6. Übungsblatt - C++
Gruppe D10

Henrik Gerdes, Manuel Eversmeyer

4. Dezember 2018

```
/*
 * Matrix.hpp
 *
 *  @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.
 */

#ifndef MATRIX_H_
#define MATRIX_H_

#include <iostream>
#include <fstream>
#include <iomanip>

#include "Vector.hpp"
#include "exceptions/OutOfBoundsException.hpp"
#include "exceptions/DivisionByZeroException.hpp"

#define _USE_MATH_DEFINES
#include <cmath>

#ifndef M_PI
#define M_PI 3.141592654
#endif
using namespace std;

namespace asteroids{

/**
 * @brief   A 4x4 matrix class implementation for use with the provided
 *          Vector types.
 */

class Matrix {
public:
    class Proxy
    {
        public:
```

```
43          /**
44           * @brief   Constructor fpr proxy
45           * @param v Gets the n-line of Matrix
46           */
47          Proxy(float* f): line(f){}
48          /**
49           * @biref   Gets the number at the index
50           *          performs bound check
51           */
52          float& operator[](int i);
53
54          private:
55          //to Save the line
56          float* line;
57      };
58      /**
59       * @brief   Default constructor. Initializes a identity matrix.
60       */
61      Matrix();
62
63      /**
64       * @brief   Initializes a matrix wit the given data array. Ensure
65       *          that the array has exactly 16 fields.
66       */
67      Matrix(float* matrix);
68      /**
69       * @brief   Initializes a matrix wit the given data array. Ensure
70       *          that the array has exactly 16 fields.
71       */
72      Matrix(const float* matrix);
73      /**
74       * @brief   Copy constructor.
75       */
76      Matrix(const Matrix& other);
77
78      /**
79       * @brief   Constructs a matrix from given axis and angle. Trys to
80       *          avoid a gimbal lock.
81       */
82      Matrix(Vector axis, float angle);
83
84      Matrix(const Vector &position, const Vector &angles);
85
86
87      ~Matrix();
88
89      /**
90       * @brief   Scales the matrix elemnts by the given factor
91       */
92
93      Matrix operator*(const float &scale) const;
94      /**
95       * @brief   Matrix-Matrix multiplication. Returns the new
96       *          matrix
97       */
98
99      Matrix operator*(const Matrix &other) const;
100     /**
```

```
101       * @brief   Matrix addition operator. Returns a new matrix
102       *
103       */
104
105      Matrix operator+(const Matrix &other) const;
106      /**
107       * @brief   Matrix addition operator
108       */
109
110      Matrix operator+=(const Matrix &other);
111
112      /**
113       * @brief   Matrix-Matrix multiplication (array based). Mainly
114       *          implemented for compatibility with other math libs.
115       *          ensure that the used array has at least 16 elements
116       *          to avoid memory access violations.
117       */
118
119      Matrix operator*(const float* &other) const;
120      /**
121       * @brief   Multiplication of Matrix and Vector types
122       */
123
124      Vector operator*(const Vector &v) const;
125
126     /**
127       * @brief   Sets the given index of the Matrix's data field
128       *          to the provided value.
129       *
130       * @param   i       Field index of the matrix
131       * @param   value   new value
132       */
133      void set(int i, float value);
134
135      /**
136       * @brief   Transposes the current matrix
137       */
138      void transpose();
139
140      /**
141       * @brief   Computes an Euler representation (x, y, z) plus three
142       *          rotation values in rad. Rotations are with respect to
143       *          the x, y, z axes.
144       */
145      void toPostionAngle(float pose[6]);
146
147
148      /**
149       * @brief   Matrix scaling with self assignment.
150       */
151      void operator*=(const float scale);
152
153      /**
154       * @brief   Matrix-Matrix multiplication with self assigment.
155       */
156      void operator*=(const Matrix& other);
157
158      /**
```

```
159      * @brief   Matrix-Matrix multiplication (array based). See \ref{
             operator*}.
160      */
161     void operator*=(const float* other);
162 
163     /**
164      * @brief   Returns the element at the given index.
165      */
166     float at(const int i) const;
167 
168     /**
169      * @brief   Indexed element (reading) access.
170      */
171     float operator[](const int index) const;
172 
173     /**
174      * @brief   Returns Proxy with float* to n-line
175      */
176     Proxy operator[](const int index);
177 
178     /**
179      * @brief   Returns a new matrix that will be scalled by scal.
180      *          Will not override the original matrix.
181      * @param scal The float to scale with.
182      */
183     Matrix operator/(const float scal) const;
184 
185     /**
186      * @brief   Returns a matrix that will be scalled by scal.
187      *          Will override the original (this) matrix.
188      * @param scal The float to scale with.
189      */
190     Matrix& operator/=(const float scal);
191 
192     /**
193      * @brief   Returns the matrix's determinant
194      */
195     float det();
196     /**
197     * @brief    Returns the internal data array. Unsafe. Will probably
198     *           removed in one of the next versions.
199     */
200     inline float* getData(){ return m;};
201 
202     /**
203      * @brief   Inverts the matrix. Success is true if operation was
             successful
204      */
205     Matrix inv(bool& success);
206 
207     /// Assignment operator
208     Matrix& operator=(const Matrix& other);
209 
210 private:
211 
212     /**
213      * @brief   Returns a sub matrix without row \ref i and column \ref j.
214      */
```

```
215     void submat(float* submat, int i, int j);
216 
217     /**
218      * @brief     Calculates the determinant of a 3x3 matrix
219      *
220      * @param     M  input 3x3 matrix
221      * @return    determinant of input matrix
222      */
223     float det3( const float *M );
224 
225 
226     /// Internal data array
227     float m[16];
228 
229     Proxy* tmp;
230 };
231 
232 /**
233  * @brief Output operator for matrices.
234  */
235 
236 inline ostream& operator<<(ostream& os, const Matrix matrix){
237     os << "Matrix:" << endl;
238     os << fixed;
239     for(int i = 0; i < 16; i++){
240         os << setprecision(4) << matrix[i] << " ";
241         if(i % 4 == 3) os << " " <<  endl;
242     }
243     os << endl;
244     return os;
245 }
246 
247 
248 } // namespace asteroids
249 #endif /* MATRIX_H_ */
```

