

NURBS Reference Manual

generated by Doxygen

Wed Dec 16 13:26:10 1998

Contents

1	Hierarchical Index	1
1.1	NURBS Class Hierarchy	1
2	Annotated Compound Index	3
2.1	NURBS Compound Index	3
3	Compound Documentation	5
3.1	BoundingBoxGL Class Reference	5
3.2	CPointGL Class Reference	7
3.3	HCPPointGL Class Reference	10
3.4	HNurbsSurface Class Reference	12
3.5	HNurbsSurfaceGL Class Reference	30
3.6	HNurbsSurfaceSP Class Reference	40
3.7	KnotGL Class Reference	47
3.8	Material Class Reference	49
3.9	MatrixRT Class Reference	51
3.10	NurbsCpolygonGL Class Reference	57
3.11	NurbsCurve Class Reference	59
3.12	NurbsCurveArray Class Reference	116
3.13	NurbsCurveGL Class Reference	121
3.14	NurbsCurveSP Class Reference	127
3.15	NurbsGL Class Reference	130
3.16	NurbsListGL Class Reference	135

3.17 NurbsSpolygonGL Class Reference	137
3.18 NurbsSurface Class Reference	139
3.19 NurbsSurfaceArray Class Reference	185
3.20 NurbsSurfaceGL Class Reference	189
3.21 NurbsSurfaceSP Class Reference	197
3.22 ObjectGL Class Reference	204
3.23 ObjectListGL Class Reference	217
3.24 ObjectRefGL Class Reference	231
3.25 ObjectRefListGL Class Reference	232
3.26 ParaCurve Class Reference	235
3.27 ParaSurface Class Reference	243
3.28 PointGL Class Reference	253
3.29 PointListGL Class Reference	255
3.30 RGBAf Struct Reference	259
3.31 SPointCurveGL Class Reference	260
3.32 SPointGL Class Reference	263
3.33 SPointHSurfaceGL Class Reference	264
3.34 SPointSurfaceGL Class Reference	267

Chapter 1

Hierarchical Index

1.1 NURBS Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically.

Material	49
MatrixRT	51
NurbsCurveArray	116
NurbsSurfaceArray	185
OpenGL	204
BoundingBoxGL	5
CPointGL	7
HPointGL	10
SPointGL	263
SPointCurveGL	260
SPointHSurfaceGL	264
SPointSurfaceGL	267
KnotGL	47
NurbsCpolygonGL	57
NurbsGL	130
HNurbsSurfaceGL	30
NurbsCurveGL	121
NurbsSurfaceGL	189
NurbsSpolygonGL	137
ObjectListGL	217
NurbsListGL	135
ObjectRefListGL	232
ObjectRefGL	231
PointGL	253
PointListGL	255

ParaCurve	235
NurbsCurve	59
NurbsCurveSP	127
ParaSurface	243
NurbsSurface	139
HNurbsSurface	12
HNurbsSurfaceSP	40
NurbsSurfaceSP	197
RGBAf	259

Chapter 2

Annotated Compound Index

2.1 NURBS Compound Index

Here are the classes, structs and unions with brief descriptions:

BoundingBoxGL (Holds a bounding box)	5
CPointGL (A class to hold a control point)	7
HCPPointGL (A class to hold a control point from a HNURBS) . . .	10
HNurbsSurface (A hierarchichal NURBS surface class)	12
HNurbsSurfaceGL (A HNURBS surface class for OpenGL)	30
HNurbsSurfaceSP (A NURBS surface with surface point)	40
KnotGL (A class to hold a knot)	47
Material (A class for the material properties of an object)	49
MatrixRT (A matrix for rotation and translation transformation) . .	51
NurbsCpolygonGL (Holds the control polygon for a curve)	57
NurbsCurve (A NURBS curve class)	59
NurbsCurveArray (An array of NurbsCurve)	116
NurbsCurveGL (A NURBS curve class with OpenGL interface) . .	121
NurbsCurveSP (A NURBS curve with surface point)	127
NurbsGL (A Virtual NURBS object class)	130
NurbsListGL (A linked list of NurbsGL)	135
NurbsSpolygonGL (Holds the control polygon for a surface)	137
NurbsSurface (A class to represent a NURBS surface)	139
NurbsSurfaceArray (An array of NurbsSurface)	185
NurbsSurfaceGL (A NURBS surface class for OpenGL)	189
NurbsSurfaceSP (A NURBS surface with surface point)	197
ObjectGL (The base class for OpenGL objects)	204
ObjectListGL (A link list of ObjectGL)	217

ObjectRefGL (A reference object for OpenGL)	231
ObjectRefListGL (A link list of ObjectRefListGL)	232
ParaCurve (An abstract parametric curve class)	235
ParaSurface (An abstract parametric surface class)	243
PointGL (A class to hold a 3D point)	253
PointListGL (A class to hold a list of points)	255
RGBAf (A class to hold rgba floating point values)	259
SPointCurveGL (A class to hold a control point from a NURBS curve or surface)	260
SPointGL (A class to hold a control point from a NURBS surface) .	263
SPointHSurfaceGL (A class to hold a HNURBS surface point) . . .	264
SPointSurfaceGL (A class to hold a surface point)	267

Chapter 3

Compound Documentation

3.1 BoundingBoxGL Class Reference

Holds a bounding box.

```
#include <nurbs/nurbsGL.hh>
```

Inherits **ObjectGL**.

Public Members

- **BoundingBoxGL** ()
an object to represent a bounding box.
- void **glObject** () const
generates a bounding box.
- void **setColorXYZ** (const Color& colX, const Color& colY, const Color& colZ)
- Point3Df **minP**
- Point3Df **maxP**

Protected Members

- Color **colorX**
 - Color **colorY**
 - Color **colorZ**
-

Detailed Description

Holds a bounding box.

Author(s):

Philippe Lavoie

Date:

23 September 1997

Member Function Documentation

3.1.1 BoundingBoxGL::BoundingBoxGL()

an object to represent a bounding box.

This will set the colors for the X,Y and Z axis to the value specified by the axisXColorDefault, axisYColorDefault and axisZColorDefault global variables. Use setColorXYZ() if you want to change this default.

Author(s):

Philippe Lavoie

Date:

23 September 1997

3.1.2 void BoundingBoxGL::glObject() const

generates a bounding box.

This function generates a bounding box around the NURBS object.

Author(s):

Philippe Lavoie

Date:

23 September 1997

Reimplemented from **ObjectGL**.

The documentation for this class was generated from the following files:

- nurbsGL.hh
- nurbsGL.cc

3.2 CPointGL Class Reference

A class to hold a control point.

`#include <nurbs/nurbsGL.hh>`

Inherits **ObjectGL**.

Inherited by **HPointGL** and **SPointGL**.

Public Members

- **CPointGL** (HPoint3Df& cp, int i, int j=-1)
- **~CPointGL** ()
- virtual void **glObject** () const

Displays a control point.

- virtual void **modify** (const HPoint3Df& v)
- virtual void **modifySym** (const HPoint3Df &v)

Modify a control point and his symmetrical point.

- void **setSize** (int s)
- HPoint3Df& **point** () const
- void **setSym** (CPointGL* sp, float x, float y, float z, float w)
- int **row** () const
- int **col** () const

Protected Members

- **CPointGL** ()
- HPoint3Df& **cpoint**
- int **psize**
- int **i0**
- int **j0**
- CPointGL* **symPoint**
- float **xCoord**
- float **yCoord**
- float **zCoord**
- float **wCoord**

Detailed Description

A class to hold a control point.

Author(s):

Philippe Lavoie

Date:

23 September 1997

Member Function Documentation

3.2.1 void CPointGL::glObject() const [virtual]

Displays a control point.

Author(s):

Philippe Lavoie

Date:

30 September 1997

Reimplemented from **ObjectGL**.

Reimplemented in **SPointSurfaceGL**, **SPointHSurfaceGL**,
SPointCurveGL and **HCPPointGL**.

3.2.2 void CPointGL::modifySym(const HPoint3Df &v) [virtual]

Modify a control point and his symmetrical point.

Parameters:

v - modify the points by this value

Author(s):

Philippe Lavoie

Date:

29 January 1998

The documentation for this class was generated from the following files:

- nurbsGL.hh
- nurbsGL.cc

3.3 HCPPointGL Class Reference

A class to hold a control point from a HNURBS.

`#include <nurbs/nurbsGL.hh>`

Inherits **CPointGL**.

Public Members

- **HCPPointGL** (HPoint3Df& off, int i, int j, HNurbsSurfaceGL* hs, int mod)
- **~HCPPointGL** ()
- virtual void **glObject** () const
Displays a control point.
- virtual void **modify** (const HPoint3Df& v)
Modifies the offset point.

Protected Members

- HPoint3Df& **offset**
 - HNurbsSurfaceGL* **s**
 - int **canModify**
-

Detailed Description

A class to hold a control point from a HNURBS.

Author(s):

Philippe Lavoie

Date:

3 November 1997

Member Function Documentation

3.3.1 void HCPoinGL::glObject() const [virtual]

Displays a control point.

Author(s):

Philippe Lavoie

Date:

3 November 1997

Reimplemented from **ObjectGL**.

3.3.2 void HCPoinGL::modify(const HPoint3Df& v) [virtual]

Modifies the offset point.

Author(s):

Philippe Lavoie

Date:

3 November 1997

Reimplemented from **CPointGL**.

The documentation for this class was generated from the following files:

- nurbsGL.hh
- nurbsGL.cc

3.4 HNurbsSurface Class Reference

A hierarchichal NURBS surface class.

`#include <nurbs/hnurbsS.hh>`

Inherits **NurbsSurface**.

Inherited by **HNurbsSurfaceSP**.

Public Members

- **HNurbsSurface** ()
The basic constructor.
- **HNurbsSurface** (const NurbsSurface<T, N>& S)
Constructs a base HNURBS.
- **HNurbsSurface** (const HNurbsSurface<T, N>& S)
Constructs a base HNURBS.
- **HNurbsSurface** (HNurbsSurface<T, N>* base)
Constructor with a base level.
- **HNurbsSurface** (HNurbsSurface<T, N>* base, const Vector<T>& xU, const Vector<T>& xV)
Constructor with a base level.
- **HNurbsSurface** (HNurbsSurface<T, N>* base, const HNurbsSurface<T, N> &surf)
A level constructor.
- virtual **~HNurbsSurface** ()
Destructor.
- HNurbsSurface<T,N>* **baseLevel** () const
- HNurbsSurface<T,N>* **nextLevel** () const
- HNurbsSurface<T,N>* **firstLevel** () const
- HNurbsSurface<T,N>* **lastLevel** () const
- void **splitUV** (int nu, int nv, Vector<T> &nU, Vector<T> &nV)
Insert n knots between each knots.
- void **splitUV** (int nu, int su, int nv, int sv, Vector<T> &nU, Vector<T> &nV)
Insert n knots between each knots.

- virtual HNurbsSurface<T,N>* **addLevel** (int nsplit)
Adds a level to this HNURBS surface.
- virtual HNurbsSurface<T,N>* **addLevel** ()
Adds a level to this HNURBS surface.
- virtual void **copy** (const HNurbsSurface<T, N>& nS)
Copies a HNurbs Surface and all it children.
- int **modifies** (T u, T v)
Specifies the level that modifies the point.
- HPoint_nD<T,N> **operator()** (T u, T v) const
- HPoint_nD<T,N> **hpointAt** (T u, T v, int lod=-1) const
- void **deriveAtH** (T u, T v, int, Matrix< HPoint_nD<T, N> >&, int lod=-1) const
Finds the derivative of the point (u,v).
- void **deriveAt** (T u, T v, int, Matrix< Point_nD<T, N> >&, int lod=-1) const
Finds the derivative of the point (u,v).
- int **movePointOffset** (T u, T v, const Point_nD<T, N>& delta)
Move a point on the surface.
- void **scale** (const Point_nD<T, N>& s)
Scales the object.
- int **initBase** (int force=0)
Initialize the base surface.
- virtual void **updateSurface** (int i0=-1, int j0=-1)
updates the NURBS surface.
- virtual void **updateLevels** (int updateLevel=-1)
Update the surface for all the levels.
- int **isoCurveU** (T u, NurbsCurve<T, N>& c, int lod=-1) const
generates an iso curve in the U direction.
- int **isoCurveV** (T v, NurbsCurve<T, N>& c, int lod=-1) const
generates an iso curve in the V direction.

- int **read** (const char* filename)
Reads a HNURBS surface from a file.
- int **write** (const char* filename) const
write a HNURBS surface to a file.
- virtual int **read** (ifstream &fin)
Read a HNURBS surface from a file stream.
- int **write** (ofstream &fout) const
Write a HNURBS surface to a file stream.
- int **maxLevel** () const
the maximum level of detail.
- int **modifiedN** () const
- void **refineKnots** (const Vector<T>& nU, const Vector<T>& nV)
Refine both knot vectors.
- void **refineKnotU** (const Vector<T>& X)
Refines the U knot vector.
- void **refineKnotV** (const Vector<T>& X)
Refines the V knot vector.
- void **axis** (int i, int j, Point_nD<T, N>& xaxis, Point_nD<T, N>& yaxis, Point_nD<T, N>& zaxis) const
- void **setFixedOffsetVector** (const Point_nD<T, N> &I, const Point_nD<T, N> &J, const Point_nD<T, N>& K)
The offset vector are fixed.
- void **setVariableOffsetVector** ()
The offset vector are variable.
- Matrix< HPoint_nD<T,N> > **offset**
- Vector<T> **rU**
- Vector<T> **rV**
- const int& **level**

Protected Members

- NurbsSurface<T,N> **baseSurf**
- HNurbsSurface<T,N>* **baseLevel_**
- HNurbsSurface<T,N> * **nextLevel_**

- HNurbsSurface<T,N> * **firstLevel_**
- HNurbsSurface<T,N> * **lastLevel_**
- Matrix< Point_nD<T,N> > **ivec**
- Matrix< Point_nD<T,N> > **jvec**
- Matrix< Point_nD<T,N> > **kvec**
- int **level_**
- int **updateN**
- int **baseUpdateN**
- int **update_**
- T **uS_**
- T **uE_**
- T **vS_**
- T **vE_**
- T **uD**
- T **vD**
- int **fixedOffset**

Detailed Description

A hierarchichal NURBS surface class.

This class can represent and manipulate a hierarchical NURBS surface. The surface is composed of points in homogenous space. It can have any degree and have any number of control points.

This does not correspond to the HBSplines given by Forsey. However I hope that it will be usefull for interactive manipulations of NURBS surfaces.

Other aspects of my implementation are different. They will be documented when the class is fully fonctionnal.

Presently there is only a limited set of functions available for this class. And honestly, until I can optimize the space requirement of the class I don't think you should build anything critical with a HNurbsSurface.

Author(s):

Philippe Lavoie

Date:

28 September 1997

Member Function Documentation

3.4.1 `HNurbsSurface::HNurbsSurface()`

The basic constructor.

Author(s):

Philippe Lavoie

Date:

7 October 1997

3.4.2 `HNurbsSurface::HNurbsSurface(const HNurbsSurface<T,N>& S)`

Constructs a base HNURBS.

Constructs a base HNURBS. This HNURBS surface is set to level 0. And it corresponds to the NURBS surface.

This constructor does not transform the NURBS surface into a HNURBS surface. It only copies the values from the NURBS surface as it's base offset values.

Parameters:

S - the NURBS surface at level 0

Author(s):

Philippe Lavoie

Date:

7 October 1997

3.4.3 `HNurbsSurface::HNurbsSurface(const HNurbsSurface<T,N>& S)`

Constructs a base HNURBS.

Constructs a base HNURBS. This HNURBS surface is set to level 0. And it corresponds to the NURBS surface.

This constructor does not transform the NURBS surface into a HNURBS surface. It only copies the values from the NURBS surface as it's base offset values.

Parameters:

S - the NURBS surface at level 0

Author(s):

Philippe Lavoie

Date:

7 October 1997

3.4.4

```
HNurbsSurface::HNurbsSurface(HNurbsSurface<T,N>*  
base)
```

Constructor with a base level.

Parameters:

base - the base level

xU - the U knots to insert in this level

xV - the V knots to insert in this level

Warning:

The base pointer must be pointing to a valid object

Author(s):

Philippe Lavoie

Date:

7 October 1997

3.4.5

```
HNurbsSurface::HNurbsSurface(HNurbsSurface<T,N>*  
base, const Vector<T>& xU, const Vector<T>&  
xV)
```

Constructor with a base level.

Parameters:

base - the base level

xU - the U knots to insert in this level

xV - the V knots to insert in this level

Warning:

The base pointer must be pointing to a valid object

Author(s):

Philippe Lavoie

Date:

7 October 1997

3.4.6

**HNurbsSurface::HNurbsSurface(HNurbsSurface<T,N>
*base, const HNurbsSurface<T,N> &surf)**

A level constructor.

Parameters:

base - the base of this level

the - values for this new level

Warning:

The base pointer must be pointing to a valid object

Author(s):

Philippe Lavoie

Date:

7 October 1997

3.4.7 HNurbsSurface::~~HNurbsSurface() [virtual]

Destructor.

Deletes all the levels.

Author(s):

Philippe Lavoie

Date:

7 October 1997

**3.4.8 void HNurbsSurface::splitUV(int nu, int nv,
Vector<T> &nU, Vector<T> &nV)**

Insert n knots between each knots.

