Curve and surface modeling

– a CAGD approach based on OpenGL and C++ –

Ágoston Róth

Department of Mathematics and Computer Science, Babeș-Bolyai University, Cluj-Napoca, Romania

(agoston.roth@gmail.com)

Seminar 2 | March 7 & March 16, 2022

Color styles

- keywords, built-in types, enumerations, constants and namespaces of C++
- keywords, built-in types, enumerations, constants and functions of OpenGL
- our types, constants, enumerations and namespaces
- comments

Descartes coordinates

Class DCoordinate3

Description

- Class DCoordinate3 forms the most elementary building block/data structure of all other classes.
- By means of Descartes coordinates we can represent curve points, tangent and acceleration vectors (i.e. higher order derivatives) of curves, surface points, higher order (mixed) partial derivatives of surfaces, normal vectors associated with surface points, control polygons, control nets, etc.
- Using operator overloading we can easily implement mathematical formulas that describe either curves or surfaces.

Descartes coordinates – header file, part I

DCoordinates3.h

```
1  #pragma once

2  #include <cmath>
3  #include <GL/glew.h>
4  #include <iostream>

5  namespace cagd
6  {
7      //--------------------
8      // class DCoordinate3
9      //--------------------
10     class DCoordinate3
11     {
12     private:
13         GLdouble _data[3];

14     public:

15         // default constructor
16         DCoordinate3();

17         // special constructor
18         DCoordinate3(GLdouble x, GLdouble y, GLdouble z = 0.0);

19         // get components by value
20         const GLdouble operator [](GLuint index) const;
21         const GLdouble x() const;
22         const GLdouble y() const;
23         const GLdouble z() const;


24         // get components by reference
25         GLdouble& operator [](GLuint index);
26         GLdouble& x();
27         GLdouble& y();
```

Descartes coordinates – header file, part II

DCoordinates3.h

```
28          GLdouble& z();

29          // change sign
30          const DCoordinate3 operator +() const;
31          const DCoordinate3 operator -() const;

32          // add
33          const DCoordinate3 operator +(const DCoordinate3& rhs) const;

34          // add to *this
35          DCoordinate3& operator +=(const DCoordinate3& rhs);

36          // subtract
37          const DCoordinate3 operator -(const DCoordinate3& rhs) const;

38          // subtract from *this
39          DCoordinate3& operator -=(const DCoordinate3& rhs);

40          // cross product
41          const DCoordinate3 operator ^(const DCoordinate3& rhs) const;

42          // cross product, result is stored by *this
43          DCoordinate3& operator ^=(const DCoordinate3& rhs);

44          // dot product
45          const GLdouble operator *(const DCoordinate3& rhs) const;

46          // scale
47          const DCoordinate3 operator *(const GLdouble& rhs) const;
48          const DCoordinate3 operator /(const GLdouble& rhs) const;

49          // scale *this
50          DCoordinate3& operator *=(const GLdouble& rhs);
51          DCoordinate3& operator /=(const GLdouble& rhs);
```

Descartes coordinates – header file, part III

DCoordinates3.h

```
52          // length
53          const GLdouble length() const;

54          // normalize
55          DCoordinate3& normalize();
56      };

57      //------------------------------------------
58      // implementation of class DCoordinate3
59      //------------------------------------------

60      // default constructor
61      inline DCoordinate3::DCoordinate3()
62      {
63          _data[0] = _data[1] = _data[2] = 0.0;
64      }

65      // special constructor
66      inline DCoordinate3::DCoordinate3(GLdouble x, GLdouble y, GLdouble z)
67      {
68          _data[0] = x;
69          _data[1] = y;
70          _data[2] = z;
71      }

72      // get components by value
73      inline const GLdouble DCoordinate3::operator [](GLuint index) const
74      {
75          return _data[index];
76      }

77      inline const GLdouble DCoordinate3::x() const
78      {
79          return _data[0];
80      }
```

Descartes coordinates – header file, part IV

DCoordinates3.h

```
81      inline const GLdouble DCoordinate3::y() const
82      {
83          // homework
84      }

85      inline const GLdouble DCoordinate3::z() const
86      {
87          // homework
88      }

89      // get components by reference
90      inline GLdouble& DCoordinate3::operator [](GLuint index)
91      {
92          return _data[index];
93      }

94      inline GLdouble& DCoordinate3::x()
95      {
96          return _data[0];
97      }

98      inline GLdouble& DCoordinate3::y()
99      {
100         // homework
101     }

102     inline GLdouble& DCoordinate3::z()
103     {
104         // homework
105     }

106     // change sign
107     inline const DCoordinate3 DCoordinate3::operator +() const
108     {
```

Descartes coordinates – header file, part V

DCoordinates3.h

```
109         return DCoordinate3(_data[0], _data[1], _data[2]);
110     }

111     inline const DCoordinate3 DCoordinate3::operator -() const
112     {
113         return DCoordinate3(-_data[0], -_data[1], -_data[2]);
114     }

115     // add
116     inline const DCoordinate3 DCoordinate3::operator +(const DCoordinate3& rhs) const
117     {
118         return DCoordinate3(_data[0] + rhs._data[0], _data[1] + rhs._data[1], _data[2] + rhs._data[2]);
119     }

120     // add to *this
121     inline DCoordinate3& DCoordinate3::operator +=(const DCoordinate3& rhs)
122     {
123         _data[0] += rhs._data[0];
124         _data[1] += rhs._data[1];
125         _data[2] += rhs._data[2];
126         return *this;
127     }

128     // subtract
129     inline const DCoordinate3 DCoordinate3::operator -(const DCoordinate3& rhs) const
130     {
131         // homework
132     }

133     // subtract from *this
134     inline DCoordinate3& DCoordinate3::operator -=(const DCoordinate3& rhs)
135     {
136         //homework
137     }
```

Descartes coordinates – header file, part VI

DCoordinates3.h

```
138     // cross product
139     inline const DCoordinate3 DCoordinate3::operator ^(const DCoordinate3& rhs) const
140     {
141         return DCoordinate3(
142                 _data[1] * rhs._data[2] - _data[2] * rhs._data[1],
143                 _data[2] * rhs._data[0] - _data[0] * rhs._data[2],
144                 _data[0] * rhs._data[1] - _data[1] * rhs._data[0]);
145     }

146     // cross product, result is stored by *this
147     inline DCoordinate3& DCoordinate3::operator ^=(const DCoordinate3& rhs)
148     {
149         // homework
150     }

151     // dot product
152     inline const GLdouble DCoordinate3::operator *(const DCoordinate3& rhs) const
153     {
154         return _data[0] * rhs._data[0] + _data[1] * rhs._data[1] + _data[2] * rhs._data[2];
155     }

156     // scale
157     inline const DCoordinate3 DCoordinate3::operator *(const GLdouble& rhs) const
158     {
159         return DCoordinate3(_data[0] * rhs, _data[1] * rhs, _data[2] * rhs);
160     }

161     inline const DCoordinate3 operator *(const GLdouble& lhs, const DCoordinate3& rhs)
162     {
163         // homework
164     }

165     inline const DCoordinate3 DCoordinate3::operator /(const GLdouble& rhs) const
166     {
167         // homework
```

