Einführung in C++ – Übung 9
Testatgruppe A (Isaak)

Rasmus Diederichsen

17. Dezember 2014

Aufgabe 9.1 Exception-Handling

Fehlerabfragen machen meines Erachtens nur beim Aufruf von `Vertex::normalize()` Sinn, und auch da nur beschränkt. Alle Arrayzugriffe durch den `[]`-operator geschehen durch Literale, weshalb hier ebenfalls kein `try`-Block nötig ist.

src/exceptions/BaseException.hpp

```
/**
 * @file BaseException.hpp
 * @author Rasmus Diederichsen (rdiederichse@uos.de)
 * @version 08.12.2014
 * @brief Contains the BaseException class declaration
 */


#ifndef BASEEXCEPTION_H

#define BASEEXCEPTION_H

#include <stdexcept>
#include <string>

namespace asteroids
{
   /**
    * @class BaseException
    * @brief A base class for exceptions
    */
   class BaseException : public std::runtime_error
   {
      protected:
         /**
          * @brief Constructor
          * @param s Message string
          */
         BaseException(std::string s);
   };
} /* namespace asteroids */
```

```
32
33 #endif /* end of include guard: BASEEXCEPTION_H */
```

src/exceptions/BaseException.cpp

```
1  #include "BaseException.hpp"
2
3  namespace asteroids
4  {
5     BaseException::BaseException(std::string s) : std::runtime_error
          (s) {}
6  } /* namespace BaseException */
```

src/exceptions/DivisionByZeroException.hpp

```
1  /**
2   * @file DivisionByZeroException.hpp
3   * @author Rasmus Diederichsen (rdiederichse@uos.de)
4   * @version 08.12.2014
5   * @brief Contains DivisionByZeroException class declaration
6   */
7
8  #ifndef DIVISIONBYZEROEXCEPTION_H
9
10 #define DIVISIONBYZEROEXCEPTION_H
11
12 #include "BaseException.hpp"
13
14 namespace asteroids
15 {
16    /**
17     * @class DivisionByZeroException
18     * @brief A class for runtime errors due to division by zero.
19     */
20    class DivisionByZeroException : public BaseException
21    {
22       public:
23          /**
24           * @see BaseException
25           */
26          DivisionByZeroException(std::string s);
27    };
28 } /* namespace asteroids */
29
30
31 #endif /* end of include guard: DIVISIONBYZEROEXCEPTION_H */
```

src/exceptions/DivisionByZeroException.cpp

```
1  #include "DivisionByZeroException.hpp"
2
3  namespace asteroids
4  {
5     DivisionByZeroException::DivisionByZeroException(std::string s)
          : BaseException(s) {}
6  } /* namespace asteroids */
```

src/exceptions/OutOfBoundsException.hpp

```
/**
 * @file OutOfBoundsException.hpp
 * @author Rasmus Diederichsen (rdiederichse@uos.de)
 * @version 08.12.2014
 * @brief Contains OutOfBoundsException class declaration
 */

#ifndef OUTOFBOUNDSEXCEPTION_H

#define OUTOFBOUNDSEXCEPTION_H

#include "BaseException.hpp"

namespace asteroids
{
   class OutOfBoundsException : public BaseException
   {
      public:
         /**
          * @see BaseException
          */
         OutOfBoundsException(std::string s);
   };
} /* namespace asteroids */

#endif /* end of include guard: OUTOFBOUNDSEXCEPTION_H */
```

src/exceptions/OutOfBoundsException.cpp

```
#include "OutOfBoundsException.hpp"

namespace asteroids
{
   OutOfBoundsException::OutOfBoundsException(std::string s) :
       BaseException(s) {}
} /* namespace asteroids */
```

