;
147      }
```

```
148
149     /*  Retrieve the vertex coordinate data */
150
151     nv = shapes[0].mesh.positions.size();
152     vertices = new GLfloat[nv];
153     for(i=0; i<nv; i++) {
154         vertices[i] = shapes[0].mesh.positions[i];
155     }
156
157     /*
158      *  Find the range of the x, y and z
159      *  coordinates.
160      */
161     xmin = ymin = zmin = 1000000.0;
162     xmax = ymax = zmax = -1000000.0;
163     for(i=0; i<nv/3; i++) {
164         if(vertices[3*i] < xmin)
165             xmin = vertices[3*i];
166         if(vertices[3*i] > xmax)
167             xmax = vertices[3*i];
168         if(vertices[3*i+1] < ymin)
169             ymin = vertices[3*i+1];
170         if(vertices[3*i+1] > ymax)
171             ymax = vertices[3*i+1];
172         if(vertices[3*i+2] < zmin)
173             zmin = vertices[3*i+2];
174         if(vertices[3*i+2] > zmax)
175             zmax = vertices[3*i+2];
176     }
177     /* compute center and print range */
178     cx = (xmin+xmax)/2.0f;
179     cy = (ymin+ymax)/2.0f;
180     cz = (zmin+zmax)/2.0f;
181     printf("X range: %f %f\n",xmin,xmax);
182     printf("Y range: %f %f\n",ymin,ymax);
183     printf("Z range: %f %f\n",zmin,zmax);
184     printf("center: %f %f %f\n",cx, cy,cz);
185
186     /*  Retrieve the vertex normals */
187
188     nn = shapes[0].mesh.normals.size();
189     normals = new GLfloat[nn];
190     for(i=0; i<nn; i++) {
191         normals[i] = shapes[0].mesh.normals[i];
192     }
193
194     nt = shapes[0].mesh.texcoords.size();
195     texcoords = new GLfloat[nt];
196     for(i=0; i<nt; i++) {
```

```
197         texcoords[i] = shapes[0].mesh.texcoords[i];
198     }
199
200     /*  Retrieve the triangle indices */
201
202     ni = shapes[0].mesh.indices.size();
203     triangles = ni/3;
204     indices = new GLuint[ni];
205     for(i=0; i<ni; i++) {
206         indices[i] = shapes[0].mesh.indices[i];
207     }
208
209     /*
210      *  load the vertex coordinate data
211      */
212     glGenBuffers(1, &vbuffer);
213     glBindBuffer(GL_ARRAY_BUFFER, vbuffer);
214     glBufferData(GL_ARRAY_BUFFER, (nv+nn+nt)*sizeof(GLfloat), NULL,
          GL_STATIC_DRAW);
215     glBufferSubData(GL_ARRAY_BUFFER, 0, nv*sizeof(GLfloat), vertices);
216     glBufferSubData(GL_ARRAY_BUFFER, nv*sizeof(GLfloat), nn*sizeof(GLfloat),
          normals);
217     glBufferSubData(GL_ARRAY_BUFFER, (nv+nn)*sizeof(GLfloat), nt*sizeof
          (GLfloat), texcoords);
218
219     /*
220      *  load the vertex indexes
221      */
222     glGenBuffers(1, &ibuffer);
223     glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, ibuffer);
224     glBufferData(GL_ELEMENT_ARRAY_BUFFER, ni*sizeof(GLuint), indices,
          GL_STATIC_DRAW);
225
226     /*
227      *  compile and build the shader program
228      */
229
230     vs = buildShader(GL_VERTEX_SHADER, "example8.vs");
231     fs = buildShader(GL_FRAGMENT_SHADER, "example8.fs");
232     program = buildProgram(vs,fs,0);
233
234     /*
235      *  link the vertex coordinates to the vPosition
236      *  variable in the vertex program.  Do the same
237      *  for the normal vectors.
238      */
239     glUseProgram(program);
240     vPosition = glGetAttribLocation(program,"vPosition");
241     glVertexAttribPointer(vPosition, 3, GL_FLOAT, GL_FALSE, 0, 0);
```