Descartes coordinates – header file, part VII

DCoordinates3.h

```
168     }

169     // scale *this
170     inline DCoordinate3& DCoordinate3::operator *=(const GLdouble& rhs)
171     {
172         _data[0] *= rhs;
173         _data[1] *= rhs;
174         _data[2] *= rhs;

175         return *this;
176     }

177     inline DCoordinate3& DCoordinate3::operator /=(const GLdouble& rhs)
178     {
179         // homework
180     }

181     // length
182     inline const GLdouble DCoordinate3::length() const
183     {
184         return std::sqrt((*this) * (*this));
185     }

186     // normalize
187     inline DCoordinate3& DCoordinate3::normalize()
188     {
189         GLdouble l = length();

190         if (l && l != 1.0)
191             *this /= l;

192         return *this;
193     }
```

Descartes coordinates – header file, part VIII

DCoordinates3.h

```
194     //-----------------------------------------------------------------------
195     // definitions of overloaded input/output from/to stream operators
196     //-----------------------------------------------------------------------

197     // output to stream
198     inline std::ostream& operator <<(std::ostream& lhs, const DCoordinate3& rhs)
199     {
200         return lhs << rhs[0] << " " << rhs[1] << " " << rhs[2];
201     }

202     // input from stream
203     inline std::istream& operator >>(std::istream& lhs, DCoordinate3& rhs)
204     {
205         // homework
206     }
207 }
```

Descartes coordinates – source file

♯ DCoordinates3.cpp

Homework

Implement all operators and methods of class DCoordinate3! Notice that for efficiency reasons all operators, methods must be inlined.

A simple template for matrices

Template class Matrix

Description

- By means of the template class Matrix we can represent collocation matrices, control polygons, control nets, grouped spline or patch information.

A simple template for matrices – header file, part I

Matrices.h

```
1  #pragma once

2  #include <iostream>
3  #include <vector>
4  #include <GL/glew.h>

5  namespace cagd
6  {
7      // forward declaration of template class Matrix
8      template <typename T>
9      class Matrix;

10     // forward declaration of template class RowMatrix
11     template <typename T>
12     class RowMatrix;

13     // forward declaration of template class ColumnMatrix
14     template <typename T>
15     class ColumnMatrix;

16         // forward declaration of template class TriangularMatrix
17     template <typename T>
18     class TriangularMatrix;

19     // forward declarations of overloaded and templated input/output from/to stream operators
20     template <typename T>
21     std::ostream& operator << (std::ostream& lhs, const Matrix<T>& rhs);

22     template <typename T>
23     std::istream& operator >>(std::istream& lhs, Matrix<T>& rhs);

24     template <typename T>
25     std::istream& operator >>(std::istream& lhs, TriangularMatrix<T>& rhs);
26
```

A simple template for matrices – header file, part II

Matrices.h

```
27      template <typename T>
28      std::ostream& operator << (std::ostream& lhs, const TriangularMatrix<T>& rhs);

29      //-------------------------
30      // template class Matrix
31      //-------------------------
32      template <typename T>
33      class Matrix
34      {
35          friend std::ostream& operator << <T>(std::ostream&, const Matrix<T>& rhs);
36          friend std::istream& operator >> <T>(std::istream&, Matrix<T>& rhs);

37      protected:
38          GLuint                          _row_count;
39          GLuint                          _column_count;
40          std::vector< std::vector<T> >   _data;
41      public:
42          // special constructor (can also be used as a default constructor)
43          Matrix(GLuint row_count = 1, GLuint column_count = 1);

44          // copy constructor
45          Matrix(const Matrix& m);

46          // assignment operator
47          Matrix& operator =(const Matrix& m);

48          // get element by reference
49          T& operator ()(GLuint row, GLuint column);

50          // get copy of an element
51          T operator ()(GLuint row, GLuint column) const;

52          // get dimensions
53          GLuint GetRowCount() const;
54          GLuint GetColumnCount() const;
```

A simple template for matrices – header file, part III

Matrices.h

```
55          // set dimensions
56          virtual GLboolean ResizeRows(GLuint row_count);
57          virtual GLboolean ResizeColumns(GLuint column_count);

58          // update
59          GLboolean SetRow(GLuint index, const RowMatrix<T>& row);
60          GLboolean SetColumn(GLuint index, const ColumnMatrix<T>& column);

61          // destructor
62          virtual ~Matrix();
63      };

64      //--------------------------------
65      // template class RowMatrix
66      //--------------------------------
67      template <typename T>
68      class RowMatrix: public Matrix<T>
69      {
70      public:
71          // special constructor (can also be used as a default constructor)
72          RowMatrix(GLuint column_count = 1);

73          // get element by reference
74          T& operator ()(GLuint column);
75          T& operator [](GLuint column);

76          // get copy of an element
77          T operator ()(GLuint column) const;
78          T operator [](GLuint column) const;

79          // a row matrix consists of a single row
80          GLboolean ResizeRows(GLuint row_count);
81      };
```

A simple template for matrices – header file, part IV

Matrices.h

```
82      //-----------------------------------
83      // template class ColumnMatrix
84      //-----------------------------------
85      template <typename T>
86      class ColumnMatrix: public Matrix<T>
87      {
88      public:
89          // special constructor (can also be used as a default constructor)
90          ColumnMatrix(GLuint row_count = 1);

91          // get element by reference
92          T& operator ()(GLuint row);
93          T& operator [](GLuint row);

94          // get copy of an element
95          T operator ()(GLuint row) const;
96          T operator [](GLuint row) const;

97          // a column matrix consists of a single column
98          GLboolean ResizeColumns(GLuint column_count);
99      };

100     //-----------------------------------
101     // template class TriangularMatrix
102     //-----------------------------------
103     template <typename T>
104     class TriangularMatrix
105     {
106         friend std::istream& operator >> <T>(std::istream&, TriangularMatrix<T>& rhs);
107         friend std::ostream& operator << <T>(std::ostream&, const TriangularMatrix<T>& rhs);

108     protected:
109         GLuint                          _row_count;
110         std::vector< std::vector<T> > _data;
```