src/math/Vertex.cpp

```
#include "Vertex.hpp"
#include <stdio.h>
#include <string>
#include "exceptions/DivisionByZeroException.hpp"
#include "exceptions/OutOfBoundsException.hpp"

namespace asteroids {

Vertex::Vertex()
{
        // Default values
        x = y = z = 0.0;
}


Vertex::Vertex(float _x, float _y, float _z)
{
```

```
18          // Set the given values
19          x = _x;
20          y = _y;
21          z = _z;
22  }
23
24  void Vertex::normalize()
25  {
26          // Normalize the vector
27          float mag2 = x * x + y * y + z * z;
28          if (fabs(mag2 - 1.0f) > TOLERANCE)
29          {
30                  float mag = sqrt(mag2);
31        // to_string is c++ 11
32        if (mag == .0f)
33           throw DivisionByZeroException("Vector to normalise has 0
                 length.");
34        x /= mag;
35        y /= mag;
36        z /= mag;
37     }
38  }
39
40  Vertex Vertex::operator+(const Vertex vec) const
41  {
42     // Add value to value
43     float tx = x + vec.x;
44     float ty = y + vec.y;
45     float tz = z + vec.z;
46     return Vertex(tx, ty, tz);
47  }
48
49  Vertex Vertex::operator-(const Vertex vec) const
50  {
51     // Subtract value from value
52     float tx = x - vec.x;
53     float ty = y - vec.y;
54     float tz = z - vec.z;
55     return Vertex(tx, ty, tz);
56  }
57
58
59  float Vertex::operator[](const int &index) const
60  {
61
62     // Get the wanted value
63     if(index == 0)
64     {
65        return x;
66     }
67
68     if(index == 1)
69     {
70        return y;
71     }
72
73     if(index == 2)
```

```
74      {
75          return z;
76      }
77
78      // to_string is c++ 11
79      throw OutOfBoundsException("Wrong index " + std::to_string(index
            ));
80  }
81
82
83  float& Vertex::operator[](const int &index)
84  {
85
86      if(index == 0)
87      {
88          return x;
89      }
90
91      if(index == 1)
92      {
93          return y;
94      }
95
96      if(index == 2)
97      {
98          return z;
99      }
100
101     throw OutOfBoundsException("Wrong index " + std::to_string(index
            ));
102 }
103
104
105
106
107 float Vertex::operator*(const Vertex vec) const
108 {
109     // Calculate the result
110     return (x * vec.x + y * vec.y + z * vec.z);
111 }
112
113 Vertex Vertex::operator*(float scale) const
114 {
115     // Calculate the result
116     float tx = x * scale;
117     float ty = y * scale;
118     float tz = z * scale;
119     return Vertex(tx, ty, tz);
120 }
121
122 void Vertex::operator+=(Vertex v)
123 {
124     // Add value to value
125     x += v.x;
126     y += v.y;
127     z += v.z;
128 }
```

```
129
130 } // namespace cpp2014
```

src/math/Quaternion.cpp

```
1  #include "Quaternion.hpp"
2  #include "exceptions/DivisionByZeroException.hpp"
3
4  namespace asteroids
5  {
6
7      Quaternion::Quaternion()
8      {
9          // Default Quaternion
10         x = 1.0;
11         y = 0.0;
12         z = 0.0;
13         w = 0.0;
14     }
15
16     Quaternion::~Quaternion()
17     {
18         // Do nothing
19     }
20
21     Quaternion::Quaternion(Vertex vec, float angle)
22     {
23         // Calculate the quaternion
24         fromAxis(vec, angle);
25     }
26
27     Quaternion::Quaternion(float _x, float _y, float _z, float
           _angle)
28     {
29         // Set the values
30         x = _x;
31         y = _y;
32         z = _z;
33         w = _angle;
34     }
35
36     Quaternion::Quaternion(float* vec, float _w)
37     {
38         // Set the values
39         x = vec[0];
40         y = vec[1];
41         z = vec[2];
42         w = _w;
43     }
44
45     void Quaternion::fromAxis(Vertex axis, float angle)
46     {
47         float sinAngle;
48         angle *= 0.5f;
49
50         // Create a copy of the given vector and normalize the new
               vector
51         Vertex vn(axis.x, axis.y, axis.z);
```

```
52          try
53          {
54             vn.normalize();
55          } catch (DivisionByZeroException &divex)
56          {
57             std::cout << divex.what() << std::endl;
58          }
59
60          // Calculate the sinus of the given angle
61          sinAngle = sin(angle);
62
63          // Get the quaternion
64          x = (vn.x * sinAngle);
65          y = (vn.y * sinAngle);
66          z = (vn.z * sinAngle);
67          w = cos(angle);
68       }
69
70       Quaternion Quaternion::getConjugate()
71       {
72          // Conjugate the given quaternion
73          return Quaternion(-x, -y, -z, w);
74       }
75
76
77       Quaternion Quaternion::operator* (const Quaternion rq)
78       {
79          // Calculate the new quaternion
80          return Quaternion(w * rq.x + x * rq.w + y * rq.z - z * rq.y,
81                w * rq.y + y * rq.w + z * rq.x - x * rq.z,
82                w * rq.z + z * rq.w + x * rq.y - y * rq.x,
83                w * rq.w - x * rq.x - y * rq.y - z * rq.z);
84       }
85
86       Vertex Quaternion::operator* (Vertex vec)
87       {
88          // Copy the vector and normalize the new vector
89          Vertex vn(vec);
90          try
91          {
92             vn.normalize();
93          } catch (DivisionByZeroException &divex)
94          {
95             std::cout << divex.what() << std::endl;
96          }
97
98          // Fill the first quaternion and...
99          Quaternion vecQuat, resQuat;
100         vecQuat.x = vn.x;
101         vecQuat.y = vn.y;
102         vecQuat.z = vn.z;
103         vecQuat.w = 0.0f;
104
105         // calculate the new quaternion
106         resQuat = vecQuat * getConjugate();
107         resQuat = *this * resQuat;
108         return (Vertex(resQuat.x, resQuat.y, resQuat.z));
```

```
109     }
110
111 }
```

