

Einführung in C++ – Übung 6

Testatgruppe A (Isaak)

Rasmus Diederichsen

21. November 2014

Aufgabe 6.1 Kamera

Listing 1: Camera.hpp

```
1  /**
2   * @file Camera.hpp
3   *
4   * Camera class definition with auxiallary struct.
5   * @author Rasmus Diederichsen
6   */
7  #ifndef CAMERA_HPP
8  #define CAMERA_HPP
9
10 #include <string>
11
12 /**
13  * @struct vector
14  *
15  * @brief Structure representing a vector.
16  */
17 typedef struct
18 {
19     float x;
20     float y;
21     float z;
22 } vector;
23
24 /**
25  * @class Camera
26  *
27  * @brief Class to represent the camera in a GLUT-rendered scene.
28  * Contains functions to set position, motion and speed of the
29  * camera.
30  */
31 class Camera
32 {
33 private:
34     vector m_up;           /**< Look up vector */
35     vector m_trans;       /**< Translation offset from initial pos
36                          */
```

```

35     vector m_l;                /**< Look at point */
36     vector m_rot;              /**< Representation of the Euler
        angles */
37     vector m_initial; /**< Initial position of the camera */
38     float m_turnSpeed;         /**< Tuning speed (in
        radians) */
39     float m_moveSpeed;         /**< Move speed in
        coordinate units */
40
41     /**
42     * Contractor surrogate since C++ has no constructor chaining
43     *
44     * @param x x coordinate of camera position.
45     * @param y y coordinate of camera position.
46     * @param z z coordinate of camera position.
47     */
48     void Init(float x, float y, float z);
49
50 public:
51     /**
52     * the camera sensitivity, i.e. the number of pixels the
53     * mouse pointer mus
54     * move to be registered as a camera motion.
55     */
56     static const double CAM_SENSITIVITY;
57
58     /**
59     * Default constructor. Will set the camera's position to
60     * (0,0,0).
61     */
62     Camera();
63
64     /**
65     * Constructor will set the camera's position to (x,y,z).
66     * @param x The x coordinate of the camera's position.
67     * @param y The y coordinate of the camera's position.
68     * @param z The z coordinate of the camera's position.
69     */
70     Camera(float x, float y, float z);
71
72     /**
73     * Method to translate the camera's current state into a <tt>
74     * gluLookAt()</tt> call.
75     */
76     void apply();
77
78     /**
79     * Move the camera upwards (translation).
80     */
81     void moveUp();
82
83     /**
84     * Move the camera upwards (translation).
85     */
86     void moveDown();

```

```

86     /**
87      * Move the camera forward (translation).
88      */
89     void moveForward();
90
91     /**
92      * Move the camera backward (translation).
93      */
94     void moveBackward();
95
96     /**
97      * Turn the camera left (rotation).
98      */
99     void turnLeft();
100
101     /**
102      * Turn the camera right (rotation).
103      */
104     void turnRight();
105
106     /**
107      * Turn the camera upwards (rotation).
108      */
109     void turnUp();
110
111     /**
112      * Turn the camera downwards (rotation).
113      */
114     void turnDown();
115
116     /**
117      * Adjust the turn speed of the camera.
118      *
119      * @param s The amount added to rotation radians for turning
120      *         motions.
121      */
122     void setTurnSpeed(float s);
123
124     /**
125      * Adjust the translation speed of camera.
126      * @param s The amount added to the translation vector for
127      *         translative
128      *         motions.
129      */
130     void setSpeed(float s);
131
132     /**
133      * Returns a string representation of the camera.
134      *
135      * @return A string holding this camera's current state.
136      */
137     std::string to_string();
138 };
139 #endif /* end of include guard: CAMERA_HPP */

```

Listing 2: Camera.cpp

```

1 #include "Camera.hpp"

