Einführung in C++ – Übung 6
Testatgruppe A (Isaak)

Rasmus Diederichsen

19. 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
9  #define CAMERA_HPP
10
11 #include <string>
12
13 /**
14  * @struct vector
15  *
16  * @brief Structure representing a vector.
17  */
18 typedef struct
19 {
20    float x;
21    float y;
22    float z;
23 } vector;
24
25 /**
26  * @class Camera
27  *
28  * @brief Class to represent the camera in a GLUT-rendered scene.
29  * Contains functions to set position, motion and speed of the
        camera.
30  */
31 class Camera
32 {
33    private:
34       vector up;                /**< Look up vector */
```

```
35          vector trans;     /**< Translatation offset from initial pos
                */
36          vector l;         /**< Look at point */
37          vector rot;               /**< Representation of the Euler
                angles */
38          vector initial;   /**< Initial position of the camera */
39          float turn_speed;         /**< Tuning speed (in radians) */
40          float move_speed;         /**< Move speed in coordinate units
                */
41
42          /**
43           * Contructor surrogate since C++ has no constructor chaining
                .
44           *
45           * @param x x coordinate of camera position.
46           * @param y y coordinate of camera position.
47           * @param z z coordinate of camera position.
48           */
49          void Init(float x, float y, float z);
50
51      public:
52          /**
53           * the camera sensitivity, i.e. the number of pixels the
                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
                (0,0,0).
60           */
61          Camera();
62
63          /**
64           * Constructor will set the camera's position to (x,y,z).
65           * @param x The x coordinate of the camera's position.
66           * @param y The y coordinate of the camera's position.
67           * @param z The z coordinate of the camera's position.
68           */
69          Camera(float x, float y, float z);
70
71          /**
72           * Method to translate the camera's current state into a <tt>
73           * gluLookAt()</tt> call.
74           */
75          void apply();
76
77          /**
78           * Move the camera upwards (translation).
79           */
80          void moveUp();
81
82          /**
83           * Move the camera upwards (translation).
84           */
85          void moveDown();
```

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

Listing 2: Camera.cpp

```cpp
#include "Camera.hpp"
#include <cmath>
#ifdef __APPLE__
#include <OpenGL/gl.h>
#include <glut.h>
#else
#include <GL/gl.h>
#include <GL/glut.h>
#endif
#include <sstream>
#include <string>

const double Camera::CAM_SENSITIVITY = 5.0;
const double PH = 1.57079632;
void Camera::Init(float x, float y, float z)
{
   vector up =  { .x = 0, .y = 1, .z = 0 };
   this->up = up;
   vector trans =  { .x = 0, .y = 0, .z = 0 };
   this->trans = trans;
   vector initial =  { .x = x, .y = y, .z = z };
   this->initial = initial;
   vector rot =  { .x = 0, .y = 0, .z = 0 };
   this->rot = rot;
   vector l =  { .x = 0, . y = 0, .z = -1 };
   this->l = l;
   turn_speed = .05;
   move_speed = 5.;
}
Camera::Camera()
{
   Init(0.,0.,0.);
}
Camera::Camera(float x, float y, float z)
{
   Init(x,y,z);
}

void Camera::apply()
{
    /* Calc look at vector based on rotation state */
    l.x =  initial.x + trans.x + sin(rot.y);
    l.z = -initial.z - trans.z - cos(rot.y);
    l.y =  initial.y + trans.y + sin(rot.x);

    /* Clear matrix stack */
    glLoadIdentity();

    /* Apply transformation */
    gluLookAt(initial.x + trans.x, initial.y + trans.y, -initial.z
        - trans.z,
              l.x, l.y, l.z,
              up.x, up.y, up.z);

