```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 "ObjectMover.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;

const int numberParticles = 360;
ObjectMover particleEmitter[numberParticles];

float t = 0.001f;           // Global variable for animation
float explosionTimer = 0.0f;

```

```
50  bool mousePressed = false;
51  bool alreadyPressed = false;
52  double mousePosX = 0.0;
53  double mousePosY = 0.0;
54
55  int windowWidth;
56  int windowHeight;
57
58  int main()
59  {
60      int errorGraphics = myGraphics.Init();      // Launch window and graphics context
61      if (errorGraphics) return 0;                //Close if something went wrong...
62
63      startup();                                  // Setup all necessary information for startup (aka. load texture, shaders, models, etc).
64
65                                                  // Mixed graphics and update functions - declared in main for simplicity.
66      glfwSetWindowSizeCallback(myGraphics.window, onResizeCallback);          // Set callback for resize
67      glfwSetKeyCallback(myGraphics.window, onKeyCallback);                    // Set Callback for keys
68
69
    // MAIN LOOP run until the window is closed
70      do {
71          double currentTime = glfwGetTime();     // retrieve timelapse
72          glfwPollEvents();                       // poll callbacks
73          update(currentTime);                    // update (physics, animation, structures, etc)
74          render(currentTime);                    // call render function.
75
76          glfwSwapBuffers(myGraphics.window);     // swap buffers (avoid flickering and tearing)
77
78          mousePressed = glfwGetMouseButton(myGraphics.window, GLFW_MOUSE_BUTTON_1) || mousePressed;
79
80          if (mousePressed && !alreadyPressed) {
81              glfwGetCursorPos(myGraphics.window, &mousePosX, &mousePosY );
82              glfwGetWindowSize(myGraphics.window, &windowWidth, &windowHeight);
83              mousePosX = (mousePosX / windowWidth -0.5);
84              mousePosY = (mousePosY / windowHeight -0.5) * -1;
85              alreadyPressed = true;
86              explosionTimer = t;
87          }
88
89
90          running &= (glfwGetKey(myGraphics.window, GLFW_KEY_ESCAPE) == GLFW_RELEASE);  // exit if escape key pressed
```

```
91          running &= (glfwWindowShouldClose(myGraphics.window) != GL_TRUE);
92      } while (running);
93
94      myGraphics.endProgram();            // Close and clean everything up...
95
96      cout << "\nPress any key to continue...\n";
97      cin.ignore(); cin.get(); // delay closing console to read debugging
          errors.
98
99      return 0;
100 }
101
102 void startup() {
103
104     // Calculate proj_matrix for the first time.
105     myGraphics.aspect = (float)myGraphics.windowWidth / (float)
          myGraphics.windowHeight;
106     myGraphics.proj_matrix = glm::perspective(glm::radians(50.0f),
          myGraphics.aspect, 0.1f, 1000.0f);
107
108     // Load Geometry
109     myCube.Load();
110
111     for (int i = 0; i < numberParticles; i++) {
112         Particle p;
113         p.Load();
114
115         glm::vec3 v = glm::vec3(cos(glm::radians((float)i)), sin
            (glm::radians((float) i)),0.0f);
116
117         particleEmitter[i] = ObjectMover(p, v);
118     }
119
120     myGraphics.SetOptimisations();      // Cull and depth testing
121 }
122
123 void update(double currentTime) {
124     if (mousePressed) {
125         for (int i = 0; i < numberParticles; i++) {
126             glm::mat4 mv_particle =
127                 glm::translate(glm::vec3((float) mousePosX, (float)
                      mousePosY, -6.0f) + (t - explosionTimer ) *
                      particleEmitter[i].v) *
128                 glm::rotate(glm::radians(i + 270.0f), glm::vec3(0.0f, 0.0f,
                      1.0f)) *
129                 glm::scale(glm::vec3(0.1f, 0.1f, 0.1f)) *
130                 glm::mat4(1.0f);
131
132             particleEmitter[i].particle.mv_matrix = mv_particle;
133             particleEmitter[i].particle.proj_matrix =
                  myGraphics.proj_matrix;
134         }
135
```

```
136     }
137     // Calculate Cube movement ( T * R * S ) http://www.opengl-tutorial.org/beginners-tutorials/tutorial-3-matrices/
138     glm::mat4 mv_matrix_cube =
139         glm::translate(glm::vec3(200.0f, 0.0f, -6.0f)) *
140         glm::rotate(glm::radians(180.0f), glm::vec3(0.0f, 0.0f, 1.0f)) *
141         //glm::rotate(t, glm::vec3(1.0f, 0.0f, 0.0f)) *
142         //glm::scale(glm::vec3(0.1f, 0.1f, 0.1f)) *
143         glm::mat4(1.0f);
144     myCube.mv_matrix = mv_matrix_cube;
145     myCube.proj_matrix = myGraphics.proj_matrix;
146
147     t += 0.1f; // increment movement variable
148
149 }
150
151 void render(double currentTime) {
152     // Clear viewport - start a new frame.
153     myGraphics.ClearViewport();
154
155     // Draw
156     myCube.Draw();
157
158     if (mousePressed && t < (explosionTimer + 1)) {
159         for (int i = 0; i < numberParticles; i++) {
160             particleEmitter[i].particle.Draw();
161         }
162     }
163     else {
164         mousePressed = false;
165         alreadyPressed = false;
166     }
167
168 }
169
170 void onResizeCallback(GLFWwindow* window, int w, int h) {   // call everytime the window is resized
171     myGraphics.windowWidth = w;
172     myGraphics.windowHeight = h;
173
174     myGraphics.aspect = (float)w / (float)h;
175     myGraphics.proj_matrix = glm::perspective(glm::radians(50.0f), myGraphics.aspect, 0.1f, 1000.0f);
176 }
177
178 void onKeyCallback(GLFWwindow* window, int key, int scancode, int action, int mods) { // called everytime a key is pressed
179     if (key == GLFW_KEY_ESCAPE && action == GLFW_PRESS)
180         glfwSetWindowShouldClose(window, GLFW_TRUE);
181
182     //if (key == GLFW_KEY_LEFT) angleY += 0.05f;
183 }
```