Matrix.hpp

```
1  /*
2   * Matrix.cpp
3   *
4   *  @date 18.11.2018
5   *  @author Thomas Wiemann
6   *
7   *  Copyright (c) 2018 Thomas Wiemann.
8   *  Restricted usage. Licensed for participants of the course "The C++
        Programming Language" only.
9   *  No unauthorized distribution.
10  */
11 
12 #include "Matrix.hpp"
13 
14 namespace asteroids
15 {
16 
17 Matrix::Matrix()
18 {
19     for(int i = 0; i < 16; i++) m[i] = 0;
```

```
20      m[0] = m[5] = m[10] = m[15] = 1;
21  }
22
23
24  Matrix::Matrix(float* matrix)
25  {
26      for(int i = 0; i < 16; i++) m[i] = matrix[i];
27  }
28
29  Matrix::Matrix(const float* matrix)
30  {
31      for(int i = 0; i < 16; i++) m[i] = matrix[i];
32  }
33
34
35  Matrix::Matrix(const Matrix& other)
36  {
37      for(int i = 0; i < 16; i++) m[i] = other.m[i];
38  }
39
40  Matrix::Matrix(Vector axis, float angle)
41  {
42      // Check for gimbal lock
43      if(fabs(angle) < 0.0001)
44      {
45
46          bool invert_z = axis.z < 0;
47
48          //Angle to yz-plane
49          float pitch = atan2(axis.z, axis.x) - M_PI_2;
50          if(pitch < 0.0f) pitch += 2.0f * M_PI;
51
52          if(axis.x == 0.0f && axis.z == 0.0) pitch = 0.0f;
53
54          //Transform axis into yz-plane
55          axis.x =  axis.x * cos(pitch) + axis.z * sin(pitch);
56          axis.z = -axis.x * sin(pitch) + axis.z * cos(pitch);
57
58          //Angle to y-Axis
59          float yaw = atan2(axis.y, axis.z);
60          if(yaw < 0) yaw += 2 * M_PI;
61
62          Matrix m1, m2, m3;
63
64          if(invert_z) yaw = -yaw;
65
66          cout << "YAW: " << yaw << " PITCH: " << pitch << endl;
67
68          if(fabs(yaw)   > 0.0001){
69              m2 = Matrix(Vector(1.0, 0.0, 0.0), yaw);
70              m3 = m3 * m2;
71          }
72
73          if(fabs(pitch) > 0.0001){
74              m1 = Matrix(Vector(0.0, 1.0, 0.0), pitch);
75              m3 = m3 * m1;
76          }
77
```

```
78          *this = m3;
79          /*this = m3 shuld be the smae as
80          for(int i = 0; i < 16; i++) m[i] = m3[i];
81          */
82
83      } else {
84          float c = cos(angle);
85          float s = sin(angle);
86          float t = 1.0f - c;
87          float tmp1, tmp2;
88
89          // Normalize axis
90          Vector a(axis);
91          a.normalize();
92
93          m[ 0] = c + a.x * a.x * t;
94          m[ 5] = c + a.y * a.y * t;
95          m[10] = c + a.z * a.z * t;
96
97          tmp1 = a.x * a.y * t;
98          tmp2 = a.z * s;
99          m[ 4] = tmp1 + tmp2;
100         m[ 1] = tmp1 - tmp2;
101
102         tmp1 = a.x * a.z * t;
103         tmp2 = a.y * s;
104         m[ 8] = tmp1 - tmp2;
105         m[ 2] = tmp1 + tmp2;
106
107         tmp1 = a.y * a.z * t;
108         tmp2 = a.x * s;
109         m[ 9] = tmp1 + tmp2;
110         m[ 6] = tmp1 - tmp2;
111
112         m[ 3] = m[ 7] = m[11] = 0.0;
113         m[12] = m[13] = m[14] = 0.0;
114         m[15] = 1.0;
115     }
116 }
117
118
119 Matrix::Matrix(const Vector &position, const Vector &angles)
120 {
121     float sx = sin(angles[0]);
122     float cx = cos(angles[0]);
123     float sy = sin(angles[1]);
124     float cy = cos(angles[1]);
125     float sz = sin(angles[2]);
126     float cz = cos(angles[2]);
127
128     m[0]  = cy*cz;
129     m[1]  = sx*sy*cz + cx*sz;
130     m[2]  = -cx*sy*cz + sx*sz;
131     m[3]  = 0.0;
132     m[4]  = -cy*sz;
133     m[5]  = -sx*sy*sz + cx*cz;
134     m[6]  = cx*sy*sz + sx*cz;
135     m[7]  = 0.0;
```

```
136     m[8]  = sy;
137     m[9]  = -sx*cy;
138     m[10] = cx*cy;
139 
140     m[11] = 0.0;
141 
142     m[12] = position[0];
143     m[13] = position[1];
144     m[14] = position[2];
145     m[15] = 1;
146 }
147 
148 Matrix::~Matrix(){}
149 
150 Matrix& Matrix::operator=(const Matrix& other)
151 {
152     if(this != &other)
153     {
154         for(int i = 0; i < 16; i++)
155         {
156             m[i] = other.m[i];
157         }
158         return *this;
159     }
160     else
161     {
162         return *this;
163     }
164 }
165 
166 Matrix Matrix::operator*(const float &scale) const
167 {
168     float new_matrix[16];
169     for(int i = 0; i < 16; i++){
170         new_matrix[i] = m[i] * scale;
171     }
172     return Matrix(new_matrix);
173 }
174 
175 Matrix Matrix::operator*(const Matrix &other) const
176 {
177     Matrix tmp;
178     for(int i = 0; i < 16; i++) tmp.m[i] = 0;
179     for (int i = 0; i < 4; i ++)
180     {
181         for (int j = 0; j < 4; j ++)
182         {
183             for (int k = 0; k < 4; k ++)
184             {
185                 tmp.m[i * 4 + j] += this->m[i * 4 + k] * other.m[k * 4 + j
                        ];
186             }
187         }
188     }
189     return tmp;
190 }
191 
192 Matrix Matrix::operator+(const Matrix &other) const
```

```
193 {
194     float new_matrix[16];
195     for(int i = 0; i < 16; i++)
196     {
197         new_matrix[i] = m[i] + other.m[i];
198     }
199     return Matrix(new_matrix);
200 }
201
202
203 Matrix Matrix::operator+=(const Matrix &other)
204 {
205      return *this + other;
206 }
207
208 Matrix Matrix::operator*(const float* &other) const
209 {
210     Matrix tmp(other);
211     return *this * tmp;
212 }
213
214 Vector Matrix::operator*(const Vector &v) const
215 {
216     float x = m[ 0] * v.x + m[ 4] * v.y + m[8 ] * v.z;
217     float y = m[ 1] * v.x + m[ 5] * v.y + m[9 ] * v.z;
218     float z = m[ 2] * v.x + m[ 6] * v.y + m[10] * v.z;
219
220     x = x + m[12];
221     y = y + m[13];
222     z = z + m[14];
223
224     return Vector(x, y, z);
225 }
226
227
228 /**
229  * @brief   Transposes the current matrix
230  */
231 void Matrix::transpose()
232 {
233     float m_tmp[16];
