5. Übungsblatt - C++
Gruppe D10

Henrik Gerdes, Manuel Eversmeyer

26. November 2018

```
/*
 *  Main.cpp
 *
 *  Created on: Nov. 04 2018
 *      Author: Thomas Wiemann
 *
 *  Copyright (c) 2018 Thomas Wiemann.
 *  Restricted usage. Licensed for participants of the course "The C++ Programming Language" only.
 *  No unauthorized distribution.
 */

#include "MainWindow.hpp"

int main(int argc, char** argv)
{
    string buffer = "../models/arrow.ply";
    asteroids::MainWindow mainWindow("Asteroids", buffer, 1024, 768);
    //asteroids::MainWindow mainWindow("Asteroids", argv[1], 1024, 768);
    mainWindow.execute();


    return 0;
}
```

Main.cpp

```
/*
 * Matrix.cpp
 *
 *  @date 18.11.2018
 *  @author Thomas Wiemann
 *
 *  Copyright (c) 2018 Thomas Wiemann.
 *  Restricted usage. Licensed for participants of the course "The C++ Programming Language" only.
 *  No unauthorized distribution.
 */

#include "Matrix.hpp"

namespace asteroids
{
```

```
16
17 Matrix::Matrix()
18 {
19     for(int i = 0; i < MatrixSize; i++)
20     {
21         m[i] = 0;
22     }
23     m[0] = m[5] = m[10] = m[15] = 1;
24
25     initializedOk = true;
26 }
27 Matrix::Matrix(Vector axis, float angle)
28 {
29     if(fabs(angle)<0.0001)
30     {
31
32     }
33 }
34
35 Matrix::Matrix(float* array)
36 {
37     for(int i = 0; i < MatrixSize; i++)
38     {
39         m[i] = array[i];
40     }
41
42     initializedOk =  true;
43 }
44
45 Matrix::Matrix(const Matrix &other)
46 {
47     for(int i = 0; i < MatrixSize; i++)
48     {
49         m[i] = other.m[i];
50     }
51     initializedOk = true;
52 }
53
54 int Matrix::determinate()
55 {
56     return 0;
57 }
58
59 Matrix& Matrix::operator=(const Matrix& other)
60 {
61     for(int i = 0; i < MatrixSize; i++)
62     {
63         this->m[i] = other.m[i];
64     }
65
66     return *this;
67 }
68
69 Matrix Matrix::operator*(const Matrix& other) const
70 {
71     Matrix tmp;
72     for(int i = 0; i < 16; i++) tmp.m[i] = 0;
73     for (int i = 0; i < 4; i ++)
```

```
74      {
75          for (int j = 0; j < 4; j ++)
76          {
77              for (int k = 0; k < 4; k ++)
78              {
79                  tmp.m[i * 4 + j] += this->m[i * 4 + k] * other.m[k * 4 + j
                        ];
80              }
81          }
82      }
83      return tmp;
84  }
85
86  Matrix& Matrix::operator*=(const Matrix& other)
87  {
88      *this = *this * other;
89      return *this;
90  }
91
92  Matrix Matrix::operator+(const Matrix& other) const
93  {
94      Matrix tmp;
95      for(int i = 0; i < MatrixSize; i++)
96      {
97          tmp.m[i] = this->m[i] + other.m[i];
98      }
99
100     return tmp;
101 }
102
103 Matrix& Matrix::operator+=(const Matrix& other)
104 {
105     *this = *this + other;
106     return *this;
107 }
108
109 Matrix Matrix::operator-(const Matrix& other) const
110 {
111     Matrix tmp;
112     for(int i = 0; i < MatrixSize; i++)
113     {
114         tmp.m[i] = this->m[i] - other.m[i];
115     }
116
117     return tmp;
118 }
119
120 Matrix& Matrix::operator-=(const Matrix& other)
121 {
122     *this = *this - other;
123     return *this;
124 }
125
126 Matrix Matrix::operator*(const float scal) const
127 {
128     Matrix tmp;
129     for(int i = 0; i < MatrixSize; i++)
130     {
```

```
131             tmp.m[i] = this->m[i] * scal;
132         }
133         return tmp;
134 }
135
136 Matrix& Matrix::operator*=(const float scal)
137 {
138         *this = *this * scal;
139         return *this;
140 }
141
142 Matrix Matrix::operator/(const float scal) const
143 {
144         return *this * (1/scal);
145 }
146
147 Matrix& Matrix::operator/=(const float scal)
148 {
149         *this = *this / scal;
150         return *this;
151 }
152
153 Vector Matrix::operator*(const Vector& vec)const
154 {
155         Vector tmp;
156         float val[4];
157         for(int i = 0; i < 4; i++)
158         {
159             val[i] = vec.x * this->m[i * 4] + vec.y * this->m[i*4 + 1] + vec.z
160                 * this->m[i*4 + 2] + 1 * this->m[i*4 + 3];
161         }
162         tmp.x = val[0];
163         tmp.y = val[1];
164         tmp.z = val[2];
165         return tmp;
166 }
167
168
169
170 Vector& Matrix::operator*=(Vector& vec)const
171 {
172         vec = *this * vec;
173         return vec;
174 }
175
176 float& Matrix::operator[](const int index)
177 {
178         /*Fuer [][] brauchte man eine proxyklasse*/
179         if(index >= 0 && index < 4)
180         {
181             return this->m[4*index];
182         }
183
184         cout << "Error, out of Matrix bounds" << endl;
185         return m[0];
186 }
187
188 void Matrix::printMatrix()
```

```
{
    int i;
    for(i = 0; i < MatrixSize; i++)
    {
        if(i%4==0)
        {
            cout << endl;
        }
        cout << m[i] << " ";
    }
    cout << endl;
}



Matrix::~Matrix(){}


} // namespace asteroids
```