```

```

2  #include <cmath>
3  #ifdef __APPLE__
4  #include <OpenGL/gl.h>
5  #include <glut.h>
6  #else
7  #include <GL/gl.h>
8  #include <GL/glut.h>
9  #endif
10 #include <sstream>
11 #include <string>
12
13 const double Camera::CAM_SENSITIVITY = 5.0;
14 const double PH = 1.57079632;
15 void Camera::Init(float x, float y, float z)
16 {
17     vector m_up = { .x = 0, .y = 1, .z = 0 };
18     this->m_up = m_up;
19     vector m_trans = { .x = 0, .y = 0, .z = 0 };
20     this->m_trans = m_trans;
21     vector m_initial = { .x = x, .y = y, .z = z };
22     this->m_initial = m_initial;
23     vector m_rot = { .x = 0, .y = 0, .z = 0 };
24     this->m_rot = m_rot;
25     vector m_l = { .x = 0, .y = 0, .z = -1 };
26     this->m_l = m_l;
27     m_turnSpeed = .05;
28     m_moveSpeed = 5.;
29 }
30 Camera::Camera()
31 {
32     Init(0.,0.,0.);
33 }
34 Camera::Camera(float x, float y, float z)
35 {
36     Init(x,y,z);
37 }
38
39 void Camera::apply()
40 {
41     /* Calc look at vector based on rotation state */
42     m_l.x = m_initial.x + m_trans.x + sin(m_rot.y);
43     m_l.z = -m_initial.z - m_trans.z - cos(m_rot.y);
44     m_l.y = m_initial.y + m_trans.y + sin(m_rot.x);
45
46     /* Clear matrix stack */
47     glLoadIdentity();
48
49     /* Apply transformation */
50     gluLookAt(m_initial.x + m_trans.x, m_initial.y + m_trans.y, -
51             m_initial.z - m_trans.z,
52             m_l.x, m_l.y, m_l.z,
53             m_up.x, m_up.y, m_up.z);
54 }
55
56 std::string Camera::to_string()
57 {

```

```

58     std::stringstream s;
59     s << "Current camera parameters:\n";
60     s << "Position:" << m_trans.x << " " << m_trans.y << " " <<
        m_trans.z << std::endl;
61     s << "Rotation:" << m_rot.x << " " << m_rot.y << " " << m_rot.z
        << std::endl;
62     s << "LookAt:" << m_l.x << " " << m_l.y << " " << m_l.z <<
        std::endl;
63     s << "ViewUp:" << m_up.x << " " << m_up.y << " " << m_up.z <<
        std::endl;
64     return s.str();
65 }
66 void Camera::moveUp()
67 {
68     m_trans.y += m_moveSpeed;
69 }
70
71 void Camera::moveDown()
72 {
73     m_trans.y -= m_moveSpeed;
74 }
75
76 void Camera::moveForward()
77 {
78     m_trans.x += m_moveSpeed * sin(m_rot.y);
79     m_trans.z += m_moveSpeed * cos(m_rot.y);
80 }
81
82 void Camera::moveBackward()
83 {
84     m_trans.x -= m_moveSpeed * sin(m_rot.y);
85     m_trans.z -= m_moveSpeed * cos(m_rot.y);
86 }
87
88 void Camera::turnLeft()
89 {
90     m_rot.y -= m_turnSpeed;
91 }
92
93 void Camera::turnRight()
94 {
95     m_rot.y += m_turnSpeed;
96 }
97
98 void Camera::turnUp()
99 {
100     if(m_rot.x < PH) m_rot.x += m_turnSpeed;
101 }
102
103 void Camera::turnDown()
104 {
105     if(m_rot.x < PH) m_rot.x -= m_turnSpeed;
106 }
107
108 void Camera::setTurnSpeed(float s)
109 {
110     if (s > 0) m_turnSpeed = s;

```

```

111 }
112
113 void Camera::setSpeed(float s)
114 {
115     if (s > 0) m_moveSpeed = s;
116 }

```

Aufgabe 6.2 Fenster

Listing 3: MainWindow.hpp

```

1  /**
2   * @file MainWindow.hpp
3   *
4   * @brief Contains MainWindow class to represent a <tt>GLUT</tt>
        window.
5   * @author Rasmus Diederichsen
6   */
7
8  #ifndef MAINWINDOW_HPP
9  #define MAINWINDOW_HPP
10
11  #include "TriangleMesh.hpp"
12  #include "Camera.hpp"
13
14  /**
15   * @class MainWindow
16   *
17   * @brief Class to represent a <tt>GLUT</tt> window with associated
        methods.
18   * There can be but one instance of this class at a time, hence it
        is made a
19   * singleton.
20   */
21  class MainWindow {
22
23      private:
24
25          /**
26           * Private default constructor.
27           */
28          MainWindow();
29
30          /**
31           * Private copy constructor.
32           */
33          MainWindow(MainWindow const&) {};
34
35          /**
36           * Private -=operator to prevent instance creation by
                assignment.
37           */
38          MainWindow& operator=(MainWindow const&) {};
39
40          Camera *m_cam; /**< The camera for this window. */
41