```
242     glEnableVertexAttribArray(vPosition);
243     vNormal = glGetAttribLocation(program, "vNormal");
244     glVertexAttribPointer(vNormal, 3, GL_FLOAT, GL_FALSE, 0, (void*)
          (nv*sizeof(GLfloat)));
245     glEnableVertexAttribArray(vNormal);
246     vTexcoord = glGetAttribLocation(program, "vTexcoord");
247     glVertexAttribPointer(vTexcoord, 2, GL_FLOAT, GL_FALSE, 0, (void*) ((nv
          +nn)*sizeof(GLfloat)));
248     glEnableVertexAttribArray(vTexcoord);
249
250         /*
251      *  Create the texture.
252      */
253     makeCheckImage();
254
255     texture = loadImage("glasswork.jpg");
256
257     glGenTextures(1, &texName);
258     glBindTexture(GL_TEXTURE_2D, texName);
259
260     /*
261     glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, checkImageWidth,
262     checkImageHeight, 0, GL_RGBA, GL_UNSIGNED_BYTE,
263     &checkImage[0][0][0]);
264     */
265
266     glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, texture->width, texture->height,
267         0, GL_RGB,GL_UNSIGNED_BYTE,  texture->data);
268     glGenerateMipmap(GL_TEXTURE_2D);
269     glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
270     glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
271     glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
272 //   glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
273     glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
          GL_LINEAR_MIPMAP_LINEAR);
274
275 }
276
277 /*
278  *  Executed each time the window is resized,
279  *  usually once at the start of the program.
280  */
281 void framebufferSizeCallback(GLFWwindow *window, int w, int h) {
282
283     // Prevent a divide by zero, when window is too short
284     // (you cant make a window of zero width).
285
286     if (h == 0)
287         h = 1;
```

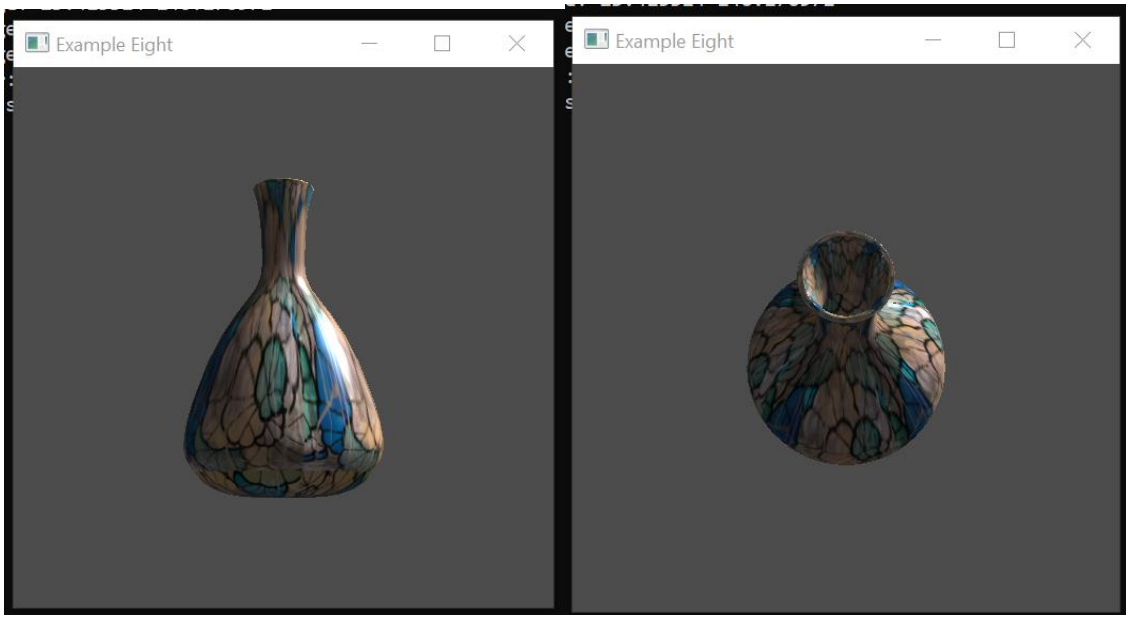
```
288
289     float ratio = 1.0f * w / h;
290
291     glfwMakeContextCurrent(window);
292
293     glViewport(0, 0, w, h);
294
295     projection = glm::perspective(0.7f, ratio, 1.0f, 800.0f);
296
297 }
298
299 /*
300  *  This procedure is called each time the screen needs
301  *  to be redisplayed
302  */
303 void display() {
304     glm::mat4 view;
305     int modelViewLoc;
306     int projectionLoc;
307     int normalLoc;
308     int eyeLoc;
309
310     view = glm::lookAt(glm::vec3(eyex, eyey, eyez),
311                     glm::vec3(cx,cy,cz),
312                     glm::vec3(0.0f, 1.0f, 0.0f));
313
314     glm::mat3 normal = glm::transpose(glm::inverse(glm::mat3(view)));
315
316     glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
317     glUseProgram(program);
318     modelViewLoc = glGetUniformLocation(program,"modelView");
319     glUniformMatrix4fv(modelViewLoc, 1, 0, glm::value_ptr(view));
320     projectionLoc = glGetUniformLocation(program,"projection");
321     glUniformMatrix4fv(projectionLoc, 1, 0, glm::value_ptr(projection));
322     normalLoc = glGetUniformLocation(program,"normalMat");
323     glUniformMatrix3fv(normalLoc, 1, 0, glm::value_ptr(normal));
324     eyeLoc = glGetUniformLocation(program, "eye");
325
326     glUniform3f(eyeLoc, eyex, eyey, eyez);
327
328     glBindTexture(GL_TEXTURE_2D, texName);
329
330     glBindVertexArray(objVAO);
331     glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, ibuffer);
332     glDrawElements(GL_TRIANGLES, 3*triangles, GL_UNSIGNED_INT, NULL);
333
334 }
335
336 /*
```