Matrix.cpp

```
/**
 *  @file Vector.cpp
 *
 *  @date 18.11.2018
 *  @author Thomas Wiemann
 *
 *  Copyright (c) 2018 Thomas Wiemann.
 *  Restricted usage. Licensed for participants of the course "The C++
    Programming Language" only.
 *  No unauthorized distribution.
 */

#include "Vector.hpp"

namespace asteroids {

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


Vector::Vector(float _x, float _y, float _z)
{
    // Set the given values
    x = _x;
    y = _y;
    z = _z;
}

void Vector::normalize()
{
    // Normalize the vector
    float mag2 = x * x + y * y + z * z;
    if (fabs(mag2 - 1.0f) > 0.00001)
```

```
36      {
37          float mag = sqrt(mag2);
38          x /= mag;
39          y /= mag;
40          z /= mag;
41      }
42  }
43
44  Vector Vector::operator+(const Vector& other)const
45  {
46      Vector tmp;
47      tmp.x = this->x + other.x;
48      tmp.y = this->y + other.y;
49      tmp.z = this->z + other.z;
50
51      return tmp;
52  }
53
54  Vector& Vector::operator+=(const Vector& other)
55  {
56      *this = *this + other;
57      return *this;
58  }
59
60  Vector Vector::operator-(const Vector& other)const
61  {
62      Vector tmp;
63      tmp.x = this->x - other.x;
64      tmp.y = this->y - other.y;
65      tmp.z = this->z - other.z;
66
67      return tmp;
68  }
69
70  Vector& Vector::operator-=(const Vector& other)
71  {
72      *this = *this - other;
73      return *this;
74  }
75
76  Vector Vector::operator*(const float scale)const
77  {
78      Vector tmp;
79      tmp.x = this->x * scale;
80      tmp.y = this->y * scale;
81      tmp.z = this->z * scale;
82
83      return tmp;
84  }
85
86  Vector& Vector::operator*=(const float scale)
87  {
88      *this = *this * scale;
89      return *this;
90  }
91
92  Vector Vector::operator/(const float scale)const
93  {
```

```
94      return *this * (1/scale);
95  }
96
97  Vector& Vector::operator/=(const float scale)
98  {
99      *this = *this / scale;
100     return *this;
101 }
102
103 float Vector::operator[](const int& index) const
104 {
105     if(index >= 0 && index<4)
106     {
107         if(index == 0)
108         {
109             return x;
110         }
111         if(index == 1)
112         {
113             return y;
114         }
115         if(index == 2)
116         {
117             return z;
118         }
119     }
120     std::cerr << "Out of Vector bounds << " << std::endl;
121     return 0;
122 }
123
124 double Vector::scalar()
125 {
126     return sqrt(this->x * this->x + this->y * this->y + this->z * this->z);
127 }
128 void Vector::print()
129 {
130     std::cout << "(" << x << ", " << y << ", " << z << ")" << std::endl;
131 }
132
133
134
135 } // namespace asteroids
```