Insert nu knots between each knots in the U vector and nv knots between each knots in the V vector.

This does not perform a split. It just generates a suitable rU and rV vector. It is suggested that splitting should be done for the level above, not the local level.

Parameters:

nu - the number of new knots between each knots in U.
nv - the number of new knots between each knots in V.
nU - the new refinement knot vector in U
nV - the new refinement knot vector in V

Author(s):

Philippe Lavoie

Date:

7 October 1997

3.4.9 void HNurbsSurface::splitUV(int nu, int su, int nv, int sv, Vector<T> &nU, Vector<T> &nV)

Insert n knots between each knots.

Insert nu knots between each knots in the U vector and nv knots between each knots in the V vector.

This doesn't not perform a split. It just generates a suitable rU and rV vector. It is suggested that splitting should be done for the level above, not the local level.

Parameters:

nu - the number of new knots between each knots in U.
su - the multiplicity of the each new knot in U
nv - the number of new knots between each knots in V.
sv - the multiplicity of the each new knot in V
nU - the new refinement knot vector in U
nV - the new refinement knot vector in V

Author(s):

Philippe Lavoie

Date:

7 October 1997

3.4.10 HNurbsSurface<T,N>* HNurbsSurface::addLevel(int n) [virtual]

Adds a level to this HNURBS surface.

Parameters:

n - the number of new knots between each knots.

Returns:

a pointer to the new level or 0 if there was an error.

Warning:

returns 0, if there is already a nextlevel.

Author(s):

Philippe Lavoie

Date:

7 October 1997

3.4.11 **HNurbsSurface<T,N>*** **HNurbsSurface::addLevel()** [virtual]

Adds a level to this HNURBS surface.

Parameters:

n - the number of new knots between each knots.

Returns:

a pointer to the new level or 0 if there was an error.

Warning:

returns 0, if there is already a nextlevel.

Author(s):

Philippe Lavoie

Date:

7 October 1997

Reimplemented in **HNurbsSurfaceSP**.

3.4.12 **void** **HNurbsSurface::copy(const** **HNurbsSurface<T,N>& nS)** [virtual]

Copies a HNurbs Surface and all it children.

Parameters:

ns - the HNurbs surface to copy

Author(s):

Philippe Lavoie

Date:

7 October 1997

Reimplemented in **HNurbsSurfaceSP**.**3.4.13 int HNurbsSurface::modifies(T u, T v)**

Specifies the level that modifies the point.

Specifies what level modifies the point (u,v) **Parameters:** u - the u parametric value v - the v parametric value**Author(s):**

Philippe Lavoie

Date:

7 October 1997

3.4.14 void HNurbsSurface::deriveAtH(T u, T v, int d, Matrix< HPoint_nD<T,N> >& ders,int lod) constFinds the derivative of the point (u,v) .Computes the matrix of derivatives at (u,v) . The value of skl(k,l) represents the derivative of the surface $S(u,v)$ with respect to u , k times and to v , l times.**Parameters:** u - the u parametric value v - the v parametric value $ders$ - the Matrix of derivatives.**Author(s):**

Philippe Lavoie

Date:

7 October 1997

3.4.15 void HNurbsSurface::deriveAt(T u, T v, int d, Matrix< Point_nD<T,N> >& ders, int lod) const

Finds the derivative of the point (u,v) .

Computes the matrix of derivatives at u,v . The value of skl(k,l) represents the derivative of the surface $S(u,v)$ with respect to u , k times and to v , l times.

Parameters:

u - the u parametric value
 v - the v parametric value
 $ders$ - the Matrix of derivatives.

Author(s):

Philippe Lavoie

Date:

7 October 1997

3.4.16 int HNurbsSurface::movePointOffset(T u, T v, const Point_nD<T,N>& delta)

Move a point on the surface.

This moves the point $s(u,v)$ by delta. As this is a HNURBS surface. It moves the offset surface by delta, it doesn't move the surface point per say.

Parameters:

u - the parameter in the u direction
 v - the parameter in the v direction
 $delta$ - the displacement of the point $s(u,v)$

Returns:

1 if the operation was succesfull, 0 otherwise

Warning:

u and v must be in a valid range.

Author(s):

Philippe Lavoie

Date:

3 June 1998

3.4.17 void HNurbsSurface::scale(const Point_nD<T,N>& s)

Scales the object.

Parameters:

s - the scaling factor

Author(s):

Philippe Lavoie

Date:

11 June 1998

3.4.18 int HNurbsSurface::initBase(int force=0)

Initialize the base surface.

Initialize the base surface from the previous level if it has been modified.

Parameters:

force - if set, this forces an update of the base surface

Returns:

1 if the base surface is modified, 0 otherwise.

Author(s):

Philippe Lavoie

Date:

15 April 1998

3.4.19 void HNurbsSurface::updateSurface(int i0=-1, int j0=-1) [virtual]

updates the NURBS surface.

Updates the NURBS surface according to the offset values and its base level. You can update only one control point from the surface if you specify a value for *i* and *j* or you can update all the points if *i0* or *j0* is below 0.

Parameters:

i0 - the row of the control point to update

j0 - the column of the control point to update

Author(s):

Philippe Lavoie

Date:

7 October 1997

Reimplemented in **HNurbsSurfaceSP**.**3.4.20 void HNurbsSurface::updateLevels(int upLevel)**
[virtual]

Update the surface for all the levels.

Parameters:*upLevel* - updates the levels up to this level of detail**Author(s):**

Philippe Lavoie

Date:

7 October 1997

Reimplemented in **HNurbsSurfaceSP**.**3.4.21 int HNurbsSurface::isoCurveU(T u,**
NurbsCurve<T,N>& c,int lod=-1) const

generates an iso curve in the U direction.

Generates an iso-parametric curve which goes through the parametric value *u* along the U direction.**Parameters:***u* - the U parametric value*c* - the iso-parametric curve*lod* - the level of detail to draw the curve with**Returns:**

0 if an error occured, 1 otherwise.

Warning:the parametric value *u* must be in a valid range

Author(s):

Philippe Lavoie

Date:

7 October, 1997

**3.4.22 int HNurbsSurface::isoCurveV(T v,
NurbsCurve<T,N>& c,int lod=-1) const**

generates an iso curve in the V direction.

Generates an iso-parametric curve which goes through the parametric value *v* along the V direction.

Parameters:*v* - the V parametric value*c* - the iso-parametric curve*lod* - the level of detail to draw the curve with**Returns:**

0 if an error occurred, 1 otherwise

Warning:the parametric value *v* must be in a valid range**Author(s):**

Philippe Lavoie

Date:

7 October, 1997

3.4.23 int HNurbsSurface::read(const char* filename)

Reads a HNURBS surface from a file.

Parameters:*filename* - the filename to read from**Returns:**

0 if an error occurs, 1 otherwise

Author(s):

Philippe Lavoie

Date:

7 October 1997

Reimplemented from **NurbsSurface**.

3.4.24 **int HNurbsSurface::write(const char* filename)** **const**

write a HNURBS surface to a file.

Parameters:

filename - the filename to write to

Returns:

1 on success, 0 on failure

Author(s):

Philippe Lavoie

Date:

7 October 1997

Reimplemented from **NurbsSurface**.

3.4.25 **int HNurbsSurface::read(istream &fin)** [virtual]

Read a HNURBS surface from a file stream.

Parameters:

fin - the input file stream

Returns:

0 if an error occurs, 1 otherwise

Author(s):

Philippe Lavoie

Date:

7 October 1997

Reimplemented from **NurbsSurface**.

Reimplemented in **HNurbsSurfaceSP**.

3.4.26 int HNurbsSurface::write(ofstream &fout) const

Write a HNURBS surface to a file stream.

Parameters:

fout - the output filestream to write to.

Returns:

1 on success, 0 on failure

Author(s):

Philippe Lavoie

Date:

7 October 1997

Reimplemented from **NurbsSurface**.

3.4.27 int HNurbsSurface::maxLevel() const

the maximum level of detail.

Finds the maximum level of detail available from this HNURBS surface

Returns:

the maximum level of detail.

Author(s):

Philippe Lavoie

Date:

7 October 1997

3.4.28 void HNurbsSurface::refineKnots(const Vector<T>& nU, const Vector<T>& nV)

Refine both knot vectors.

Parameters:

nU - the U knot vector to refine from

nV - the V knot vector to refine from

Author(s):

Philippe Lavoie

Date:

24 January, 1997

Reimplemented from **NurbsSurface**.

Reimplemented in **HNurbsSurfaceSP**.

3.4.29 void HNurbsSurface::refineKnotU(const Vector<T>& X)

Refines the U knot vector.

Parameters:

X - the knot vector to refine from

Author(s):

Philippe Lavoie

Date:

24 January, 1997

Reimplemented from **NurbsSurface**.

Reimplemented in **HNurbsSurfaceSP**.

3.4.30 void HNurbsSurface::refineKnotV(const Vector<T>& X)

Refines the V knot vector.

Parameters:

X - the knot vector to refine from

Author(s):

Philippe Lavoie

Date:

24 January, 1997

Reimplemented from **NurbsSurface**.

Reimplemented in **HNurbsSurfaceSP**.

3.4.31 void HNurbsSurface::setFixedOffsetVector(const Point_nD<T,N> &I, const Point_nD<T,N> &J, const Point_nD<T,N>& K)

The offset vector are fixed.

Fixes the offset vector direction to a unique value. The offset vector's direction won't depend on its base layer.

Parameters:

I - the i vector

J - the j vector

K - the k vector

Author(s):

Philippe Lavoie

Date:

11 June 1998

3.4.32 void HNurbsSurface::setVariableOffsetVector()

The offset vector are variable.

Fixes the offset vector direction to a variable value. The value depends on its base layer.

Author(s):

Philippe Lavoie

Date:

11 June 1998

The documentation for this class was generated from the following files:

- hnurbsS.hh
- hnurbsS.cc

3.5 HNurbsSurfaceGL Class Reference

a HNURBS surface class for OpenGL.

```
#include <nurbs/nurbsGL.hh>
```

Inherits **NurbsGL**.

Public Members

- **HNurbsSurfaceGL** ()
Default constructor.
- **HNurbsSurfaceGL** (const NurbsSurface& nS)
Constructor from a surface.
- **HNurbsSurfaceGL** (const HNurbsSurfaceGL& bS)
Copy constructor with patch information.
- **HNurbsSurfaceGL** (const HNurbsSurfaceGL* bS)
Copy constructor.
- virtual **~HNurbsSurfaceGL** ()
- void **setLevelOfDetail** (int l)
- int **levelOfDetail** () const
- void **increaseLevelOfDetail** ()
- void **decreaseLevelOfDetail** ()
Activates the patch at a higher level.
- void **highestLevelOfDetail** ()
- int **maxLevelOfDetail** ()
- void **gluNurbs** () const
creates a HNURBS surface for OpenGL.
- void **point** (float &u, float &v, int pSize, const Color& colorP, int cp_flag=0) const
draws a point at the location C(u).
- void **resetBoundingBox** ()
resets the minP and maxP values of bbox.
- void **resetCPoints** ()
Reset the control point information.

- void **resetPolygon** ()
Reset the control point information stored in cpoints.
- void **resetKnots** ()
- int **read** (const char*f)
- int **write** (const char* f) const
- int **read** (ifstream &fin)
Reads the information from a stream.
- int **write** (ofstream &fout) const
Writes a NurbsCurveGL to an output stream.
- int **writeRIB** (ofstream &fout) const
- int **writePOVRAY** (ofstream &fout) const
- void **selectBasePatch** ()
- void **selectNextPatch** ()
Activates the next patch at the same level.
- void **selectPrevPatch** ()
Activates the previous patch at the same level.
- void **selectHigherLevel** ()
Activates the patch at a higher level.
- void **selectLowerLevel** ()
Activates the patch at a higher level.
- void **selectHighestLevel** ()
Activates the patch at a higher level.
- int **editLevel** ()
- void **updateUpToLOD** ()
- HNurbsSurfaceSPf* **addLevel** ()
Adds a level to the Hierarchical surface.
- void **applyTransform** ()
apply the local transformation to the surface.
- void **modifyPoint** (float u, float v, float dx, float dy, float dz)
Modifies a point on the surface.
- ObjectGL* **copy** ()
- void **setSym** (int set, int uDir, float x, float y, float z, float w)
Sets the symmetry for the control points.

- void **axis** (int i, int j, Point3Df& xaxis, Point3Df& yaxis, Point3Df& zaxis)
const

Protected Members

- int **lod**
 - HNurbsSurfaceGL* **activePatch**
-

Detailed Description

a HNURBS surface class for OpenGL.

Author(s):

Philippe Lavoie

Date:

23 September 1997

Member Function Documentation

3.5.1 HNurbsSurfaceGL::HNurbsSurfaceGL()

Default constructor.

Author(s):

Philippe Lavoie

Date:

7 November 1997

3.5.2 HNurbsSurfaceGL::HNurbsSurfaceGL(const NurbsSurface& nS)

Constructor from a surface.

Parameters:

nS - a Nurbs Surface

Author(s):

Philippe Lavoie

Date:

7 November 1997

3.5.3 HNurbsSurfaceGL::HNurbsSurfaceGL(const HNurbsSurfaceGL& bS)

Copy constructor with patch information.

Parameters:

bS - the object to copy
us - the start of the U parametric patch
ue - the end of the U parametric patch
vs - the start of the V parametric patch
ve - the end of the V parametric patch

Author(s):

Philippe Lavoie

Date:

7 November 1997

3.5.4 HNurbsSurfaceGL::HNurbsSurfaceGL(const HNurbsSurfaceGL* bS)

Copy constructor.

Parameters:

bS - a pointer to the object to copy

Author(s):

Philippe Lavoie

Date:

7 November 1997

3.5.5 void HNurbsSurfaceGL::decreaseLevelOfDetail()

Activates the patch at a higher level.

Author(s):

Philippe Lavoie

Date:

3 November 1997

3.5.6 void HNurbsSurfaceGL::gluNurbs() const

creates a HNURBS surface for OpenGL.

This draws a HNURBS. Presently only isocurves are drawn to represent the surface.

Author(s):

Philippe Lavoie

Date:

23 September 1997

Reimplemented from **NurbsGL**.

3.5.7 void HNurbsSurfaceGL::point(float &u, float &v, int pSize, const Color& colorP, int cp_flag) const

draws a point at the location $C(u)$.

This function calls between a glBegin/glEnd the proper functions to represent the point which is at $S(u,v)$ on the hierarchical surface.

Parameters:

u - the U parametric value

v - the V parametric value

$pSize$ - the size of the control points

$colorP$ - the color of the control points

Author(s):

Philippe Lavoie

Date:

23 September 1997

Reimplemented from **NurbsGL**.

3.5.8 void HNurbsSurfaceGL::resetBoundingBox()

resets the minP and maxP values of bbox.

Resets the minP and maxP values for the bounding box.

Warning:

Calling this function without a proper surface initialized might result in strange results.

Author(s):

Philippe Lavoie

Date:

23 September 1997

Reimplemented from **NurbsGL**.

3.5.9 void HNurbsSurfaceGL::resetCPoints()

Reset the control point information.

Reset the control point information stored in cpoints

Author(s):

Philippe Lavoie

Date:

23 September 1997

Reimplemented from **NurbsGL**.

3.5.10 void HNurbsSurfaceGL::resetPolygon()

Reset the control point information stored in cpoints.

Author(s):

Philippe Lavoie

Date:

29 January 1998

Reimplemented from **NurbsGL**.

3.5.11 int HNurbsSurfaceGL::read(istream &fin)

Reads the information from a stream.

Parameters:

fin - the input stream

Returns:

1 on success, 0 on failure

Author(s):

Philippe Lavoie

Date:

19 June 1998

Reimplemented from **ObjectGL**.

3.5.12 int HNurbsSurfaceGL::write(ofstream &fout) const

Writes a **NurbsCurveGL** to an output stream.

Parameters:

fout - the output stream

Returns:

0 if an error occurs, 1 otherwise

Author(s):

Philippe Lavoie

Date:

19 June 1998

Reimplemented from **ObjectGL**.

3.5.13 void HNurbsSurfaceGL::selectNextPatch()

Activates the next patch at the same level.

Author(s):

Philippe Lavoie

Date:

3 November 1997

3.5.14 void HNurbsSurfaceGL::selectPrevPatch()

Activates the previous patch at the same level.

Author(s):

Philippe Lavoie

Date:

3 November 1997

3.5.15 void HNurbsSurfaceGL::selectHigherLevel()

Activates the patch at a higher level.

Author(s):

Philippe Lavoie

Date:

3 November 1997

3.5.16 void HNurbsSurfaceGL::selectLowerLevel()

Activates the patch at a higher level.

Author(s):

Philippe Lavoie

Date:

3 November 1997

3.5.17 void HNurbsSurfaceGL::selectHighestLevel()

Activates the patch at a higher level.

Author(s):

Philippe Lavoie

Date:

28 January 1998

3.5.18 HNurbsSurfaceSPf* HNurbsSurfaceGL::addLevel()

Adds a level to the Hierarchical surface.

Author(s):

Philippe Lavoie

Date:

28 January 1998

3.5.19 void HNurbsSurfaceGL::applyTransform()

apply the local transformation to the surface.

