Chapter 1

NURBS

On top of the CGAL kernel we provide an application module containing a basic representation for non-uniform rational B-spline curves and surfaces (NURBS) in 2 and 3 dimensional Euclidean space.

The eventual goal is to define a class that can be used as a basis for the manipulation of NURBS curves in 2D, NURBS curves in 3D, and NURBS surfaces in 3D, both for modelling purposes and for sweeping of arrangements of NURBS.

For the moment we provide only a limited base of operations on (open) curves and surfaces. We put most emphasis on efficiency and short computation time, more than on precision. The algorithms are provided only for number types like double, although the overall design is with template classes and functions where the user can define ones own number type as well as using the standard one provided as part of the CGAL application module. We will describe this approach in the following.

The design and implementation is due to Wolfgang Freiseisen (RISC-Linz), Daniela Vasaru (RISC-Linz), Carl Van Geem (RISC-Linz), and Markus Wabro (Uni. Linz). In the initial stage there was some discussion and exchange of expertise with Han Dataport. In a later stage the interaction with Quôc-Nam Trân (RISC-Linz) led to the development of the current state of this application module and to a scientifically relevant result in CAGD.

During the design stage of the classes - resulting in the specification described below - we set ourselves the goal that it should be easy to extend the package for NURBS in future so that it can be used for many more purposes. For a list of such purposes we kept in mind, we refer to the last section of this description.

Also these classes for NURBS are in the orthodox canonical form, as are all classes in CGAL .

Some classes have a traits class as their template parameter. In this traits class one defines the number type over which the computations will be done, and the type of points which is going to be used for the storage of control points of the NURBS curve. The dimension of the NURBS curve is implicitely determined by the dimension of the type of points used. We provide two standard traits classes: one for 2 dimensional curves, and one for 3 dimensional curves and surfaces. The user may use the standard NURBS traits class defined in file *NURBS_standard_traits_2.h* for 2 dimensional NURBS curves, and the standard NURBS traits class defined in file *NURBS_standard_traits_3.h* for 3 dimensional NURBS curves and surfaces.

We refer to the wide range of literature for all details on the theory of NURBS, in particular to [Far88], [Fuj88], and [PieTil96].

NOTE: All three dimensional algorithms are tested only briefly.

1.1 Knotvector for NURBS

Definition

A knotvector is nothing more than a sorted sequence of numbers, on which some operations are defined which are of particular use for applications related to NURBS. The class is templated with a NURBS traits class. This traits class provides the number type of the knots. We provide a standard NURBS traits class in files *NURBS_standard_traits_2.h* and *NURBS_standard_traits_2.h*.

Types

Traits::R stands for the number type which is the type of the knots. For instance, in the standard NURBS traits classes *NURBS_standard_traits_2* and *NURBS_standard_traits_3* this type is preinstantiated by *double*.

Creation

;   Default constructor.

(CGAL_Knotvector<Traits> knotvect);

Default copy constructor.

(vector<Traits::R> vect);

This constructor assigns the *STL* vector to which *vect* points to the knotvector.

Operations

vector<Traits::R >

knots()   returns the sequence of knots as a *STL* vector

int   *find_knot_span(Traits::R t)*

Returns -1 if t is outside the range of the knot vector. Returns leftmost endpoint of the segment defined by the knotvector entries in which t lies, i.e. returns i, if $t\ [i] \leq t < t\ [i{+}1]$ or if $[i] < t \leq t\ [i{+}1]$ if t equals the last element of the knot vector.

Implementation

The sorted list is an *STL container*. The algorithm finding the knot span was taken from [PieTil96]: page 68, algorithm A2.1.

1.2 Control Polygon (for curves)

Definition

In order to be compliant to [PieTil96] (Chapter 13 about exchange of data) the sequence of control points is defined as a type on its own: *CGAL_Control_polygon*. The control polygon is nothing more than a sequence of points on which some operations are defined which are of significance in relation to NURBS. The class is a template class where the template parameter is a NURBS traits class. The dimension of the points depends on the instantiation in the NURBS traits class. We provide a standard NURBS traits class both for 2 and 3 dimensional points.