Aufgabe 9.2 Timestamps und Logging

Timestamp

src/time/Timestamp.hpp

```
1   /**
2    * @file Timestamp.hpp
3    * @author Rasmus Diederichsen (rdiederichse@uos.de)
4    * @version 08.12.2014
5    */
6
7
8   #ifndef TIMESTAMP_H
9
10  #define TIMESTAMP_H
11
12  #include <iostream>
13  #include <sys/time.h>
14
15  namespace asteroids
16  {
17      /**
18       * @class Timestamp
19       * @brief Represents a point in time.
20       */
21      class Timestamp
22      {
23          public:
24              /**
25               * @brief Constructor. Time is initialised to current
                     system time.
26               */
27              Timestamp();
28
29              /**
30               * @brief Get current system time.
31               * @return Current system time in milliseconds (from UNIX
                     epoch)
32               */
33              unsigned long getCurrentTimeInMs() const;
34
35              /**
36               * @brief Get time elapsed since instance creation.
37               * @return The time elapsed since instance creation in
                     milliseconds
38               * (from UNIX epoch)
39               */
40              unsigned long getElapsedTimeInMs() const;
41
42              /**
```

```
43            * @see Timestamp::getCurrentTimeInMs()
44            */
45           unsigned long getCurrentTimeInS() const;
46
47           /**
48            * @see Timestamp::getElapsedTimeInMs()
49            */
50           unsigned long getElapsedTimeInS() const;
51
52           /**
53            * @brief Reset the timer to current system time.
54            */
55           void resetTimer();
56
57           /**
58            * @brief Get string representation of time elapsed since
                 creation.
59            * @return The elapsed time as a string.
60            */
61           std::string getElapsedTime() const;
62
63           /**
64            * @brief Operator to print to a stream.
65            */
66           friend std::ostream& operator<<(std::ostream& os, const
                 Timestamp& ts); // why is friend necessary?
67
68       private:
69           unsigned long m_startTime;
70   };
71 } /* namespace asteroids */
72
73 #endif /* end of include guard: TIMESTAMP_H */
```

src/time/Timestamp.cpp

```
1  #include "Timestamp.hpp"
2  #include <cstddef>
3  #include <stdexcept>
4  #include <sstream>
5
6  namespace asteroids
7  {
8
9      Timestamp::Timestamp()
10     {
11         resetTimer();
12     }
13     unsigned long Timestamp::getCurrentTimeInMs() const
14     {
15         struct timeval tv;
16         struct timezone tz;
17         if (gettimeofday(&tv, &tz) == -1)
18             throw std::runtime_error("Error while attempting to get
                   system time.");
19         return 1000 * tv.tv_sec + (unsigned long) (tv.tv_usec / 1000)
               ;
20     }
```

```
21     unsigned long Timestamp::getElapsedTimeInMs() const
22     {
23        return getCurrentTimeInMs() - m_startTime;
24     }
25     unsigned long Timestamp::getCurrentTimeInS() const
26     {
27        return getCurrentTimeInMs() / 1000;
28     }
29     unsigned long Timestamp::getElapsedTimeInS() const
30     {
31        return getElapsedTimeInMs() / 1000;
32     }
33     void Timestamp::resetTimer()
34     {
35        m_startTime = getCurrentTimeInMs();
36     }
37     std::string Timestamp::getElapsedTime() const
38     {
39        unsigned long elapsed = getElapsedTimeInMs();
40        unsigned long hours = elapsed / (1000 * 60 * 60);
41        elapsed -= hours * 1000 * 60 * 60;
42        unsigned long minutes = elapsed / (1000 * 60);
43        elapsed -= minutes * 1000 * 60;
44        unsigned long seconds = elapsed / 1000;
45        elapsed -= seconds;
46        unsigned long milliseconds = elapsed;
47        char buffer[17];
48        buffer[16] = '\0'; // necessary?
49        sprintf(buffer, "[%02lu:%02lu:%02lu␣-␣%03lu]", hours, minutes
              , seconds, milliseconds);
50        return std::string(buffer);
51     }
52     std::ostream& operator<<(std::ostream& os, const Timestamp& ts)
53     {
54        os << ts.getElapsedTime();
55        return os;
56     }
57  } /* namespace asteroids */
```