Apply the local transformation to the surface. This is necessary if you want to get the proper position for the control points before doing anymore processing on them.

Author(s):

Philippe Lavoie

Date:

7 November 1997

Reimplemented from **ObjectGL**.

3.5.20 void HNurbsSurfaceGL::modifyPoint(float u, float v, float dx, float dy, float dz)

Modifies a point on the surface.

Parameters:

u - the u parametric value

v - the v parametric value

dx - the delta value in the *xx*-axis direction

dy - the delta value in the *yy*-axis direction

dz - the delta value in the *zz*-axis direction

Author(s):

Philippe Lavoie

Date:

7 November 1997

Reimplemented from **NurbsGL**.

3.5.21 void HNurbsSurfaceGL::setSym(int set, int uDir, float x, float y, float z, float w)

Sets the symmetry for the control points.

Parameters:

true - 1 if it should be in symmetrical mode

Author(s):

Philippe Lavoie

Date:

29 January 1998

Reimplemented from **NurbsGL**.

The documentation for this class was generated from the following files:

- nurbsGL.hh
- nurbsGL.cc

3.6 HNurbsSurfaceSP Class Reference

A NURBS surface with surface point.

Inherits **HNurbsSurface**.

Public Members

- **HNurbsSurfaceSP** ()
- **HNurbsSurfaceSP** (const NurbsSurface<T, N>& S)
- **HNurbsSurfaceSP** (const HNurbsSurface<T, N>& S)
- **HNurbsSurfaceSP** (const HNurbsSurfaceSP<T, N>& S)
- **HNurbsSurfaceSP** (HNurbsSurface<T, N>* base)
- **HNurbsSurfaceSP** (HNurbsSurface<T, N>* base, const Vector<T>& xU, const Vector<T>& xV)
- **HNurbsSurfaceSP** (HNurbsSurface<T, N>* base, const HNurbsSurface<T, N> &surf)
- virtual void **resizeKeep** (int Pu, int Pv, int DegU, int DegV)
- virtual void **refineKnots** (const Vector<T>& nU, const Vector<T>& nV)
- virtual void **refineKnotU** (const Vector<T>& X)
- virtual void **refineKnotV** (const Vector<T>& X)
- virtual void **mergeKnots** (const Vector<T>& nU, const Vector<T>& nV)
- virtual void **mergeKnotU** (const Vector<T>& X)
- virtual void **mergeKnotV** (const Vector<T>& X)
- virtual void **updateSurface** (int i0=-1, int j0=-1)
Updates the NURBS surface.
- virtual void **updateLevels** (int updateLevel=-1)
Update the surface for all the levels.
- virtual HNurbsSurfaceSP<T,N>* **addLevel** (int nsplit, int s=1)
Adds a level to this HNURBS surface.
- virtual HNurbsSurfaceSP<T,N>* **addLevel** ()
Adds a level to this HNURBS surface.
- virtual void **copy** (const HNurbsSurface<T, N>& nS)
Copies a HNurbs Surface and all it children.
- virtual int **read** (ifstream &fin)
Read a HNURBS surface from an input file stream.

- void **modSurfCPby** (int i, int j, const HPoint_nD<T, N>& a)
Modifies the surface point by a certain value.
- void **modOnlySurfCPby** (int i, int j, const HPoint_nD<T, N>& a)
Moves the surface point only.
- T **maxAtUV** (int i, int j) const
- T **maxAtU** (int i) const
- T **maxAtV** (int i) const
- HPoint_nD<T,N> **surfP** (int i, int j) const
- HPoint_nD<T,N> **surfP** (int i, int j, int lod) const
- void **updateMaxUV** ()
- void **updateMaxU** ()
Updates the basis value in the U direction.
- void **updateMaxV** ()
Updates the basis value in the V direction.
- int **okMax** ()

Protected Members

- Vector<T> **maxU**
- Vector<T> **maxV**
- Vector<T> **maxAtU_**
- Vector<T> **maxAtV_**

Detailed Description

A NURBS surface with surface point.

A Nurbs surface with surface point manipulators. This allows someone to modify the point on a surface for which a control point has maximal influence over it. This might provide a more intuitive method to modify a surface.

Author(s):

Philippe Lavoie

Date:

14 May, 1998

Member Function Documentation

3.6.1 void HNurbsSurfaceSP::updateSurface(int i0=-1, int j0=-1) [virtual]

Updates the NURBS surface.

Updates the NURBS surface according to the offset values and its base level. You can update only one control point from the surface if you specify a value for i and j or you can update all the points if i0 or j0 is below 0.

Parameters:

i0 - the row of the control point to update

j0 - the column of the control point to update

Author(s):

Philippe Lavoie

Date:

7 October 1997

Reimplemented from **HNurbsSurface**.

3.6.2 void HNurbsSurfaceSP::updateLevels(int upLevel) [virtual]

Update the surface for all the levels.

Parameters:

upLevel - updates the levels up to this level of detail

Author(s):

Philippe Lavoie

Date:

7 October 1997

Reimplemented from **HNurbsSurface**.

3.6.3 HNurbsSurfaceSP<T,N>* HNurbsSurfaceSP::addLevel(int n, int s) [virtual]

Adds a level to this HNURBS surface.

Parameters:

n - the number of new knots between each knots.
 s - the multiplicity of each of these knots

Returns:

a pointer to the new level or 0 if there was an error.

Warning:

returns 0, if there is already a nextlevel.

Author(s):

Philippe Lavoie

Date:

14 May 1998

3.6.4 HNurbsSurfaceSP<T,N>* HNurbsSurfaceSP::addLevel() [virtual]

Adds a level to this HNURBS surface.

Parameters:

n - the number of new knots between each knots.

Returns:

a pointer to the new level or 0 if there was an error.

Warning:

returns 0, if there is already a nextlevel.

Author(s):

Philippe Lavoie

Date:

14 May 1998

Reimplemented from **HNurbsSurface**.

3.6.5 void HNurbsSurfaceSP::copy(const HNurbsSurface<T,N>& nS) [virtual]

Copies a HNurbs Surface and all it children.

Parameters:

ns - the HNurbs surface to copy

Author(s):

Philippe Lavoie

Date:

14 May 1998

Reimplemented from **HNurbsSurface**.

3.6.6 **int HNurbsSurfaceSP::read(istream &fin)** [virtual]

Read a HNURBS surface from an input file stream.

Parameters:

fin - the input file stream

Returns:

0 if an error occurs, 1 otherwise

Author(s):

Philippe Lavoie

Date:

7 October 1997

Reimplemented from **NurbsSurface**.

3.6.7 **void HNurbsSurfaceSP::modSurfCPby(int i, int j, const HPoint_nD<T,N>& a)**

Modifies the surface point by a certain value.

Parameters:

i - the row of the surface point

j - the column of the surface point

a - modify the surface point by this value

Warning:

The degree in U and V of the surface must be of 3 or less.

Author(s):

Philippe Lavoie

Date:

14 May, 1998

3.6.8 void HNurbsSurfaceSP::modOnlySurfCPby(int i, int j, const HPoint_nD<T,N>& a)

Moves the surface point only.

Moves only the specified surface point. The other surface points normally affected by moving this point are {\em not} moved.

The point a is in the 4D homogenous space, but only the x,y,z value are used. The weight is not moved by this function.

Parameters:

i - the row of the surface point to move

j - the column of the surface point to move

a - move that surface point by that amount.

Warning:

The degree of the curve must be of 3 or less.

Author(s):

Philippe Lavoie

Date:

7 June, 1998

3.6.9 void HNurbsSurfaceSP::updateMaxU()

Updates the basis value in the U direction.

Updates the basis value at which a control point has maximal influence. It also finds where the control point has maximal influence.

Warning:

The degree in U of the surface must be of 3 or less.

Author(s):

Philippe Lavoie

Date:

14 May, 1998

3.6.10 void HNurbsSurfaceSP::updateMaxV()

Updates the basis value in the V direction.

Updates the basis value at which a control point has maximal influence. It also finds where the control point has maximal influence.

Warning:

The degree in V of the surface must be of 3 or less.

Author(s):

Philippe Lavoie

Date:

14 May, 1998

The documentation for this class was generated from the following files:

- hnurbsS_sp.hh
- hnurbsS_sp.cc

3.7 KnotGL Class Reference

A class to hold a knot.

```
#include <nurbs/nurbsGL.hh>
```

Inherits **ObjectGL**.

Public Members

- **KnotGL** (const Point3Df& p3d, int kU, int kV)
- void **glObject** () const
Displays a knot.
- void **modify** (const Point3Df& v)
- void **set** (const Point3Df& v)
- void **setSize** (int s)
- float **x** () const
- float **y** () const
- float **z** () const
- const Point3Df& **point** () const
- ObjectGL* **copy** ()
- int **knotU** ()
- int **knotV** ()

Protected Members

- Point3Df **p**
- int **ku**
- int **kv**
- int **psize**

Detailed Description

A class to hold a knot.

Author(s):

Philippe Lavoie

Date:

23 September 1997

Member Function Documentation

3.7.1 void KnotGL::glObject() const

Displays a knot.

Author(s):

Philippe Lavoie

Date:

30 September 1997

Reimplemented from **ObjectGL**.

The documentation for this class was generated from the following files:

- nurbsGL.hh
- nurbsGL.cc

3.8 Material Class Reference

a class for the material properties of an object.

```
#include <nurbs/nurbsGL.hh>
```

Public Members

- **Material ()**
default constructor.
- void **glMaterial ()**
- RGBAf **frontAmbient**
- RGBAf **backAmbient**
- RGBAf **frontDiffuse**
- RGBAf **backDiffuse**
- RGBAf **frontSpecular**
- RGBAf **backSpecular**
- RGBAf **frontEmission**
- RGBAf **backEmission**
- float **frontShininess**
- float **backShininess**
- RGBAf **pigment**
- double **pigment_transfer**
- double **pigment_transmit**
- char* **pigment_userdefined**
- double **bump**
- double **bump_scale**
- char* **normal_userdefined**
- Color **ambient**
- double **diffuse**
- double **brilliance**
- double **phong**
- double **specular**
- double **roughness**
- double **metallic**
- Color **reflection**
- double **refraction**
- double **ior**
- double **caustics**
- double **fade_distance**
- double **fade_power**
- double **irid_thick**
- Vector< Point3Df > **irid_turbulence**
- double **crand**
- char* **material**

Detailed Description

a class for the material properties of an object.

This holds the information about the material properties of the object. This is a long list, since it has properties which are usefull for more than one rendering method.

You should not use this class yet. It is not used by any other class.

Author(s):

Philippe Lavoie

Date:

28 September 1997

Member Function Documentation

3.8.1 Material::Material()

default constructor.

This sets the values of all the variables to their default values (according to their manual).

Author(s):

Philippe Lavoie

Date:

13 October 1997

The documentation for this class was generated from the following files:

- nurbsGL.hh
- nurbsGL.cc

3.9 MatrixRT Class Reference

a matrix for rotation and translation transformation.

```
#include <nurbs/matrixRT.hh>
```

Public Members

- **MatrixRT** (T ax, T ay, T az, T x, T y, T z)
constructor with the angles and translation parameters specified.
- **MatrixRT** ()
default constructor.
- **MatrixRT** (T* p)
default constructor.
- **MatrixRT<T>& rotate** (T ax, T ay, T az)
creates a rotation matrix of angle (ax,ay,az).
- **MatrixRT<T>& rotateXYZ** (T ax, T ay, T az)
creates a rotation matrix of angle (ax,ay,az).
- **MatrixRT<T>& translate** (T x, T y, T z)
Generates a translation matrix.
- **MatrixRT<T>& scale** (T x, T y, T z)
Generates a scaling matrix.
- **MatrixRT<T>& rotateDeg** (T ax, T ay, T az)
A rotation with the angles specified in degree.
- **MatrixRT<T>& rotateDegXYZ** (T ax, T ay, T az)
A rotation in the X,Y and Z order with the angles specified in degree.
- **MatrixRT<T>& operator=** (const Matrix<T>& M)
The assignment operator with a matrix.
- **MatrixRT<T>& operator=** (const MatrixRT<T>& M)

Protected Members

- **int created**

indicate if the data was initialized by this class or not.

Detailed Description

a matrix for rotation and translation transformation.

This is a matrix for the rotation, translation and scaling of a point in 3D or 4D.

Author(s):

Philippe Lavoie

Date:

25 July 1997

Member Function Documentation

3.9.1 MatrixRT::MatrixRT(T ax, T ay, T az, T x, T y, T z)

constructor with the angles and translation parameters specified.

Sets the matrixRT to be a rotation of ax,ay,az then a translation x,y,z around the origin.

Parameters:

ax - specifies the rotation around the x-axis

ay - specifies the rotation around the y-axis

az - specifies the rotation around the z-axis

x - specifies the translation along the x-axis

y - specifies the translation along the y-axis

z - specifies the translation along the z-axis

Author(s):

Philippe Lavoie

Date:

25 July, 1997

3.9.2 MatrixRT::MatrixRT

default constructor.

Default constructor

Author(s):

Philippe Lavoie

Date:

25 July, 1997

3.9.3 MatrixRT::MatrixRT(T* p)

default constructor.

Default constructor

Author(s):

Philippe Lavoie

Date:

25 July, 1997

3.9.4 MatrixRT<T>& MatrixRT::rotate(T ax, T ay, T az)

creates a rotation matrix of angle (ax,ay,az).

The rotation are clockwise around the main axes. The rotation is performed in that order: a rotation around the z-axis, then the y-axis and finally around the x-axis.

Parameters:

ax - specifies the rotation around the x-axis

ay - specifies the rotation around the y-axis

az - specifies the rotation around the z-axis

See also:

`rotateXYZ()`

Author(s):

Philippe Lavoie

Date:

25 July, 1997

3.9.5 `MatrixRT<T>& MatrixRT::rotateXYZ(T ax, T ay, T az)`

creates a rotation matrix of angle (ax,ay,az).

The rotation are clockwise around the main axes. The rotation is performed in that order: a rotation around the x-axis, then the y-axis and finally around the z-axis.

Parameters:

ax - specifies the rotation around the x-axis

ay - specifies the rotation around the y-axis

az - specifies the rotation around the z-axis

See also:

`rotate()`

Author(s):

Philippe Lavoie

Date:

25 July, 1997

3.9.6 `MatrixRT<T>& MatrixRT::translate(T x, T y, T z)`

Generates a translation matrix.

Parameters:

x - specifies the translation along the *\$x\$*-axis

y - specifies the translation along the *\$y\$*-axis

z - specifies the translation along the *\$z\$*-axis

Warning:

This resets the rotation part of the matrix

Author(s):

Philippe Lavoie

Date:

25 July, 1997

3.9.7 MatrixRT<T>& MatrixRT::scale(T x, T y, T z)

Generates a scaling matrix.

Parameters:

- x - specifies the scaling along the x -axis
- y - specifies the scaling along the y -axis
- z - specifies the scaling along the z -axis

Author(s):

Philippe Lavoie

Date:

25 July, 1997

3.9.8 MatrixRT<T>& MatrixRT::rotateDeg(T ax, T ay, T az)

A rotation with the angles specified in degree.

3.9.9 MatrixRT<T>& MatrixRT::rotateDegXYZ(T ax, T ay, T az)

A rotation in the X,Y and Z order with the angles specified in degree.

3.9.10 MatrixRT<T>& MatrixRT::operator=(const Matrix<T>& M)

The assignment operator with a matrix.

Parameters:

M - the matrix

Returns:

A reference to itself

Warning:

The matrix *must* be of size 4×4

Author(s):

Philippe Lavoie

Date:

25 July, 1997

Member Data Documentation

3.9.11 int MatrixRT::created [protected]

indicate if the data was initialized by this class or not.

The documentation for this class was generated from the following files:

- matrixRT.hh
- matrixRT.cc

3.10 NurbsCpolygonGL Class Reference

Holds the control polygon for a curve.

```
#include <nurbs/nurbsGL.hh>
```

Inherits **ObjectGL**.

Public Members

- **NurbsCpolygonGL** (NurbsCurve& c)
- void **glObject** () const
draws the polygon joining the control points.

Protected Members

- NurbsCurve& **curve**
-

Detailed Description

Holds the control polygon for a curve.

Author(s):

Philippe Lavoie

Date:

23 September 1997

Member Function Documentation

3.10.1 void NurbsCpolygonGL::glObject() const

draws the polygon joining the control points.

This function calls between a glBegin/glEnd the proper functions to represent the polygon joining all the control points of the NURBS curve.

Author(s):

Philippe Lavoie

Date:

23 September 1997

Reimplemented from **ObjectGL**.

The documentation for this class was generated from the following files:

- nurbsGL.hh
- nurbsGL.cc

3.11 NurbsCurve Class Reference

A NURBS curve class.

`#include <nurbs/nurbs.hh>`

Inherits **ParaCurve**.

Inherited by **NurbsCurveSP**.