```
337  *  Called each time a key is pressed on
338  *  the keyboard.
339  */
340 static void key_callback(GLFWwindow* window, int key, int scancode, int
      action, int mods)
341 {
342     if (key == GLFW_KEY_ESCAPE && action == GLFW_PRESS)
343         glfwSetWindowShouldClose(window, GLFW_TRUE);
344
345     if (key == GLFW_KEY_A && action == GLFW_PRESS)
346         phi -= 0.1;
347     if (key == GLFW_KEY_D && action == GLFW_PRESS)
348         phi += 0.1;
349     if (key == GLFW_KEY_W && action == GLFW_PRESS)
350         theta += 0.1;
351     if (key == GLFW_KEY_S && action == GLFW_PRESS)
352         theta -= 0.1;
353
354     eyex = (float)(r*sin(theta)*cos(phi));
355     eyey = (float)(r*sin(theta)*sin(phi));
356     eyez = (float)(r*cos(theta));
357
358 }
359
360 void error_callback(int error, const char* description)
361 {
362     fprintf(stderr, "Error: %s\n", description);
363 }
364
365
366 int main(int argc, char **argv) {
367     GLFWwindow *window;
368
369     glfwSetErrorCallback(error_callback);
370
371     // initialize glfw
372
373     if (!glfwInit()) {
374         fprintf(stderr, "can't initialize GLFW\n");
375     }
376
377     // create the window used by our application
378
379     window = glfwCreateWindow(512, 512, "Example Eight", NULL, NULL);
380
381     if (!window)
382     {
383         glfwTerminate();
384         exit(EXIT_FAILURE);
```

```
385     }
386
387     // establish framebuffer size change and input callbacks
388
389     glfwSetFramebufferSizeCallback(window, framebufferSizeCallback);
390
391     glfwSetKeyCallback(window, key_callback);
392
393     /*
394      *  initialize glew
395      */
396     glfwMakeContextCurrent(window);
397     GLenum error = glewInit();
398     if(error != GLEW_OK) {
399         printf("Error starting GLEW: %s\n",glewGetErrorString(error));
400         exit(0);
401     }
402
403     eyex = 0.0;
404     eyey = 0.0;
405     eyez = 500.0;
406
407     theta = 0.5;
408     phi = 1.5;
409     r = 500.0;
410
411     init();
412
413     glEnable(GL_DEPTH_TEST);
414     glClearColor(0.3,0.3,0.3,1.0);
415
416     projection = glm::perspective(0.7f, 1.0f, 1.0f, 800.0f);
417
418     glfwSwapInterval(1);
419
420     // GLFW main loop, display model, swapbuffer and check for input
421
422     while (!glfwWindowShouldClose(window)) {
423         display();
424         glfwSwapBuffers(window);
425         glfwPollEvents();
426     }
427
428     glfwTerminate();
429
430 }
```

```
1  /*
2   *  Simple vertex shader for the first
3   *  texture example.
4   */
5
6  #version 330 core
7
8  in vec4 vPosition;
9  in vec3 vNormal;
10 in vec2 vTexcoord;
11
12 uniform mat4 modelView;
13 uniform mat4 projection;
14 uniform mat3 normalMat;
15
16 out vec2 texCoord;
17 out vec3 normal;
18
19 void main(void) {
20     gl_Position = projection * modelView * vPosition;
21     normal = normalMat * vNormal;
22     texCoord = vTexcoord * 4;
23 }
```

```
1  /*
2   *  Simple fragment shader for the first
3   *  texture example.  Just look up the texture
4   *  value at the current text coordinate
5   */
6
7  #version 330 core
8
9  in vec2 texCoord;
10 uniform sampler2D tex;
11
12 in vec3 normal;
13 in vec4 position;
14 uniform vec3 eye;
15
16 void main(void) {
17     vec3 N;
18     vec3 L = vec3(1.0, 1.0, 0.0);
19     vec4 Lcolour = vec4(1.0, 1.0, 1.0, 1.0);
20     vec3 H = normalize(L + vec3(0.0, 0.0, 1.0));
21
22     float diffuse;
23     float specular;
24     float n = 100.0;
25
26     N = normalize(normal);
27     L = normalize(L);
28     diffuse = dot(N,L);
29
30     if(diffuse < 0.0) {
31         diffuse = 0.0;
32         specular = 0.0;
33     } else {
34         specular = pow(max(0.0, dot(N,H)),n);
35     }
36
37     vec4 colour = texture(tex, texCoord);
38
39     gl_FragColor = min(0.3*colour + diffuse*colour*Lcolour + Lcolour*specular,
          vec4(1.0));
40     gl_FragColor.a = colour.a;
41 }
```


Example Eight
Example Eight