A simple template for matrices – header file, part V

Matrices.h

```
111     public:
112         // special constructor (can also be used as a default constructor)
113         TriangularMatrix(GLuint row_count = 1);
114
115         // get element by reference
116         T& operator ()(GLuint row, GLuint column);
117
118         // get copy of an element
119         T operator ()(GLuint row, GLuint column) const;
120
121         // get dimension
122         GLuint GetRowCount() const;
123
124         // set dimension
125         GLboolean ResizeRows(GLuint row_count);
126     };
127
128     //------------------------------------------------------
129     // homework: implementation of template class Matrix
130     //------------------------------------------------------
131
132     //------------------------------------------------------------
133     // homework: implementation of template class RowMatrix
134     //------------------------------------------------------------
135
136
137     //---------------------------------------------------------
138     // homework: implementation of template class ColumnMatrix
139     //---------------------------------------------------------
140
141     //-----------------------------------------------------------------
142     // homework: implementation of template class TriangularMatrix
143     //-----------------------------------------------------------------
```

A simple template for matrices – header file, part VI

Matrices.h

```
135     //------------------------------------------------------------------------------
136     // definitions of Matrix-related overloaded and templated input/output from/to stream operators
137     //------------------------------------------------------------------------------

138     // output to stream
139     template <typename T>
140     std::ostream& operator <<(std::ostream& lhs, const Matrix<T>& rhs)
141     {
142         lhs << rhs._row_count << " " << rhs._column_count << std::endl;
143         for (typename std::vector< std::vector<T> >::const_iterator row = rhs._data.begin();
144              row != rhs._data.end(); ++row)
145         {
146             for (typename std::vector<T>::const_iterator column = row->begin();
147                  column != row->end(); ++column)
148                 lhs << *column << " ";
149             lhs << std::endl;
150         }
151         return lhs;
152     }

153     // input from stream
154     template <typename T>
155     std::istream& operator >>(std::istream& lhs, Matrix<T>& rhs)
156     {
157         // homework
158     }

159     //------------------------------------------------------------------------------
160     // definitions of TringularMatrix-related overloaded and templated input/output from/to
161     // stream operators
162     //------------------------------------------------------------------------------

163     // homework
164 }
```

A simple template for matrices – source file

♯ Matrices.cpp

Homework

Implement all operators, methods and friend functions of the template classes Matrix, RowMatrix, and ColumnMatrix! The implementation must be done in the header file Matrices.h.

Real square matrices

Derived class RealSquareMatrix

Description

- Matrix<GLdouble> is the base class of the class RealSquareMatrix.
- Some numerical methods (e.g. data point interpolation, degree elevation, LU-decompisition, solutions of linear systems) require real square matrices.

Real square matrices – header file, part I

RealSquareMatrices.h

```
1  #pragma once

2  #include "DCoordinates3.h"
3  #include <GL/glew.h>
4  #include <limits>
5  #include "Matrices.h"

6  namespace cagd
7  {
8      class RealSquareMatrix: public Matrix<GLdouble>
9      {
10     private:
11         GLboolean            _lu_decomposition_is_done;
12         std::vector<GLuint> _row_permutation;

13     public:
14         // special constructor
15         RealSquareMatrix(GLuint size);

16         // homework: copy constructor
17         RealSquareMatrix(const RealSquareMatrix& m);

18         // homework: assignment operator
19         RealSquareMatrix& operator =(const RealSquareMatrix& rhs);

20         // homework: square matrices have the same number of rows and columns!
21         GLboolean ResizeRows(GLuint row_count);
22         GLboolean ResizeColumns(GLuint row_count);

23         // tries to determine the LU decomposition of this square matrix
24         GLboolean PerformLUDecomposition();

25         // Solves linear systems of type A * x = b, where A is a regular square matrix,
26         // while b and x are row or column matrices with elements of type T.
27         // Here matrix A corresponds to *this.
```

Real square matrices – header file, part II

RealSquareMatrices.h

```
28          // Advantage: T can be either GLdouble or DCoordinate,
29          // or any other type which has similar mathematical operators.
30          template <class T>
31          GLboolean SolveLinearSystem(const Matrix<T>& b, Matrix<T>& x,
32                                      GLboolean represent_solutions_as_columns = GL_TRUE);
33      };

34      template <class T>
35      GLboolean RealSquareMatrix::SolveLinearSystem(const Matrix<T>& b, Matrix<T>& x,
36                                                    GLboolean represent_solutions_as_columns)
37      {
38          if (!_lu_decomposition_is_done)
39              if (!PerformLUDecomposition())
40                  return GL_FALSE;

41          if (represent_solutions_as_columns)
42          {
43              GLint size = static_cast<GLint>(GetColumnCount());
44              if (static_cast<GLint>(b.GetRowCount()) != size)
45                      return GL_FALSE;

46              x = b;

47              for (GLuint k = 0; k < b.GetColumnCount(); ++k)
48              {
49                  GLint ii = 0;
50                  for (GLint i = 0; i < size; ++i)
51                  {
52                      GLuint ip = _row_permutation[i];
53                      T sum = x(ip, k);
54                      x(ip, k) = x(i, k);
55                      if (ii != 0)
56                          for (GLint j = ii - 1; j < i; ++j)
57                              sum -= _data[i][j] * x(j, k);
58                      else
```

Real square matrices – header file, part III

RealSquareMatrices.h

```
59                      if (sum != 0.0)
60                          ii = i + 1;
61                  x(i, k) = sum;
62              }

63              for (GLint i = size - 1; i >= 0; --i)
64              {
65                  T sum = x(i, k);
66                  for (GLint j = i + 1; j < size; ++j)
67                      sum -= _data[i][j] * x(j, k);
68                  x(i, k) = sum /= _data[i][i];
69              }
70          }
71      }
72      else
73      {
74          GLint size = static_cast<GLint>(GetRowCount());
75          if (static_cast<GLint>(b.GetColumnCount()) != size)
76              return GL_FALSE;

77          x = b;

78          for (GLuint k = 0; k < b.GetRowCount(); ++k)
79          {
80              GLint ii = 0;
81              for (GLint i = 0; i < size; ++i)
82              {
83                  GLuint ip = _row_permutation[i];
84                  T sum = x(k, ip);
85                  x(k, ip) = x(k, i);
86                  if (ii != 0)
87                      for (GLint j = ii - 1; j < i; ++j)
88                          sum -= _data[i][j] * x(k, j);
89                  else
90                      if (sum != 0.0)
```

Real square matrices – header file, part IV

RealSquareMatrices.h

```
91                              ii = i + 1;
92                          x(k, i) = sum;
93                      }

94                      for (GLint i = size - 1; i >= 0; --i)
95                      {
96                          T sum = x(k, i);
97                          for (GLint j = i + 1; j < size; ++j)
98                              sum -= _data[i][j] * x(k, j);
99                          x(k, i) = sum /= _data[i][i];
100                     }
101                 }
102             }

103             return GL_TRUE;
104         }

105 }
```