Public Members

- **NurbsCurve** ()
default constructor.
- **NurbsCurve** (const NurbsCurve<T, N>& nurb)
A copy constructor.
- **NurbsCurve** (const Vector< HPoint_nD<T, N> >& P1, const Vector<T> &U1, int degree=3)
Constructor with control points in 4D.
- **NurbsCurve** (const Vector< Point_nD<T, N> >& P1, const Vector<T> &W, const Vector<T> &U1, int degree=3)
Constructor with control points in 3D.
- virtual **~NurbsCurve** ()
- void **resize** (int n, int Deg)
Resizes a NURBS curve.
- virtual void **reset** (const Vector< HPoint_nD<T, N> >& P1, const Vector<T> &U1, int degree)
Resets a NURBS curve to new values.
- virtual NurbsCurve& **operator=** (const NurbsCurve<T, N>&)
The assignment operator for a NURBS curve.
- virtual HPoint_nD<T,N> **operator()** (T u) const
Evaluates the curve in 4D at parameter u.
- HPoint_nD<T,N> **hpointAt** (T u) const
< calls operator().
- HPoint_nD<T,N> **hpointAt** (T u, int span) const
Evaluates the curve in homogenous space at parameter u.

- void **deriveAtH** (T u, int, Vector< HPoint_nD<T, N> >&) const
< returns the curvePoint in 3D.
- void **deriveAt** (T u, int, Vector< Point_nD<T, N> >&) const
Computes the derivative at the parameter u.
- void **deriveAtH** (T u, int, int, Vector< HPoint_nD<T, N> >&) const
Computes the derivative of degree d of the curve at parameter u.
- void **deriveAt** (T u, int, int, Vector< Point_nD<T, N> >&) const
Computes the derivative of the curve at the parameter u.
- Point_nD<T,N> **derive3D** (T u, int d) const
Computes the derivative of degree d of the curve at parameter u.
- HPoint_nD<T,N> **derive** (T u, int d) const
Computes the derivative of degree of the curve at parameter u.
- Point_nD<T,N> **normal** (T u, const Point_nD<T, N>& v) const
Computes the normal of the curve at u from a vector.
- HPoint_nD<T,N> **firstD** (T u) const
Computes the first derivative Computes the first derivative in the 4D homogenous space.
- HPoint_nD<T,N> **firstD** (T u, int span) const
Computes the first derivative.
- Point_nD<T,N> **firstDn** (T u) const
Computes the first derivative.
- Point_nD<T,N> **firstDn** (T u, int span) const
Computes the first derivative.
- T **basisFun** (T u, int i, int p=-1) const
Computes the basis function of the curve.
- void **basisFuns** (T u, int span, Vector<T>& N) const
computes the non-zero basis functions of the curve.
- void **dersBasisFuns** (int n, T u, int span, Matrix<T>& N) const
Compute the derivatives functions at u of the NURBS curve.

- **T minKnot** () const
< the minimal value for the knot vector.
- **T maxKnot** () const
< the maximal value for the knot vector.
- **int findSpan** (T u) const
Determines the knot span index.
- **void findMultSpan** (T u, int& r, int& s) const
Finds the multiplicity of a knot at a parametric value.
- **int findMult** (int r) const
Finds the multiplicity of a knot.
- **int findKnot** (T u) const
Finds the knot k for which u is in the range $[u_k, u_{k+1})$.
- **T getRemovalBnd** (int r, int s) const
Get the knot removal error bound for an internal knot.
- **virtual void removeKnot** (int r, int s, int num)
Removes an internal knot from a curve. This is A5.8 on p185 from the NURB book modified to not check for tolerance before removing the knot.
- **virtual void removeKnotsBound** (const Vector<T>& ub, Vector<T>& ek, T E)
Remove knots from a curve without exceeding an error bound.
- **virtual void knotInsertion** (T u, int r, NurbsCurve<T, N>& nc)
- **virtual void refineKnotVector** (const Vector<T>& X)
Refine the curve knot vector.
- **virtual void mergeKnotVector** (const Vector<T> &Um)
Merges the knot vector of a curve with another knot vector.
- **int leastSquares** (const Vector< Point_nD<T, N> >& Q, int degC, int n)
A least squares curve approximation.
- **int leastSquares** (const Vector< Point_nD<T, N> >& Q, int degC, int n, const Vector<T>& ub)
A least squares curve approximation.

- **int leastSquaresH** (const Vector< HPoint_nD<T, N> >& Q, int degC, int n, const Vector<T>& ub)
A least squares curve approximation.
- **int leastSquares** (const Vector< Point_nD<T, N> >& Q, int degC, int n, const Vector<T>& ub, const Vector<T>& knot)
A least squares curve approximation.
- **int leastSquaresH** (const Vector< HPoint_nD<T, N> >& Q, int degC, int n, const Vector<T>& ub, const Vector<T>& knot)
- **void globalApproxErrBnd** (Vector< Point_nD<T, N> >& Q, int degree, T E)
Approximation of a curve bounded to a certain error.
- **void globalApproxErrBnd** (Vector< Point_nD<T, N> >& Q, Vector<T>& ub, int degree, T E)
Approximation of a curve bounded to a certain error.
- **void globalApproxErrBnd2** (Vector< Point_nD<T, N> >& Q, int degC, T E)
- **void globalApproxErrBnd3** (Vector< Point_nD<T, N> >& Q, int degC, T E)
- **void globalApproxErrBnd3** (Vector< Point_nD<T, N> >& Q, const Vector<T> &ub, int degC, T E)
- **void globalInterp** (const Vector< Point_nD<T, N> >& Q, int d)
global curve interpolation with points in 3D.
- **void globalInterp** (const Vector< Point_nD<T, N> >& Q, const Vector<T>& ub, int d)
global curve interpolation with points in 3D.
- **void globalInterpH** (const Vector< HPoint_nD<T, N> >& Q, int d)
global curve interpolation with points in 4D.
- **void globalInterpH** (const Vector< HPoint_nD<T, N> >& Q, const Vector<T>& U, int d)
global curve interpolation with 4D points and a knot vector defined.
- **void globalInterpH** (const Vector< HPoint_nD<T, N> >& Q, const Vector<T>& ub, const Vector<T>& U, int d)
global curve interpolation with 4D points, a knot vector defined and the parametric value vector defined.
- **void globalInterpD** (const Vector< Point_nD<T, N> >& Q, const Vector< Point_nD<T, N> >& D, int d, int unitD, T a=1.0)

global curve interpolation with the 1st derivatives of the points specified.

- void **projectTo** (const Point_nD<T, N>& p, T guess, T& u, Point_nD<T, N>& r, T e1=0.001, T e2=0.001, int maxTry=100) const
- T **length** (T eps=0.001, int n=100) const

Computes the length of the curve.

- T **lengthIn** (T us, T ue, T eps=0.001, int n=100) const

Computes the length of the curve inside [u_s, u_e].

- T **lengthF** (T) const

The function used by length length needs to integrate a function over an interval to determine the length of the NURBS curve. Well, this is the function.

- T **lengthF** (T, int) const
- void **makeCircle** (const Point_nD<T, N>& O, const Point_nD<T, N>& X, const Point_nD<T, N>& Y, T r, double as, double ae)

generates a circular curve.

- void **makeCircle** (const Point_nD<T, N>& O, T r, double as, double ae)

generates a circular curve.

- void **makeLine** (const Point_nD<T, N>& P0, const Point_nD<T, N>& P1, int d)

Generate a straight line.

- virtual void **degreeElevate** (int t)
- void **decompose** (NurbsCurveArray<T, N>& c) const

Decompose the curve into Bzier segments.

- int **splitAt** (T u, NurbsCurve<T, N>& cl, NurbsCurve<T, N>& cu) const

Splits the curve into two curves.

- int **mergeOf** (const NurbsCurve<T, N>& cl, const NurbsCurve<T, N>& cu)

The curve is the result of mergin two curves.

- void **transform** (const MatrixRT<T>& A)

Performs geometrical modifications.

- void **modCP** (int i, const HPoint_nD<T, N>& a)

- void **modCPby** (int i, const HPoint_nD<T, N>& a)
- virtual void **modKnot** (const Vector<T>& knot)
- int **movePoint** (T u, const Point_nD<T, N>& delta)
Moves a point on the NURBS curve.
- int **movePoint** (T u, const Vector< Point_nD<T, N> >& delta)
Moves a point in the NURBS curve.
- int **movePoint** (const Vector<T>& ur, const Vector< Point_nD<T, N> >& D)
Moves a point with some constraint.
- int **movePoint** (const Vector<T>& ur, const Vector< Point_nD<T, N> >& D, const Vector_INT& Dr, const Vector_INT& Dk)
Moves a point with some constraint.
- int **movePoint** (const Vector<T>& ur, const Vector< Point_nD<T, N> >& D, const Vector_INT& Dr, const Vector_INT& Dk, const Vector_INT& fixCP)
Moves a point with some constraint.
- int **read** (const char*)
Reads a NurbsCurve<T,N> from a file.
- int **write** (const char*) const
Writes a NurbsCurve<T,N> to a file.
- virtual int **read** (ifstream &fin)
reads a NurbsCurve<T,N> from a file.
- int **write** (ofstream &fout) const
Writes a NurbsCurve<T,N> to an output stream.
- int **writePS** (const char*, int cp=0, T magFact=T(-1), T dash=T(5)) const
Writes the curve in the postscript format to a file.
- int **writePSp** (const char*, const Vector< Point_nD<T, N> >&, const Vector< Point_nD<T, N> >&, int cp=0, T magFact=0.0, T dash=5.0) const
Writes a post-script file representing the curve.
- int **writeVRML** (const char* filename, T radius, int K, const Color& color, int Nu, int Nv, T u_s, T u_e) const

Writes the curve to a VRML file.

- int **writeVRML** (const char* filename, T radius=1, int K=5, const Color& color=whiteColor, int Nu=20, int Nv=20) const
- void **drawImg** (Image_UBYTE& Img, unsigned char color=255, T step=0.01)
- void **drawImg** (Image_Color& Img, const Color& color, T step=0.01)

Draws a NURBS curve on an image.

- void **drawAaImg** (Image_Color& Img, const Color& color, int precision=3, int alpha=1)

Draws an anti-aliased NURBS curve on an image.

- void **drawAaImg** (Image_Color& Img, const Color& color, const NurbsCurve<T, 3>& profile, int precision=3, int alpha=1)

draws an anti-aliased NURBS curve on an image.

- NurbsSurface<T,3> **drawAaImg** (Image_Color& Img, const Color& color, const NurbsCurve<T, 3>& profile, const NurbsCurve<T, 3> &scaling, int precision=3, int alpha=1)

Draws an anti-aliased NURBS curve on an image.

- BasicList<Point_nD<T,N> > **tessellate** (T tolerance, BasicList<T>*uk) const

Generates a list of points from the curve.

- const int& **degree**

a reference to the degree of the curve.

- const Vector< HPoint_nD<T,N> >& **CtrlP**

a reference to the vector of control points.

- const Vector<T>& **Knot**

a reference to the vector of knots.

Protected Members

- Vector< HPoint_nD<T,N> > **P**
- Vector<T> **U**
- int **deg**

Related Functions

(Note that these are not member functions.)

- **T chordLengthParam** (const Vector< Point_nD<T,N> >& Q, Vector<T> &ub)
chord length parameterization.
-

Detailed Description

A NURBS curve class.

This class is used to represent and manipulate NURBS curve. The curves are composed of points in 4D. They can have any degree and have any number of control points.

Author(s):

Philippe Lavoie

Date:

4 Oct. 1996

Member Function Documentation

3.11.1 NurbsCurve::NurbsCurve()

default constructor.

Author(s):

Philippe Lavoie

Date:

24 January 1997

3.11.2 NurbsCurve::NurbsCurve(const NurbsCurve<T,N>& nurb)

A copy constructor.

Parameters:

nurb - the NURBS curve to copy

Author(s):

Philippe Lavoie

Date:

24 January 1997

3.11.3 NurbsCurve::NurbsCurve(const Vector<HPoint_nD<T,N> >& P1, const Vector<T> &U1, int Degree)

Constructor with control points in 4D.

Parameters:

P1 - the control points

U1 - the knot vector

Degree - the degree of the curve

Warning:

The size of P1,U1 and Degree must agree: $P.n + degree + 1 = U.n$

Author(s):

Philippe Lavoie

Date:

24 January 1997

3.11.4 NurbsCurve::NurbsCurve(const Vector<Point_nD<T,N> >& P1, const Vector<T> &W, const Vector<T> &U1, int Degree)

Constructor with control points in 3D.

Parameters:

P1 - -> the control point vector

W - -> the weight for each control points

U1 - -> the knot vector

Degree - -> the degree of the curve

Warning:

The size of P1,U1 and Degree must agree: $P.n + \text{degree} + 1 = U.n$

Author(s):

Philippe Lavoie

Date:

24 January 1997

3.11.5 void NurbsCurve::resize(int n, int Deg)

Resizes a NURBS curve.

Resizes a NURBS curve. The old values are lost and new ones have to be created.

Parameters:

n - the new number of control points for the curve

Deg - the new degree for the curve

Author(s):

Philippe Lavoie

Date:

24 January 1997

3.11.6 void NurbsCurve::reset(const Vector<HPoint_nD<T,N> >& P1, const Vector<T> &U1, int Degree [virtual])

Resets a NURBS curve to new values.

Parameters:

P1 - the new values for the control points

U1 - the new values for the knot vector

Degree - the new degree of the curve

Warning:

The size of P1,U1 and Degree must agree: $P.n + \text{degree} + 1 = U.n$

Author(s):

Philippe Lavoie

Date:

24 January 1997

Reimplemented in **NurbsCurveSP**.

```

3.11.7 virtual NurbsCurve<T,N>&
        NurbsCurve::operator=(const
        NurbsCurve<T,N>& curve) [virtual]

```

The assignment operator for a NURBS curve.

Parameters:

curve - the NURBS curve to copy

Returns:

A reference to itself

Warning:

The curve being copied must be valid, otherwise strange results might occur.

Author(s):

Philippe Lavoie

Date:

24 January 1997

Reimplemented in `NurbsCurveSP` and `NurbsCurveSP`.

```

3.11.8 HPoint_nD<T,N> NurbsCurve::operator()(T u)
        const [virtual]

```

Evaluates the curve in 4D at parameter u .

It evaluates the NURBS curve in 4D at the parametric point u . Using the following equation

$$C(u) = \sum_{i=0}^n N_{i,p} P_i \quad a \leq u \leq b \quad (3.1)$$

where P_i are the control points and $N_{i,p}$ are the p th degree B-spline basis functions.

For more details on the algorithm, see A4.1 on page 124 of the Nurbs Book.

Parameters:

u - the parametric value at which the curve is evaluated

Returns:

the 4D point at $C(u)$

Warning:

the parametric value must be in a valid range

Author(s):

Philippe Lavoie

Date:

24 January, 1997

Reimplemented from **ParaCurve**.

3.11.9 HPoint_nD<T,N> NurbsCurve::hpointAt(T u) const

< calls operator().

Reimplemented from **ParaCurve**.

3.11.10 HPoint_nD<T,N> NurbsCurve::hpointAt(T u, int span) const

Evaluates the curve in homogenous space at parameter u .

It evaluates the NURBS curve in 4D at the parametric point u . Using the following equation

$$C(u) = \sum_{i=0}^n N_{i,p} P_i \quad a \leq u \leq b \quad (3.2)$$

where P_i are the control points and $N_{i,p}$ are the p th degree B-spline basis functions.

For more details on the algorithm, see A4.1 on page 124 of the Nurbs Book.

Parameters:

u - the parametric value at which the curve is evaluated
 $span$ - the span of u

Returns:

the 4D point at $C(u)$

Warning:

the parametric value must be in a valid range

Author(s):

Philippe Lavoie

Date:

24 January, 1997

Reimplemented from **ParaCurve**.**3.11.11 void NurbsCurve::deriveAtH(T u,int d, Vector<HPoint_nD<T,N> >& ders) const**

< returns the curvePoint in 3D.

For more information on the algorithm used, see A3.2 on p 93 of the NurbsBook.

Parameters:

u - the parametric value to evaluate at
d - the degree of the derivative
ders - a vector containing the derivatives of the curve at *u*.

Warning:*u* and *d* must be in a valid range.**Author(s):**

Philippe Lavoie

Date:

24 January, 1997

Reimplemented from **ParaCurve**.**3.11.12 void NurbsCurve::deriveAt(T u, int d, Vector<Point_nD<T,N> >& ders) const**Computes the derivative at the parameter *u*.**Parameters:**

u - the parameter at which the derivative is computed
d - the degree of derivation
ders - the vector containing the derivatives of the point at *u*.

\wanring *u* and *d* must be in a valid range.**Author(s):**

Philippe Lavoie

Date:

24 January, 1997

Reimplemented from **ParaCurve**.

3.11.13 void NurbsCurve::deriveAtH(T u, int d, int span, Vector< HPoint_nD<T,N> >& ders) const

Computes the derivative of degree d of the curve at parameter u .

For more information on the algorithm used, see A3.2 on p 93 of the NurbsBook.

Parameters:

u - the parametric value to evaluate at

d - the degree of the derivative

$span$ - the span of u

$ders$ - a vector containing the derivatives of the curve at u .

Warning:

u and d must be in a valid range.

Author(s):

Philippe Lavoie

Date:

9 October, 1998

3.11.14 void NurbsCurve::deriveAt(T u, int d, int span, Vector< Point_nD<T,N> >& ders) const

Computes the derivative of the curve at the parameter u .

Parameters:

u - the parameter at which the derivative is computed

d - the degree of derivation

$span$ - the span of u .