234     m_tmp[0]  = m[0];
235     m_tmp[4]  = m[1];
236     m_tmp[8]  = m[2];
237     m_tmp[12] = m[3];
238     m_tmp[1]  = m[4];
239     m_tmp[5]  = m[5];
240     m_tmp[9]  = m[6];
241     m_tmp[13] = m[7];
242     m_tmp[2]  = m[8];
243     m_tmp[6]  = m[9];
244     m_tmp[10] = m[10];
245     m_tmp[14] = m[11];
246     m_tmp[3]  = m[12];
247     m_tmp[7]  = m[13];
248     m_tmp[11] = m[14];
249     m_tmp[15] = m[15];
250     for(int i = 0; i < 16; i++) m[i] = m_tmp[i];
```

```
251 }
252
253 /**
254  * @brief   Computes an Euler representation (x, y, z) plus three
255  *          rotation values in rad. Rotations are with respect to
256  *          the x, y, z axes.
257  */
258 void Matrix::toPostionAngle(float pose[6])
259 {
260     if(pose != 0){
261         float _trX, _trY;
262         if(m[0] > 0.0) {
263             pose[4] = asin(m[8]);
264         } else {
265             pose[4] = (float)M_PI - asin(m[8]);
266         }
267         // rPosTheta[1] =  asin( m[8]);       // Calculate Y-axis angle
268
269         float  C    =  cos( pose[4] );
270         if ( fabs( C ) > 0.005 )  {           // Gimball lock?
271             _trX      =  m[10] / C;           // No, so get X-axis angle
272             _trY      =  -m[9] / C;
273             pose[3]  = atan2( _trY, _trX );
274             _trX      =  m[0] / C;            // Get Z-axis angle
275             _trY      = -m[4] / C;
276             pose[5]  = atan2( _trY, _trX );
277         } else {                              // Gimball lock has occurred
278             pose[3] = 0.0;                    // Set X-axis angle to zero
279             _trX      =  m[5];  //1           // And calculate Z-axis angle
280             _trY      =  m[1];  //2
281             pose[5]  = atan2( _trY, _trX );
282         }
283
284         pose[0] = m[12];
285         pose[1] = m[13];
286         pose[2] = m[14];
287     }
288 }
289
290 void Matrix::operator*=(const float scale)
291 {
292     *this = *this * scale;
293 }
294
295 void Matrix::operator*=(const Matrix& other)
296 {
297     *this = *this * other;
298 }
299
300 void Matrix::operator*=(const float* other)
301 {
302     *this = *this * other;
303 }
304
305 Matrix Matrix::operator/(const float scal) const
306 {
307     if(scal == 0)
308     {
```

```
309         throw DivisionByZeroException("Matrixdivision durch 0", "Matrix /
                float");
310     }
311     return *this * (1/scal);
312 }
313
314 Matrix& Matrix::operator/=(const float scal)
315 {
316     if(scal == 0)
317     {
318         throw DivisionByZeroException("Matrixdivision durch 0", "Matrix /=
                float");
319     }
320     *this = *this / scal;
321     return *this;
322 }
323
324
325 /**
326  * @brief    Returns a Proxy that hollds a float*.
327  *           Performs bound check.
328  */
329 Matrix::Proxy Matrix::operator[](const int index)
330 {
331     if(index > 3 || index < 0)
332     {
333         throw OutOfBoundsException("Fehler beim Zugriff der ersten Mattrix-
                Dimension", index);
334     }
335     return Proxy(m + 4 * index);
336 }
337
338 float& Matrix::Proxy::operator[](const int index)
339 {
340     if(index > 3 || index < 0)
341     {
342         throw OutOfBoundsException("Fehler beim Zugriff der zweiten Matrix-
                Dimension" , index);
343     }
344     return line[index];
345 }
346
347 /**
348  * @brief    Returns the matrix's determinant
349  */
350 float Matrix::det()
351 {
352     float det, result = 0, i = 1.0;
353     float Msub3[9];
354     int     n;
355     for ( n = 0; n < 4; n++, i *= -1.0 ) {
356         submat( Msub3, 0, n );
357         det     = det3( Msub3 );
358         result += m[n] * det * i;
359     }
360     return( result );
361 }
362
```

```
363 Matrix Matrix::inv(bool& success)
364 {
365     Matrix Mout;
366     float  mdet = det();
367     if(mdet == 0)
368     {
369         throw DivisionByZeroException("Divison druch 0", "Beim berechner
            der Determinante");
370     }
371     if ( fabs( mdet ) < 0.00000000000005 ) {
372         cout << "Error matrix inverting! " << mdet << endl;
373         return Mout;
374     }
375     float  mtemp[9];
376     int     i, j, sign;
377     for ( i = 0; i < 4; i++ ) {
378         for ( j = 0; j < 4; j++ ) {
379             sign = 1 - ( (i +j) % 2 ) * 2;
380             submat( mtemp, i, j );
381             Mout[j*4][i] = ( det3( mtemp ) * sign ) / mdet;
382 
383             /*Should do the same*/
384             //Mout[i+j*4] = ( det3( mtemp ) * sign ) / mdet;
385         }
386     }
387     return Mout;
388 }
389 
390 
391 /**
392  * @brief   Returns a sub matrix without row \ref i and column \ref j.
393  */
394 void Matrix::submat(float* submat, int i, int j)
395 {
396     int di, dj, si, sj;
397     // loop through 3x3 submatrix
398     for( di = 0; di < 3; di ++ ) {
399         for( dj = 0; dj < 3; dj ++ ) {
400             // map 3x3 element (destination) to 4x4 element (source)
401             si = di + ( ( di >= i ) ? 1 : 0 );
402             sj = dj + ( ( dj >= j ) ? 1 : 0 );
403             // copy element
404             submat[di * 3 + dj] = m[si * 4 + sj];
405         }
406     }
407 }
408 
409 /**
410  * @brief    Calculates the determinant of a 3x3 matrix
411  *
412  * @param    M  input 3x3 matrix
413  * @return   determinant of input matrix
414  */
415 float Matrix::det3(const float *M )
416 {
417     float det;
418     det = (double)(  M[0] * ( M[4]*M[8] - M[7]*M[5] )
419                    - M[1] * ( M[3]*M[8] - M[6]*M[5] )
```

```
420                     + M[2] * ( M[3]*M[7] - M[6]*M[4] ));
421     return ( det );
422 }
423
424 } // namespace asteroids
```