Types

Traits::R stands for the number type which is the type of the coordinates of the control points. For instance, in the standard NURBS traits classes *NURBS_standard_traits_2* and *NURBS_standard_traits_3* this type is preinstantiated by *double*.

Traits::Point stands for the type of points which are stored in the control polygon. For instance, in the standard NURBS traits class for 2 dimensional NURBS curves this type is preinstantiated by *CGAL_Point_2<CGAL_Homogeneous<double> >*, and in the standard NURBS traits class for 3 dimensional NURBS curves this type is preinstantiated by *CGAL_Point_3<CGAL_Homogeneous<double> >*.

Creation

; Default constructor.

(CGAL_Controlpolygon<I > cp);

Default copy constructor.

(vector<Traits::Point > vct);

Constructor instantiating the sequence of control points by the values in the *STL* vector *vct*.

Operations

vector<Traits::Point >

controlpoints() Returns the sequence of control points as an *STL* vector.

void *change_weight(int position, Traits::R new_weight)*

Changes the weight of the control point at position *position* into value *new_weight*. That is, the homogenizing coordinate of the control point at position *position* turns into *new_weight*, while all other coordinates are devided by the old homogenizing coordinate and multiplied by *new_weight*.

1.3 NURBS curve

Definition

A NURBS curve *CGAL_NURBS_curve* is defined by a set of control points, a knotvector, and the degree. The control points are represented as homogeneous points whereby the homogeneous coordinate plays the role of the weight. The class is templated with the NURBS traits class. The dimension of the curve depends on the NURBS curve traits class only. We provide two standard NURBS curve traits classes: one for the 2 dimensional and one for the 3 dimensional NURBS curve.

Types

Traits::R stands for the number type which is the type of the coordinates of the control points as well as for the type of the values of the knots. For instance, in the standard NURBS traits classes *NURBS_standard_traits_2* and *NURBS_standard_traits_3* this type is preinstantiated by *double*.

Traits::Point stands for the type of points which are stored in the control polygon. For instance, in the standard NURBS traits class for 2 dimensional NURBS curves this type is preinstantiated by

CGAL_Point_2<*CGAL_Homogeneous*<*double*> >, and in the standard NURBS traits class for 3 dimensional NURBS curves this type is preinstantiated by *CGAL_Point_3*<*CGAL_Homogeneous*<*double*> >.
Traits stands for the template parameter which represents the whole NURBS traits class. We provide the user with two predefined standard NURBS traits classes: in the file named *NURBS_standard_traits_2.h* one finds the standard NURBS traits class for 2 dimensions, and in the file named *NURBS_standard_traits_3.h* one finds the standard NURBS traits class for 3 dimensions. For more details on the standard NURBS traits classes we refer to Section 1.7.1. The user can also define ones own NURBS traits class. For more details on user defined NURBS traits classes we refer to Section 1.7.2.

Creation

; Default constructor.

(CGAL_NURBS_curve<*I*> *nurbscurve);*
Default copy constructor.

(CGAL_Controlpolygon<*Traits*> *controlpol, CGAL_Knotvector*<*Traits*> *knotvec, int deg);*
Constructor which instantiates the nurbs curve with given control polygon *controlpol*, knotvector *knotvec*, and degree *deg*.

Operations

CGAL_Controlpolygon<*Traits*> *controlpolygon()*
Returns the control polygon of the curve.