$ders$ - the vector containing the derivatives of the point at u .

Warning:

u and d must be in a valid range.

Author(s):

Philippe Lavoie

Date:

9 October 1998

3.11.15 `Point_nD<T,N> NurbsCurve::derive3D(T u, int d) const`

Computes the derivative of degree d of the curve at parameter u .

For more information on the algorithm used, see A3.2 on p 93 of the NurbsBook.

Parameters:

u - the parametric value to evaluate at
 d - the degree of the derivative

Returns:

The derivative d in norma space at the parameter u

Warning:

u and d must be in a valid range.

Author(s):

Philippe Lavoie

Date:

24 January, 1997

3.11.16 `HPoint_nD<T,N> NurbsCurve::derive(T u, int d) const`

Computes the derivative of degree d of the curve at parameter u .

For more information on the algorithm used, see A3.2 on p 93 of the NurbsBook.

Parameters:

u - the parametric value to evaluate at
 d - the degree of the derivative

Returns:

The derivative d in 4D at the parameter u

Warning:

u and d must be in a valid range.

Author(s):

Philippe Lavoie

Date:

24 January, 1997

3.11.17 `Point_nD<T,N> NurbsCurve::normal(T u, const Point_nD<T,N>& v) const`

Computes the normal of the curve at u from a vector.

Computes the normal of the curve at u from a vector. If the curve lies only in the xy-plane, then calling the function with the vector $v = (0,0,1)$ (the z\$axis) will yield a proper normal for this curve.

Parameters:

u - the parameter at which the normal is computed
 v - the vector to compute the normal with

Returns:

the normal vector in 3D.

Warning:

u must be in a valid range.

Author(s):

Philippe Lavoie

Date:

2 September, 1997

3.11.18 `HPoint_nD<T,N> NurbsCurve::firstD(T u) const`

Computes the first derivative Computes the first derivative in the 4D homogenous space.

Parameters:

u - \rightarrow compute the derivative at this parameter

Returns:

The first derivative in homogenous space

Warning:

u must be in the valid range

Author(s):

Philippe Lavoie

Date:

13 October 1998

3.11.19 HPoint_nD<T,D> NurbsCurve::firstD(T u, int span) const

Computes the first derivative.

Computes the first derivative in the homogenous space with the span given.

Parameters:

u - compute the derivative at this parameter
span - the span of u

Returns:

The first derivative of the point in the homogeneous space

Warning:

u and span must be in a valid range

Author(s):

Philippe Lavoie

Date:

13 October 1998

3.11.20 Point_nD<T,N> NurbsCurve::firstDn(T u) const

Computes the first derivative.

Computes the first derivative in the normal space.

Parameters:

u - compute the derivative at this parameter
span - the span of u

Returns:

The first derivative in normal space

Warning:

u and span must be in a valid range

Author(s):

Philippe Lavoie

Date:

13 October 1998

3.11.21 `Point_nD<T,N> NurbsCurve::firstDn(T u, int span) const`

Computes the first derivative.

Computes the first derivative in the normal space (3D or 2D).

Parameters:

u - \rightarrow compute the derivative at this parameter

span - \rightarrow the span of *u*

Warning:

u and *span* must be in a valid range

Author(s):

Philippe Lavoie

Date:

13 October 1998

3.11.22 `T NurbsCurve::basisFun(T u, int i, int p) const`

Computes the basis function of the curve.

Computes the *i* basis function of degree *p* of the curve at parameter *u*.

The basis function is noted as $N_{ip}(u)$. The B-spline basis function of *p*-degree is defined as

$$N_{i,0}(u) = \begin{cases} 1 & \text{if } u_i \leq u < u_{i+1} \\ 0 & \text{otherwise} \end{cases}$$

$$N_{i,p}(u) = \frac{u - u_i}{u_{i+p} - u_i} N_{i,p-1}(u) + \frac{u_{i+p+1} - u}{u_{i+p+1} - u_{i+1}} N_{i+1,p-1}(u)$$

where the u_i define the knot vector $U = \{u_0, \dots, u_m\}$ as a nondecreasing sequence of real numbers, *i.e.*, $u_i \leq u_{i+1}$ for $i = 0, \dots, m-1$. And *m* is related to the number of control points *n* and the degree of the curve *p* with the relation $m = n + p + 1$. The knot vector has the form

$$U = \{\underbrace{a, \dots, a}_{p+1}, u_{p+1}, \dots, u_{m-p-1}, \underbrace{b, \dots, b}_{p+1}\} \quad (3.3)$$

Parameters:

u - the parametric variable

i - specifies which basis function to compute

p - the degree to which the basis function is computed

Returns:

the value of $N_{ip}(u)$

Author(s):

Philippe Lavoie

Date:

24 January 1997

3.11.23 void NurbsCurve::basisFuns(T u, int i, Vector<T>& N) const

computes the non-zero basis functions of the curve.

Computes the non-zero basis functions and puts the result into N . N has a size of $\text{deg}+1$. To relate N to the basis functions, $\text{Basis}[\text{span} - \text{deg} + i] = N[i]$ for $i=0 \dots \text{deg}$.

The B-spline basis function of p -degree is defined as

$$N_{i,0}(u) = \begin{cases} 1 & \text{if } u_i \leq u < u_{i+1} \\ 0 & \text{otherwise} \end{cases}$$

$$N_{i,p}(u) = \frac{u - u_i}{u_{i+p} - u_i} N_{i,p-1}(u) + \frac{u_{i+p+1} - u}{u_{i+p+1} - u_{i+1}} N_{i+1,p-1}(u)$$

where the u_i define the knot vector $U = \{u_0, \dots, u_m\}$ as a nondecreasing sequence of real numbers, *i.e.*, $u_i \leq u_{i+1}$ for $i = 0, \dots, m-1$. And m is related to the number of control points n and the degree of the curve p with the relation $m = n + p + 1$. The knot vector has the form

$$U = \{\underbrace{a, \dots, a}_{p+1}, u_{p+1}, \dots, u_{m-p-1}, \underbrace{b, \dots, b}_{p+1}\} \quad (3.4)$$

The B-spline basis function are non-zero for at most $p + 1$ of the $N_{i,p}$. The relationship between the non-zero basis functions in N and $N_{i,p}$ is as follows, $N_{\text{span} - \text{deg} + j, p} = N[j]$ for $j = 0, \dots, \text{deg}$. Where span is the non-zero span of the basis functions. This non-zero span for can be found by calling `findSpan(u)`.

Parameters:

- u - the parametric value
- i - the non-zero span of the basis functions
- N - the non-zero basis functions

Warning:

u and i must be valid values

Author(s):

Philippe Lavoie

Date:

24 January 1997

3.11.24 void NurbsCurve::dersBasisFuns(int n, T u, int span, Matrix<T>& ders) const

Compute the derivatives functions at u of the NURBS curve.

For information on the algorithm, see A2.3 on p72 of the NURBS book.

The result is stored in the *ders* matrix, where *ders* is of size $(n+1, deg+1)$ and the derivative $N'_i(u) = ders(1, i=span-deg+j)$ where $j=0...deg+1$.

Parameters:

n - the degree of the derivation

u - the parametric value

span - the span for the basis functions

ders - A matrix containing the derivatives of the curve.

Warning:

n , u and *span* must be valid values.

Author(s):

Philippe Lavoie

Date:

24 January 1997

3.11.25 T NurbsCurve::minKnot() const

< the minimal value for the knot vector.

Reimplemented from **ParaCurve**.

3.11.26 T NurbsCurve::maxKnot() const

< the maximal value for the knot vector.

Reimplemented from **ParaCurve**.

3.11.27 int NurbsCurve::findSpan(T u) const

Determines the knot span index.

Determines the knot span for which there exists non-zero basis functions. The span is the index k for which the parameter u is valid in the $[u_k, u_{k+1}]$ range.

Parameters:

u - the parametric value

Returns:

the span index at u .

Warning:

u must be in a valid range

Author(s):

Philippe Lavoie

Date:

24 January 1997

3.11.28 void NurbsCurve::findMultSpan(T u, int& r, int& s) const

Finds the multiplicity of a knot at a parametric value.

Finds the index of the knot at parametric value u and returns its multiplicity.

Parameters:

u - the parametric value

r - the knot of interest

s - the multiplicity of this knot

Warning:

u must be in a valid range.

Author(s):

Philippe Lavoie

Date:

24 January, 1997

3.11.29 int NurbsCurve::findMult(int r) const

Finds the multiplicity of a knot.

Parameters:

r - the knot to observe

Returns:

the multiplicity of the knot

Warning:

r must be a valid knot index

Author(s):

Philippe Lavoie

Date:

24 January, 1997

3.11.30 int NurbsCurve::findKnot(T u) const

Finds the knot k for which u is in the range $[u_k, u_{k+1})$.

Parameters:

u - parametric value

Returns:

the index k

Warning:

u must be in a valid range.

Author(s):

Philippe Lavoie

Date:

24 January, 1997

**3.11.31 T NurbsCurve::getRemovalBnd(int r, int s)
const**

Get the knot removal error bound for an internal knot.

Get the knot removal error bound for an internal knot r (non-rational). For more information on the algorithm, see A9.8 from the Nurbs book on page 428.

Parameters:

curve - a NURBS curve
r - the index of the internal knot to check
s - the multiplicity of that knot

Returns:

The maximum distance between the new curve and the old one

Author(s):

Philippe Lavoie

Date:

24 January, 1997

3.11.32 void NurbsCurve::removeKnot(int r, int s, int num) [virtual]

Removes an internal knot from a curve. This is A5.8 on p185 from the NURB book modified to not check for tolerance before removing the knot.

Parameters:

r - the knot to remove
s - the multiplicity of the knot
num - the number of times to try to remove the knot

Warning:

r **must** be an internal knot.

Author(s):

Philippe Lavoie

Date:

24 January 1997

Reimplemented in **NurbsCurveSP**.

3.11.33 void NurbsCurve::removeKnotsBound(const Vector<T>& ub, Vector<T>& ek, T E [virtual]

Remove knots from a curve without exceeding an error bound.

For more information about the algorithm, see A9.9 on p429 of the NURB book.

Parameters:

ub - the knot coefficients
ek - the error after removing

Author(s):

Philippe Lavoie

Date:

24 January 1997

Reimplemented in **NurbsCurveSP**.

3.11.34 void NurbsCurve::refineKnotVector(const Vector<T>& X) [virtual]

Refine the curve knot vector.

For more information, see A5.4 on page 164 of the NURBS book

Parameters:

X - the new knots to insert in the knot vector

Author(s):

Philippe Lavoie

Date:

24 January, 1997

Reimplemented in **NurbsCurveSP**.

3.11.35 void NurbsCurve::mergeKnotVector(const Vector<T> &Um) [virtual]

Merges the knot vector of a curve with another knot vector.

Will merge the Knot vector *U* with the one from the curve and it will refine the curve appropriately.

Parameters:

Um - the knot vector to merge with

Warning:

the knot *U* must be common with the one from the curve *c*

Author(s):

Philippe Lavoie

Date:

24 January, 1997

Reimplemented in **NurbsCurveSP**.**3.11.36 int NurbsCurve::leastSquares(const Vector<Point_nD<T,N> >& Q, int degC, int n)**

A least squares curve approximation.

This routine solves the following problem: find the NURBS curve C satisfying

- $Q_0 = C(0)$ and $Q_m = C(1)$
- the remaining Q_k are approximated in the least squares sense, *i.e.*

$$\sum_{k=1}^{m-1} |Q_k - C(\bar{u}_k)|^2$$

in a minimum with respect to the n variable P_i ; the \bar{u} are the parameter values computed with the chord length method.

The resulting curve will generally not pass through Q_k and $C(\bar{u}_k)$ is not the closest point on $C(u)$ to Q_k .

For more details, see section 9.4.1 on page 491 of the NURBS book.

Parameters: Q - the vector of 3D points $degC$ - the degree of the curve n - the number of control points in the new curve.**Warning:** deg must be smaller than $Q.n$.**Author(s):**

Philippe Lavoie

Date:

24 January, 1997

3.11.37 `int NurbsCurve::leastSquares(const Vector<Point_nD<T,N> >& Q, int degC, int n, const Vector<T>& ub)`

A least squares curve approximation.

This routine solves the following problem: find the NURBS curve C satisfying

- $Q_0 = C(0)$ and $Q_m = C(1)$
- the remaining Q_k are approximated in the least squares sense, *i.e.*

$$\sum_{k=1}^{m-1} |Q_k - C(\bar{u}_k)|^2$$

in a minimum with respect to the n variable P_i ; the \bar{u} are the precomputed parameter values.

The resulting curve will generally not pass through Q_k and $C(\bar{u}_k)$ is not the closest point on $C(u)$ to Q_k .

For more details, see section 9.4.1 on page 491 of the NURBS book.

Parameters:

Q - the vector of 3D points
 $degC$ - the degree of the curve
 n - the number of control points in the new curve
 ub - the knot coefficients

Warning:

the variable curve **must** contain a valid knot vector.

Author(s):

Philippe Lavoie

Date:

24 January 1997

3.11.38 `int NurbsCurve::leastSquaresH(const Vector<HPoint_nD<T,N> >& Q, int degC, int n, const Vector<T>& ub)`

A least squares curve approximation.

This routine solves the following problem: find the NURBS curve C satisfying

- $Q_0 = C(0)$ and $Q_m = C(1)$
- the remaining Q_k are approximated in the least squares sense, *i.e.*

$$\sum_{k=1}^{m-1} |Q_k - C(\bar{u}_k)|^2$$

in a minimum with respect to the n variable P_i ; the \bar{u} are the precomputed parameter values.

The resulting curve will generally not pass through Q_k and $C(\bar{u}_k)$ is not the closest point on $C(u)$ to Q_k .

For more details, see section 9.4.1 on page 491 of the NURBS book.

Parameters:

Q - the vector of 4D points
 $degC$ - the degree of the curve
 n - the number of control points in the new curve
 ub - the knot coefficients

Warning:

the variable curve **must** contain a valid knot vector.

Author(s):

Philippe Lavoie

Date:

24 January 1997

3.11.39 `int NurbsCurve::leastSquares(const Vector<Point_nD<T,N> >& Q, int degC, int n, const Vector<T>& ub, const Vector<T>& knot)`

A least squares curve approximation.

This routine solves the following problem: find the NURBS curve C satisfying

- $Q_0 = C(0)$ and $Q_m = C(1)$
- the remaining Q_k are approximated in the least squares sense, *i.e.*

$$\sum_{k=1}^{m-1} |Q_k - C(\bar{u}_k)|^2$$

in a minimum with respect to the n variable P_i ; the \bar{u} are the precomputed parameter values.

The resulting curve will generally not pass through Q_k and $C(\bar{u}_k)$ is not the closest point on $C(u)$ to Q_k .

For more details, see section 9.4.1 on page 491 of the NURBS book.

Parameters:

Q - the vector of 3D points
 $degC$ - the degree of the curve
 n - the number of control points in the new curve
 ub - the knot coefficients
 $knot$ - the knot vector to use for the curve

Returns:

1 if succesfull, 0 if the number of points to approximate the curve with is too big compared to the number of points.

Warning:

the variable curve **must** contain a valid knot vector.

Author(s):

Philippe Lavoie

Date:

24 January 1997

3.11.40 void NurbsCurve::globalApproxErrBnd(Vector<Point_nD<T,N> >& Q, int degC, T E)

Approximation of a curve bounded to a certain error.

It is a type II approximation: it starts with a lot of control points then tries to eliminate as much as it can as long as the curve stays within a certain error bound.

The method uses least squares fitting along with knot removal techniques. It is the algorithm A9.10 on p 431 of the NURBS book.

Parameters:

Q - the points to approximate
 $degree$ - the degree of the approximation curve
 E - the maximum error allowed

Author(s):

Philippe Lavoie

Date:

24 January 1997

3.11.41 void NurbsCurve::globalApproxErrBnd(Vector< Point_nD<T,N> >& Q, Vector<T>& ub, int degC, T E)

Approximation of a curve bounded to a certain error.

It is a type II approximation: it starts with a lot of control points then tries to eliminate as much as it can as long as the curve stays within a certain error bound.

The method uses least squares fitting along with knot removal techniques. It is the algorithm A9.10 on p 431 of the NURBS book.

