```cpp
// Simplified Renderer application for GP course
// Code is similar to the one in lab 1 but all the graphics sections were refactored into the Graphics Class.
// Extra improvements:
// Reduced OpenGL version from 4.5 to 3.3 to allow it to render in older laptops.
// Added Shapes library for rendering cubes, spheres and vectors.
// Added examples of matrix multiplication on Update.
// Added resize screen and keyboard callbacks.
//
// Update 2018/01 updated libraries and created project for VS2015.

// Suggestions or extra help please do email me S.Padilla@hw.ac.uk
//
// Note: Do not forget to link the libraries correctly and add the GLEW DLL in your debug/release folder.

#include <iostream>
#include <vector>
using namespace std;

#include <GL/glew.h>
#include <GLFW/glfw3.h>
#include <glm/glm.hpp>
#define GLM_ENABLE_EXPERIMENTAL
#include <glm/gtx/transform.hpp>


#include "graphics.h"
#include "shapes.h"
#include "Swarm.h"

// FUNCTIONS
void render(double currentTime);
void update(double currentTime);
void startup();
void onResizeCallback(GLFWwindow* window, int w, int h);
void onKeyCallback(GLFWwindow* window, int key, int scancode, int action, int mods);

// VARIABLES
bool        running = true;

Graphics    myGraphics;     // Runing all the graphics in this object

Cube        myCube;
Sphere      mySphere;
Arrow       arrowX;
Arrow       arrowY;
Arrow       arrowZ;

float t = 0.001f;           // Global variable for animation
float leftRight = 1.0f;
```

```
50  std::vector<Boid> boids;
51  std::vector<Cube> visualBoids;
52  Swarm swarm;
53  float prevUpdateTime = 0;
54
55
56  int main()
57  {
58      int errorGraphics = myGraphics.Init();      // Launch window and graphics context
59      if (errorGraphics) return 0;                //Close if something went wrong...
60
61      // define swarm members
62
63      int zoom = 8;
64
65      boids.push_back(Boid(MyVector(1, 0, -6-zoom), MyVector(1, 0, 0)));
66      boids.push_back(Boid(MyVector(1.5, 0, -6 - zoom), MyVector(1, 1, 0)));
67      boids.push_back(Boid(MyVector(1, 0.5, -6.5 - zoom), MyVector(0, 1, 0)));
68      boids.push_back(Boid(MyVector(2.5, 2, -8 - zoom), MyVector(1, 0, 0)));
69      boids.push_back(Boid(MyVector(2, 3, -8 - zoom), MyVector(1, 0, 0)));
70      boids.push_back(Boid(MyVector(2, 2, -7 - zoom), MyVector(1, 1, 0)));
71      boids.push_back(Boid(MyVector(2, 2, -8 - zoom), MyVector(1, 0, 1)));
72      boids.push_back(Boid(MyVector(1, 0, -6 - zoom), MyVector(1, 0, 0)));
73      boids.push_back(Boid(MyVector(1.5, 0, -6 - zoom), MyVector(1, 1, 0)));
74      boids.push_back(Boid(MyVector(1, 0.5, -6.5 - zoom), MyVector(0, 1, 0)));
75      boids.push_back(Boid(MyVector(2.5, 2, -8 - zoom), MyVector(1, 0, 0)));
76      boids.push_back(Boid(MyVector(2, 3, -8 - zoom), MyVector(1, 0, 0)));
77      boids.push_back(Boid(MyVector(2, 2, -7 - zoom), MyVector(1, 1, 0)));
78      boids.push_back(Boid(MyVector(2, 2, -8 - zoom), MyVector(1, 0, 1)));
79      boids.push_back(Boid(MyVector(1, 0, -6 - zoom), MyVector(1, 0, 0)));
80      boids.push_back(Boid(MyVector(1.5, 0, -6 - zoom), MyVector(1, 1, 0)));
81      boids.push_back(Boid(MyVector(1, 0.5, -6.5 - zoom), MyVector(0, 1, 0)));
82      boids.push_back(Boid(MyVector(2.5, 2, -8 - zoom), MyVector(1, 0, 0)));
83      boids.push_back(Boid(MyVector(2, 3, -8 - zoom), MyVector(1, 0, 0)));
84      boids.push_back(Boid(MyVector(2, 2, -7 - zoom), MyVector(1, 1, 0)));
85      boids.push_back(Boid(MyVector(2, 2, -8 - zoom), MyVector(1, 0, 1)));
86
87      //create swarm
88      swarm = Swarm(&boids);
89
90      startup();                                  // Setup all necessary information for startup (aka. load texture, shaders, models, etc).