Logger

src/logging/Logger.hpp

```
1  /**
2   * @file Logger.hpp
3   * @author Rasmus Diederichsen (rdiederichse@uos.de)
4   * @version 08.12.2014
5   */
6
7  #ifndef LOGGER_H
8
9  #define LOGGER_H
10
11 #include "time/Timestamp.hpp"
12 #include <ostream>
13
14 namespace asteroids
```

```
{
   /**
    * @class Logger
    * @brief Singleton Class to log program events
    */
   class Logger
   {
      public:
         /**
          * @brief Get the singleton instance
          * @return the singleton
          */
         static Logger& instance();

         /**
          * @brief specify destination
          * @param filename File to which log should go.
          */
         void setOutputFile(std::string filename);

         /**
          * @brief reset loggin to stdout
          */
         void setOutputToStdout();

         /**
          * @brief Print log message.
          * @param s Message to log.
          */
         Logger& operator<<(const std::string& s);

         /**
          * @brief Desctructor
          */
         ~Logger();
      private:
         static Logger* logger;
         static Timestamp stamp;
         std::ostream* out;
         Logger();
         Logger(const Logger& l);
         void operator=(const Logger& l);

   };
} /* namespace asteroids */

#endif /* end of include guard: LOGGER_H */
```

src/logging/Logger.cpp

```
#include "Logger.hpp"
#include <iostream>
#include <fstream>
#include <cstddef>
#include <typeinfo>

namespace asteroids
{
```

```
9     Timestamp Logger::stamp;
10    Logger* Logger::logger = NULL;
11    Logger::Logger()
12    {
13       setOutputToStdout();
14    }
15
16    Logger& Logger::operator<<(const std::string& s)
17    {
18       *out << stamp;
19       *out << " - ";
20       *out << s << std::endl;
21       return *this;
22    }
23
24    void Logger::setOutputFile(std::string filename)
25    {
26       out = new std::ofstream(filename.c_str());
27    }
28
29    void Logger::setOutputToStdout()
30    {
31       out = &std::cout;
32    }
33
34    Logger& Logger::instance()
35    {
36       if (logger == NULL)
37          logger = new Logger;
38       return *logger;
39    }
40
41    Logger::~Logger()
42    {
43       if (typeid(*out) == typeid(std::ofstream))
44       {
45          ((std::ofstream*)out)->close();
46          delete out;
47       }
48    }
49 } /* namespace asteroids */
```

Geloggt wurde in der `MainWindow`-Klasse, um die Zeit für das Rendering zu prüfen.