Parameters:

Q - the points to approximate
ub - the vector of parameters where the points are located
degree - the degree of the approximation curve
E - the maximum error allowed

Warning:

ub and *Q* must be of the same size

Author(s):

Philippe Lavoie

24 January 1997

*/

```
template <class T, int N>
```

```
void NurbsCurve<T,N>::globalApproxErrBnd(Vector< Point_nD<T,N> >& Q, Vector<T>& ub, int degC, T E){
```

```
    Vector<T> ek(Q.n) ;
```

```
    Vector<T> Uh(Q.n) ;
```

```
    NurbsCurve<T,N> tcurve ;
```

```
    int i,j,degL ;
```

```
    if(ub.n != Q.n){
```

```
        Error err("globalApproxErrBnd");
```

```
        err << "The data vector and the parameter vectors are not of the same size!\n" ;
```

```
        err << "Q.n = " << Q.n << ", ub.n = " << ub.n << endl ;
```

```
        err.fatal() ;
```

```
    }
```

```
    resize(Q.n,1) ;
```

```
    // Initialize U
```

```
    deg = 1 ;
```

```
    for(i=0;i<ub.n;i++){
```

```
        U[i+deg] = ub[i] ;
```

```
    }
```

```
    U[0] = 0 ;
```

```

U[U.n-1] = 1.0 ;
// Initialize P
for(i=0;i<P.n;i++)
    P[i] = Q[i] ;

for(degL=1; degL<=degC+1 ; degL++){
    removeKnotsBound(ub,ek,E) ;

    if(degL==degC)
        break ;

    if(degL<degC){

        // Find the degree elevated knot vector
        Uh.resize(U.n*2) ;

        Uh[0] = U[0] ;
        j = 1 ;
        for(i=1;i<U.n;++i){
            if(U[i]>U[i-1])
                Uh[j++] = U[i-1] ;
            Uh[j++] = U[i] ;
        }
        Uh[j++] = U[U.n-1] ;
        Uh.resize(j) ;
        tcurve = *this ;
        if(!leastSquares(Q,degL+1,Uh.n-degL-1-1,ub,Uh)){
this = tcurve ;
            degreeElevate(1);
        }
    }
    else{
        tcurve = *this ;
        if(!leastSquares(Q,degL,P.n,ub,U)){
this = tcurve ;
        }
    }
}

// Project the points from curve to Q and update ek and ub
//    for(i=0;i<Q.n;i++){
for(i=0;i<Q.n;i++){
    T u_i ;
    Point_nD<T,N> r_i ;
    projectTo(Q[i],ub[i],u_i,r_i) ;
    ek[i] = norm(r_i-Q[i]) ;
    ub[i] = u_i ;
}
}
}

```

```

template <class T, int N>
void NurbsCurve<T,N>::globalApproxErrBnd2(Vector< Point_nD<T,N> >& Q,
                                         int degC,
                                         T E){

    Vector<T> ub(Q.n) ;
    Vector<T> ek(Q.n) ;
    Vector<T> Uh(Q.n) ;
    NurbsCurve<T,N> tcurve ;
    int i,degL ;

    resize(Q.n,1) ;

    chordLengthParam(Q,ub) ;

    // Initialize U
    deg = 1 ;
    for(i=0;i<ub.n;i++){
        U[i+deg] = ub[i] ;
    }
    U[0] = 0 ;
    U[U.n-1] = 1.0 ;
    // Initialize P
    for(i=0;i<P.n;i++){
        P[i] = Q[i] ;
    }

    for(degL=1; degL<degC ; degL++){
        degreeElevate(1);

        // Project the points from curve to Q and update ek and ub
        // for(i=0;i<Q.n;i++){
        for(i=0;i<Q.n;i++){
            T u_i ;
            Point_nD<T,N> r_i ;
            projectTo(Q[i],ub[i],u_i,r_i) ;
            ek[i] = norm(r_i-Q[i]) ;
            ub[i] = u_i ;
        }
        removeKnotsBound(ub,ek,E) ;
    }
}

template <class T, int N>
void NurbsCurve<T,N>::globalApproxErrBnd3(Vector< Point_nD<T,N> >& Q,int degC,T E){
    //NurbsCurve<T,N> tCurve(1) ;
    Vector<T> ub(Q.n) ;
    Vector<T> ek(Q.n) ;
    int i ;

```

```

        resize(Q.n,1) ;

        chordLengthParam(Q,ub) ;

        // Initialize U
        deg = 1 ;
        for(i=0;i<ub.n;i++){
            U[i+deg] = ub[i] ;
        }
        U[0] = 0 ;
        U[U.n-1] = 1.0 ;

        // Initialize P
        for(i=0;i<P.n;i++)
            P[i] = Q[i] ;

        //removeKnotsBoundCurve(curve,ub,ek,E/10.0,curve) ;
        degreeElevate(degC-1) ;
        removeKnotsBound(ub,ek,E) ;
    }

template <class T, int N>
void NurbsCurve<T,N>::globalApproxErrBnd3(Vector< Point_nD<T,N> >& Q,
                                           const Vector<T> &ub,
                                           int degC,
                                           T E){

    Vector<T> ek(Q.n) ;
    int i ;

    resize(Q.n,1) ;

    // Initialize U
    deg = 1 ;
    for(i=0;i<ub.n;i++){
        U[i+deg] = ub[i] ;
    }
    U[0] = 0 ;
    U[U.n-1] = 1.0 ;

    // Initialize P
    for(i=0;i<P.n;i++)
        P[i] = Q[i] ;

    degreeElevate(degC-1) ;
    removeKnotsBound(ub,ek,E) ;
}

```

```

template <class T, int N>
void NurbsCurve<T,N>::projectTo(const Point_nD<T,N>& p, T guess, T& u, Point_nD<T,N>& r, T e1, T e2,
    T un ;
    T c1, c2;
    Vector< Point_nD<T,N> > Cd ;
    Point_nD<T,N> c, cd,cdd ;
    int t = 0 ;
    u = guess ;

    if(u<U[0]) u = U[0] ;
    if(u>U[U.n-1]) u = U[U.n-1] ;

    while(1) {
        ++t ;
        if(t>maxTry){
            r = c ;
            return ;
        }
        c = pointAt(u) ;
        Cder(u,*this,2,Cd) ;
        cd = Cd[1] ;
        cdd = Cd[2] ;
        c1 = norm2(c-p) ;
        if(c1<e1*e1){
            r = c ;
            return ;
        }
        c2 = norm((Point_nD<T,N>)(cd*(c-p))) ;
        //c2 *= c2 ;
        c2 /= norm(cd)*norm(c-p) ;
        //if(c2<e2*e2){
        if(c2<e2){
            r = c ;
            return ;
        }

        un = u- cd*(c-p)/(cdd*(c-p)+norm2(cd)) ;

        if(un<U[0]) un = U[0] ;
        if(un>U[U.n-1]) un = U[U.n-1] ;

        if(norm2((un-u)*cd)<e1*e1){
            r = c ;
            return ;
        }
        u = un ;
    }
}

```

```

template <class T, int N>
void NurbsCurve<T,N>::degreeElevate(int t){
    if(t<=0){
        return ;
    }

    NurbsCurve<T,N> c(*this) ;

    int i,j,k ;
    int n = c.CtrlP.n-1;
    int p = c.degree ;
    int m = n+p+1;
    int ph = p+t ;
    int ph2 = ph/2 ;
    Matrix<T> bezalfs(p+t+1,p+1) ; // coefficients for degree elevating the Bezier segment
    Vector< HPoint_nD<T,N> > bpts(p+1) ; // pth-degree Bezier control points of the current
    Vector< HPoint_nD<T,N> > ebpts(p+t+1) ; // (p+t)th-degree Bezier control points of the
    Vector< HPoint_nD<T,N> > Nextbpts(p-1) ; // leftmost control points of the next Bezier
    Vector<T> alphas(p-1) ; // knot instertion alphas.

    // Compute the binomial coefficients
    Matrix<T> Bin(ph+1,ph2+1) ;
    binomialCoef(Bin) ;

    // Compute Bezier degree elevation coefficients
    T inv,mpi ;
    bezalfs(0,0) = bezalfs(ph,p) = 1.0 ;
    for(i=1;i<=ph2;i++){
        inv= 1.0/Bin(ph,i) ;
        mpi = minimum(p,i) ;
        for(j=maximum(0,i-t); j<=mpi; j++){
            bezalfs(i,j) = inv*Bin(p,j)*Bin(t,i-j) ;
        }
    }

    for(i=ph2+1;i<ph ; i++){
        mpi = minimum(p,i) ;
        for(j=maximum(0,i-t); j<=mpi ; j++){
            bezalfs(i,j) = bezalfs(ph-i,p-j) ;
        }
    }

    resize(c.P.n+c.P.n*t,ph) ; // Allocate more control points than necessary

    int mh = ph ;
    int kind = ph+1 ;
    T ua = c.U[0] ;
    T ub = 0.0 ;
    int r=-1 ;

```

```

int oldr ;
int a = p ;
int b = p+1 ;
int cind = 1 ;
int rbz,lbz = 1 ;
int mul,save,s;
T alf;
int first, last, kj ;
T den,bet,gam,numer ;

P[0] = c.P[0] ;
for(i=0; i <= ph ; i++){
    U[i] = ua ;
}

// Initialize the first Bezier segment

for(i=0;i<=p ;i++)
    bpts[i] = c.P[i] ;

while(b<m){ // Big loop thru knot vector
    i=b ;
    while(b<m && c.U[b] >= c.U[b+1]) // for some odd reasons... == doesn't work
        b++ ;
    mul = b-i+1 ;
    mh += mul*t ;
    ub = c.U[b] ;
    oldr = r ;
    r = p-mul ;
    if(oldr>0)
        lbz = (oldr+2)/2 ;
    else
        lbz = 1 ;
    if(r>0)
        rbz = ph-(r+1)/2 ;
    else
        rbz = ph ;
    if(r>0){ // Insert knot to get Bezier segment
        numer = ub-ua ;
        for(k=p;k>mul;k--){
            alphas[k-mul-1] = numer/(c.U[a+k]-ua) ;
        }
        for(j=1;j<=r;j++){
            save = r-j ; s = mul+j ;
            for(k=p;k>=s;k--){
                bpts[k] = alphas[k-s] * bpts[k]+(1.0-alphas[k-s])*bpts[k-1] ;
            }
            Nextbpts[save] = bpts[p] ;
        }
    }
}

```



```

for(i=lbz;i<=ph;i++){ // Degree elevate Bezier, only the points lbz,...,ph are used
    ebpts[i] = 0.0 ;
    mpi = minimum(p,i) ;
    for(j=maximum(0,i-t); j<=mpi ; j++)
        ebpts[i] += bezalfs(i,j)*bpts[j] ;
}

if(olldr>1){ // Must remove knot u=c.U[a] oldr times
    // if(olldr>2) // Alphas on the right do not change
    //     alfj = (ua-U[kind-1])/(ub-U[kind-1]) ;
    first = kind-2 ; last = kind ;
    den = ub-ua ;
    bet = (ub-U[kind-1])/den ;
    for(int tr=1; tr<olldr; tr++){ // Knot removal loop
        i = first ; j = last ;
        kj = j-kind+1 ;
        while(j-i>tr){ // Loop and compute the new control points for one removal step
            if(i<cind){
                alf=(ub-U[i])/(ua-U[i]) ;
                P[i] = alf*P[i] + (1.0-alf)*P[i-1] ;
            }
            if( j>= lbz){
                if(j-tr <= kind-ph+olldr){
                    gam = (ub-U[j-tr])/den ;
                    ebpts[kj] = gam*ebpts[kj] + (1.0-gam)*ebpts[kj+1] ;
                }
                else{
                    ebpts[kj] = bet*ebpts[kj]+(1.0-bet)*ebpts[kj+1] ;
                }
            }
            ++i ; --j; --kj ;
        }
        --first ; ++last ;
    }
}

if(a!=p) // load the knot u=c.U[a]
    for(i=0;i<ph-oldr; i++){
        U[kind++] = ua ;
    }
for(j=lbz; j<=rbz ; j++) { // load control points onto the curve
    P[cind++] = ebpts[j] ;
}

if(b<m){ // Set up for next pass thru loop
    for(j=0;j<r;j++)
        bpts[j] = Nextbpts[j] ;
    for(j=r;j<=p;j++)
        bpts[j] = c.P[b-p+j] ;
}

```

```

        a=b ;
        b++ ;
        ua = ub ;
    }
    else{
        for(i=0;i<=ph;i++)
            U[kind+i] = ub ;
    }
}
resize(mh-ph,ph) ; // Resize to the proper number of control points
}

```

, Philippe Lavoie

Date:

24 January 1997

3.11.42 void NurbsCurve::globalInterp(const Vector< Point_nD<T,N> >& Q, int d)

global curve interpolation with points in 3D.

Parameters:

Q - the 3D points to interpolate
d - the degree of the interpolation

Warning:

The number of points to interpolate must be greater than the degree specified for the curve.

Author(s):

Philippe Lavoie

Date:

24 January, 1997

3.11.43 void NurbsCurve::globalInterp(const Vector< Point_nD<T,N> >& Q, const Vector<T>& ub, int d)

global curve interpolation with points in 3D.

Global curve interpolation with points in 3D and with the parametric values specified.

Parameters:

Q - the 3D points to interpolate
ub - the parametric values
d - the degree of the interpolation

Warning:

The number of points to interpolate must be greater than the degree specified for the curve.

Author(s):

Philippe Lavoie

Date:

24 January, 1997

3.11.44 void NurbsCurve::globalInterpH(const Vector<HPoint_nD<T,N> >& Q, int d)

global curve interpolation with points in 4D.

Global curve interpolation with points in 4D

Parameters:

Q - the points in 4D to interpolate
d - the degree of the curve interpolation

Warning:

The number of points to interpolate must be greater than the degree specified for the curve.

Author(s):

Philippe Lavoie

Date:

24 January, 1997

3.11.45 void NurbsCurve::globalInterpH(const Vector<HPoint_nD<T,N> >& Q, const Vector<T>& Uc, int d)

global curve interpolation with 4D points and a knot vector defined.

Global curve interpolation with 4D points and a knot vector defined.

Parameters:

Q - the 3D points to interpolate
 Uc - the knot vector to set the curve to
 d - the degree of the interpolation

Warning:

The number of points to interpolate must be greater than the degree specified for the curve.

Author(s):

Philippe Lavoie

Date:

24 January, 1997

3.11.46 `void NurbsCurve::globalInterpH(const Vector<HPoint_nD<T,N> >& Q, const Vector<T>& ub, const Vector<T>& Uc, int d)`

global curve interpolation with 4D points, a knot vector defined and the parametric value vector defined.

Global curve interpolation with 4D points, a knot vector defined and the parametric value vector defined.

Parameters:

Q - the 3D points to interpolate
 ub - the parametric values vector
 Uc - the knot vector to set the curve to
 d - the degree of the interpolation

Warning:

The number of points to interpolate must be greater than the degree specified for the curve. Uc must be compatible with the values given for $Q.n$, $ub.n$ and d .

Author(s):

Philippe Lavoie

Date:

24 January, 1997

3.11.47 `void NurbsCurve::globalInterpD(const Vector< Point_nD<T,N> >& Q, const Vector< Point_nD<T,N> >& D, int d, int unitD, T a=1.0)`

global curve interpolation with the 1st derivatives of the points specified.

Global curve interpolation with the 1st degree derivative specified for each of the points to interpolate. This will generate a number of control points 2 times greater than the number of interpolation points.

If the derivative specified is of unit length, *i.e.* the tangent vectors are given, then the derivative at each point will be multiplied by the chord length. A second multiplicative factor can be specified to make the derivative even greater. This second number should be close to 1.0.

For more information about the algorithm, refer to section 9.2.4 on page 373 of the NURBS book.

Parameters:

Q - the points to interpolate

D - the first derivative at these points

d - the degree of the curve

unitD - a flag specifying if the derivatives are unit vectors

a - a multiplicative factor for the tangent (must be greater than 0 or the function aborts).

Warning:

The number of points to interpolate must be greater than the degree specified for the curve.

Author(s):

Philippe Lavoie

Date:

3 September, 1997

3.11.48 `T NurbsCurve::length(T eps,int n) const`

Computes the length of the curve.

Computes an approximation of the length of the curve using a numerical automatic integrator.

That integrator uses a Chebyshev Series Expansion to perform its approximation. This is why you can change the value *n* which sets the number of elements in the series.

The method is simple, integrate between each span. This is necessary in case the tangent of a point at `u_i` is undefined. Add the result and return this as the approximation.

Parameters:

n - the number of element in the Chebyshev series
eps - the accepted relative error

Returns:

the length of the NURBS curve.

Author(s):

Philippe Lavoie

Date:

22 September 1998

3.11.49 T NurbsCurve::lengthIn(T us, T ue, T eps, int n) const

Computes the length of the curve inside `[u_s, u_e]`.

Computes an approximation of the length of the curve using a numerical automatic integrator. The length is computed for the range `[u_s, u_e]`

That integrator uses a Chebyshev Series Expansion to perform its approximation. This is why you can change the value *n* which sets the number of elements in the series.

The method is similar to the one used by `length` excepted that it needs to check for the range.

Parameters:

us - the starting range
ue - the end of the range
n - the number of element in the Chebyshev series
eps - the accepted relative error

Returns:

the length of the NURBS curve.

Warning:

`ue` must be greater than `us` and both must be in a valid range.

Author(s):

Philippe Lavoie

Date:

22 September 1998

3.11.50 T NurbsCurve::lengthF(T u) const

The function used by length *length* needs to integrate a function over an interval to determine the length of the NURBS curve. Well, this is the function.

Parameters:

u - \rightarrow the parameter

Returns:

square root of the square of the x,y and z value

Author(s):

Philippe Lavoie

Date:

22 September 1998

**3.11.51 void NurbsCurve::makeCircle(const
Point_nD<T,N>& O, const Point_nD<T,N>&
X, const Point_nD<T,N>& Y, T r, double as,
double ae)**

generates a circular curve.

Generates parts of a circle, starting at angle *as* and finishing at *ae* with a radius *r* and having the origin located at *O*. The *X* and *Y* vector describe the local x-axis and the local y-axis of the circle.

The degrees are specified in radians.

