

```
1 #include <fstream>
2 #include <string>
3 #include <vector>
4 #include <iostream>
5
6 #include <GL/glew.h>
7
8 #include <SDL2/SDL_main.h>
9 #include <SDL2/SDL.h>
10 #include <SDL2/SDL_opengl.h>
11 #undef main
12
13 #define GLM_FORCE_RADIANS
14 #include <glm/glm.hpp>
15 #include <glm/gtc/matrix_transform.hpp>
16 #include <glm/gtc/type_ptr.hpp>
17
18 #include "myShader.h"
19
20 using namespace std;
21
22 // SDL variables
23 SDL_Window* window;
24 SDL_GLContext glContext;
25
26 bool quit = false;
27
28 int mouse_position[2];
29 bool button_pressed = false;
30
31 int window_height = 863;
32 int window_width = 1646;
33
34 float fovy = 45.0f;
35 float znear = 1.0f;
36 float zfar = 2000.0f;
37
38 glm::vec3 camera_eye = glm::vec3(0.0f, 0.0f, 2.0f);
39 glm::vec3 camera_up = glm::vec3(0.0f, 1.0f, 0.0f);
40 glm::vec3 camera_forward = glm::vec3(0.0f, 0.0f, -1.0f);
41
42
43 void rotate(glm::vec3 & inputvec, glm::vec3 rotation_axis, float theta, bool tonormalize = false)
44 {
45     const float cos_theta = cos(theta);
46     const float dot = glm::dot(inputvec, rotation_axis);
47     glm::vec3 cross = glm::cross(inputvec, rotation_axis);
48
```

```
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,
109                                                         camera_up));
110
111     glm::vec3 tomovein_direction = -camera_right * vx + -camera_up * vy;
112
113     glm::vec3 rotation_axis = glm::normalize(glm::cross(tomovein_direction,
114                                                         camera_forward));
115
116     rotate(camera_forward, rotation_axis, theta, true);
117     rotate(camera_up, rotation_axis, theta, true);
118     rotate(camera_eye, rotation_axis, theta, false);
119
120     break;
121 }
122 case SDL_MOUSEWHEEL:
123 {
124     if (current_event.wheel.y < 0)
125         camera_eye -= 0.1f * camera_forward;
126     else if (current_event.wheel.y > 0)
127         camera_eye += 0.1f * camera_forward;
128 }
129 default:
130     break;
131 }
132 }
133
134 int main(int argc, char *argv[])
135 {
136     // Initialize video subsystem
137     SDL_Init(SDL_INIT_TIMER | SDL_INIT_VIDEO);
138
139     // Using OpenGL 3.1 core
140     SDL_GL_SetAttribute(SDL_GL_CONTEXT_MAJOR_VERSION, 3);
141     SDL_GL_SetAttribute(SDL_GL_CONTEXT_MINOR_VERSION, 1);
142     SDL_GL_SetAttribute(SDL_GL_CONTEXT_PROFILE_MASK,
143                         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_MULTISAMPLES, 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     myShader *shader = new myShader("basic-vertexshader.glsl", "basic-      ↗
        fragments shader.glsl");
163     shader->start();
164
165
166     vector<glm::vec3> vertices;
167     vertices.push_back(glm::vec3(0.0f, 0.0f, 0.0f));
168     vertices.push_back(glm::vec3(1.0f, 0.0f, 0.0f));
169     vertices.push_back(glm::vec3(0.0f, 1.0f, 0.0f));
170     vertices.push_back(glm::vec3(0.0f, 0.0f, 1.0f));
171
172     vector<glm::ivec3> indices;
173     indices.push_back(glm::ivec3(1,2,3));
174     indices.push_back(glm::ivec3(0,1,2));
175     indices.push_back(glm::ivec3(2,0,3));
176     indices.push_back(glm::ivec3(1,0,3));
177
178     GLuint vao;
179     glGenVertexArrays(1, &vao);
180     glBindVertexArray(vao);
181
182     GLuint buffers[2];
183     glGenBuffers(2, buffers);
184
185     unsigned int location;
186
187     location = 0;
188     glBindBuffer(GL_ARRAY_BUFFER, buffers[0]);
189     glBufferData(GL_ARRAY_BUFFER, vertices.size() * sizeof(glm::vec3),      ↗
        &vertices[0], GL_STATIC_DRAW);
```

```
190     glEnableVertexAttribArray(location);
191     glVertexAttribPointer(location, 3, GL_FLOAT, GL_FALSE, 0, 0);
192
193     glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, buffers[1]);
194     glBufferData(GL_ELEMENT_ARRAY_BUFFER, indices.size() * sizeof(glm::ivec3),
195                 &indices[0], GL_STATIC_DRAW);
196
197     glBindVertexArray(0);
198
199     // display loop
200     while (!quit)
201     {
202         glViewport(0, 0, window_width, window_height);
203
204         glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
205
206         glm::mat4 projection_matrix = glm::perspective(glm::radians(fovy),
207             static_cast<float>(window_width) / static_cast<float>(window_height),
208             znear, zfar);
209         glUniformMatrix4fv(glGetUniformLocation(shader->shaderprogram,
210             "myprojection_matrix"), 1, GL_FALSE, glm::value_ptr
211             (projection_matrix));
212
213         glm::mat4 view_matrix = glm::lookAt(camera_eye, camera_eye +
214             camera_forward, camera_up);
215         glUniformMatrix4fv(glGetUniformLocation(shader->shaderprogram,
216             "myview_matrix"), 1, GL_FALSE, glm::value_ptr(view_matrix));
217
218         glUniform4fv(glGetUniformLocation(shader->shaderprogram,
219             "input_color"), 1, glm::value_ptr(glm::vec4(1.0f, 0.0f, 0.0f,
220             1.0f)));
221
222         glBindVertexArray(vao);
223         glDrawElements(GL_TRIANGLES, static_cast<GLsizei>(indices.size() * 3),
224             GL_UNSIGNED_INT, 0);
225         glBindVertexArray(0);
226
227         SDL_GL_SwapWindow(window);
228
229         SDL_Event current_event;
230         while (SDL_PollEvent(&current_event) != 0)
231             processEvents(current_event);
232     }
233
234     // Freeing resources before exiting.
```

```
229     // Destroy window
230     if (glContext) SDL_GL_DeleteContext(glContext);
231     if (window) SDL_DestroyWindow(window);
232
233     // Quit SDL subsystems
234     SDL_Quit();
235
236     return 0;
237 }
```