Real square matrices – source file, part I

RealSquareMatrix.cpp

```
1  #include "RealSquareMatrices.h"

2  using namespace cagd;
3  using namespace std;

4  RealSquareMatrix::RealSquareMatrix(GLuint size):
5          Matrix<GLdouble>(size, size),
6          _lu_decomposition_is_done(GL_FALSE)
7  {
8  }

9  GLboolean RealSquareMatrix::PerformLUDecomposition()
10 {
11     if (_lu_decomposition_is_done)
12         return GL_TRUE;

13     if (_row_count <= 1)
14         return GL_FALSE;

15     const GLdouble tiny = numeric_limits<GLdouble>::min();

16     GLuint size = static_cast<GLuint>(_data.size());
17     vector<GLdouble> implicit_scaling_of_each_row(size);

18     _row_permutation.resize(size);

19     GLdouble row_interchanges = 1.0;

20     //------------------------------------------------------------
21     // loop over rows to get the implicit scaling information
22     //------------------------------------------------------------
23     vector<GLdouble>::iterator its = implicit_scaling_of_each_row.begin();
24     for (vector<vector<GLdouble> >::const_iterator itr = _data.begin(); itr < _data.end(); ++itr)
25     {
26         GLdouble big = 0.0;
```

Real square matrices – source file, part II

RealSquareMatrix.cpp

```
27          for (vector<GLdouble>::const_iterator itc = itr->begin(); itc < itr->end(); ++itc)
28          {
29              GLdouble temp = abs(*itc);
30              if (temp > big)
31                      big = temp;
32          }

33          if (big == 0.0)
34          {
35              // the matrix is singular
36              return GL_FALSE;
37          }
38          *its = 1.0 / big;
39          ++its;
40      }

41      //-----------------------------------------
42      // search for the largest pivot element
43      //-----------------------------------------
44      for (GLuint k = 0; k < size; ++k)
45      {
46          GLuint imax = k;
47          GLdouble big = 0.0;
48          for (GLuint i = k; i < size; ++i)
49          {
50              GLdouble temp = implicit_scaling_of_each_row[i] * abs(_data[i][k]);
51              if (temp > big)
52              {
53                  big = temp;
54                  imax = i;
55              }
56          }

57          // do we need to interchange rows?
58          if (k != imax)
```

Real square matrices – source file, part III

RealSquareMatrix.cpp

```
59          {
60              for (GLuint j = 0; j < size; ++j)
61              {
62                  GLdouble temp = _data[imax][j];
63                  _data[imax][j] = _data[k][j];
64                  _data[k][j] = temp;
65              }
66              // change the parity of row_interchanges
67              row_interchanges = -row_interchanges;
68              // also interchange the scale factor
69              implicit_scaling_of_each_row[imax] = implicit_scaling_of_each_row[k];
70          }

71          _row_permutation[k] = imax;
72          if (_data[k][k] == 0.0)
73              _data[k][k] = tiny;

74          for (GLuint i = k + 1; i < size; ++i)
75          {
76              // divide by pivot element
77              GLdouble temp = _data[i][k] /= _data[k][k];

78              // reduce remaining submatrix
79              for (GLuint j = k + 1; j < size; ++j)
80                  _data[i][j] -= temp * _data[k][j];
81          }
82      }

83      _lu_decomposition_is_done = GL_TRUE;

84      return GL_TRUE;
85  }
```

Real square matrices – source file, part IV

RealSquareMatrix.cpp

Homework

Implement all unfinished methods and operators of the class RealSquareMatrix!

Description

- Class GenericCurve3 can also be used as a base class for any type of curve.
- This class provides methods only for rendering and updating (i.e. does not implement coordinate or blending functions).
- When using inheritance, the coordinates of the curve points, first and second order derivatives must be set either by one of the methods (e.g. constructor) of the derived class, or via the inherited methods

```
DCoordinate3& operator ()(GLuint order, GLuint index);

GLboolean SetDerivative(GLuint order, GLuint index, GLdouble x, GLdouble y, GLdouble z = 0.0);
GLboolean SetDerivative(GLuint order, GLuint index, const DCoordinate3 &d);
```

- The rendering is based on vertex buffer objects. Notice that, for efficiency reasons all double coordinates are truncated to float numbers when creating/loading the data of vertex buffer objects.

Generic curves – header file, part I

GenericCurves3.h

```
1  #pragma once

2  #include "DCoordinates3.h"
3  #include <GL/glew.h>
4  #include "Matrices.h"
5  #include <iostream>

6  namespace cagd
7  {
8      //--------------------------
9      // class GenericCurve3
10     //--------------------------
11     class GenericCurve3
12     {
13         //------------------------------------
14         // input/output from/to stream
15         //------------------------------------
16         friend std::ostream& operator <<(std::ostream& lhs, const GenericCurve3& rhs);
17         friend std::istream& operator >>(std::istream& lhs, GenericCurve3& rhs);

18     protected:
19         GLenum                 _usage_flag;
20         RowMatrix<GLuint>      _vbo_derivative;
21         Matrix<DCoordinate3>   _derivative;

22     public:
23         // default and special constructor
24         GenericCurve3(
25                 GLuint maximum_order_of_derivatives = 2,
26                 GLuint point_count = 0,
27                 GLenum usage_flag = GL_STATIC_DRAW);

28         // special constructor
29         GenericCurve3(const Matrix<DCoordinate3>& derivative, GLenum usage_flag = GL_STATIC_DRAW);
```

Generic curves – header file, part II

GenericCurves3.h

```
30          // copy constructor
31          GenericCurve3(const GenericCurve3& curve);
32
33          // assignment operator
34          GenericCurve3& operator =(const GenericCurve3& rhs);
35
36          // vertex buffer object handling methods
37          GLvoid DeleteVertexBufferObjects();
38          GLboolean RenderDerivatives(GLuint order, GLenum render_mode) const;
39          GLboolean UpdateVertexBufferObjects(GLenum usage_flag = GL_STATIC_DRAW);
40
41          GLfloat* MapDerivatives(GLuint order, GLenum access_mode = GL_READ_ONLY) const;
42          GLboolean UnmapDerivatives(GLuint order) const;
43
44          // get derivative by value
45          DCoordinate3 operator ()(GLuint order, GLuint index) const;
46
47          // get derivative by reference
48          DCoordinate3& operator ()(GLuint order, GLuint index);
49
50          // other update and query methods
51          GLboolean SetDerivative(GLuint order, GLuint index, GLdouble x, GLdouble y, GLdouble z = 0.0);
52          GLboolean SetDerivative(GLuint order, GLuint index, const DCoordinate3& d);
53          GLboolean GetDerivative(GLuint order, GLuint index, GLdouble& x, GLdouble& y, GLdouble& z) const;
54          GLboolean GetDerivative(GLuint order, GLuint index, DCoordinate3& d) const;
55
56          GLuint GetMaximumOrderOfDerivatives() const;
57          GLuint GetPointCount() const;
58          GLenum GetUsageFlag() const;
59
60          // destructor
61          virtual ~GenericCurve3();
62      };
63  }
```