Vector.cpp

```
1  #include "MainWindow.hpp"
2
3  int main(int argc, char const *argv[])
4  {
5      float test1[16] = { 1,2,3,4,
6                          5,6,7,8,
7                          9,10,11,12,
8                          13,14,15,16};
9      float test2[16] = { 1,2,3,4,
10                         5,6,7,8,
11                         9,10,11,12,
12                         13,14,15,17};
13
```

```
    asteroids::Matrix tm1(test1);
    asteroids::Matrix tm2(test2);
    asteroids::Vector tv1(1,2,3);
    asteroids::Vector tv2(4,3,6);
    asteroids::Quaternion tq1(1,2,3,4);
    asteroids::Quaternion tq2(5,3,6,7);

    /*Matrix - Matrix Tests*/
    (tm1 + tm2).printMatrix();
    (tm1 - tm2).printMatrix();
    (tm1 * tm2).printMatrix();
    (tm1 += tm2).printMatrix();
    (tm1 -= tm2).printMatrix();
    (tm1 *= tm2).printMatrix();
    (tm1 = tm2).printMatrix();
    (tm1 * 5.0).printMatrix();
    (tm1 / 5.0).printMatrix();
    (tm1 *= 5.0).printMatrix();
    (tm1 /= 5.0).printMatrix();

    /*Vector - Vector Tests*/
    (tv1 * 5.0).print();
    (tv1 / 5.0).print();
    (tv1 + tv2).print();
    (tv1 - tv2).print();
    (tv1 += tv2).print();
    (tv1 -= tv2).print();
    (tv1 *= 5.0).print();
    (tv1 /= 5.0).print();

    /*Matrix - Vector Tests*/
    (tm1 * tv1).print();

    /*Quaternion Test*/
    (tq1 * tq2).print();
    (tq1 *= tq2).print();
    (tq1 * tv1).print();
    (tq1 *= tv1).print();


    return 0;
}
```

Test.cpp

```
/**
 *  @file Quaternion.cpp
 *
 *  @date 18.11.2018
 *  @author Thomas Wiemann
 *
 *  Copyright (c) 2018 Thomas Wiemann.
 *  Restricted usage. Licensed for participants of the course "The C++
    Programming Language" only.
 *  No unauthorized distribution.
 */

#include "Quaternion.hpp"
```

```
13
14 namespace asteroids
15 {
16
17 Quaternion::Quaternion()
18 {
19     // Default Quaternion
20     x = 1.0;
21     y = 0.0;
22     z = 0.0;
23     w = 0.0;
24 }
25
26 Quaternion::~Quaternion()
27 {
28     // Do nothing
29 }
30
31 Quaternion::Quaternion(const Vector& vec, float angle)
32 {
33     // Calculate the quaternion
34     fromAxis(vec, angle);
35 }
36
37 Quaternion::Quaternion(float _x, float _y, float _z, float _angle)
38 {
39     // Set the values
40     x = _x;
41     y = _y;
42     z = _z;
43     w = _angle;
44 }
45
46 Quaternion::Quaternion(float* vec, float _w)
47 {
48     // Set the values
49     x = vec[0];
50     y = vec[1];
51     z = vec[2];
52     w = _w;
53 }
54
55 Quaternion::Quaternion(const Quaternion& other)
56 {
57     w = other.w;
58     x = other.x;
59     y = other.y;
60     z = other.z;
61 }
62
63 void Quaternion::fromAxis(const Vector& axis, float angle)
64 {
65     float sinAngle;
66     angle *= 0.5f;
67
68     // Create a copy of the given vector and normalize the new vector
69     Vector vn(axis.x, axis.y, axis.z);
70     vn.normalize();
```

```
71
72      // Calculate the sinus of the given angle
73      sinAngle = sin(angle);
74
75      // Get the quaternion
76      x = (vn.x * sinAngle);
77      y = (vn.y * sinAngle);
78      z = (vn.z * sinAngle);
79      w = cos(angle);
80  }
81
82  Quaternion& Quaternion::operator=(const Quaternion &other)
83  {
84      this->x = other.x;
85      this->y = other.y;
86      this->z = other.z;
87      this->w = other.w;
88
89      return *this;
90  }
91
92  Quaternion Quaternion::operator*(const Quaternion& rq)const
93  {
94      return Quaternion(  w * rq.x + x * rq.w + y * rq.z - z * rq.y,
95                          w * rq.y + y * rq.w + z * rq.x - x * rq.z,
96                          w * rq.z + z * rq.w + x * rq.y - y * rq.x,
97                          w * rq.w - x * rq.x - y * rq.y - z * rq.z);
98  }
99
100 Quaternion& Quaternion::operator*=(const Quaternion& rq)
101 {
102     *this = *this * rq;
103     return *this;
104 }
105
106 Vector Quaternion::operator* (const Vector& vector) const
107 {
108     Vector vn(vector);
109     vn.normalize();
110     Quaternion vecQuat, resQuat;
111     vecQuat.x = vn.x;
112     vecQuat.y = vn.y;
113     vecQuat.z = vn.z;
114     vecQuat.w = 0.0f;
115
116     resQuat = vecQuat * Quaternion(-x, -y, -z, w);
117     resQuat = *this * resQuat;
118     return (Vector(resQuat.x, resQuat.y, resQuat.z));
119 }
120
121 void Quaternion::print()
122 {
123     std::cout << "(" << this->x << ", " << this->y << ", " << this->z
124         << ", " << this->w << ")" << std::endl;
125 }
126
127 Vector& Quaternion::operator*=(Vector& vector) const
128 {
```

```
    vector = *this * vector;
    return vector;
}


}
```