Matrix.cpp

```
1  #ifndef __BASEEXCEPTION_HPP__
2  #define __BASEEXCEPTION_HPP__
3
4  #include <stdexcept>
5  #include <sstream>
6
7  class BaseException: public std::exception
8  {
9      public:
10     BaseException();
11     BaseException(const char* _msg);
12     inline const char* what(){return msg;};
13     ~BaseException() noexcept;
14
15
16     protected:
17     const char* msg = "Es ist ein Feher aufgetreten!";
18
19 };
20 #endif
```

exceptions/BaseException.hpp

```
1  #include "BaseException.hpp"
2
3  BaseException::BaseException(const char* _msg):
4      std::exception(),
5      msg(_msg)
6  {
7  }
8
9  BaseException::~BaseException(){}
```

exceptions/BaseException.cpp

```
1  #ifndef __OUTOFBOUNDSEXCEPTION_HPP__
2  #define __OUTOFBOUNDSEXCEPTION_HPP__
3
4  #include "BaseException.hpp"
5
6  class OutOfBoundsException: public BaseException
7  {
8      int index;
9      public:
10     OutOfBoundsException(const char* _obj);
11     OutOfBoundsException(const char* _obj, const int _index);
12     inline int getIndex(){return index;};
13     const char* getInfo();
14 };
15
```

```
#endif
```

exceptions/OutOfBoundsException.hpp

```
#include "OutOfBoundsException.hpp"

OutOfBoundsException::OutOfBoundsException(const char* _obj):BaseException(
    _obj)
{
}

OutOfBoundsException::OutOfBoundsException(const char* _obj, const int
    _index):
    BaseException(_obj),
    index(_index)
{
}

const char* OutOfBoundsException::getInfo()
{
    std::ostringstream str;
    str << "Es ist ein Dimensionsfehler aufgetreten.\nFehlermeldung: " <<
        msg
        << " at Index: " << index;
    return str.str().c_str();
}
```

exceptions/OutOfBoundsException.cpp

```
#ifndef __DIVISIONBYZEROEXCEPTION_HPP__
#define __DIVISIONBYZEROEXCEPTION_HPP__

#include "BaseException.hpp"

class DivisionByZeroException: public BaseException
{
    const char* obj;
    public:
    DivisionByZeroException(const char* _obj);
    DivisionByZeroException(const char* _obj, const char* _inFunction);
    inline const char* getFunktion(){return obj;};
    const char* getInfo();
};

#endif
```

exceptions/DivisionByZeroException.hpp

```
#include "DivisionByZeroException.hpp"

DivisionByZeroException::DivisionByZeroException(const char* _obj):
    BaseException(_obj)
{
}

DivisionByZeroException::DivisionByZeroException(const char* _obj, const
    char* _inFunction):
    BaseException(_obj),
```

```
    obj(_inFunction)
{
}

const char* DivisionByZeroException::getInfo()
{
    std::ostringstream str;
    str << "Es wurde versucht durch 0 zu Teilen.\nFehlermeldung: " << msg
        << " Fehler in in Funktion " << obj << " aufgetreten.";
    return str.str().c_str();
}
```

exceptions/DivisionByZeroException.cpp

```
#ifndef __RENDERABLE_HPP__
#define __RENDERABLE_HPP__

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

namespace asteroids
{
    /**
    * @brief Interface class for all objects that support rendering.
    */
    class Renderable
    {
        public:
            /**
             * @brief   Virtual function that exery extending class has to
                implement
             */
            virtual void render() = 0;

            /**
             * @brief   Set the Color object. Interfacefunctin
             *
             * @param r float-value of read
             * @param g float-value of green
             * @param b float-value of blue
             */
            virtual void setColor(float r, float g, float b) = 0;
    };

}

#endif
```

renderOBJ/Renderable.hpp

```
#ifndef __RENDERABLE3D_HPP__
#define __RENDERABLE3D_HPP__

#include "Renderable.hpp"

namespace asteroids
{

```

```
/**
 * @brief Base class for al 3D renderable objects
 *
 */
class Renderable3D : public Renderable
{
    public:

        /**
         * @brief Virtual function for sub class
         *
         */
        virtual void render() = 0;

        /**
         * @brief   Set the Color object. Interfacefunctin
         *
         * @param r float-value of read
         * @param g float-value of green
         * @param b float-value of blue
         */
        void setColor(float r, float g, float b);

    protected:
        /**
         * @brief Get the Color R object
         *
         * @return float for Color value
         */
        inline float getColorR(){return color3D[0];}

        /**
         * @brief Get the Color G object
         *
         * @return float for Color value
         */
        inline float getColorG(){return color3D[1];}

        /**
         * @brief Get the Color B object
         *
         * @return float for Color value
         */
        inline float getColorB(){return color3D[2];}

    private:

        //Save the color value. Default: whites
        float color3D[3] = {1.0f,1.0f,1.0f};
};

}// namespace asteroids

#endif
```