Generic curves – source file, part I

GenericCurves3.cpp

```
1  #include "GenericCurves3.h"

2  using namespace cagd;
3  using namespace std;

4  //--------------------------------------------
5  // implementation of class GenericCurve3
6  //--------------------------------------------

7  // default and special constructor
8  GenericCurve3::GenericCurve3(GLuint maximum_order_of_derivatives, GLuint point_count, GLenum usage_flag):
9          _usage_flag(usage_flag),
10         _vbo_derivative(maximum_order_of_derivatives + 1),
11         _derivative(maximum_order_of_derivatives + 1, point_count)
12 {
13 }

14 // special constructor
15 GenericCurve3::GenericCurve3(const Matrix<DCoordinate3>& derivative, GLenum usage_flag):
16         _usage_flag(usage_flag),
17         _vbo_derivative(RowMatrix<GLuint>(derivative.GetRowCount())),
18         _derivative(derivative)
19 {
20 }

21 // copy constructor
22 GenericCurve3::GenericCurve3(const GenericCurve3& curve):
23         _usage_flag(curve._usage_flag),
24         _vbo_derivative(RowMatrix<GLuint>(curve._vbo_derivative.GetColumnCount())),
25         _derivative(curve._derivative)
26 {
27     GLboolean vbo_update_is_possible = GL_TRUE;
28     for (GLuint i = 0; i < curve._vbo_derivative.GetColumnCount(); ++i)
29         vbo_update_is_possible &= curve._vbo_derivative(i);
```

Generic curves – source file, part II

GenericCurves3.cpp

```
30      if (vbo_update_is_possible)
31          UpdateVertexBufferObjects(_usage_flag);
32  }

33  // assignment operator
34  GenericCurve3& GenericCurve3::operator =(const GenericCurve3& rhs)
35  {
36      if (this != &rhs)
37      {
38          DeleteVertexBufferObjects();

39          _usage_flag = rhs._usage_flag;
40          _derivative = rhs._derivative;

41          GLboolean vbo_update_is_possible = GL_TRUE;
42          for (GLuint i = 0; i < rhs._vbo_derivative.GetColumnCount(); ++i)
43              vbo_update_is_possible &= rhs._vbo_derivative(i);

44          if (vbo_update_is_possible)
45              UpdateVertexBufferObjects(_usage_flag);
46      }
47      return *this;
48  }

49  // vertex buffer object handling methods
50  GLvoid GenericCurve3::DeleteVertexBufferObjects()
51  {
52      for (GLuint i = 0; i < _vbo_derivative.GetColumnCount(); ++i)
53      {
54          if (_vbo_derivative(i))
55          {
56              glDeleteBuffers(1, &_vbo_derivative(i));
57              _vbo_derivative(i) = 0;
58          }
59      }
```

Generic curves – source file, part III

GenericCurves3.cpp

```
60  }
61  GLboolean GenericCurve3::RenderDerivatives(GLuint order, GLenum render_mode) const
62  {
63      GLuint max_order = _derivative.GetRowCount();
64      if (order >= max_order || !_vbo_derivative(order))
65          return GL_FALSE;
66      GLuint point_count = _derivative.GetColumnCount();
67      glEnableClientState(GL_VERTEX_ARRAY);
68          glBindBuffer(GL_ARRAY_BUFFER, _vbo_derivative(order));
69              glVertexPointer(3, GL_FLOAT, 0, (const GLvoid *)0);
70              if (!order)
71              {
72                  if (render_mode != GL_LINE_STRIP &&
73                      render_mode != GL_LINE_LOOP  &&
74                      render_mode != GL_POINTS)
75                  {
76                      glBindBuffer(GL_ARRAY_BUFFER, 0);
77                      glDisableClientState(GL_VERTEX_ARRAY);
78                      return GL_FALSE;
79                  }
80                  glDrawArrays(render_mode, 0, point_count);
81              }
82              else
83              {
84                  if (render_mode != GL_LINES && render_mode != GL_POINTS)
85                  {
86                      glBindBuffer(GL_ARRAY_BUFFER, 0);
87                      glDisableClientState(GL_VERTEX_ARRAY);
88                      return GL_FALSE;
89                  }
```

Generic curves – source file, part IV

GenericCurves3.cpp

```
90                  glDrawArrays(render_mode, 0, 2 * point_count);
91              }

92          glBindBuffer(GL_ARRAY_BUFFER, 0);
93      glDisableClientState(GL_VERTEX_ARRAY);

94      return GL_TRUE;
95  }

96  GLboolean GenericCurve3::UpdateVertexBufferObjects(GLenum usage_flag)
97  {
98      if (usage_flag != GL_STREAM_DRAW  && usage_flag != GL_STREAM_READ  &&
99          usage_flag != GL_STREAM_COPY &&
100         usage_flag != GL_DYNAMIC_DRAW && usage_flag != GL_DYNAMIC_READ &&
101         usage_flag != GL_DYNAMIC_COPY &&
102         usage_flag != GL_STATIC_DRAW  && usage_flag != GL_STATIC_READ  &&
103         usage_flag != GL_STATIC_COPY)
104         return GL_FALSE;

105     DeleteVertexBufferObjects();

106     _usage_flag = usage_flag;

107     for(GLuint d = 0; d < _vbo_derivative.GetColumnCount(); ++d)
108     {
109         glGenBuffers(1, &_vbo_derivative(d));

110         if (!_vbo_derivative(d))
111         {
112             for (GLuint i = 0; i < d; ++i)
113             {
114                 glDeleteBuffers(1, &_vbo_derivative(i));
115                 _vbo_derivative(i) = 0;
116             }
```

Generic curves – source file, part V

GenericCurves3.cpp

```
117             return GL_FALSE;
118         }
119     }

120     GLuint curve_point_count = _derivative.GetColumnCount();

121     GLfloat *coordinate = 0;

122     // curve points
123     GLuint curve_point_byte_size = 3 * curve_point_count * sizeof(GLfloat);

124     glBindBuffer(GL_ARRAY_BUFFER, _vbo_derivative(0));
125     glBufferData(GL_ARRAY_BUFFER, curve_point_byte_size, 0, _usage_flag);

126     coordinate = (GLfloat*)glMapBuffer(GL_ARRAY_BUFFER, GL_WRITE_ONLY);

127     if (!coordinate)
128     {
129         glBindBuffer(GL_ARRAY_BUFFER, 0);
130         DeleteVertexBufferObjects();
131         return GL_FALSE;
132     }

133     for (GLuint i = 0; i < curve_point_count; ++i)
134     {
135         for (GLuint j = 0; j < 3; ++j)
136         {
137             *coordinate = (GLfloat)_derivative(0,i)[j];
138             ++coordinate;
139         }
140     }

141     if (!glUnmapBuffer(GL_ARRAY_BUFFER))
142     {
```