Quaternion.cpp

```
/*
 *  Model.cpp
 *
 *  Created on: Nov. 04 2018
 *      Author: Thomas Wiemann
 *
 *  Copyright (c) 2018 Thomas Wiemann.
 *  Restricted usage. Licensed for participants of the course "The C++ Programming Language" only.
 *  No unauthorized distribution.
 */

#include "Model.hpp"
#include "PLYIO.hpp"

#include <iostream>

/* OpenGL / glew Headers */
#define GL3_PROTOTYPES 1
#include <GL/glew.h>

namespace asteroids
{

Model::Model()
{
    // Init member variables
    m_numFaces      = 0;
    m_numVertices   = 0;
    m_vertexBuffer  = 0;
    m_indexBuffer   = 0;

    // Setup rotation and position
    m_xAxis         = Vector(1.0, 0.0, 0.0);
    m_yAxis         = Vector(0.0, 1.0, 0.0);
    m_zAxis         = Vector(0.0, 0.0, 1.0);
    m_position      = Vector(0.0, 0.0, 0.0);

    // Init initial position
    initTransformations();
}

Model::Model(const Model& other)
{
    m_numFaces = other.m_numFaces;
    m_numVertices = other.m_numVertices;
    m_vertexBuffer = new float[3 * m_numVertices];
    m_indexBuffer = new int[3 * m_numFaces];