CGAL_Knotvector<*Traits*> *knotvector()*
Returns the knot vector of the curve.

int *degree()*
Returns the degree of the curve.

void *change_a_weight(int position, Traits::R new_weight)*
Changes the weight of the control point on place *position* to value *new_weight*.

bool *point_on_nurbs(Traits::R t, Traits::Point* &*result)*
Computes the point corresponding to parameter value t on the NURBS curve, using the De Boor Algorithm. The resulting point is placed in *result* and the function returns boolean value *True*.

int *refine_knot_vector(vector*<*I::R*> *newknots)*
Precondition: The sequence of knots *newknots* must be sorted in increasing order. The sequence of knots *newknots* are the knots to be inserted. Returns -*1* if insertion succeeds, -*2* if the knots are outside the range and i if the ith point couldn't be inserted. Only when the return value is -*1*, knots are inserted.

int *remove_curve_knot(int r, int num, I::R Tol)*
Removes the rth knot at most *num* times. In case the rth knot cannot be removed exactly *num* times, it removes the knot as many times as possible. Returns the number of times the knot was removed.

Implementation

The De Boor algorithm implemented for the computation of a point on a NURBS curve is the one described in [HosLas89], page 198 – an in the literature well-known standard approach.
The insertion of new knots (the refinement of the knotvector) and the removal of a knot implemented here are described in [PieTil96] pages 164 (algorithm A5.4) and 185 (algorithm A5.8) respectively.

1.4 pmatrix (for surfaces)

The pmatrix is a help structure for the implementation of a control polyhedron. It is merely more than an *STL* vector of an *STL* vector of objects. The pmatrix is templated by I, the object. (For a control polyhedron this object will be a point with homogeneous representation.)

Constants

There are two constants defined, used in the operation 'set': *PMATRIX_OK*, *PMATRIX_OUT_OF_BOUNDS*.

Creation

(const int i, const int j);

Constructs an empty pmatrix with fixed size *(0..i-1)* times *(0..j-1)*.

(pmatrix<I> mat);

Copy constructor.

(vector< vector<I> > mat);

Constructor copying the values in the *STL* vector of an *STL* vector of objects of type I stored in *mat*.

Operations

int *set(const int i, const int j, I el)*

Sets in the pmatrix on position *(i,j)* the value to *el*. Returns *PMATRIX_OUT_OF_BOUNDS* if the pmatrix has size smaller than i or j, returns *PMATRIX_OK* if insertion succeeded.

vector< vector<I> > *vecvec()*

Returns the pmatrix as an *STL* vector of *STL* vector.

int *size1()*

Returns the size of the pmatrix in the i-direction (the number of entries possible in the i-direction).

int *size2()*

Returns the size of the pmatrix in the j-direction(the number of entries possible in the j-direction).

void *cut(const int cut1, const int cut2)*

Cuts the matrix to the format *(0..cut1)* times *(0..cut2)*.

1.5 Control Polyhedron (for surfaces)

A Control polyhedron is implemented as a pmatrix of points. The control polyhedron is templated by a NURBS traits class. We provide a standard NURBS traits class for 3 dimensional points in file *NURBS_standard_traits_3.h*.

Types

Traits::R stands for the number type which is the type of the coordinates of the control points. For instance, in the *NURBS_standard_traits_3* this type is preinstantiated by *double*.

Traits::Point stands for the type of points which are stored in the control polyhedron. For instance, in the standard NURBS traits class for 3 dimensional NURBS curves this type is preinstantiated by *CGAL_Point_3<CGAL_Homogeneous<double> >*.

Creation

; Default constructor.

(CGAL_Controlpolyhedron<Traits> cp);

Default copy constructor.

(pmatrix<Traits::Point> polyh);

Constructor, copies the pmatrix *polyh* to the control polyhedron.

Operations

pmatrix<Traits::Point>

controlpoints()

void *change_weight(int pos_x, int pos_y, Traits::R new_weight)*

Changes the weight of the control point on position i times j to *new_weight*.

vector<Traits::Point>

column_vector(const int i)

Returns the ith column vector.

vector<Traits::Point>

row_vector(const int j)

Returns the jth row vector.