Generic curves – source file, part VI

GenericCurves3.cpp

```
143             glBindBuffer(GL_ARRAY_BUFFER, 0);
144             DeleteVertexBufferObjects();
145             return GL_FALSE;
146         }
147
148         // higher order derivatives
149         GLuint higher_order_derivative_byte_size = 2 * curve_point_byte_size;
150
151         for (GLuint d = 1; d < _derivative.GetRowCount(); ++d)
152         {
153             glBindBuffer(GL_ARRAY_BUFFER, _vbo_derivative(d));
154             glBufferData(GL_ARRAY_BUFFER, higher_order_derivative_byte_size, 0, _usage_flag);
155
156             coordinate = (GLfloat*)glMapBuffer(GL_ARRAY_BUFFER, GL_WRITE_ONLY);
157
158             if (!coordinate)
159             {
160                 glBindBuffer(GL_ARRAY_BUFFER, 0);
161                 DeleteVertexBufferObjects();
162                 return GL_FALSE;
163             }
164
165             for (GLuint i = 0; i < curve_point_count; ++i)
166             {
167                 DCoordinate3 sum = _derivative(0, i);
168                 sum += _derivative(d, i);
169
170                 for (GLint j = 0; j < 3; ++j)
171                 {
172                     *coordinate = (GLfloat)_derivative(0, i)[j];
173                     *(coordinate + 3) = (GLfloat)sum[j];
174                     ++coordinate;
175                 }
176
177                 coordinate += 3;
```

Generic curves – source file, part VII

GenericCurves3.cpp

```
171         }

172         if (!glUnmapBuffer(GL_ARRAY_BUFFER))
173         {
174             glBindBuffer(GL_ARRAY_BUFFER, 0);
175             DeleteVertexBufferObjects();
176             return GL_FALSE;
177         }
178     }

179     glBindBuffer(GL_ARRAY_BUFFER, 0);

180     return GL_TRUE;
181 }

182 GLfloat* GenericCurve3::MapDerivatives(GLuint order, GLenum access_mode) const
183 {
184     if (order >= _derivative.GetRowCount())
185         return 0;

186     if (access_mode != GL_READ_ONLY && access_mode != GL_WRITE_ONLY && access_mode != GL_READ_WRITE)
187         return 0;

188     glBindBuffer(GL_ARRAY_BUFFER, _vbo_derivative(order));

189     return (GLfloat*)glMapBuffer(GL_ARRAY_BUFFER, access_mode);
190 }

191 GLboolean GenericCurve3::UnmapDerivatives(GLuint order) const
192 {
193     if (order >= _derivative.GetRowCount())
194         return GL_FALSE;

195     glBindBuffer(GL_ARRAY_BUFFER, _vbo_derivative(order));
```

Generic curves – source file, part VIII

GenericCurves3.cpp

```
196     return glUnmapBuffer(GL_ARRAY_BUFFER);
197 }

198 // get derivative by value
199 DCoordinate3 GenericCurve3::operator ()(GLuint order, GLuint index) const
200 {
201     return _derivative(order, index);
202 }

203 // get derivative by reference
204 DCoordinate3& GenericCurve3::operator ()(GLuint order, GLuint index)
205 {
206     return _derivative(order, index);
207 }

208 // other update and query methods
209 GLboolean GenericCurve3::SetDerivative(GLuint order, GLuint index, GLdouble x, GLdouble y, GLdouble z)
210 {
211     if (order >= _derivative.GetRowCount() || index >= _derivative.GetColumnCount())
212         return GL_FALSE;

213     _derivative(order, index)[0] = x;
214     _derivative(order, index)[1] = y;
215     _derivative(order, index)[2] = z;

216     return GL_TRUE;
217 }

218 GLboolean GenericCurve3::SetDerivative(GLuint order, GLuint index, const DCoordinate3& d)
219 {
220     if (order >= _derivative.GetRowCount() || index >= _derivative.GetColumnCount())
221         return GL_FALSE;

222     _derivative(order, index) = d;
```

Generic curves – source file, part IX

GenericCurves3.cpp

```
223     return GL_TRUE;
224 }

225 GLboolean GenericCurve3::GetDerivative(GLuint order, GLuint index, GLdouble& x, GLdouble& y, GLdouble& z)
226 {
227     if (order >= _derivative.GetRowCount() || index >= _derivative.GetColumnCount())
228         return GL_FALSE;

229     x = _derivative(order, index)[0];
230     y = _derivative(order, index)[1];
231     z = _derivative(order, index)[2];

232     return GL_TRUE;
233 }

234 GLboolean GenericCurve3::GetDerivative(GLuint order, GLuint index, DCoordinate3& d) const
235 {
236     if (order >= _derivative.GetRowCount() || index >= _derivative.GetColumnCount())
237         return GL_FALSE;

238     d = _derivative(order, index);

239     return GL_TRUE;
240 }

241 GLuint GenericCurve3::GetMaximumOrderOfDerivatives() const
242 {
243     return _derivative.GetRowCount() - 1;
244 }

245 GLuint GenericCurve3::GetPointCount() const
246 {
247     return _derivative.GetColumnCount();
248 }
```

Generic curves – source file, part X

GenericCurves3.cpp

```
249 GLenum GenericCurve3::GetUsageFlag() const
250 {
251     return _usage_flag;
252 }

253 // destructor
254 GenericCurve3::~GenericCurve3()
255 {
256     DeleteVertexBufferObjects();
257 }

258 //----------------------------------
259 // input/output from/to stream
260 //----------------------------------
261 ostream& cagd::operator <<(ostream& lhs, const GenericCurve3& rhs)
262 {
263     return lhs << rhs._usage_flag << " " << rhs._derivative << endl;
264 }

265 std::istream& cagd::operator >>(std::istream& lhs, GenericCurve3& rhs)
266 {
267     rhs.DeleteVertexBufferObjects();

268     return lhs >> rhs._usage_flag >> rhs._derivative;
269 }
```

Description

- Class ParametricCurve3 is based on classes DCoordinate3, Matrix, and GenericCurve3, and provides methods for creating and rendering smooth parametric curves of the form

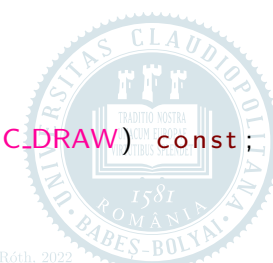
$$\mathbf{c}(u) = \begin{bmatrix} x(u) \\ y(u) \\ z(u) \end{bmatrix}, \; u \in [u_{\min}, u_{\max}] .$$

- Create a new folder Parametric aside of your project file. Copy the files ParametricCurves3.h and ParametricCurves3.cpp into this folder and finish their implementation.

