```cpp
#include <fstream>
#include <string>
#include <vector>
#include <iostream>

#include <GL/glew.h>

#include <SDL2/SDL_main.h>
#include <SDL2/SDL.h>
#include <SDL2/SDL_opengl.h>
#undef main

#define GLM_FORCE_RADIANS
#include <glm/glm.hpp>
#include <glm/gtc/matrix_transform.hpp>
#include <glm/gtc/type_ptr.hpp>

#define STB_IMAGE_IMPLEMENTATION
#include "STB/stb_image.h"

#include "myShader.h"

using namespace std;

// SDL variables
SDL_Window* window;
SDL_GLContext glContext;

bool quit = false;

int mouse_position[2];
bool button_pressed = false;

int window_height = 863;
int window_width = 1646;

float fovy = 45.0f;
float znear = 1.0f;
float zfar = 2000.0f;

glm::vec3 camera_eye = glm::vec3(0.0f, 0.0f, 2.0f);
glm::vec3 camera_up = glm::vec3(0.0f, 1.0f, 0.0f);
glm::vec3 camera_forward = glm::vec3(0.0f, 0.0f, -1.0f);


void rotate(glm::vec3 & inputvec, glm::vec3 rotation_axis, float theta, bool tonormalize = false)
{
    const float cos_theta = cos(theta);
```

```
49      const float dot = glm::dot(inputvec, rotation_axis);
50      glm::vec3 cross = glm::cross(inputvec, rotation_axis);
51
52      inputvec.x *= cos_theta; inputvec.y *= cos_theta; inputvec.z *= cos_theta;
53      inputvec.x += rotation_axis.x * dot * (float)(1.0 - cos_theta);
54      inputvec.y += rotation_axis.y * dot * (float)(1.0 - cos_theta);
55      inputvec.z += rotation_axis.z * dot * (float)(1.0 - cos_theta);
56
57      inputvec.x -= cross.x * sin(theta);
58      inputvec.y -= cross.y * sin(theta);
59      inputvec.z -= cross.z * sin(theta);
60
61      if (tonormalize) inputvec = glm::normalize(inputvec);
62  }
63
64  // Process the event.
65  void processEvents(SDL_Event current_event)
66  {
67      switch (current_event.type)
68      {
69          // window close button is pressed
70      case SDL_QUIT:
71      {
72          quit = true;
73          break;
74      }
75      case SDL_KEYDOWN:
76      {
77          if (current_event.key.keysym.sym == SDLK_ESCAPE)
78              quit = true;
79      }
80      case SDL_MOUSEBUTTONDOWN:
81      {
82          mouse_position[0] = current_event.button.x;
83          mouse_position[1] = window_height - current_event.button.y;
84          button_pressed = true;
85          break;
86      }
87      case SDL_MOUSEBUTTONUP:
88      {
89          button_pressed = false;
90          break;
91      }
92      case SDL_MOUSEMOTION:
93      {
94          if (button_pressed == false) break;
95
96          int x = current_event.motion.x;
97          int y = window_height - current_event.motion.y;
```

```
 98
 99          int dx = x - mouse_position[0];
100          int dy = y - mouse_position[1];
101
102          if (dx == 0 && dy == 0) break;
103
104          mouse_position[0] = x;
105          mouse_position[1] = y;
106
107          float vx = (float)dx / (float)window_width;
108          float vy = (float)dy / (float)window_height;
109          float theta = 4.0f * (fabs(vx) + fabs(vy));
110
111          glm::vec3 camera_right = glm::normalize(glm::cross(camera_forward,
               camera_up));
112
113          glm::vec3 tomovein_direction = -camera_right * vx + -camera_up * vy;
114
115          glm::vec3 rotation_axis = glm::normalize(glm::cross(tomovein_direction,
               camera_forward));
116
117          rotate(camera_forward, rotation_axis, theta, true);
118          rotate(camera_up, rotation_axis, theta, true);
119          rotate(camera_eye, rotation_axis, theta, false);
120
121          break;
122      }
123      case SDL_MOUSEWHEEL:
124      {
125          if (current_event.wheel.y < 0)
126              camera_eye -= 0.1f * camera_forward;
127          else if (current_event.wheel.y > 0)
128              camera_eye += 0.1f * camera_forward;
129      }
130      default:
131          break;
132      }
133  }
134
135  int main(int argc, char *argv[])
136  {
137      // Initialize video subsystem
