```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 "MyVector.h"
#include "graphics.h"
#include "shapes.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  MyVector position = MyVector(0, 3, -6);
51  MyVector velocity = MyVector(0, 0, 0);
52  MyVector acceleration = MyVector(0, -0.001, 0);
53  float friction = 0.95;
54  MyVector spin = MyVector(0, 0, 1.0);
55
56  bool moving = true;
57
58
59
60  int main()
61  {
62      int errorGraphics = myGraphics.Init();      // Launch window and graphics context
63      if (errorGraphics) return 0;                //Close if something went wrong...
64
65      startup();                                  // Setup all necessary information for startup (aka. load texture, shaders, models, etc).
66
67                                                  // Mixed graphics and update functions - declared in main for simplicity.
68      glfwSetWindowSizeCallback(myGraphics.window, onResizeCallback);          // Set callback for resize
69      glfwSetKeyCallback(myGraphics.window, onKeyCallback);                    // Set Callback for keys
70
71
    // MAIN LOOP run until the window is closed
72      do {
73          double currentTime = glfwGetTime();     // retrieve timelapse
74          glfwPollEvents();                       // poll callbacks
75          update(currentTime);                    // update (physics, animation, structures, etc)
76          render(currentTime);                    // call render function.
77
78          glfwSwapBuffers(myGraphics.window);     // swap buffers (avoid flickering and tearing)
79
80          leftRight = (float)glfwGetKey(myGraphics.window, GLFW_KEY_LEFT);
81
82          running &= (glfwGetKey(myGraphics.window, GLFW_KEY_ESCAPE) == GLFW_RELEASE);  // exit if escape key pressed
83          running &= (glfwWindowShouldClose(myGraphics.window) != GL_TRUE);
84      } while (running);
85
86      myGraphics.endProgram();            // Close and clean everything up...
87
88      cout << "\nPress any key to continue...\n";
89      cin.ignore(); cin.get(); // delay closing console to read debugging errors.
90
91      return 0;
```

```
92  }
93
94  void startup() {
95
96      // Calculate proj_matrix for the first time.
97      myGraphics.aspect = (float)myGraphics.windowWidth / (float)
          myGraphics.windowHeight;
98      myGraphics.proj_matrix = glm::perspective(glm::radians(50.0f),
          myGraphics.aspect, 0.1f, 1000.0f);
99
100
101     mySphere.Load();
102     mySphere.fillColor = glm::vec4(0.0f, 1.0f, 0.0f, 1.0f); // You can
          change the shape fill colour, line colour or linewidth
103
104     myGraphics.SetOptimisations();      // Cull and depth testing
105 }
106
107 void update(double currentTime) {
108
109     if (moving) {
110
111
112         if (position.y < -2.5) {
113             velocity.y = velocity.y * friction;
114             velocity = velocity * -1;
115             spin = spin * -1;
116             if (velocity.y < 0.05) {
117                 moving = false;
118             }
119         }
120
121         position = position + velocity;
122         velocity = velocity + acceleration;
123
124
125         // calculate Sphere movement
126         glm::mat4 mv_matrix_sphere =
127             glm::translate(glm::vec3(position.x, position.y, position.z)) *
128             glm::rotate(-t, glm::vec3(0.0f, 0.0f, spin.z)) *
129             glm::mat4(1.0f);
130         mySphere.mv_matrix = mv_matrix_sphere;
131         mySphere.proj_matrix = myGraphics.proj_matrix;
132
133         t += 0.01f; // increment movement variable
134     }
135 }
136
137 void render(double currentTime) {
138     // Clear viewport - start a new frame.
139     myGraphics.ClearViewport();
140
141     // Draw
```

```
142
143     mySphere.Draw();
144
145 }
146
147 void onResizeCallback(GLFWwindow* window, int w, int h) {   // call
      everytime the window is resized
148     myGraphics.windowWidth = w;
149     myGraphics.windowHeight = h;
150
151     myGraphics.aspect = (float)w / (float)h;
152     myGraphics.proj_matrix = glm::perspective(glm::radians(50.0f),
        myGraphics.aspect, 0.1f, 1000.0f);
153 }
154
155 void onKeyCallback(GLFWwindow* window, int key, int scancode, int action,
      int mods) { // called everytime a key is pressed
156     if (key == GLFW_KEY_ESCAPE && action == GLFW_PRESS)
157         glfwSetWindowShouldClose(window, GLFW_TRUE);
158
159     //if (key == GLFW_KEY_LEFT) angleY += 0.05f;
160 }
```