```

```
49      m_xAxis = other.m_xAxis;
50      m_yAxis = other.m_yAxis;
51      m_zAxis = other.m_zAxis;
52      m_position = other.m_position;
53      m_rotation = other.m_rotation;
54      m_transformation = other.m_transformation;
55  }
56
57  Model::Model(int* faces, float* vertices, int a, int b)
58  {
59      // Save mesh information and buffers
60      m_numFaces       = a;
61      m_numVertices    = b;
62      m_vertexBuffer   = vertices;
63      m_indexBuffer    = faces;
64
65      // Init initial position
66      initTransformations();
67  }
68
69  Model::Model(const std::string& filename)
70  {
71      LoadPLY(
72          filename,
73          m_vertexBuffer, m_indexBuffer,
74          m_numVertices, m_numFaces);
75
76      // Init initial position
77      initTransformations();
78  }
79
80  void Model::initTransformations()
81   {
82      // Setup rotation and position
83      m_xAxis          = Vector(1.0, 0.0, 0.0);
84      m_yAxis          = Vector(0.0, 1.0, 0.0);
85      m_zAxis          = Vector(0.0, 0.0, 1.0);
86      m_position       = Vector(0.0, 0.0, 0.0);
87      m_rotation.fromAxis(Vector(0.0, 0.0, 1.0), 0.0f);
88
89   }
90
91  void Model::rotate(ACTION axis, float s)
92  {
93      Quaternion qr;
94      switch(axis)
95      {
96      case PITCH: qr.fromAxis(m_yAxis, s);
97                  m_xAxis = qr * m_xAxis;
98                  m_zAxis = qr * m_zAxis; break;
99
100     case YAW:   qr.fromAxis(m_xAxis, s);
101                 m_yAxis = qr * m_yAxis;
102                 m_zAxis = qr * m_zAxis; break;
103
104     case ROLL:  qr.fromAxis(m_zAxis, s);
105                 m_yAxis = qr * m_yAxis;
106                 m_xAxis = qr * m_xAxis; break;
```

```
107
108     default: std::cout << "Will handel that key later" << std::endl;
109     }
110 }
111
112 void Model::move(ACTION axis, float speed)
113 {
114     switch (axis)
115     {
116         case ACCEL:
117             m_position += m_xAxis * -speed;
118             break;
119         case STRAFE:
120             m_position += m_yAxis * -speed;
121             break;
122         case LIFT:
123             m_position += m_zAxis * speed;
124             break;
125         default:
126             std::cout << "Bewegungsrichtung nicht definiert" << std::endl;
127             break;
128     }
129 }
130
131 void Model::computeMatrix()
132 {
133     float* matrix_trans = m_transformation.getData();
134
135     matrix_trans[0]  = m_yAxis[0];
136     matrix_trans[1]  = m_yAxis[1];
137     matrix_trans[2]  = m_yAxis[2];
138
139     matrix_trans[4]  = m_xAxis[0];
140     matrix_trans[5]  = m_xAxis[1];
141     matrix_trans[6]  = m_xAxis[2];
142
143     matrix_trans[8]  = m_zAxis[0];
144     matrix_trans[9]  = m_zAxis[1];
145     matrix_trans[10] = m_zAxis[2];
146
147     matrix_trans[12] = m_position[0];
148     matrix_trans[13] = m_position[1];
149     matrix_trans[14] = m_position[2];
150 }
151
152 void Model::printModelInformation()
153 {
154     std::cout << "Model statistics: " << std::endl;
155     std::cout << "Num vertices: " << m_numVertices << std::endl;
156     std::cout << "Num faces:    " << m_numFaces << std::endl;
157 }
158
159 void Model::printBuffers()
160 {
161     for(int i = 0; i < m_numVertices; i++)
162     {
163         std::cout << "v: " << m_vertexBuffer[3 * i]     << " "
164                            << m_vertexBuffer[3 * i + 1] << " "
```

```
165                             << m_vertexBuffer[3 * i + 2] << std::endl;
166     }
167     for(int i = 0; i < m_numFaces; i++)
168     {
169         std::cout << "f: " << m_indexBuffer[3 * i]     << " "
170                             << m_indexBuffer[3 * i + 1] << " "
171                             << m_indexBuffer[3 * i + 2] << std::endl;
172     }
173 }
174
175 void Model::render()
176 {
177     // Compute transformation matrix
178     computeMatrix();
179     glPushMatrix();
180     glMultMatrixf(m_transformation.getData());
181     // Render mesh
182     for(int i = 0; i < m_numFaces; i++)
183     {
184         // Get position og current triangle in buffer
185         int index = 3 * i;
186
187         // Get vertex indices of triangle vertices
188         int a = 3 * m_indexBuffer[index];
189         int b = 3 * m_indexBuffer[index + 1];
190         int c = 3 * m_indexBuffer[index + 2];
191
192         // Render wireframe model
193         glBegin(GL_LINE_LOOP);
194         glColor3f(1.0, 1.0, 1.0);
195         glVertex3f(m_vertexBuffer[a], m_vertexBuffer[a + 1], m_vertexBuffer
                [a + 2]);
196         glVertex3f(m_vertexBuffer[b], m_vertexBuffer[b + 1], m_vertexBuffer
                [b + 2]);
197         glVertex3f(m_vertexBuffer[c], m_vertexBuffer[c + 1], m_vertexBuffer
                [c + 2]);
198         glEnd();
199
200     }
201     glPopMatrix();
202 }
203
204 Model::~Model()
205 {
206     if(m_vertexBuffer)
207     {
208         delete[] m_vertexBuffer;
209     }
210     if(m_indexBuffer)
211     {
212         delete[] m_indexBuffer;
213     }
214 }
215 } // namespace asteroids
```

Model.cpp