138      SDL_Init(SDL_INIT_TIMER | SDL_INIT_VIDEO);
139
140      // Using OpenGL 3.1 core
141      SDL_GL_SetAttribute(SDL_GL_CONTEXT_MAJOR_VERSION, 3);
142      SDL_GL_SetAttribute(SDL_GL_CONTEXT_MINOR_VERSION, 1);
143      SDL_GL_SetAttribute(SDL_GL_CONTEXT_PROFILE_MASK,
           SDL_GL_CONTEXT_PROFILE_CORE);
```

```
144     SDL_GL_SetAttribute(SDL_GL_DOUBLEBUFFER, 1);
145     SDL_GL_SetAttribute(SDL_GL_MULTISAMPLEBUFFERS, 1);
146     SDL_GL_SetAttribute(SDL_GL_MULTISAMPLESAMPLES, 4);
147     SDL_GL_SetAttribute(SDL_GL_ACCELERATED_VISUAL, 1);
148
149     // Create window
150     window = SDL_CreateWindow("IT-5102E", SDL_WINDOWPOS_CENTERED,
          SDL_WINDOWPOS_CENTERED,
151         window_width, window_height, SDL_WINDOW_OPENGL | SDL_WINDOW_RESIZABLE);
152
153     // Create OpenGL context
154     glContext = SDL_GL_CreateContext(window);
155
156     // Initialize glew
157     glewInit();
158     glEnable(GL_DEPTH_TEST);
159     glDepthFunc(GL_LESS);
160     glEnable(GL_TEXTURE_2D);
161     glDisable(GL_CULL_FACE);
162
163     glClearColor(0.1f, 0.1f, 0.1f, 1.0f);
164
165     myShader *shader = new myShader("textureimage-vertexshader.glsl",
          "textureimage-fragmentshader.glsl");
166     shader->start();
167
168
169     vector<glm::vec3> vertices;
170     vertices.push_back(glm::vec3(0.0f, 0.0f, 0.0f));
171     vertices.push_back(glm::vec3(1.0f, 0.0f, 0.0f));
172     vertices.push_back(glm::vec3(0.0f, 1.0f, 0.0f));
173     vertices.push_back(glm::vec3(0.0f, 0.0f, 1.0f));
174
175     vector<glm::ivec3> indices;
176     indices.push_back(glm::ivec3(1, 2, 3));
177     indices.push_back(glm::ivec3(0, 1, 2));
178     indices.push_back(glm::ivec3(2, 0, 3));
179     indices.push_back(glm::ivec3(1, 0, 3));
180
181     vector<glm::vec3> normals;
182     normals.push_back(glm::vec3(-1, -1, -1));
183     normals.push_back(glm::vec3(1, 0, 0));
184     normals.push_back(glm::vec3(0, 1, 0));
185     normals.push_back(glm::vec3(0, 0, 1));
186
187     /*********************************************************************/
188     vector<glm::vec2> texturecoordinates;
189     texturecoordinates.push_back(glm::vec2(0,0));
190     texturecoordinates.push_back(glm::vec2(0, 1));
```

```
191     texturecoordinates.push_back(glm::vec2(1, 0));
192     texturecoordinates.push_back(glm::vec2(1, 1));
193     /*****************************************************************/
194
195     GLuint vao;
196     glGenVertexArrays(1, &vao);
197     glBindVertexArray(vao);
198
199     GLuint buffers[4];
200     glGenBuffers(4, buffers);
201
202     unsigned int location;
203
204     location = 0;
205     glBindBuffer(GL_ARRAY_BUFFER, buffers[0]);
206     glBufferData(GL_ARRAY_BUFFER, vertices.size() * sizeof(glm::vec3),
          &vertices[0], GL_STATIC_DRAW);
207     glVertexAttribPointer(location, 3, GL_FLOAT, GL_FALSE, 0, 0);
208     glEnableVertexAttribArray(location);
209
210     location = 1;
211     glBindBuffer(GL_ARRAY_BUFFER, buffers[1]);
212     glBufferData(GL_ARRAY_BUFFER, normals.size() * sizeof(glm::vec3), &normals
          [0], GL_STATIC_DRAW);
213     glEnableVertexAttribArray(location);
214     glVertexAttribPointer(location, 3, GL_FLOAT, GL_FALSE, 0, 0);
215
216     /*****************************************************************/
217     location = 2;
218     glBindBuffer(GL_ARRAY_BUFFER, buffers[3]);
219     glBufferData(GL_ARRAY_BUFFER, texturecoordinates.size() * sizeof
          (glm::vec2), &texturecoordinates[0], GL_STATIC_DRAW);
220     glEnableVertexAttribArray(location);
221     glVertexAttribPointer(location, 2, GL_FLOAT, GL_FALSE, 0, 0);
222     /*****************************************************************/
223
224     glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, buffers[2]);
225     glBufferData(GL_ELEMENT_ARRAY_BUFFER, indices.size() * sizeof(glm::ivec3),
          &indices[0], GL_STATIC_DRAW);