91
92                                                  // Mixed graphics and update functions - declared in main for simplicity.
93      glfwSetWindowSizeCallback(myGraphics.window, onResizeCallback);          // Set callback for resize
94      glfwSetKeyCallback(myGraphics.window, onKeyCallback);                    // Set Callback for keys
95
96
```

```
    // MAIN LOOP run until the window is closed
 97     do {
 98         double currentTime = glfwGetTime();      // retrieve timelapse
 99         glfwPollEvents();                        // poll callbacks
100         update(currentTime);                     // update (physics,
              animation, structures, etc)
101         render(currentTime);                     // call render function.
102
103         glfwSwapBuffers(myGraphics.window);      // swap buffers (avoid
              flickering and tearing)
104
105         leftRight = (float)glfwGetKey(myGraphics.window, GLFW_KEY_LEFT);
106
107         running &= (glfwGetKey(myGraphics.window, GLFW_KEY_ESCAPE) ==
              GLFW_RELEASE);  // exit if escape key pressed
108         running &= (glfwWindowShouldClose(myGraphics.window) != GL_TRUE);
109     } while (running);
110
111     myGraphics.endProgram();            // Close and clean everything up...
112
113     cout << "\nPress any key to continue...\n";
114     cin.ignore(); cin.get(); // delay closing console to read debugging
          errors.
115
116     return 0;
117 }
118
119 void startup() {
120
121     // Calculate proj_matrix for the first time.
122     myGraphics.aspect = (float)myGraphics.windowWidth / (float)
          myGraphics.windowHeight;
123     myGraphics.proj_matrix = glm::perspective(glm::radians(50.0f),
          myGraphics.aspect, 0.1f, 1000.0f);
124
125     // Load Geometry
126
127     for (Boid &b : *swarm.boids){
128         Cube visualBoid;
129         visualBoid.Load();
130         visualBoids.push_back(visualBoid);
131
132
133     }
134
135     myCube.Load();
136
137     myGraphics.SetOptimisations();      // Cull and depth testing
138 }
139
140 void update(double currentTime) {
141     float deltaSeconds = currentTime - prevUpdateTime;
142     swarm.UpdateSwarm(t);
```

```
143
144     for (Boid &b : *swarm.boids)
145     {
146         b.Position = b.Position + b.Velocity * t;
147     }
148
149     for (int i = 0; i < visualBoids.size(); i++) {
150         glm::mat4 mv_matrix_cube =
151             glm::translate(glm::vec3(swarm.boids->at(i).Position.x,
                  swarm.boids->at(i).Position.y, swarm.boids->at(i).Position.z))
                   *
152             glm::scale(glm::vec3(0.1f, 0.1f, 0.1f)) *
153             glm::mat4(1.0f);
154         visualBoids.at(i).mv_matrix = mv_matrix_cube;
155         visualBoids.at(i).proj_matrix = myGraphics.proj_matrix;
156     }
157
158
159
160     prevUpdateTime = currentTime;
161     t = 0.01f; // increment movement variable
162
163 }
164
165 void render(double currentTime) {
166     // Clear viewport - start a new frame.
167     myGraphics.ClearViewport();
168
169     // Draw
170     for (Cube &c : visualBoids){
171         c.Draw();
172     }
173
174
175 }
176
177 void onResizeCallback(GLFWwindow* window, int w, int h) {    // call
      everytime the window is resized
178     myGraphics.windowWidth = w;
179     myGraphics.windowHeight = h;
180
181     myGraphics.aspect = (float)w / (float)h;
182     myGraphics.proj_matrix = glm::perspective(glm::radians(50.0f),
        myGraphics.aspect, 0.1f, 1000.0f);
183 }
184
185 void onKeyCallback(GLFWwindow* window, int key, int scancode, int action,
      int mods) { // called everytime a key is pressed
186     if (key == GLFW_KEY_ESCAPE && action == GLFW_PRESS)
187         glfwSetWindowShouldClose(window, GLFW_TRUE);
188
189     //if (key == GLFW_KEY_LEFT) angleY += 0.05f;
190 }
```