renderOBJ/Renderable3D.hpp

```
#include "Renderable3D.hpp"
```

```
2
3
4  namespace asteroids
5  {
6
7  void Renderable3D::setColor(float r, float g, float b)
8  {
9      color3D[0] = r;
10     color3D[1] = g;
11     color3D[2] = b;
12 }
13
14 }
```

renderOBJ/Renderable3D.cpp

```
1  #ifndef __SPHERE_HPP__
2  #define __SPHERE_HPP__
3
4  #include "Renderable3D.hpp"
5  #include "../Vector.hpp"
6
7  namespace asteroids
8  {
9
10 class Sphere: public Renderable3D
11 {
12     public:
13         /**
14          * @brief Construct a new Sphere object
15          *
16          * @param position      Initial position of the sphere
17          * @param radius        Radius
18          * @param numSides      Number of horizontal intersections
19          * @param numStack      Number of vertical intersections
20          */
21         Sphere(const Vector& position, float radius, int numSides = 10, int
                numStacks = 10);
22
23         /**
24          * @brief Renders a sphere in OpenGl
25          *
26          */
27         void render();
28
29         /**
30          * @brief Destroy the Sphere object
31          */
32         ~Sphere();
33
34
35     private:
36         // Position
37         Vector m_position;
38
39         // Radius
40         float m_radius;
41
```

```
42          int m_numSides;
43
44          int m_numStacks;
45  };
46
47  }
48
49  #endif
```

renderOBJ/Sphere.hpp

```
1   #include "Sphere.hpp"
2
3   namespace asteroids
4   {
5
6   Sphere::Sphere(const Vector& position, float radius, int numSides, int
        numStack)
7   {
8       m_position = position;
9       m_radius = radius;
10      m_numSides = numSides;
11      m_numStacks = numStack;
12  }
13
14  void Sphere::render()
15  {
16
17      float curRadius, curTheta, curRho, deltaTheta, deltaRho, curX,curY,curZ
            ;
18      int curStack, curSlice, numVerts = (m_numStacks-1)*m_numSides;
19      Vector points[numVerts];
20      int curVert = 0;
21      int t;
22
23      deltaTheta = (2*M_PI) / m_numSides;
24      deltaRho = M_PI / m_numStacks;
25
26          for (curStack=1; curStack< m_numStacks; curStack++)
27          {
28              curRho = (3.141/2.0) - curStack*deltaRho;
29              curY = sin(curRho) * m_radius;
30              curRadius = cos(curRho) * m_radius;
31              for (curSlice=0; curSlice< m_numSides; curSlice++)
32              {
33                  curTheta = curSlice * deltaTheta;
34                  curX = curRadius * cos(curTheta);
35                  curZ = -curRadius * sin(curTheta);
36                  points[curVert++] = Vector(curX,curY,curZ);
37              }
38          }
39
40          glBegin(GL_TRIANGLE_FAN);
41          //Added color fpr Shpere
42          glColor3f(getColorR(), getColorG(), getColorB());
43          glNormal3d(0,1,0);
44          glVertex3d(0, m_radius,0);
45          for (t=0; t< m_numSides; t++)
```

```
46          {
47              curX = points[t].x;
48              curY = points[t].y;
49              curZ = points[t].z;
50              glNormal3d(curX, curY, curZ);
51              glVertex3d(curX, curY, curZ);
52          }
53              curX = points[0].x;
54              curY = points[0].y;
55              curZ = points[0].z;
56          glNormal3d(curX, curY, curZ);
57          glVertex3d(curX, curY, curZ);
58          glEnd();
59
60          int vertIndex;
61      for (curStack=0; curStack< m_numStacks-2; curStack++)
62      {
63          vertIndex = curStack * m_numSides;
64          glBegin(GL_QUAD_STRIP);
65          //Added color cor Shpere
66          glColor3f(getColorR(), getColorG(), getColorB());
67              for (curSlice=0; curSlice< m_numSides; curSlice++)
68              {
69                  glNormal3d(points[vertIndex+curSlice].x, points[vertIndex+
                        curSlice].y, points[vertIndex+curSlice].z);
70                  glVertex3d(points[vertIndex+curSlice].x, points[vertIndex+
                        curSlice].y, points[vertIndex+curSlice].z);
71
72                  glNormal3d(points[vertIndex+ m_numSides + curSlice].x,
                        points[vertIndex+m_numSides+curSlice].y, points[
                        vertIndex+m_numSides+curSlice].z);
73                  glVertex3d(points[vertIndex+ m_numSides + curSlice].x,
                        points[vertIndex+m_numSides+curSlice].y, points[
                        vertIndex+m_numSides+curSlice].z);
74              }
75              glNormal3d(points[vertIndex].x, points[vertIndex].y, points[
                    vertIndex].z);
76              glVertex3d(points[vertIndex].x, points[vertIndex].y, points[
                    vertIndex].z);
77              glNormal3d(points[vertIndex+ m_numSides].x, points[vertIndex+
                    m_numSides].y, points[vertIndex+m_numSides].z);
78              glVertex3d(points[vertIndex+ m_numSides].x, points[vertIndex+
                    m_numSides].y, points[vertIndex+m_numSides].z);
79          glEnd();
80      }
81
82      glBegin(GL_TRIANGLE_FAN);
83          glNormal3d(0,-1,0);
84          glVertex3d(0,- m_radius,0);
85          for (t=0; t< m_numSides-1; t++)
86          {
87              curX = points[numVerts-1-t].x;
88              curY = points[numVerts-1-t].y;
89              curZ = points[numVerts-1-t].z;
90              glNormal3d(curX, curY, curZ);
91              glVertex3d(curX, curY, curZ);
92          }
93              curX = points[numVerts-1].x;
```

```
94              curY = points[numVerts-1].y;
95              curZ = points[numVerts-1].z;
96          glNormal3d(curX, curY, curZ);
97          glVertex3d(curX, curY, curZ);
98      glEnd();
99  }
100
101 // void Sphere::setColor(float r, float g, float b)
102 // {
103 //      color3D[0] = r;
104 //      color3D[1] = g;
105 //      color3D[2] = b;
106 // }
107
108 Sphere::~Sphere()
109 {
110     //DO Nothing
111 }
112
113 }
```