Parameters:

O - the center of the circle
X - unit length vector lying in the plane of the circle
Y - unit length vector lying in the plane of the circle
r - the radius of the circle
as - start angle in radians measured with respect to $\$X\$$
ae - end angle in radians measured with respect to $\$X\$$

Author(s):

Philippe Lavoie

Date:

25 July, 1997

3.11.52 void NurbsCurve::makeCircle(const Point_nD<T,N>& O, T r, double as, double ae)

generates a circular curve.

Generates a circular curve of radius r at origin O . The curve is drawn in the xy -axis.

Parameters:

O - the center of the circle

r - the radius of the circle

as - start angle measured with respect to the x-axis

ae - end angle measured with respect to the y-axis

Author(s):

Philippe Lavoie

Date:

25 July, 1997

3.11.53 void NurbsCurve::makeLine(const Point_nD<T,N>& P0, const Point_nD<T,N>& P1, int d)

Generate a straight line.

Generate a straight line going from point $P0$ to point $P1$ of degree d .

Parameters:

$P0$ - the beginning of the line

$P1$ - the end of the line

d - the degree of the curve

Warning:

d must be greater or equal to 1

Author(s):

Philippe Lavoie

Date:

22 September 1998

3.11.54 **void**
 NurbsCurve::decompose(NurbsCurveArray<T,N>&
 c) const

Decompose the curve into Bzier segments.

This function decomposes the curve into an array of 4D Bzier segments.

Parameters:

c - an array of Bzier segments

Author(s):

Philippe Lavoie

Date:

16 February 1997

3.11.55 **int NurbsCurve::splitAt(T u,**
 NurbsCurve<T,N>& cl, NurbsCurve<T,N>&
 cu) const

Splits the curve into two curves.

Parameters:

u - splits at this parametric value

cl - the lower curve

cu - the upper curve

Returns:

1 if the operation, 0 otherwise.

Warning:

You *must* make sure that you split at a valid parametric value. You can't split a curve at its end points.

Author(s):

Philippe Lavoie

Date:

16 October 1997

3.11.56 `int NurbsCurve::mergeOf(const
 NurbsCurve<T,N>& cl, const
 NurbsCurve<T,N> &cu)`

The curve is the result of merging two curves.

Parameters:

cl - the lower curve
cu - the upper curve

Returns:

1 if the operation is successful, 0 otherwise.

Warning:

You must make sure the knot vectors are compatible, *i.e.* that the end knots of the lower curve are the same as the first knots of the upper curve. The curves must also be of the same degree.

Author(s):

Philippe Lavoie

Date:

16 October 1997

3.11.57 `void NurbsCurve::transform(const
 MatrixRT<T>& A)`

Performs geometrical modifications.

Each control point will be modified by a rotation-translation matrix.

Parameters:

A - the rotation-translation matrix

Author(s):

Philippe Lavoie

Date:

22 August 1997

3.11.58 `int NurbsCurve::movePoint(T u, const Point_nD<T,N>& delta)`

Moves a point on the NURBS curve.

This modifies the curve such that the point $C(u)$ is moved by δ .

See section 11.5.1 of the NURBS book for an explanation of the algorithm.

Parameters:

u - the point to move

δ - the movement in 3D of the point at $C(u)$

Author(s):

Philippe Lavoie

Date:

24 January 1997

3.11.59 `int NurbsCurve::movePoint(T u, const Vector< Point_nD<T,N> >& delta)`

Moves a point in the NURBS curve.

This modifies the curve such that the point $C(u)$ is moved by δ . δ is a vector containing the movement as $D^{\{k\}}$ where $\{k\}$ specifies the derivative. Thus at $D[0]$, this specifies the 0th derivative movement, at $D[1]$ it specifies the 1st derivative movement of the point. *i.e.* Suppose that $C(u) = (10,20,3)$ then a $D[0] = (10,10,10)$ will move the point to $C(u) = (20,30,13)$

See section 11.5.1 of the NURBS book for an explanation of the algorithm.

Parameters:

u - the point to move

δ - the vector of movement

Author(s):

Philippe Lavoie

Date:

24 January 1997

3.11.60 `int NurbsCurve::movePoint(const Vector<T>& ur, const Vector< Point_nD<T,N> >& D)`

Moves a point with some constraint.

This will modify the NURBS curve by respecting a certain number of constraints. `ur` specifies the parameters on which the constraints should be applied. The constraint are defined by `D` which specifies the vector by which the points should move.

For example, if you want to move the point $C(0.5)$ by $(10,0,10)$ and fix the point $C(0.6)$ on the current curve (a move of $(0,0,0)$). `ur = 0.5, 0.6` and `D = (10,0,10), (0,0,0)`

The `ur` vector should be in an increasing order.

See section 11.5.1 of the NURBS book for an explanation of the algorithm.

Parameters:

`ur` - the vector of parameters on which a constraint is applied

`D` - a vector of the value of $D_r^{\{0\}}$

Returns:

1 if the operation is possible, 0 if the problem is ill defined *i.e.* there isn't enough information to find a unique solution (the system is overdetermined) or that the system has non-independant components.

Warning:

`ur` and `D` must have the same size and the values inside `ur` should not repeat and they should be ordered

Author(s):

Philippe Lavoie

Date:

24 January 1997

3.11.61 `int NurbsCurve::movePoint(const Vector<T>& ur, const Vector< Point_nD<T,N> >& D, const Vector_INT& Dr, const Vector_INT& Dk)`

Moves a point with some constraint.

This will modify the NURBS curve by respecting a certain number of constraints. `ur` specifies the parameters on which the constraints should be applied. The constraint are defined by $D_r^{\{k\}}$ which requires 3 vectors to

fully qualify. D specifies the value of the constraint and D_r and D_k are used to specify on which parameter the constraint is applied and of what degree.

For example, if you want to move the point $C(0.5)$ by $(10,0,10)$ and fix the point $C(0.6)$ on the current curve (a move of $(0,0,0)$) but change its 1st derivative by $(0,20,0)$. Then the following values must be inputed to the routine. $ur = [0.5, 0.6]$, $D = [(10,0,10), (0,0,0), (0,20,0)]$, $D_r = [0, 1, 1]$ and $D_k = [0, 0, 1]$.

The values in D should be ordered in respect with r and k . {*\em i.e.*} for $D_i = D_{\{r_i\}^{\{k_i\}}}$, then $i < j$ implies that $r_i < r_j$ and that either $r_i < r_j$ or $k_i < k_j$.

See section 11.5.1 of the NURBS book for an explanation of the algorithm.

Parameters:

ur - the vector of parameters on which a constraint is applied
 D - a vector of the value of $D_r^{\{k\}}$
 Dr - a vector specifying the value of r for D
 Dk - a vector specifying the value of k for D

Returns:

1 if the operation is possible, 0 if the problem is ill defined *i.e.* there isn't enough information to find a unique solution (the system is overdetermined) or that the system has non-independant components.

Warning:

The values inside ur should *not* repeat. D, Dr and Dk must be of the same size.

Author(s):

Philippe Lavoie

Date:

24 January 1997

3.11.62 `int NurbsCurve::movePoint(const Vector<T>& ur, const Vector< Point_nD<T,N> >& D, const Vector_INT& Dr, const Vector_INT& Dk, const Vector_INT& fixCP)`

Moves a point with some constraint.

This will modify the NURBS curve by respecting a certain number of constraints. ur specifies the parameters on which the constraints should be applied. The constraint are defined by $D_r^{\{k\}}$ which requires 3 vectors to fully

qualify. D specifies the value of the constraint and D_r and D_k are used to specify on which parameter the constraint is applied and of what degree.

A second constraint *fixCP* consists of specifying which control points can not be moved by the routine.

For example, if you want to move the point $C(0.5)$ by $(10,0,10)$ and fix the point $C(0.6)$ on the current curve (a move of $(0,0,0)$) but change its 1st derivative by $(0,20,0)$. Doing this without modifying control point 4. Then the following values must be inputted to the routine. $u_r = [0.5, 0.6]$, $D = [(10,0,10), (0,0,0), (0,20,0)]$, $D_r = [0, 1, 1]$, $D_k = [0, 0, 1]$ and *fixCP* = 4.

The values in D should be ordered in respect with r and k . *i.e.* for $D_i = D_{\{r_i\}^{\{k_i\}}}$, then $i < j$ implies that $r_i < r_j$ and that either $r_i < r_j$ or $k_i < k_j$.

See section 11.5.1 of the NURBS book for an explanation of the algorithm.

Parameters:

ur - the vector of parameters on which a constraint is applied

D - a vector of the value of $D_r^{\{k\}}$

D_r - a vector specifying the value of r for D

D_k - a vector specifying the value of k for D

fixCP - a vector specifying which control points $\{\text{em can not}\}$ be modified.

Returns:

1 if the operation is possible, 0 if the problem is ill defined *i.e.* there isn't enough information to find a unique solution (the system is overdetermined) or that the system has non-independant components.

Warning:

The values of *ur* should *not* repeat. D , D_r and D_k must be of the same size.

Author(s):

Philippe Lavoie

Date:

24 January 1997

3.11.63 int NurbsCurve::read(const char* filename)

Reads a NurbsCurve<T,N> from a file.

Parameters:

filename - the filename to read the curve from

Returns:

0 if an error occurs, 1 otherwise

Author(s):

Philippe Lavoie

Date:

24 January 1997

3.11.64 `int NurbsCurve::write(const char* filename)` `const`

Writes a NurbsCurve<T,N> to a file.

Parameters:

filename - the filename to write to.

Returns:

0 if an error occurs, 1 otherwise

Author(s):

Philippe Lavoie

Date:

24 January 1997

3.11.65 `int NurbsCurve::read(istream &fin)` [virtual]

reads a NurbsCurve<T,N> from a file.

Parameters:

fin - an input file stream

Returns:

0 if an error occurs, 1 otherwise

Author(s):

Philippe Lavoie

Date:

24 January 1997

Reimplemented in **NurbsCurveSP**.

3.11.66 int NurbsCurve::write(ofstream &fout) const

Writes a NurbsCurve<T,N> to an output stream.

Parameters:

fout - the output stream

Returns:

0 if an error occurs, 1 otherwise

Author(s):

Philippe Lavoie

Date:

24 January 1997

3.11.67 int NurbsCurve::writePS(const char* filename,int cp,T magFact, T dash) const

Writes the curve in the postscript format to a file.

Parameters:

filename - the file to write the postscript file to

cp - a flag indicating if the control points should be drawn, 0 = no and 1 = yes

magFact - a magnification factor, the 2D point of the control points will be magnified by this value. The size is measured in postscript points. If the magFact is set to a value smaller or equal to 0, than the program will try to guess a magnification factor such that the curve is large enough to fill the page.

dash - the size of the dash in postscript points . A size smaller or equal to 0 indicates that the line joining the control points is plain.

Returns:

0 if an error occurs, 1 otherwise

Warning:

If the weights of the curve are not all at 1, the result might not be representative of the true NURBS curve.

Author(s):

Philippe Lavoie

Date:

16 February 1997

3.11.68 `int NurbsCurve::writePSp(const char* filename, const Vector< Point_nD<T,N> >& points, const Vector< Point_nD<T,N> >& vectors, int cp, T magFact, T dash) const`

Writes a post-script file representing the curve.

Writes the curve in the postscript format to a file, it also draws the points defined in \$points\$ with their associated vectors if \$vector\$ is used.

Parameters:

filename - the file to write the postscript file to

points - draws these additional points as empty circles

vectors - specify a vector associated with the points (this can be an empty vector)

cp - a flag indicating if the control points should be drawn, 0 = no and 1 = yes

magFact - a magnification factor, the 2D point of the control points will be magnified by this value. The size is measured in postscript points. If the magFact is set to a value smaller or equal to 0, than the program will try to guess a magnification factor such that the curve is large enough to fill the page.

dash - the size of the dash in postscript points . A size smaller or equal to 0 indicates that the line joining the control points is plain.

Returns:

0 if an error occurs, 1 otherwise

Warning:

If the weights of the curve are not all at 1, the result might not be representative of the true NURBS curve. If vectors is used, then it must be of the same size as points. If a vector element is (0,0,0) it will not be drawn.

Author(s):

Philippe Lavoie

Date:

16 February 1997

3.11.69 `int NurbsCurve::writeVRML(const char* filename, T radius, int K, const Color& color, int Nu, int Nv, T u_s, T u_e) const`

Writes the curve to a VRML file.

A circle is swept around the trajectory made by the curve. The resulting surface is saved as a VRML file.

Parameters:

filename - the name of the VRML file to save to
radius - the radius of the line
K - the minimum number of interpolation
color - the color of the line
Nu - the number of points for the circle
Nv - the number of points along the path
u_s - the starting parametric value for *u*
u_e - the end parametric value for *u*

Returns:

returns 1 on success, 0 otherwise.

Author(s):

Philippe Lavoie

Date:

25 July 1997

3.11.70 void NurbsCurve::drawImg(Image_Color& Img,const Color& color,T step)

Draws a NURBS curve on an image.

This will draw very primitively the NURBS curve on an image. The drawing assumes the line is only in the xy plane (the z is not used for now).

The algorithm finds the points on the curve at a *step* parametric intervall between them and join them by a line. No fancy stuff.

Parameters:

Img - draws the nurbs curve to this Image
color - the line is drawn in this color
step - the parametric distance between two computed points.

Author(s):

Philippe Lavoie

Date:

24 January 1997

3.11.71 void NurbsCurve::drawAaImg(Image_Color& Img, const Color& color, int precision, int alpha)

Draws an anti-aliased NURBS curve on an image.

This will draw the NURBS by using a circular brush profile. The drawing is performed by averaging the intensity of the profile at the pixels.

Parameters:

Img - draws the nurbs curve to this Image

color - the line is drawn in this color

precision - this number influences the number of points used for averaging purposes.

alpha - a flag indicating if the profile is used as an alpha channel. If so, the line doesn't overwrite, it blends the line with the image already present in *Img*.

Warning:

This routine is very *slow*; use normal drawing for speed.

Author(s):

Philippe Lavoie

Date:

25 July 1997

3.11.72 void NurbsCurve::drawAaImg(Image_Color& Img, const Color& color, const NurbsCurve<T,3>& profile, int precision, int alpha)

draws an anti-aliased NURBS curve on an image.

This will draw the NURBS by using a user-defined brush profile. The drawing is performed by averaging the intensity of the profile at the pixels.

Parameters:

Img - draws the nurbs curve to this Image

color - the line is drawn in this color

profile - the profile of the NURBS curve to draw

precision - this number influences the number of points used for averaging purposes.

alpha - a flag indicating if the profile is used as an alpha channel. If so, the line doesn't overwrite, it blends the line with the image already present in *Img*.

Warning:

This routine is very *slow*; use normal drawing for speed.

Author(s):

Philippe Lavoie

Date:

22 August 1997

3.11.73 NurbsSurface<T,3> NurbsCurve::drawAaImg(Image_Color& Img, const Color& color, const NurbsCurve<T,3>& profile, const NurbsCurve<T,3> &scaling, int precision=3,int alpha=1)

Draws an anti-aliased NURBS curve on an image.

This will draw the NURBS by using a brush profile. The drawing is performed by averaging the intensity of the profile at the pixels.

This function generates a sweep surface by using the profile given in its argument. The sweep is always performed by following the y-axis of the profile. A scaling function is also used when sweeping. This is used to vary the shape of the profile while it's being swept (see the sweep member function of **NurbsSurface<T,N>** for more details).

Parameters:

Img - draws the nurbs curve to this Image

color - the line is drawn in this color

profile - the profile of the NURBS curve to draw

scaling - the scaling to give the profile while drawing the curve

precision - this number influences the number of points used for averaging purposes.

alpha - a flag indicating if the profile is used as an alpha channel. If so, the line doesn't overwrite, it blends the line with the image already present in *Img*.

Warning:

This routine is very *slow*; use normal drawing for speed or lower the precision factor.

Author(s):

Philippe Lavoie

Date:

25 July 1997

3.11.74 **BasicList<Point_nD<T,N>** **NurbsCurve::tessellate(T tolerance, BasicList<T>** ***uk) const**

Generates a list of points from the curve.

Generates a list of points from the curve. The list is generated within a user specified tolerance.

Parameters:

tolerance - -> the tolerance for the tessellation.

Returns:

The list of points.

Author(s):

Philippe Lavoie

Date:

17 October 1997

3.11.75 **T chordLengthParam(const Vector<** **Point_nD<T,N> >& Q, Vector<T> &ub)**

chord length parameterization.

Performs chord length parameterization from a vector of points. The chord length parameterization works as follows:

- The total chord length is defined as

$$d = \sum_{k=1}^{n-1} |Q_k - Q_{k-1}| \quad (3.5)$$

- $\bar{u}_0 = 0$ and $\bar{u}_{n-1} = 1$

-

$$\bar{u}_k = \bar{u}_{k-1} + \frac{|Q_k - Q_{k-1}|}{d} \quad k = 1, \dots, n-2 \quad (3.6)$$

Where n is the size of the vector Q .

Parameters:

Q - a vector of 3D points

ub - the result of chord length parameterization

Returns:

the total chord length of the points.

Author(s):

Philippe Lavoie

Date:

24 January, 1997

Member Data Documentation**3.11.76** `const int& NurbsCurve::degree`

a reference to the degree of the curve.