src/view/MainWindow.cpp

```
1  #include "MainWindow.hpp"
2  #include "io/TriangleMeshFactory.hpp"
3  #include "io/Read3DS.hpp"
4  #include "rendering/TexturedMesh.hpp"
5  #include "logging/Logger.hpp"
6
7  using std::cout;
8  using std::endl;
9
10 namespace asteroids
11 {
12
```

```
13  MainWindow* MainWindow::master = 0;
14
15  Camera MainWindow::m_cam;
16
17  MainWindow::MainWindow(string filename)
18  {
19
20          // Save pointer to current instance. Later on we
21          // will create a proper singleton here...
22          MainWindow::master = this;
23
24      // Init glut main window
25      initGlut();
26      initGL();
27
28          for(int i = 0; i < 256; i++)
29          {
30                  m_keyStates[i] = false;
31                  m_specialkeyStates[i] = false;
32          }
33
34
35     /* // Timestamp this point */
36     /* Timestamp t; */
37     Logger::instance() << "Start loading mesh.";
38
39          // Create a triangle mesh instance
40          m_mesh = TriangleMeshFactory::instance().getMesh(filename);
41
42          // Create a sky box. We assume that a model was loaded
                beforehand
43          // to ensure that the base path in texture factory was set
                correctly.
44          // If not set it manually before creating the sky box!!
45          string names[6];
46          names[0] = "box1.ppm";
47          names[1] = "box2.jpg";
48          names[2] = "box3.ppm";
49          names[3] = "box4.jpg";
50          names[4] = "box5.ppm";
51          names[5] = "box6.jpg";
52
53     Logger::instance() << "Start rendering.";
54
55     m_skybox = new Skybox(2048, names);
56
57     Logger::instance() << "Finished rendering.";
58
59     /* // print elapsed time */
60     /* std::cout << t << std::endl; */
61
62     // Call main loop
63     glutMainLoop();
64  }
65
66  MainWindow::~MainWindow()
67  {
```

```
68      // Check if objects exists an delete if necessary
69      if(m_mesh)
70      {
71          delete m_mesh;
72          m_mesh = 0;
73      }
74
75      if(m_skybox)
76      {
77          delete m_skybox;
78          m_skybox = 0;
79      }
80  }
81
82
83  void MainWindow::initGlut()
84  {
85      // Create dummy arguments for command line options
86      int dummy_argc = 1;
87      char *dummy_argv[1];
88      dummy_argv[0] = new char[255];
89      dummy_argv[0] = (char*)"Main Window";
90
91      // Initialize glut toolkit
92      glutInit(&dummy_argc, dummy_argv);
93
94      glutInitDisplayMode(GLUT_DEPTH | GLUT_DOUBLE | GLUT_RGBA);
95      glutInitWindowPosition(100,100);
96      glutInitWindowSize(762, 576);
97      glutCreateWindow("3D-Viewer");
98      glutSetKeyRepeat(1);
99
100     //Register callback functions
101     glutDisplayFunc(MainWindow::callback_render);
102     glutReshapeFunc(MainWindow::callback_reshape);
103     glutMouseFunc(MainWindow::callback_mouse);
104     glutMotionFunc(MainWindow::callback_motion);
105     glutKeyboardFunc(MainWindow::callback_key);
106     glutKeyboardUpFunc(MainWindow::callback_keyUp);
107     glutSpecialFunc(MainWindow::callback_specialkey);
108     glutSpecialUpFunc(MainWindow::callback_specialkeyUp);
109     glutTimerFunc(15, MainWindow::callback_timer, 0);
110     // glutIdleFunc(MainWindow::callback_render);
111
112     // Init OpenGL stuff
113     initGL();
114 }
115
116 void MainWindow::initGL()
117 {
118     glMatrixMode(GL_MODELVIEW);
119     glPolygonMode (GL_FRONT_AND_BACK, GL_FILL);
120
121     // Setup two light sources
122     float light0_position[4];
123     float light0_ambient[4];
124     float light0_diffuse[4];
```

```
125
126     float light1_position[4];
127     float light1_ambient[4];
128     float light1_diffuse[4];
129
130     light0_position[0] =   1.0f; light0_ambient[0] = 0.8f;
            light0_diffuse[0] = 0.8f;
131     light0_position[1] =   1.0f; light0_ambient[1] = 0.8f;
            light0_diffuse[1] = 0.8f;
132     light0_position[2] =   0.0f; light0_ambient[2] = 0.8f;
            light0_diffuse[2] = 0.8f;
133     light0_position[3] =   1.0f; light0_ambient[3] = 0.1f;