226
227     glBindVertexArray(0);
228
229
230     /
          ************************************************************************
          */
231     GLuint texture_id;
232     {
233         int size, width, height;
```

```
234
235         GLubyte *mytexture = stbi_load("scenary.jpg", &width, &height, &size,
              4);
236
237         glGenTextures(1, &texture_id);
238         glBindTexture(GL_TEXTURE_2D, texture_id);
239
240         glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
241         glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
242         glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
              GL_LINEAR_MIPMAP_LINEAR);
243         glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
244
245         glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, static_cast<GLuint>(width),
              static_cast<GLuint>(height), 0, GL_RGBA, GL_UNSIGNED_BYTE,
              mytexture);
246
247         delete mytexture;
248         glGenerateMipmap(GL_TEXTURE_2D);
249         glBindTexture(GL_TEXTURE_2D, 0);
250     }
251     /
        ************************************************************************
        */
252
253
254     // display loop
255     while (!quit)
256     {
257         glViewport(0, 0, window_width, window_height);
258
259         glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
260
261         glm::mat4 projection_matrix = glm::perspective(glm::radians(fovy),
              static_cast<float>(window_width) / static_cast<float>(window_height),
               znear, zfar);
262         glUniformMatrix4fv(glGetUniformLocation(shader->shaderprogram,
              "myprojection_matrix"), 1, GL_FALSE, glm::value_ptr
              (projection_matrix));
263
264         glm::mat4 view_matrix = glm::lookAt(camera_eye, camera_eye +
              camera_forward, camera_up);
265         glUniformMatrix4fv(glGetUniformLocation(shader->shaderprogram,
              "myview_matrix"), 1, GL_FALSE, glm::value_ptr(view_matrix));
266
267
268         glUniform4fv(glGetUniformLocation(shader->shaderprogram,
              "input_color"), 1, glm::value_ptr(glm::vec4(1.0f, 0.0f, 0.0f,
              1.0f)));
```

```
269
270         /*****************************************************************
              *****/
271         int texture_offset = 8;
272         glActiveTexture(GL_TEXTURE0 + texture_offset);
273         glBindTexture(GL_TEXTURE_2D, texture_id);
274
275         glUniform1i(glGetUniformLocation(shader->shaderprogram, "imagetex"),
              texture_offset);
276
277         shader->setUniform("imagetex", static_cast<int>(texture_offset));
278         /*****************************************************************
              *****/
279
280         glBindVertexArray(vao);
281         glDrawElements(GL_TRIANGLES, static_cast<GLsizei>(indices.size() * 3),
              GL_UNSIGNED_INT, 0);
282         glBindVertexArray(0);
283
284
285         glUniform4fv(glGetUniformLocation(shader->shaderprogram,
              "input_color"), 1, glm::value_ptr(glm::vec4(1.0f, 1.0f, 1.0f,
              1.0f)));
286         glBegin(GL_LINES);
287         for (unsigned int i = 0; i < vertices.size(); ++i)
288         {
289             glm::vec3 v = vertices[i] + glm::normalize(normals[i]);
290             glVertex3fv(&vertices[i][0]);
291             glVertex3fv(&v[0]);
292         }
293         glEnd();
294
295
296         SDL_GL_SwapWindow(window);
297
298         SDL_Event current_event;
299         while (SDL_PollEvent(&current_event) != 0)
300             processEvents(current_event);
301     }
302
303     // Freeing resources before exiting.
304
305     // Destroy window
306     if (glContext) SDL_GL_DeleteContext(glContext);
307     if (window) SDL_DestroyWindow(window);
308
309     // Quit SDL subsystems
```

```
310     SDL_Quit();
311
312     return 0;
313 }
```