1.6 NURBS surface

The NURBS surface is a tensor surface. It contains a control polyhedron (the control points ordered in a (u,v)-grid), a knot vector in the u-direction, a knot vector in the v-direction, and a degree for the u-direction and a degree for the v-direction. The class is templated by a NURBS traits class. A standard NURBS traits class is provided in file *Nurbs_standard_traits_3.h*.

Constants

There are two constants defined: *U_DIRECTION*, *V_DIRECTION*.

Creation

; Empty constructor.

(CGAL_NURBS_surface nurbssurface);

Copy constructor.

(CGAL_Controlpolyhedron<Traits> controlpol,
CGAL_Knotvector<Traits> knotvecu,
CGAL_Knotvector<Traits> knotvecv,
int degu,
int degv)

Constructor, setting the control polyhedron to *controlpol*, the knotvector in u-direction to *knotvecu*, the knotvector in v-direction to *knotvecv*, the degree for direction u to *degu*, and the degree for direction v to *degv*.

Operations

CGAL_Controlpolyhedron<Traits>
controlpolyhedron()

Returns the control polyhedron.

CGAL_Knotvector<Traits>
knotvector_u() Returns the knotvector in the u-direction.

CGAL_Knotvector<Traits>
knotvector_v() Returns the knotvector in the v-direction.

int *degree_u()* Returns the degree in the u-direction.

int *degree_v()* Returns the degree in the v-direction.

Traits::Point
point_on_nurbs(Traits::R s, Traits::R t)

Returns the point on the surface for parameter value s in the u-direction and parameter value t in the v-direction.

void *change_a_weight(int pos_u, int pos_v, Traits::R new_weight)*

Changes the weight of the control point at position *pos_u* times *pos_v* in the control polyhedron to value *new_weight*.

void *find_area(const Traits::R s, const Traits::R t, int &ku, int &kv)*

Traits::Point
derivative_b_spline(const int k, const int l, Traits::R u, Traits::R v)

Returns the *(k,l)*-derivative of the NURBS as b-spline in E^4 at *(u,v)*.

vector< vector<Traits::Point> >
derivative(const int d, Traits::R u, Traits::R v)

Returns all the *(k,l)*-derivatives with $0 <= k + l <= d$.

CGAL_NURBS_surface<Traits>
knot_refine(const int dir, vector<Traits::R> add_knots)
Refine the knots in direction *dir* by inserting *add_knots*.

CGAL_NURBS_surface<Traits>
remove_knot(const int dir, const int r, IR tol, const int num, int &t)
Tries to remove knot with index *r* in direction *dir* *num* times. After execution the number of real removals is stored in *t*. Precondition: It is assumed that knot *r* is not equal to knot *r+1*.

CGAL_NURBS_surface<Traits>
degree_elevate(const int dir, const int t)
Degree elevate the surface *t* times in direction *dir*.

CGAL_NURBS_surface<Traits>
degree_reduce(const int dir)
Degree reduce the surface in direction *dir*.

Implementation

The algorithms implemented here for the computation of a point on the surface, the derivative, knot refinement, knot removal, degree elevation and degree reduction are taken from [PieTil96].

1.7 NURBS Traits Classes

1.7.1 Standard NURBS Traits Classes

We provide two standard NURBS traits classes: one for 2 dimensional space in the file named *NURBS_standard_traits_2.h* and one for 3 dimensional space in the file named *NURBS_standard_traits_3.h*.

Parameters

The template parameter *R* represents the number type over which all computations will be done and is instantiated with *double*.

The template parameter *Point* represents the points which will be used (as the representation by homogeneous coordinates) in the control polygon of the NURBS curve. Instantiating this template parameter implicitely determines the space in which one works. In the standard NURBS traits class for 2 dimensional space this is instantiated with *CGAL_Point_2<CGAL_Homogeneous<R> >*. In the standard NURBS traits class for 3 dimensional space this is instantiated with *CGAL_Point_3<CGAL_Homogeneous<R> >*.