renderOBJ/Sphere.cpp

```
1  #ifndef __RENDERABLE2D_HPP__
2  #define __RENDERABLE2D_HPP__
3
4  #include "Renderable.hpp"
5
6  namespace asteroids
7  {
8
9  class MainWindow;
10
11 class Renderable2D : public Renderable
12 {
13     public:
14
15         /**
16          * @brief Construct a new Renderable 2D object
17          *
18          * @param mainWindow Pointer the MainWindow for dimensions
19          */
20         Renderable2D(MainWindow* mainWindow);
21
22         /**
23          * @brief   Context change form 3D to 2D.
24          *          Accsessable for subclasses
25          */
26         void prerender();
27
28         /**
29          * @brief   Context change form 2D to 3D.
30          *          Accsessable for subclasses
31          */
32         void postrender();
33
34         /**
35          * @brief Virtual function for subclasses
```

```
36          *
37          */
38         virtual void render() = 0;
39
40         /**
41          * @brief Set the Color object
42          *
43          * @param r Value for red
44          * @param g Value for green
45          * @param b Value for blue
46          */
47         void setColor(float r, float g, float b);
48
49     protected:
50
51         /**
52          * @brief Get the Color R object
53          *
54          * @return float of red
55          */
56         inline float getColorR(){return color2D[0];}
57
58         /**
59          * @brief Get the Color G object
60          *
61          * @return float of green
62          */
63         inline float getColorG(){return color2D[1];}
64
65         /**
66          * @brief Get the Color B object
67          *
68          * @return float blue
69          */
70         inline float getColorB(){return color2D[2];}
71
72     private:
73         MainWindow* m_window;
74
75         float color2D[3] = {1.0,0.0,0.0};
76 };
77
78 }// namespace asteroids
79
80 #endif
```

renderOBJ/Renderable2D.hpp

```
1  #include "Renderable2D.hpp"
2  #include "../MainWindow.hpp"
3
4  namespace asteroids
5  {
6
7  Renderable2D::Renderable2D(MainWindow* mainWindow)
8  {
9      m_window = mainWindow;
10 }
```

```
11
12 void Renderable2D::prerender()
13 {
14     // Enter modelview mode and save current view
15     // matrix. Set transformation to indentity to
16     // 'undo' current look at transformation
17     glMatrixMode(GL_MODELVIEW);
18     glPushMatrix();
19     glLoadIdentity();
20
21     // Enter projection mode and set ortho projection
22     // according to current window size
23     glMatrixMode(GL_PROJECTION);
24     glPushMatrix();
25     glLoadIdentity();
26     glOrtho(0.0f, m_window->width(), m_window->height(), 0.0f, -10.0f, 10.0
           f);
27
28 }
29
30 void Renderable2D::postrender()
31 {
32     // Delete current ortho projection, enter model
33     // view mode and restore previous look at matrix
34     glPopMatrix();
35     glMatrixMode(GL_MODELVIEW);
36     glPopMatrix();
37 }
38
39 void Renderable2D::setColor(float r, float g, float b)
40 {
41     color2D[0] = r;
42     color2D[1] = g;
43     color2D[2] = b;
44 }
45
46 }// namespace asteroids
```

renderOBJ/Renderable2D.cpp

```
1 #ifndef __CIRCLE_HPP__
2 #define __CIRCLE_HPP__
3
4 #include "Renderable2D.hpp"
5
6 #define _USE_MATH_DEFINES
7 #include <cmath>
8
9 namespace asteroids
10 {
11
12 class Circle : public Renderable2D
13 {
14     public:
15         /**
16          * @brief Construct a new Circle object
17          *
18          * @param _mainWindow   Pointer to a MainWindow for high and wigth
```

```
19          * @param x Absult x-value for position
20          * @param y Absult y-value for position
21          * @param radius    Radius of the Circle
22          * @param segments
23          */
24         Circle(MainWindow* _mainWindow, float x, float y, float radius, int
                segments);
25
26         /**
27          * @brief Renders a Circle
28          *
29          */
30         virtual void render();
31
32     private:
33         // Position x
34         float m_x;
35
36         //Position y
37         float m_y;
38
39         //Radius
40         float m_radius;
41
42         int m_segments;
43 };
44
45 }
46
47 #endif
```

renderOBJ/Circle.hpp

```
1  #include "Circle.hpp"
2
3  namespace asteroids
4  {
5
6  Circle::Circle(MainWindow* _mainWindow, float x, float y, float radius, int
       segments):Renderable2D(_mainWindow)
7  {
8      m_x = x;
9      m_y = y;
10     m_radius = radius;
11     m_segments = segments;
12 }
13
14 void Circle::render()
15 {
16     Renderable2D::prerender();
17
18     float theta = 2 * 3.1415926 / float(m_segments);
19     float tangetial_factor = tanf(theta);//calculate the tangential factor
20
21     float radial_factor = cosf(theta);//calculate the radial factor
22
23     float x = m_radius;//we start at angle = 0
24
```

```
25      float y = 0;
26
27      glBegin(GL_LINE_LOOP);
28      for(int ii = 0; ii < m_segments; ii++)
29      {
30          glColor3f(getColorR(), getColorG(), getColorB());
31          glVertex2f(x + m_x, y + m_y);//output vertex
32
33          //calculate the tangential vector
34          //remember, the radial vector is (x, y)
35          //to get the tangential vector we flip those coordinates and negate
                one of them
36          float tx = -y;
37          float ty = x;
38
39          //add the tangential vector
40          x += tx * tangetial_factor;
41          y += ty * tangetial_factor;
42
43          //correct using the radial factor
44          x *= radial_factor;
45          y *= radial_factor;
46      }
47
48      glEnd();
49
50      Renderable2D::postrender();
51  }
52
53  }
```

renderOBJ/Circle.cpp

```
1   #ifndef __RECTANGLE_HPP__
2   #define __RECTANGLE_HPP__
3
4   #include "Renderable2D.hpp"
5
6   namespace asteroids
7   {
8
9   class MainWindow;
10  class Rectangle : public Renderable2D
11  {
12      public:
13          /**
14           * @brief Construct a new Rectangle object
15           *
16           * @param _mainWin Pointer to MainWindow for high and wigh
17           * @param x Position x
18           * @param y Position y
19           * @param w wigh of the Rectangle
20           * @param h hight of the Rectange
21           */
22          Rectangle(MainWindow* _mainWin, float x, float y, float w, float h)
                ;
23
24          /**
```

```
25          * @brief Renders a Rectange
26          *
27          */
28         void render();
29
30     private:
31         // Position x
32         float m_x;
33
34         // Position y
35         float m_y;
36
37         // wigh
38         float m_w;
39
40         // high
41         float m_h;
42 };
43
44 }// namespace asteroids
45
46 #endif
```