```

```

42     TriangleMesh *m_mesh; /**< The model (.ply) rendered in this
        window. */
43
44     int m_sizeX, m_sizeY, /**< The window size */
45         m_mouseState, m_mouseButton, /**< last pressed buttons */
46         m_oldX, m_oldY; /**< last recorded mouse pointer *
        coordniates. */
47
48     static MainWindow* window; /**< The singleton object. */
49
50     /**
51      * Initialises the callbacks for <tt>GLUT</tt>.
52      */
53     void initCallbacks();
54
55     /**
56      * Initialises <tt>GLUT</tt> framework.
57      */
58     void initGlut();
59
60 public:
61     /**
62      * Destructor. Deletes the singleton object.
63      */
64     ~MainWindow();
65
66     /**
67      * Acquire the singleton object of this class.
68      * @return The singleton object.
69      */
70     static MainWindow* getInstance();
71
72     /**
73      * Set the camera for this window.
74      * @param cam Pointer to a {@link Camera} for this window.
75      */
76     void setCamera(Camera *cam);
77
78     /**
79      * Set the model rendered in this window.
80      * @param mesh Pointer to a {@link TriangleMesh} for this
        window.
81      */
82     void setMesh(TriangleMesh *mesh);
83
84     /**
85      * Start the rendering process.
86      */
87     static void render();
88
89     /**
90      * Resize the window.
91      * @param x The new horizontal size.
92      * @param y The new vertical size.
93      */
94     static void reshape(int x, int y);
95

```

```

96     /**
97      * Callback for key presses.
98      * @param key The key pressed.
99      * @param x The current mouse pointer position.
100     * @param y The current mouse pointer position.
101     */
102     static void keyPressed(unsigned char key, int x, int y);
103
104     /**
105      * Callback for passive mouse motion.
106      * @param x The current x position.
107      * @param y The current y position.
108      */
109     static void mouseMoved(int x, int y);
110
111     /**
112      * Callback for mouse clicks.
113      * @param button The mouse button pressed.
114      * @param state The state of the button (pressed, release etc
115      *             .).
116      * @param x The current x position.
117      * @param y The current y position.
118      */
119     static void mousePressed(int button, int state, int x, int y)
120     ;
121 };
122
123 #endif /* end of include guard: MAINWINDOW_HPP */

```

Listing 4: MainWindow.cpp

```

1  /**
2   * @file MainWindow.cpp
3   *
4   * @author Rasmus Diederichsen
5   *
6   * Contains the MainWindow class implementation.
7   */
8
9  #include "MainWindow.hpp"
10 #ifdef __APPLE__
11 #include <glut.h>
12 #else
13 #include <GL/glut.h>
14 #endif
15 #include <cmath>
16 #include <iostream>
17
18 MainWindow* MainWindow::window = NULL;
19 MainWindow* MainWindow::getInstance()
20 {
21     if (window == NULL) window = new MainWindow;
22     return window;
23 }
24
25 MainWindow::MainWindow()
26 {
27     m_sizeX = 762;

```



```

28     m_sizeY = 576;
29     m_cam = NULL;
30     m_mesh = NULL;
31     m_oldX = m_oldY = 0;
32     initGlut();
33     initCallbacks();
34 }
35
36 void MainWindow::initGlut()
37 {
38     int dummy1 = 0;
39     char *dummy2 = NULL;
40     glutInit(&dummy1, &dummy2);
41     glutInitDisplayMode(GLUT_DEPTH | GLUT_DOUBLE | GLUT_RGBA);
42     glutInitWindowPosition(100,100);
43     glutInitWindowSize(m_sizeX,m_sizeY);
44     glutCreateWindow("Main Window");
45 }
46
47 void MainWindow::initCallbacks()
48 {
49     glutDisplayFunc(render);
50     glutReshapeFunc(reshape);
51     glutKeyboardFunc(keyPressed);
52     glutMotionFunc(mouseMoved);
53     glutMouseFunc(mousePressed);
54 }
55 void MainWindow::setCamera(Camera *cam)
56 {
57     this->m_cam = cam;
58 }
59 void MainWindow::setMesh(TriangleMesh *mesh)
60 {
61     this->m_mesh = mesh;
62 }
63 void MainWindow::render()
64 {
65     glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
66     /* Set camera position and direction */
67     getInstance()->m_cam->apply();
68     if (getInstance()->m_mesh != NULL)
69     {
70         getInstance()->m_mesh->render();
71         glutSwapBuffers();
72     }
73     else std::cerr << "Error: No mesh to render." << std::endl;
74 }
75 void MainWindow::reshape(int w, int h)
76 {
77     if (h == 0) h = 1;
78
79     float ratio = w * 1.0 / h;
80     glMatrixMode(GL_PROJECTION);
81     glLoadIdentity();
82     glViewport(0, 0, w, h);
83     gluPerspective(45,ratio,1,10000);
84     glMatrixMode(GL_MODELVIEW);