The template parameter *Vector* represents vectors, and their dimension must correspond to the dimension of the template parameter *Point*. In the standard NURBS traits class for 2 dimensional space this is instantiated with *CGAL_Vector_2<CGAL_Homogeneous<R> >*. In the standard NURBS traits class for 3 dimensional space this is instantiated with *CGAL_Vector_3<CGAL_Homogeneous<R> >*.

The template parameter *CPoint* represents the cartesian representation of the points which will be used in the control polygon of the NURBS curve. It is necessary to provide such a type for internal conversions between the homogeneous representation defined by *Point*. In the standard NURBS traits class for 2 dimensional space this is instantiated with *CGAL_Point_2<CGAL_Cartesian<R> >*. In the standard NURBS traits class for 3 dimensional space this is instantiated with *CGAL_Point_3<CGAL_Cartesian<R> >*.

1.7.2 Requirements for NURBS traits classes

The user can also define ones own NURBS traits class. A NURBS traits class must provide the following functionality for each of the template parameters.

A template parameter R must be instantiated with a number type which allows arithmetic operations sum, difference, and multiplication.

A template parameter *Point* must represent a point in homogeneuos space. The coordinates must be accessible by operations *hx()*, *hy()*, (also *hz()* for the 3 dimensional NURBS curve), and *hw()*, the latter for the homogenizing coordinate.

A template parameter *Vector* must represent a vector in the same dimensional space as for which template parameter *Point* is defined. The *Vector* must be provided with the arithmetic operations sum (adding two vectors) and multiplication with a scalar of the number type defined with template parameter R.

A template parameter *CPoint* must represent a point in cartesian space. The coordinates must be accessible by operations *x()* and *y()*, (also *z()* for the 3 dimensional NURBS curve).

Conversions between *Point* and *CPoint* as well as functions for the computation of the squared distance between two points must be provided.

1.8 Possible Extensions.

For the design as presented above we considered a number of possible extensions in future. Although we do not foresee the implementation of these extensions in the near future due to limitation in time, we list them here.

1.8.1 Intersection of Curves.

Algorithms for dealing with intersection should be included for possible use in sweeping a set of curve segments, and in case one wants to deal with offsets of curves and surfaces.

1.8.2 Sweeping Arrangements.

A typical technique in Computational Geometry consists of sweeping an arrangement (in 2D with a line, in 3D with a plane). We think of the possibility to sweep also arrangements of NURBS curves and/or surfaces. For this purpose it will be necessary to have the possibility of intersecting NURBS with a straight (vertical) line or plane.

1.8.3 Offsets.

Computing offsets for NURBS curves and surfaces is an interesting application. We might want to implement offsets in two ways: as done in industrial applications by numerical approximatif techniques, and on a local base in an exact algebraic way.

1.8.4 NURBS Surfaces on Triangulations.

The specification for NURBS with triangles as a basis can be very similar to the one above using patches. Roughly we can use a Triangulation instead of two knotvectors, and assign to each triangle a Control Point. Obviously the algorithms for surfaces based on triangles will differ from those for surfaces based on patches. In that respect the proposed specification takes in account also such a possible extension of the application domain of NURBS.

the curve represented by an instance of this class will be written to

Bibliography

[Far88] Gerald Farin, *Curves and Surfaces for Computer Aided Geometric Design, A Practical Guide*, CSSC, Academic Press, 1988.

[Fuj88] Fujio Yamaguchi, *Curves and Surfaces in Computer Aided Geometric Design*, Springer-Verlag, 1988.

[HosLas89] Josef Hoschek, Dieter Lasser, *Fundamentals of Computer Aided Geometric Design*, A K Peters, Wllesley, Massachusetts, 1993.

[PieTil96] Les Piegl, Wayne Tiller, *The NURBS Book (second edition)*, Springer-Verlag, 1996.