renderOBJ/Rectangle.hpp

```
1  #include "Rectangle.hpp"
2
3  namespace asteroids
4  {
5
6  Rectangle::Rectangle(MainWindow* _mainWin, float x, float y, float w, float
       h):Renderable2D(_mainWin)
7  {
8      m_x = x;
9      m_y = y;
10     m_w = w;
11     m_h = h;
12 }
13
14 void Rectangle::render()
15 {
16     Renderable2D::prerender();
17
18     glBegin(GL_LINE_LOOP);
19     glColor3f(getColorR(), getColorG(), getColorB());
20     glVertex2d(m_x, m_y);
21     glVertex2d(m_x + m_w, m_y);
22     glVertex2d(m_x + m_w, m_y + m_h);
23     glVertex2d(m_x, m_y + m_h);
24     glEnd();
25
26     Renderable2D::postrender();
27 }
28
29
30 }// namespace asteroids
```

renderOBJ/Rectangle.cpp

```
/*
 *  MainWindow.hpp
 *
 *  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.
 */

#ifndef __MAINWINDOW_HPP__
#define __MAINWINDOW_HPP__

#include <string>

#include <SDL2/SDL.h>

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

#include "Camera.hpp"
#include "renderOBJ/TriangleMesh.hpp"
#include "renderOBJ/Sphere.hpp"

namespace asteroids
{
class Renderable2D;
/**
 * @brief   Represents the main window of the game. This
 *          class contains the main loop, handles all
 *          user input and renders all objects
 *
 */
class MainWindow
{
public:

    /**
     * @brief Construct a new Main Window object
     *
     * @param title    The title of the window
     * @param plyname  A .ply file to render
     * @param w        The window width
     * @param h        The window heigt
     */
    MainWindow(const std::string& title, const std::string& plyname, int w,
        int h);

    /**
     * @brief Start the window's main loop
     */
    void execute();

    /**
     * @brief Destroys the Main Window object
```

```
56          *
57          */
58         ~MainWindow();
59
60         /// Returns the width of the window
61         int width();
62
63         /// Returns the height of the windows
64         int height();
65
66     private:
67
68         /// A pointer to a model to render
69         TriangleMesh*           m_mesh;
70
71         /// The virtual camera
72         Camera                  m_camera;
73
74         /// The window width
75         int                     m_width;
76
77         /// The window height
78         int                     m_height;
79
80         /// The SDL Window
81         SDL_Window*             m_sdlWindow;
82
83         /// The SDL OpenGL rendering context
84         SDL_GLContext           m_sdlGlcontext;
85
86         // For relyed dependecies
87         Renderable2D* _rend2d;
88     };
89
90     }// namespace asteroids
91
92     #endif
```