Parametric curves – header file, part I

ParametricCurves3.h

```
1  #pragma once
2  #include "../Core/DCoordinates3.h"
3  #include "../Core/GenericCurves3.h"
4  #include "../Core/Matrices.h"
5  namespace cagd
6  {
7      //--------------------------------
8      // class ParametricCurve3
9      //--------------------------------
10     class ParametricCurve3
11     {
12     public:
13         typedef DCoordinate3 (*Derivative)(GLdouble);
14     private:
15         // definition domain
16         GLdouble _u_min, _u_max;
17         // derivatives of coordinate functions
18         RowMatrix<Derivative> _derivatives;
19     public:
20         // special constructor
21         ParametricCurve3(const RowMatrix<Derivative>& derivatives, GLdouble u_min, GLdouble u_max);
22         // calculate derivative at the parameter value u
23         DCoordinate3 operator ()(GLuint order, GLdouble u) const;
24         // generate image/arc
25         GenericCurve3* GenerateImage(GLuint div_point_count, GLenum usage_flag = GL_STATIC_DRAW) const;
26         // set/get definition domain
27         GLvoid SetDefinitionDomain(GLdouble u_min, GLdouble u_max);
```



Parametric curves – header file, part II

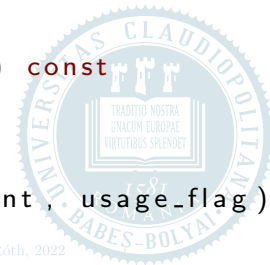
ParametricCurves3.h

```
28          GLvoid GetDefinitionDomain(GLdouble& u_min, GLdouble& u_max) const;

29          // set derivatives
30          GLvoid SetDerivatives(const RowMatrix<Derivative>& derivatives);
31      };
32  }
```

Parametric curves – source file, part I

ParametricCurves3.cpp

```
1  #include "ParametricCurves3.h"

2  using namespace cagd;
3  using namespace std;

4  //--------------------------------------------
5  // implementation of class ParametricCurve3
6  //--------------------------------------------

7  // special constructor
8  ParametricCurve3::ParametricCurve3(
9          const RowMatrix<Derivative>& derivatives,
10         GLdouble u_min, GLdouble u_max):
11         _u_min(u_min), _u_max(u_max),
12         _derivatives(derivatives)
13 {
14 }

15 // calculate derivative at parameter u:
16 // 0th order derivative corresponds to the curve point at parameter u
17 // 1st order derivative corresponds to the tangent vector at parameter u
18 // 2nd order derivative corresponds to the acceleration vector at parameter u
19 // etc.
20 DCoordinate3 ParametricCurve3::operator ()(GLuint order, GLdouble u) const
21 {
22     return _derivatives(order)(u);
23 }

24 // generate image of the parametric curve
25 GenericCurve3* ParametricCurve3::GenerateImage(GLuint div_point_count, GLenum usage_flag) const
26 {
27     GenericCurve3* result = nullptr;

28     result = new (nothrow) GenericCurve3(_derivatives.GetColumnCount() - 1, div_point_count, usage_flag);
```



Parametric curves – source file, part II

ParametricCurves3.cpp

```
29      if (!result)
30      {
31          return nullptr;
32      }

33      // set derivatives at the endpoints of the parametric curve
34      for (GLuint order = 0; order < _derivatives.GetColumnCount(); ++order)
35      {
36          (*result)(order, 0) = _derivatives(order)(_u_min);
37          (*result)(order, div_point_count - 1) = _derivatives(order)(_u_max);
38      }

39      // calculate derivatives at inner curve points
40      GLdouble u_step = (_u_max - _u_min) / (div_point_count - 1);
41      GLdouble u = _u_min;

42      for (GLuint i = 1; i < div_point_count - 1; ++i)
43      {
44          u += u_step;

45          for (GLuint order = 0; order < _derivatives.GetColumnCount(); ++order)
46          {
47              (*result)(order, i) = _derivatives(order)(u);
48          }
49      }

50      return result;
51  }

52  // set/get definition domain
53  GLvoid ParametricCurve3::SetDefinitionDomain(GLdouble u_min, GLdouble u_max)
54  {
55      // homework
56  }
57
```

Parametric curves – source file, part III

ParametricCurves3.cpp

```
58 GLvoid ParametricCurve3::GetDefinitionDomain(GLdouble& u_min, GLdouble& u_max) const
59 {
60     // homework
61 }

62 // set derivatives
63 GLvoid ParametricCurve3::SetDerivatives(const RowMatrix<Derivative> &derivatives)
64 {
65     // homework
66 }
```

Testing parametric curves

- Create a new folder Test aside of your project file. Copy the files TestFunctions.h and TestFunctions.cpp into this folder.
- Modify the files GLWidget.h and GLWidget.cpp accordingly to the next slides.

Testing parametric curves – declaring/defining test functions

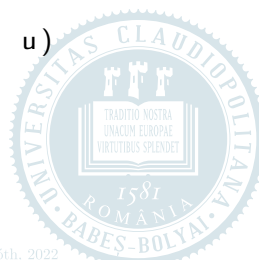
TestFunctions.h/cpp

- TestFunctions.h

```
1  #pragma once

2  #include "../Core/DCoordinates3.h"

3  namespace cagd
4  {
5      namespace spiral_on_cone
6      {
7          extern GLdouble u_min, u_max;

8          DCoordinate3 d0(GLdouble);
9          DCoordinate3 d1(GLdouble);
10         DCoordinate3 d2(GLdouble);
11     }

12     ...

13     namespace ...
14     {
15         extern GLdouble u_min, u_max;

16         DCoordinate3 d0(GLdouble);
17         DCoordinate3 d1(GLdouble);
18         DCoordinate3 d2(GLdouble);
19         DCoordinate3 d3(GLdouble);
20         ...
21     }

22     ...
23 }
```