```

```

85 }
86 void MainWindow::keyPressed(unsigned char key, int x, int y)
87 {
88
89     switch(key) {
90         case 'w':
91             getInstance()->m_cam->moveForward();
92             break;
93         case 's':
94             getInstance()->m_cam->moveBackward();
95             break;
96         case 'd':
97             getInstance()->m_cam->turnRight();
98             break;
99         case 'a':
100             getInstance()->m_cam->turnLeft();
101             break;
102         case 't':
103             getInstance()->m_cam->turnUp();
104             break;
105         case 'g':
106             getInstance()->m_cam->turnDown();
107             break;
108     }
109     glutPostRedisplay();
110 }
111
112 void MainWindow::mousePressed(int button, int state, int, int)
113 {
114     getInstance()->m_mouseButton = button;
115     getInstance()->m_mouseState = state;
116 }
117 void MainWindow::mouseMoved(int x, int y)
118 {
119     /* Difference between old and current mouse position */
120     int dx;
121     int dy;
122
123     /* Update mouse coordinates */
124     dx = x - getInstance()->m_oldX;
125     dy = y - getInstance()->m_oldY;
126
127
128     /* Left button controls camera movement */
129     if(getInstance()->m_mouseButton == GLUT_LEFT_BUTTON)
130     {
131         /* Move cam left or right if x coordinates differ */
132         if(fabs(dx) > Camera::CAM_SENSITIVITY)
133         {
134             if(dx < 0) getInstance()->m_cam->turnRight();
135             else getInstance()->m_cam->turnLeft();
136         }
137
138         /* Move forward and backward if y coordinates differ */
139         if(fabs(dy) > Camera::CAM_SENSITIVITY)
140         {
141             if(dy > 0) getInstance()->m_cam->moveForward();

```

```

142         else getInstance()->m_cam->moveBackward();
143     }
144 }
145
146 /* Right button controls head movement */
147 if(getInstance()->m_mouseButton == GLUT_RIGHT_BUTTON)
148 {
149     if(fabs(dy) > Camera::CAM_SENSITIVITY)
150     {
151         if(dy > 0) getInstance()->m_cam->turnUp();
152         else getInstance()->m_cam->turnDown();
153     }
154
155     if(fabs(dx) > Camera::CAM_SENSITIVITY)
156     {
157         if(dx > 0) getInstance()->m_cam->turnRight();
158         else getInstance()->m_cam->turnLeft();
159     }
160 }
161
162 /* Middle button controls height */
163 if(getInstance()->m_mouseButton == GLUT_MIDDLE_BUTTON)
164 {
165     if(fabs(dy) > Camera::CAM_SENSITIVITY)
166     {
167         if(dy > 0) getInstance()->m_cam->moveUp();
168         else getInstance()->m_cam->moveDown();
169     }
170 }
171
172 getInstance()->m_oldX = x;
173 getInstance()->m_oldY = y;
174
175 glutPostRedisplay();
176 }
177
178 MainWindow::~MainWindow()
179 {
180     delete window;
181 }

```

Aufgabe 6.3 Mainfunktion

Listing 5: Main.cpp

```

1  /**
2   * @file Main.cpp
3   *
4   * @brief Main function to start the viewer.
5   * @author Rasmus Diederichsen
6   */
7
8  #include "MainWindow.hpp"
9  #include "Camera.hpp"
10 #include "TriangleMesh.hpp"
11 #ifdef __APPLE__

```

```

12 #include <glut.h>
13 #else
14 #include <GL/glut.h>
15 #endif
16 #include <iostream>
17
18 using std::cout;
19 using std::endl;
20 using std::string;
21
22 int main(int argc, const char **argv)
23 {
24     string s(argv[1]);
25     TriangleMesh mesh(s);
26     Camera cam(0.,0.,-1000.);
27     cout << cam.to_string() << endl;
28     MainWindow::getInstance()->setMesh(&mesh);
29     MainWindow::getInstance()->setCamera(&cam);
30     glutMainLoop();
31     return 0;
32 }

```