MainWindow.hpp

```
1  /*
2   *  MainWindow.cpp
3   *
4   *  Created on: Nov. 04 2018
5   *      Author: Thomas Wiemann
6   *
7   *  Copyright (c) 2018 Thomas Wiemann.
8   *  Restricted usage. Licensed for participants of the course "The C++
        Programming Language" only.
9   *  No unauthorized distribution.
10  */
11
12 #include "MainWindow.hpp"
13 #include "renderOBJ/Rectangle.hpp"
14 #include "renderOBJ/Renderable2D.hpp"
15 #include "renderOBJ/Circle.hpp"
16
17 #include <iostream>
```

```
18
19 namespace asteroids
20 {
21
22 MainWindow::MainWindow(
23     const std::string& title,
24     const std::string& plyname, int w, int h)
25     : m_camera(Vector(0.0f, 0.0f, -700.0f), 0.05f, 5.0f)
26 {
27     // Save width and height
28     m_height = h;
29     m_width = w;
30
31     // Setup window
32     m_sdlWindow = SDL_CreateWindow(
33         "SDL Main Window",
34         SDL_WINDOWPOS_CENTERED, SDL_WINDOWPOS_CENTERED,
35         m_width, m_height, SDL_WINDOW_OPENGL);
36
37     if(!m_sdlWindow)
38     {
39         std::cout << "MainWindow: Unable to create SDL window" << std::endl
              ;
40     }
41
42     m_sdlGlcontext = SDL_GL_CreateContext(m_sdlWindow);
43
44     if(!m_sdlGlcontext)
45     {
46         std::cout << "MainWindow: Unable to creade SDL GL context" << std::
              endl;
47     }
48
49     if(m_sdlWindow && m_sdlGlcontext)
50     {
51         // Set our OpenGL version.
52         // SDL_GL_CONTEXT_CORE gives us only the newer version, deprecated
              functions are disabled
53         SDL_GL_SetAttribute(SDL_GL_CONTEXT_PROFILE_MASK,
              SDL_GL_CONTEXT_PROFILE_CORE);
54
55         // 3.2 is part of the modern versions of OpenGL,
56         // but most video cards whould be able to run it
57         SDL_GL_SetAttribute(SDL_GL_CONTEXT_MAJOR_VERSION, 3);
58         SDL_GL_SetAttribute(SDL_GL_CONTEXT_MINOR_VERSION, 2);
59
60         // Turn on double buffering with a 24bit Z buffer.
61         // You may need to change this to 16 or 32 for your system
62         SDL_GL_SetAttribute(SDL_GL_DOUBLEBUFFER, 1);
63
64         // This makes our buffer swap syncronized with the monitor's
              vertical refresh
65         SDL_GL_SetSwapInterval(1);
66
67 #ifndef __APPLE__
68         glewExperimental = GL_TRUE;
69         glewInit();
70 #endif
```

```
71          SDL_GL_SwapWindow(m_sdlWindow);
72
73          // Init OpenGL projection matrix
74          glClearColor(0.0, 0.0, 0.0, 1.0);
75          float ratio = m_width * 1.0 / m_height;
76          glMatrixMode(GL_PROJECTION);
77          glLoadIdentity();
78          glViewport(0, 0, m_width, m_height);
79          gluPerspective(45, ratio, 1, 10000);
80
81          // Enter model view mode
82          glMatrixMode(GL_MODELVIEW);
83      }
84
85      // Load model
86      m_mesh = new TriangleMesh(plyname);
87  }
88
89  int MainWindow::width()
90  {
91      return m_width;
92  }
93
94  int MainWindow::height()
95  {
96      return m_height;
97  }
98
99  void MainWindow::execute()
100 {
101     int x = m_width / 2;
102     int y = m_height / 2;
103     int w = 200;
104     int h = 100;
105
106     Circle circle(this, x, y, 100, 20);
107     circle.setColor(1.0, 0.0, 0.0);
108
109     Rectangle rect(this, x - w / 2, y - h / 2, w, h);
110     rect.setColor(0.0, 1.0, 2.0);
111
112     Sphere sphere(Vector(0, 0, 0), 10);
113     //sphere.setColor(0.5, 0.6,1.0);
114
115     if(m_mesh && m_sdlWindow && m_sdlGlcontext)
116     {
117         bool loop = true;
118         const Uint8* keyStates;
119
120         while (loop)
121         {
122             //Clear background
123             glClear(GL_COLOR_BUFFER_BIT );
124
125             //Apply camera, also loads indentity matrix
126             m_camera.apply();
127
128             //Markers for mouse buttons
```

```
129             bool r_pressed = false;
130             bool l_pressed = false;
131
132             //Handle events
133             SDL_Event event;
134             while (SDL_PollEvent(&event))
135             {
136                 switch(event.type)
137                 {
138                     //Window was closed, exit main loop
139                     case SDL_QUIT:
140                         loop = false;
141                         break;
142                     //Handle mouse motion
143                     case SDL_MOUSEMOTION:
144
145                             //Check if left button is pressed
146                             if(event.motion.state & SDL_BUTTON_LMASK)
147                             {
148                                 l_pressed = true;
149                             }
150
151                             //Check if right button is pressed
152                             if(event.motion.state & SDL_BUTTON_RMASK)
153                             {
154                                 r_pressed = true;
155                             }
156
157                             /*Handle motion for pressed L button while R is
                                    not
158                             pressed*/
159                             if(l_pressed & !r_pressed)
160                             {
161                                if(event.motion.xrel > -3)
162                                 {
163                                     m_camera.turn(Camera::LEFT);
164                                 }
165                                 if(event.motion.xrel < 3)
166                                 {
167                                     m_camera.turn(Camera::RIGHT);
168                                 }
169                                 if(event.motion.yrel > 3)
170                                 {
171                                     m_camera.turn(Camera::UP);
172                                 }
173                                 if(event.motion.yrel < -3)
174                                 {
175                                     m_camera.turn(Camera::DOWN);
176                                 }
177                             }
178
179                             /*Handle motion for pressed R button while L is
                                    not
180                             pressed*/
181                             if(r_pressed & !l_pressed)
182                             {
183                                if(event.motion.xrel > 3)
184                                {
```

```
185                                 m_camera.move(Camera::RIGHT);
186                             }
187                             if(event.motion.xrel < -3)
188                             {
189                                 m_camera.move(Camera::LEFT);
190                             } if(event.motion.yrel > 3)
191                             {
192                                 m_camera.move(Camera::FORWARD);
193                             }
194                             if(event.motion.yrel < -3)
195                             {
196                                 m_camera.move(Camera::BACKWARD);
197                             }
198                         }
199                         break;
200                     default:
201                         break;
202                 }
203 
204                 //Get keyboard states and handle model movement
205                 keyStates = SDL_GetKeyboardState(NULL);
206 
207                 if(keyStates[SDL_SCANCODE_UP])
208                 {
209                     m_mesh->rotate(TriangleMesh::YAW, 0.05);
210                 }
211                 if(keyStates[SDL_SCANCODE_DOWN])
212                 {
213                     m_mesh->rotate(TriangleMesh::YAW, -0.05);
214                 }
215                 if(keyStates[SDL_SCANCODE_LEFT])
216                 {
217                     m_mesh->rotate(TriangleMesh::ROLL, 0.05);
218                 }
219                 if(keyStates[SDL_SCANCODE_RIGHT])
220                 {
221                     m_mesh->rotate(TriangleMesh::ROLL, -0.05);
222                 }
223                 if(keyStates[SDL_SCANCODE_W])
224                 {
225                     m_mesh->move(TriangleMesh::ACCEL, 3);
226                 }
227                 if(keyStates[SDL_SCANCODE_S])
228                 {
229                     m_mesh->move(TriangleMesh::ACCEL, -3);
230                 }
231                 if(keyStates[SDL_SCANCODE_A])
232                 {
233                     m_mesh->move(TriangleMesh::STRAFE, 3);
234                 }
235                 if(keyStates[SDL_SCANCODE_D])
236                 {
237                     m_mesh->move(TriangleMesh::STRAFE, -3);
238                 }
239             }
240 
241             /*Render model*/
242             m_mesh->render();
```

```
243
244                 circle.render();
245                 rect.render();
246                 sphere.render();
247
248                 // Bring up back buffer
249                 SDL_GL_SwapWindow(m_sdlWindow);
250             }
251         }
252 }
253
254 MainWindow::~MainWindow()
255 {
256     // Delete model
257     if (m_mesh)
258     {
259         delete m_mesh;
260     }
261
262     // Cleanup SDL stuff
263     SDL_GL_DeleteContext(m_sdlGlcontext);
264
265     // Destroy our window
266     SDL_DestroyWindow(m_sdlWindow);
267
268     // Shutdown SDL 2
269     SDL_Quit();
270 }
271
272 } // namespace asteroids
```

MainWindow.cpp

```
1 Es ist sinnvoll, dass Renderable2D einen eigenen Konstructor hat, weil
2 alle 2D-Objekte die Dimensionen des MainWindows brauchen. Man könnte
3 Dimensionen in alle 2D-Objekte einzeln implementieren, würde dann
4 aber reduendanten Code haben.
5 Auch ist dieser Code dann nicht mehr so leicht austauschbar.
```

INFO.txt