            light0_diffuse[3] = 1.0f;
134
135     light1_position[0] =   0.0f; light1_ambient[0] = 0.1f;
            light1_diffuse[0] = 0.5f;
136     light1_position[1] =  -1.0f; light1_ambient[1] = 0.1f;
            light1_diffuse[1] = 0.5f;
137     light1_position[2] =   0.0f; light1_ambient[2] = 0.1f;
            light1_diffuse[2] = 0.5f;
138     light1_position[3] =   1.0f; light1_ambient[3] = 1.0f;
            light1_diffuse[3] = 1.0f;
139
140     // Light 1
141     glLightfv(GL_LIGHT0, GL_AMBIENT,  light0_ambient);
142     glLightfv(GL_LIGHT0, GL_DIFFUSE,  light0_diffuse);
143     glLightfv(GL_LIGHT0, GL_POSITION, light0_position);
144     glEnable(GL_LIGHT0);
145
146     // Light 2
147     glLightfv(GL_LIGHT1, GL_AMBIENT,  light1_ambient);
148     glLightfv(GL_LIGHT1, GL_DIFFUSE,  light1_diffuse);
149     glLightfv(GL_LIGHT1, GL_POSITION, light1_position);
150     glEnable(GL_LIGHT1);
151
152     // Enable lighting
153     glEnable(GL_LIGHTING);
154
155     // Enable z buffer and gouroud shading
156     glEnable(GL_DEPTH_TEST);
157     glDepthFunc(GL_LESS);
158     glShadeModel (GL_SMOOTH);
159 }
160
161
162 void MainWindow::callback_key(unsigned char key, int x, int y)
163 {
164     // Re-route key callback
165     MainWindow::master->keyPressed(key, x, y);
166     glutPostRedisplay();
167 }
168
169 void MainWindow::callback_keyUp(unsigned char key, int x, int y)
170 {
171     // Re-route key callback
172     MainWindow::master->keyUp(key, x, y);
173     glutPostRedisplay();
```

```
174 }
175
176 void MainWindow::callback_specialkey(int key, int x, int y)
177 {
178     // Re-route special key callback
179     MainWindow::master->specialkeyPressed(key, x, y);
180     glutPostRedisplay();
181 }
182
183 void MainWindow::callback_specialkeyUp(int key, int x, int y)
184 {
185     // Re-route special key callback
186     MainWindow::master->specialkeyUp(key, x, y);
187     glutPostRedisplay();
188 }
189
190 void MainWindow::callback_reshape(int w, int h)
191 {
192     // Re-route reshape callback
193     MainWindow::master->reshape(w, h);
194 }
195
196 void MainWindow::callback_render()
197 {
198     // Re-route render callback
199     MainWindow::master->render();
200 }
201
202 void MainWindow::callback_mouse(int button, int state, int x, int y
        )
203 {
204     // Re-route mouse callback
205     MainWindow::master->mousePressed(button, state, x, y);
206     glutPostRedisplay();
207 }
208
209 void MainWindow::callback_motion(int x, int y)
210 {
211     // Re-route motion callback
212     MainWindow::master->mouseMoved(x, y);
213     glutPostRedisplay();
214 }
215
216 void MainWindow::callback_timer (int value)
217 {
218     glutPostRedisplay();
219     glutTimerFunc(15, callback_timer, 0);
220 }
221
222 void MainWindow::keyPressed(unsigned char key, int x, int y)
223 {
224     // State of key is pressed
225     m_keyStates[key] = true;
226 }
227
228 void MainWindow::keyUp (unsigned char key, int x, int y)
229 {
```

```
230     // State of key is unpressed
231     m_keyStates[key] = false;
232 }
233 
234 void MainWindow::specialkeyPressed(int key, int x, int y)
235 {
236     // State of key is pressed
237     m_specialkeyStates[key] = true;
238 }
239 
240 void MainWindow::specialkeyUp(int key, int x, int y)
241 {
242     // State of key is unpressed
243     m_specialkeyStates[key] = false;
244 }
245 
246 void MainWindow::keyOperations(void)
247 {
248     // Controller for moving and rotation
249     if (m_keyStates['q'])
250     {
251         m_mesh->rotate(ROLL, 0.05);
252     }
253 
254     if (m_keyStates['e'])
255     {
256         m_mesh->rotate(ROLL, -0.05);
257     }
258 
259     if (m_keyStates['a'])
260     {
261         m_mesh->rotate(YAW, 0.05);
262     }
263 
264     if (m_keyStates['d'])
265     {
266         m_mesh->rotate(YAW, -0.05);
267     }
268 
269     if (m_keyStates['w'])
270     {
271         m_mesh->rotate(PITCH, 0.05);
272     }
273 
274     if (m_keyStates['s'])
275     {
276         m_mesh->rotate(PITCH, -0.05);
277     }
278 
279     if (m_specialkeyStates[GLUT_KEY_UP])
280     {
