```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>

#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);
    const float dot = glm::dot(inputvec, rotation_axis);
    glm::vec3 cross = glm::cross(inputvec, rotation_axis);

```

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

```
 98
 99          if (dx == 0 && dy == 0) break;
100
101          mouse_position[0] = x;
102          mouse_position[1] = y;
103
104          float vx = (float)dx / (float)window_width;
105          float vy = (float)dy / (float)window_height;
106          float theta = 4.0f * (fabs(vx) + fabs(vy));
107
108          glm::vec3 camera_right = glm::normalize(glm::cross(camera_forward,
               camera_up));
109
110          glm::vec3 tomovein_direction = -camera_right * vx + -camera_up * vy;
111
112          glm::vec3 rotation_axis = glm::normalize(glm::cross(tomovein_direction,
               camera_forward));
113
114          rotate(camera_forward, rotation_axis, theta, true);
115          rotate(camera_up, rotation_axis, theta, true);
116          rotate(camera_eye, rotation_axis, theta, false);
117
118          break;
119      }
120      case SDL_MOUSEWHEEL:
121      {
122          if (current_event.wheel.y < 0)
123              camera_eye -= 0.1f * camera_forward;
124          else if (current_event.wheel.y > 0)
125              camera_eye += 0.1f * camera_forward;
126      }
127      default:
128          break;
129      }
130  }
131
132  int main(int argc, char *argv[])
133  {
134      // Initialize video subsystem
135      SDL_Init(SDL_INIT_TIMER | SDL_INIT_VIDEO);
136
137      // Using OpenGL 3.1 core
138      SDL_GL_SetAttribute(SDL_GL_CONTEXT_MAJOR_VERSION, 3);
139      SDL_GL_SetAttribute(SDL_GL_CONTEXT_MINOR_VERSION, 1);
140      SDL_GL_SetAttribute(SDL_GL_CONTEXT_PROFILE_MASK,
           SDL_GL_CONTEXT_PROFILE_CORE);
141      SDL_GL_SetAttribute(SDL_GL_DOUBLEBUFFER, 1);
142      SDL_GL_SetAttribute(SDL_GL_MULTISAMPLEBUFFERS, 1);
143      SDL_GL_SetAttribute(SDL_GL_MULTISAMPLESAMPLES, 4);
```

```
144     SDL_GL_SetAttribute(SDL_GL_ACCELERATED_VISUAL, 1);
145
146     // Create window
147     window = SDL_CreateWindow("IT-5102E", SDL_WINDOWPOS_CENTERED,
          SDL_WINDOWPOS_CENTERED,
148         window_width, window_height, SDL_WINDOW_OPENGL | SDL_WINDOW_RESIZABLE);
149
150     // Create OpenGL context
151     glContext = SDL_GL_CreateContext(window);
152
153     // Initialize glew
154     glewInit();
155     glEnable(GL_DEPTH_TEST);
156     glDepthFunc(GL_LESS);
157     glEnable(GL_TEXTURE_2D);
158     glDisable(GL_CULL_FACE);
159
160     glClearColor(0.1f, 0.1f, 0.1f, 1.0f);
161
162
163     myShader *shader = new myShader("basic-vertexshader.glsl", "basic-
          fragmentshader.glsl");
164     shader->start();
165
166
167     vector<glm::vec3> vertices;
168     vertices.push_back(glm::vec3(-1.0f, 0.0f, 0.0f));
169     vertices.push_back(glm::vec3(0.5f, 0.0f, 0.0f));
170     vertices.push_back(glm::vec3(0.5f, 0.5f, 0.0f));
171
172     vector<glm::ivec3> indices;
173     indices.push_back(glm::ivec3(0, 1, 2));
174
175
176     GLuint buffers[2];
177     glGenBuffers(2, buffers);
178
179     glBindBuffer(GL_ARRAY_BUFFER, buffers[0]);
180     glBufferData(GL_ARRAY_BUFFER, vertices.size() * sizeof(glm::vec3),
          &vertices[0], GL_STATIC_DRAW);
181
182     glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, buffers[1]);
183     glBufferData(GL_ELEMENT_ARRAY_BUFFER, indices.size() * sizeof(glm::ivec3),
          &indices[0], GL_STATIC_DRAW);
184
185
186     // display loop
187     while (!quit)
188     {
```

```
189         glViewport(0, 0, window_width, window_height);
190
191         glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
192
193         glm::mat4 projection_matrix = glm::perspective(glm::radians(fovy),
              static_cast<float>(window_width) / static_cast<float>(window_height),
               znear, zfar);
194         glUniformMatrix4fv(glGetUniformLocation(shader->shaderprogram,
              "myprojection_matrix"), 1, GL_FALSE, glm::value_ptr
              (projection_matrix));
195
196         glm::mat4 view_matrix = glm::lookAt(camera_eye, camera_eye +
              camera_forward, camera_up);
197         glUniformMatrix4fv(glGetUniformLocation(shader->shaderprogram,
              "myview_matrix"), 1, GL_FALSE, glm::value_ptr(view_matrix));
198
199         glUniform4fv(glGetUniformLocation(shader->shaderprogram,
              "input_color"), 1, glm::value_ptr(glm::vec4(1.0f, 0.0f, 0.0f,
              1.0f)));
200
201
202
203         unsigned int location;
204
205         location = 0;
206         glBindBuffer(GL_ARRAY_BUFFER, buffers[0]);
207         glEnableVertexAttribArray(location);
208         glVertexAttribPointer(location, 3, GL_FLOAT, GL_FALSE, 0, 0);
209
210         glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, buffers[1]);
211
212         glDrawElements(GL_TRIANGLES, static_cast<GLsizei>(indices.size() * 3),
              GL_UNSIGNED_INT, 0);
213
214         glDisableVertexAttribArray(0);
215
216
217         SDL_GL_SwapWindow(window);
218
219         SDL_Event current_event;
220         while (SDL_PollEvent(&current_event) != 0)
221             processEvents(current_event);
222     }
223
224     // Freeing resources before exiting.
225
226     // Destroy window
227     if (glContext) SDL_GL_DeleteContext(glContext);
228     if (window) SDL_DestroyWindow(window);
```

```
229
230     // Quit SDL subsystems
231     SDL_Quit();
232
233     return 0;
234 }
```