}

std::string Camera::to_string()
```

```
57  {
58      std::stringstream s;
59      s << "Current␣camera␣parameters:␣\n";
60      s << "Position:␣" << trans.x << "␣" << trans.y << "␣" << trans.z
            << std::endl;
61      s << "Rotation:␣" << rot.x << "␣" << rot.y << "␣" << rot.z <<
            std::endl;
62      s << "LookAt␣␣:␣" << l.x << "␣" << l.y << "␣" << l.z << std::
            endl;
63      s << "ViewUp␣␣:␣" << up.x << "␣" << up.y << "␣" << up.z << std::
            endl;
64      return s.str();
65  }
66  void Camera::moveUp()
67  {
68      trans.y += move_speed;
69  }
70
71  void Camera::moveDown()
72  {
73      trans.y -= move_speed;
74  }
75
76  void Camera::moveForward()
77  {
78      trans.x += move_speed * sin(rot.y);
79      trans.z += move_speed * cos(rot.y);
80  }
81
82  void Camera::moveBackward()
83  {
84      trans.x -= move_speed * sin(rot.y);
85      trans.z -= move_speed * cos(rot.y);
86  }
87
88  void Camera::turnLeft()
89  {
90      rot.y -= turn_speed;
91  }
92
93  void Camera::turnRight()
94  {
95      rot.y += turn_speed;
96  }
97
98  void Camera::turnUp()
99  {
100     if(rot.x < PH) rot.x += turn_speed;
101 }
102
103 void Camera::turnDown()
104 {
105     if(rot.x < PH) rot.x -= turn_speed;
106 }
107
108 void Camera::setTurnSpeed(float s)
109 {
```

```
110     if (s > 0) turn_speed = s;
111 }
112
113 void Camera::setSpeed(float s)
114 {
115     if (s > 0) move_speed = 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 *cam; /**< The camera for this window.  */
```

```
41
42          TriangleMesh *mesh; /**< The model (.ply) rendered in this
                window.  */
43
44          int sizex, sizey, /**< The window size */
45              mouse_state, mouse_button, /**< last pressed buttons */
46              old_x, old_y;/**< 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          * @param x The current x position.
116          * @param y The current y position.
117          */
118         static void mousePressed(int button, int state, int x, int y)
                ;
119 };
120
121 #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      sizex = 762;
28      sizey = 576;
29      cam = NULL;
30      mesh = NULL;
31      old_x = old_y = 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(sizex,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->cam = cam;
58  }
59  void MainWindow::setMesh(TriangleMesh *mesh)
60  {
61      this->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()->cam->apply();
68      if (getInstance()->mesh != NULL)
69      {
70          getInstance()->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()->cam->moveForward();
92              break;
93          case 's':
94              getInstance()->cam->moveBackward();
95              break;
96          case 'd':
97              getInstance()->cam->turnRight();
98              break;
99          case 'a':
100             getInstance()->cam->turnLeft();
101             break;
102         case 't':
103             getInstance()->cam->turnUp();
104             break;
105         case 'g':
106             getInstance()->cam->turnDown();
107             break;
108     }
109     glutPostRedisplay();
110 }
111
112 void MainWindow::mousePressed(int button, int state, int, int)
113 {
114     getInstance()->mouse_button = button;
115     getInstance()->mouse_state = 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()->old_x;
125     dy = y - getInstance()->old_y;
126
127
128     /* Left button controls camera movement */
129     if(getInstance()->mouse_button == 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()->cam->turnRight();
135             else getInstance()->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()->cam->moveForward();
142             else getInstance()->cam->moveBackward();
143         }
144     }
145
146     /* Right button controls head movement */
147     if(getInstance()->mouse_button == GLUT_RIGHT_BUTTON)
148     {
149         if(fabs(dy) > Camera::CAM_SENSITIVITY)
150         {
151             if(dy > 0) getInstance()->cam->turnUp();
152             else getInstance()->cam->turnDown();
153         }
154
155         if(fabs(dx) > Camera::CAM_SENSITIVITY)
156         {
157             if(dx > 0) getInstance()->cam->turnRight();
158             else getInstance()->cam->turnLeft();
159         }
160     }
161
162     /* Middle button contols height */
163     if(getInstance()->mouse_button == GLUT_MIDDLE_BUTTON)
164     {
165         if(fabs(dy) > Camera::CAM_SENSITIVITY)
166         {
167             if(dy > 0) getInstance()->cam->moveUp();
168             else getInstance()->cam->moveDown();
169         }
170     }
171
172     getInstance()->old_x = x;
173     getInstance()->old_y = 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 funtion 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 namespace std;
19
20 int main(int argc, const char **argv)
21 {
22    string s(argv[1]);
23    TriangleMesh mesh(s);
24    Camera cam(0.,0.,-1000.);
25    cout << cam.to_string() << endl;
26    MainWindow::getInstance()->setMesh(&mesh);
27    MainWindow::getInstance()->setCamera(&cam);
28    glutMainLoop();
29    return 0;
30 }
```