281         m_mesh->move(STRAFE, -10);
282     }
283 
284     if (m_specialkeyStates[GLUT_KEY_DOWN])
285     {
286         m_mesh->move(STRAFE, 10);
```

```
287     }
288
289     if (m_specialkeyStates[GLUT_KEY_LEFT])
290     {
291        m_mesh->move(LIFT, 5);
292     }
293
294     if (m_specialkeyStates[GLUT_KEY_RIGHT])
295     {
296        m_mesh->move(LIFT, -5);
297     }
298
299     if (m_specialkeyStates[GLUT_KEY_PAGE_UP])
300     {
301        m_mesh->move(ACCEL, 5);
302     }
303
304     if (m_specialkeyStates[GLUT_KEY_PAGE_DOWN])
305     {
306        m_mesh->move(ACCEL, -5);
307     }
308 }
309
310 void MainWindow::mousePressed(int button, int state, int x, int y)
311 {
312    // Save state and button id. We need this information
313    // when the mouse is moved to call the proper reaction
314    // callback of the camera
315    m_button      = button;
316    m_buttonState = state;
317
318 }
319
320 void MainWindow::mouseMoved(int x, int y)
321 {
322
323    // Get number the number of pixel between the last
324    // und current mouse position
325    int dx = x - m_mouseX;
326    int dy = y - m_mouseY;
327
328    // Check which button was pressend and apply action
329    if(m_button == GLUT_LEFT_BUTTON)
330    {
331       moveXY(dx, dy);
332    }
333
334    if(m_button == GLUT_RIGHT_BUTTON)
335    {
336       moveHead(dx, dy);
337    }
338
339    if(m_button == GLUT_MIDDLE_BUTTON)
340    {
341       moveZ(dy);
342    }
343
```

```
344     // Transform viewport
345     m_cam.apply();
346 
347     // Save new coodinates
348     m_mouseX = x;
349     m_mouseY = y;
350 
351 }
352 
353 void MainWindow::moveXY(int dx, int dy)
354 {
355 
356     if(fabs(dx) > MOUSE_SENSITY)
357     {
358         if(dx > 0)
359         {
360             m_cam.turnRight();
361         }
362 
363         else
364         {
365             m_cam.turnLeft();
366         }
367 
368     }
369 
370     if(fabs(dy) > MOUSE_SENSITY)
371     {
372         if(dy > 0)
373         {
374             m_cam.moveBackward();
375         }
376 
377         else
378         {
379             m_cam.moveForward();
380         }
381     }
382 }
383 
384 void MainWindow::moveZ(int dy)
385 {
386 
387     if(fabs(dy) > MOUSE_SENSITY)
388     {
389         if(dy > 0)
390         {
391             m_cam.moveUp();
392         }
393 
394         else
395         {
396             m_cam.moveDown();
397         }
398     }
399 }
400 
```

```
401 void MainWindow::moveHead(int dx, int dy)
402 {
403
404     if(fabs(dy) > MOUSE_SENSITY)
405     {
406         if(dy > 0)
407         {
408             m_cam.turnUp();
409         }
410
411         else
412         {
413             m_cam.turnDown();
414         }
415     }
416
417     if(fabs(dx) > MOUSE_SENSITY)
418     {
419         if(dx > 0)
420         {
421             m_cam.turnRight();
422         }
423
424         else
425         {
426             m_cam.turnLeft();
427         }
428     }
429 }
430
431
432 void MainWindow::reshape(int w, int h)
433 {
434     if(h == 0)
435     {
436         h = 1;
437     }
438
439     float ratio = 1.0* w / h;
440
441     // Reset the coordinate system before modifying
442     glMatrixMode(GL_PROJECTION);
443     glLoadIdentity();
444
445     // Set the viewport to be the entire window
446     glViewport(0, 0, w, h);
447
448     // Set the correct perspective.
449     gluPerspective(45,ratio,1,100000);
450     glMatrixMode(GL_MODELVIEW);
451
452     // Set 'LookAt'
453     m_cam.apply();
454 }
455
456 void MainWindow::render()
457 {
```

```
458     MainWindow::keyOperations();
459     // Set black background color
460     glClearColor(0.0, 0.0, 0.0, 0.0);
461 
462     // Clear bg color and enable depth test (z-Buffer)
463     glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
464 
465     // Render objects
466     if(m_skybox)
467     {
468        m_skybox->render();
469     }
470 
471     if(m_mesh)
472     {
473        m_mesh->render();
474     }
475 
476 
477 
478     glFinish();
479 
480     // Call back buffer
481     glutSwapBuffers();
482 }
483 
484 }
```