- TestFunctions.cpp

```
1  #include <cmath>
2  #include "TestFunctions.h"

3  using namespace cagd;
4  using namespace std;

5  GLdouble spiral_on_cone::u_min = -6.2831;
6  GLdouble spiral_on_cone::u_max = +6.2831;

7  DCoordinate3 spiral_on_cone::d0(GLdouble u)
8  {
9      return DCoordinate3(
10             u * cos(u),
11             u * sin(u),
12             u);
13 }

14 DCoordinate3 spiral_on_cone::d1(GLdouble u)
15 {
16     return DCoordinate3(
17             cos(u) - u * sin(u),
18             sin(u) + u * cos(u),
19             1.0);
20 }

21 DCoordinate3 spiral_on_cone::d2(GLdouble u)
22 {
23     return DCoordinate3(
24             -2.0 * sin(u) - u * cos(u),
25             2.0 * cos(u) - u * sin(u),
26             0);
27 }
```



Testing parametric curves – modifying GLWidget.h

GLWidget.h

```
1  #pragma once

2  #include <GL/glew.h>
3  #include <QOpenGLWidget>
4  #include "../Parametric/ParametricCurves3.h"

5  namespace cagd
6  {
7      class GLWidget: public QOpenGLWidget
8      {
9          Q_OBJECT

10     private:

11         ...

12         ParametricCurve3* _pc;
13         GenericCurve3*    _image_of_pc;

14     public:

15         ...

16         virtual ~GLWidget();

17     public slots:

18         ...

19     };
20 }
```

Testing parametric curves – modifying GLWidget.cpp, part I

GLWidget.cpp

```
1  #include "GLWidget.h"
2  #include <iostream>
3  #include "../Core/Matrices.h"
4  #include "../Test/TestFunctions.h"

5  using namespace cagd;
6  using namespace std;

7  ...

8  GLWidget::~GLWidget()
9  {
10     if (_pc)
11         delete _pc, _pc= nullptr;

12     if (_image_of_pc)
13         delete _image_of_pc, _image_of_pc = nullptr;
14 }

15 void GLWidget::initializeGL()
16 {
17     ...

18     try
19     {
20         // initializing the OpenGL Extension Wrangler library
21         GLenum error = glewInit();

22         if (error != GLEW_OK)
23         {
24             throw Exception("Could not initialize the OpenGL Extension Wrangler Library!");
25         }
```

Testing parametric curves – modifying GLWidget.cpp, part II

GLWidget.cpp

```
26          if (!glewIsSupported("GL_VERSION_2_0"))
27          {
28              throw Exception("Your graphics card is not compatible with OpenGL 2.0+! "
29                              "Try to update your driver or buy a new graphics adapter!");
30          }

31          // create and store your geometry in display lists or vertex buffer objects
32          // ...
33          RowMatrix<ParametricCurve3::Derivative> derivative(3);
34          derivative[0] = spiral_on_cone::d0;
35          derivative[1] = spiral_on_cone::d1;
36          derivative[2] = spiral_on_cone::d2;

37          _pc = nullptr;
38          _pc = new (nothrow) ParametricCurve3(derivative, spiral_on_cone::u_min, spiral_on_cone::u_max);

39          if (!_pc)
40          {
41              // error: either close the application, or handle this exception
42          }

43          GLuint div_point_count = 200;
44          GLenum usage_flag = GL_STATIC_DRAW;

45          _image_of_pc = nullptr;
46          _image_of_pc = _pc->GenerateImage(div_point_count, usage_flag);

47          if (!_image_of_pc)
48          {
49              throw Exception("Could not generate the image of the parametric curve!");
50          }
51
```

Testing parametric curves – modifying GLWidget.cpp, part III

GLWidget.cpp

```
52          if (!_image_of_pc->UpdateVertexBufferObjects(usage_flag))
53          {
54              throw Exception("Could not create the vertex buffer object of the parametric curve!");
55          }

56          glEnable(GL_POINT_SMOOTH);
57          glHint(GL_POINT_SMOOTH_HINT, GL_NICEST);

58          glEnable(GL_LINE_SMOOTH);
59          glHint(GL_LINE_SMOOTH_HINT, GL_NICEST);

60          glEnable(GL_POLYGON_SMOOTH);
61          glHint(GL_POLYGON_SMOOTH_HINT, GL_NICEST);

62          glHint(GL_PERSPECTIVE_CORRECTION_HINT, GL_NICEST);

63          glEnable(GL_DEPTH_TEST);
64      }
65      catch (Exception &e)
66      {
67          cout << e << endl;
68      }
69  }


70  void GLWidget::paintGL()
71  {
72      glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

73      // saving the current modelview matrix
74      glPushMatrix();

75          // transformations
```

Testing parametric curves – modifying GLWidget.cpp, part IV

GLWidget.cpp

```
76          ...

77          if (_image_of_pc)
78          {
79              glColor3f(1.0f, 0.0f, 0.0f);
80              _image_of_pc->RenderDerivatives(0, GL_LINE_STRIP);

81              glPointSize(5.0f);

82                  glColor3f(0.0f, 0.5f, 0.0f);
83                  _image_of_pc->RenderDerivatives(1, GL_LINES);
84                  _image_of_pc->RenderDerivatives(1, GL_POINTS);

85                  glColor3f(0.1f, 0.5f, 0.9f);
86                  _image_of_pc->RenderDerivatives(2, GL_LINES);
87                  _image_of_pc->RenderDerivatives(2, GL_POINTS);

88              glPointSize(1.0f);

89          }

90      // restoring the former modelview matrix
91      glPopMatrix();
92  }
```

A spiral on a cone

$$\begin{cases} x(u) &= u\cos(u), \\ y(u) &= u\sin(u), \\ z(u) &= u, \end{cases}$$

$u \in [-2\pi, 2\pi]$.

Fig. 1: A spiral lying on the surface of a perpendicular cone. The green and blue vectors represent the first (i.e., the tangent vectors) and second order derivatives (i.e., acceleration vectors), respectively.

A torus knot

$$\begin{cases} x(u) &= \left(2+\cos\left(\frac{2u}{3}\right)\right)\cos(u), \\ y(u) &= \left(2+\cos\left(\frac{2u}{3}\right)\right)\sin(u), \\ z(u) &= \sin\left(\frac{2u}{3}\right), \end{cases}$$

$$u \in [0, 6\pi].$$

Homework

- Append the files TestFunctions.h and TestFunctions.cpp with at least 6 new parametric curve declarations and definitions.
- At least 2 of these new parametric curves should lie on smooth parametric surfaces (e.g., on spheres, ellipsoids, tori, cylinders, hyperboloids, Dupin cyclides), but you may also define other 2- and 3-dimensional continuous/smooth open or closed parametric curves (such as circles, ellipses, cycloids, epi- and hypocycloids, epi- and hypotrochoids, Lissajous-curves, foliums, rose and butterfly curves, witch of Agnesi, helices, Viviani curves, etc.).
- Declare/define/render at least 6 different smooth parametric curves!
- Update the GUI of the application (you should be able to interactively switch between different parametric curves, to show/hide/scale their first- and second-order derivatives, and to increase/decrease the number of subdivision points of the definition domain)!