3.11.77 `const Vector< HPoint_nD<T,N> >& NurbsCurve::CtrlP`

a reference to the vector of control points.

3.11.78 `const Vector<T>& NurbsCurve::Knot`

a reference to the vector of knots.

The documentation for this class was generated from the following files:

- nurbs.hh
- nurbs.cc

3.12 NurbsCurveArray Class Reference

an array of **NurbsCurve**.

```
#include <nurbs/nurbs.hh>
```

Public Members

- **NurbsCurveArray** (NurbsCurve<T, N>* Ca, int size)
Constructor from a pointer to an array of curves.
- **NurbsCurveArray** ()
- virtual ~**NurbsCurveArray** ()
- virtual NurbsCurve<T,N>& **operator[]** (int i)
- virtual NurbsCurve<T,N> **operator[]** (int i) const
- virtual void **resize** (int s)
Resize the NurbsCurveArray.
- void **init** (NurbsCurve<T, N>* Ca, int size)
Initialize the array of curves with a vector of nurbs curve.
- int **read** (const char *filename)
Reads a NurbsCurveArray from a file.
- int **write** (const char *filename)
Writes a NurbsCurveArray from a file.
- int **writePS** (const char*, int cp=0, T magFact=T(-1), T dash=T(5)) const
Writes a post-script file representing an array of curves.
- int **writePSp** (const char*, const Vector< Point_nD<T, N> >&, const Vector< Point_nD<T, N> >&, int cp=0, T magFact=0.0, T dash=5.0) const
Writes a post-script file representing the array of curves.
- const int& **n**

Protected Members

- NurbsCurve<T,N>& **curve** (int i)
- NurbsCurve<T,N> **curve** (int i) const
- int **size**
- int **rsize**

- NurbsCurve<T,N>** C
-

Detailed Description

an array of **NurbsCurve**.

This class represents an array of **NurbsCurve**.

Author(s):

Philippe Lavoie

Date:

4 Oct. 1996

Member Function Documentation

3.12.1

NurbsCurveArray::NurbsCurveArray(NurbsCurve<T,N>*
Ca, int s)

Constructor from a pointer to an array of curves.

Author(s):

Philippe Lavoie

Date:

24 January 1997

3.12.2 void NurbsCurveArray::resize(int size) [virtual]

Resize the NurbsCurveArray.

Parameters:

size - the new size

Author(s):

Philippe Lavoie

Date:

24 January 1997

3.12.3 void NurbsCurveArray::init(NurbsCurve<T,N>* ca,int size)

Initialize the array of curves with a vector of nurbs curve.

Parameters:

ca - a pointer to a vector of NURBS curve
size - the size of the array

Author(s):

Philippe Lavoie

Date:

24 January 1997

3.12.4 int NurbsCurveArray::read(const char* filename)

Reads a NurbsCurveArray from a file.

Parameters:

filename - the filename to read the curve array from

Returns:

0 if an error occurs, 1 otherwise

Author(s):

Philippe Lavoie

Date:

24 January 1997

3.12.5 int NurbsCurveArray::write(const char* filename)

Writes a NurbsCurveArray from a file.

Parameters:

filename - -> the filename to read the curve array from

Returns:

0 if an error occurs, 1 otherwise

Author(s):

Philippe Lavoie

Date:

24 January 1997

3.12.6 int NurbsCurveArray::writePS(const char* filename,int cp,T magFact, T dash) const

Writes a post-script file representing an array of curves.

Parameters:

filename - the file to write the postscript file to

cp - a flag indicating if the control points should be drawn, 0 = no and 1 = yes

magFact - a magnification factor, the 2D point of the control points will be magnified by this value. The size is measured in postscript points. If the *magFact* is set to a value smaller or equal to 0, than the program will try to guess a magnification factor such that the curve is large enough to fill the page.

dash - the size of the dash in postscript points . A size smaller or equal to 0 indicates that the line joining the control points is plain.

Returns:

0 if an error occurs, 1 otherwise

Warning:

If the weights of the curve are not all at 1, the result might not be representative of the true NURBS curve.

Author(s):

Philippe Lavoie

Date:

7 October 1998

3.12.7 int NurbsCurveArray::writePSp(const char* filename,const Vector< Point_nD<T,N> >& points, const Vector< Point_nD<T,N> >& vectors, int cp, T magFact, T dash) const

Writes a post-script file representing the array of curves.

Writes the array of curves in the postscript format to a file, it also draws the points defined in `$points$` with their associated vectors if `$vector$` is used.

Parameters:

filename - the file to write the postscript file to

points - draws these additional points as empty circles

vectors - specify a vector associated with the points (this can be an empty vector)

cp - a flag indicating if the control points should be drawn, 0 = no and 1 = yes

magFact - a magnification factor, the 2D point of the control points will be magnified by this value. The size is measured in postscript points. If the *magFact* is set to a value smaller or equal to 0, than the program will try to guess a magnification factor such that the curve is large enough to fill the page.

dash - the size of the dash in postscript points . A size smaller or equal to 0 indicates that the line joining the control points is plain.

Returns:

0 if an error occurs, 1 otherwise

Warning:

If the weights of the curve are not all at 1, the result might not be representative of the true NURBS curve. If vectors is used, then it must be of the same size as points. If a vector element is (0,0,0) it will not be drawn.

Author(s):

Philippe Lavoie

Date:

7 October 1998

The documentation for this class was generated from the following files:

- nurbs.hh
- nurbsArray.cc

3.13 NurbsCurveGL Class Reference

A NURBS curve class with OpenGL interface.

`#include <nurbs/nurbsGL.hh>`

Inherits **NurbsGL**.

Public Members

- **NurbsCurveGL** ()
- **NurbsCurveGL** (const NurbsCurve& nurb)
- **NurbsCurveGL** (const NurbsCurveGL& nurb)
- **NurbsCurveGL** (const Vector<HPoint3Df>& P1, const Vector<float>& U1, int degree=3)
- **NurbsCurveGL** (const Vector<Point3Df>& P1, const Vector<float>& W, const Vector<float>& U1, int degree=3)
- void **gluNurbs** () const
creates a nurbs curve for OpenGL.
- void **point** (float &u, float &v, int pSize, const Color& colorP, int cp_flag=0) const
draws a point at the location C(u).
- NurbsCurveGL& **operator=** (const NurbsCurveGL& a)
Copies another Nurbs Curve GL.
- NurbsCurveGL& **operator=** (const NurbsCurvef& a)
- void **resetBoundingBox** ()
resets the minP and maxP values of bbox.
- void **resetCPoints** ()
reset the control point information.
- void **resetPolygon** ()
- void **resetKnots** ()
Reset the knots information.
- ObjectGL* **copy** ()
- int **read** (ifstream &fin)
Reads the information from a stream.
- int **write** (ofstream &fout) const
Writes a NurbsCurveGL to an output stream.

- void **applyTransform** ()
apply the local transformation to the curve.
 - void **modifyPoint** (float u, float v, float dx, float dy, float dz)
Modifies a point on the curve.
 - void **setSym** (int set, int uDir, float x, float y, float z, float w)
-

Detailed Description

A NURBS curve class with OpenGL interface.

Author(s):

Philippe Lavoie

Date:

23 September 1997

Member Function Documentation

3.13.1 void NurbsCurveGL::gluNurbs() const

creates a nurbs curve for OpenGL.

This function calls between a gluBeginCurve/gluEndCurve the proper functions to generate a NURBS curve.

Author(s):

Philippe Lavoie

Date:

23 September 1997

Reimplemented from **NurbsGL**.

3.13.2 void NurbsCurveGL::point(float &u,float &v,int pSize,const Color& colorP, int cp_flag) const

draws a point at the location $C(u)$.

This function calls between a glBegin/glEnd the proper functions to represent the point which is at $C(u)$ on the curve.

Parameters:

u - the parametric value
v - a dummy variable so the call is the same as with a NURBS surface.
psize - the size of the control points
colorP - the color of the control points

Author(s):

Philippe Lavoie

Date:

23 September 1997

Reimplemented from **NurbsGL**.

3.13.3 NurbsCurveGL& NurbsCurveGL::operator=(const NurbsCurveGL &a)

Copies another Nurbs Curve GL.

Parameters:

a - the Nurbs curve to copy

Author(s):

Philippe Lavoie

Date:

6 November 1997

3.13.4 void NurbsCurveGL::resetBoundingBox()

resets the minP and maxP values of bbox.

Resets the minP and maxP values for the bounding box.

Warning:

Calling this function without a proper curve initialized might result in strange results.

Author(s):

Philippe Lavoie

Date:

23 September 1997

Reimplemented from **NurbsGL**.**3.13.5 void NurbsCurveGL::resetCPoints()**

reset the control point information.

Reset the control points information stored in cpoints.

Author(s):

Philippe Lavoie

Date:

23 September 1997

Reimplemented from **NurbsGL**.**3.13.6 void NurbsCurveGL::resetKnots()**

Reset the knots information.

Reset the knot information stored in knots.

Author(s):

Philippe Lavoie

Date:

23 September 1997

Reimplemented from **NurbsGL**.**3.13.7 int NurbsCurveGL::read(ifstream &fin)**

Reads the information from a stream.

Parameters:*fin* - the input stream

Returns:

1 on success, 0 on failure

Author(s):

Philippe Lavoie

Date:

19 June 1998

Reimplemented from **ObjectGL**.

3.13.8 int NurbsCurveGL::write(ofstream &fout) const

Writes a NurbsCurveGL to an output stream.

Parameters:

fout - the output stream

Returns:

0 if an error occurs, 1 otherwise

Author(s):

Philippe Lavoie

Date:

19 June 1998

Reimplemented from **ObjectGL**.

3.13.9 void NurbsCurveGL::applyTransform()

apply the local transformation to the curve.

Apply the local transformation to the curve. This is necessary if you want to get the proper position for the control points before doing anymore processing on them.

Author(s):

Philippe Lavoie

Date:

23 September 1997

Reimplemented from **ObjectGL**.

3.13.10 void NurbsCurveGL::modifyPoint(float u, float v, float dx, float dy, float dz)

Modifies a point on the curve.

Parameters:

u - the u parametric value
v - the v parametric value
dx - the delta value in the *x*-axis direction
dy - the delta value in the *y*-axis direction
dz - the delta value in the *z*-axis direction

Author(s):

Philippe Lavoie

Date:

7 November 1997

Reimplemented from **NurbsGL**.

The documentation for this class was generated from the following files:

- nurbsGL.hh
- nurbsGL.cc

3.14 NurbsCurveSP Class Reference

A NURBS curve with surface point.

```
#include <nurbs/nurbs_sp.hh>
```

Inherits **NurbsCurve**.

Public Members

- **NurbsCurveSP** ()
- **NurbsCurveSP** (const NurbsCurve<T, N>& nurb)
- **NurbsCurveSP** (const NurbsCurveSP<T, N>& nurb)
- **NurbsCurveSP** (const Vector< HPoint_nD<T, N> >& P1, const Vector<T> &U1, int degree=3)
- **NurbsCurveSP** (const Vector< Point_nD<T, N> >& P1, const Vector<T> &W, const Vector<T> &U1, int degree=3)
- virtual void **reset** (const Vector< HPoint_nD<T, N> >& P1, const Vector<T> &U1, int degree)
- virtual NurbsCurve<T,N>& **operator=** (const NurbsCurve<T, N>& a)
- NurbsCurveSP<T,N>& **operator=** (const NurbsCurveSP<T, N>& a)
- virtual void **modKnot** (const Vector<T>& knot)
- virtual void **removeKnot** (int r, int s, int num)
- virtual void **removeKnotsBound** (const Vector<T>& ub, Vector<T>& ek, T E)
- virtual void **refineKnotVector** (const Vector<T>& X)
- virtual void **mergeKnotVector** (const Vector<T> &Um)
- virtual void **knotInsertion** (T u, int r, NurbsCurveSP<T, N>& nc)
- virtual void **degreeElevate** (int t)
- int **read** (ifstream &fin)
- void **modSurfCPby** (int i, const HPoint_nD<T, N>& a)
- void **modSurfCP** (int i, const HPoint_nD<T, N>& a)
- void **modOnlySurfCPby** (int i, const HPoint_nD<T, N>& a)

Move the surface point only.

- void **modOnlySurfCP** (int i, const HPoint_nD<T, N>& a)
- T **maxAt** (int i) const
- HPoint_nD<T,N> **surfP** (int i) const
- void **updateMaxU** ()

Updates the basis value.

- int **okMax** ()

Protected Members

- Vector<T> **maxU**
 - Vector<T> **maxAt_**
-

Detailed Description

A NURBS curve with surface point.

A Nurbs curve with surface point manipulators. This allows someone to modify the point on a curve for which a ControlPoint has maximal influence over it. This might provide a more intuitive method to modify a curve.

Author(s):

Philippe Lavoie

Date:

7 May, 1998

Member Function Documentation

3.14.1 void NurbsCurveSP::modOnlySurfCPby(int i, const HPoint_nD<T,N>& a)

Move the surface point only.

Moves only the specified surface point. The other surface points normally affected by moving this point are {\em not} moved.

The point a is in the 4D homogenous space, but only the x,y,z value are used. The weight is not moved by this function.

Parameters:

- i* - the surface point to move
- a* - move that surface point by that amount.

Warning:

The degree of the curve must be of 3 or less.

Author(s):

Philippe Lavoie

Date:

7 June, 1998

3.14.2 void NurbsCurveSP::updateMaxU()

Updates the basis value.

Updates the basis value at which a control point has maximal influence. It also finds where the control point has maximal influence.

Warning:

The degree of the curve must be of 3 or less.

Author(s):

Philippe Lavoie

Date:

7 May, 1998

The documentation for this class was generated from the following files:

- nurbs_sp.hh
- nurbs_sp.cc

3.15 NurbsGL Class Reference

a Virtual NURBS object class.

`#include <nurbs/nurbsGL.hh>`

Inherits **ObjectGL**.

Inherited by **HNurbsSurfaceGL**, **NurbsCurveGL** and **NurbsSurfaceGL**.

Public Members

- **NurbsGL** ()
the default constructor.
- virtual **~NurbsGL** ()
- virtual NurbsGL& **operator=** (const NurbsGL& a)
Copies another Nurbs Curve GL.
- virtual void **gluNurbs** () const = 0
- void **setNurbsRenderer** (GLUnurbsObj *nobj)
- virtual void **glObject** () const
displays the object if it's not hiding.
- virtual void **point** (float &u, float &v, int pSize, const Color& colorP, int cp_flag=0) const = 0
- virtual void **resetAll** ()
- virtual void **resetBoundingBox** ()
- virtual void **resetCPoints** ()
- virtual void **resetPolygon** ()
- virtual void **resetKnots** ()
- void **setObjectColor** (const Color& a, const Color& b, const Color& c)
- void **setBBoxColor** (const Color& a, const Color& b, const Color& c, const Color& d, const Color& e, const Color& f)
- void **setPolygonColor** (const Color& a, const Color& b, const Color& c)
- void **setCPointColor** (const Color& a, const Color& b, const Color& c, const Color& d)
- void **setKnotsColor** (const Color& a, const Color& b, const Color& c)
- void **viewBBox** ()
- void **viewCPoints** ()
- void **viewCpolygon** ()
- void **viewNurbs** ()
- void **viewKnots** ()
- void **hideBBox** ()
- void **hideCPoints** ()

- void **hideCpolygon** ()
- void **hideNurbs** ()
- void **hideKnots** ()
- void **select** ()
- void **deselect** ()
- void **activate** ()
- void **deactivate** ()
- virtual void **setSym** (int set, int uDir, float x, float y, float z, float w) = 0
- virtual void **modifyPoint** (float u, float v, float dx, float dy, float dz) = 0
- int **editSurfacePoints** () const
- int **editControlPoints** () const
- int **editSurfacePoints** (int a)
- int **editControlPoints** (int a)
- int **editFixPoints** () const
- int **editFixPoints** (int a)
- void **setULines** (int u)
- void **setVLines** (int v)
- int **ULines** () const
- int **VLines** () const
- Color **colorCP**
- Color **colorCP0**
- Color **colorPolygon**
- Color **colorKnot**
- ObjectListGL **cpoints**
- ObjectListGL **knots**
- ObjectGL* **polygon**
- BoundingBoxGL **bbox**
- ObjectGLState **nurbsState**

Protected Members

- int **editSP**
- int **editFix**
- GLUnurbsObj* **nurbsRenderer**
- int **nULines**
- int **nVlines**

Related Functions

(Note that these are not member functions.)

- NurbsGL* **readNurbsObject** (const char* filename)
reads a Nurb object from a file.
-

Detailed Description

a Virtual NURBS object class.

Author(s):

Philippe Lavoie

Date:

28 September 1997

Member Function Documentation

3.15.1 NurbsGL::NurbsGL()

the default constructor.

Initialized the colors of the objects.

Author(s):

Philippe Lavoie

Date:

23 September 1997

3.15.2 NurbsGL& NurbsGL::operator=(const NurbsGL &a) [virtual]

Copies another Nurbs Curve GL.

Parameters:

a - the Nurbs curve to copy

Author(s):

Philippe Lavoie

Date:

6 November 1997

3.15.3 void NurbsGL::glObject() const [virtual]

displays the object if it's not hiding.

Displays the object if it's not hiding.

Author(s):

Philippe Lavoie

Date:

23 September 1997

Reimplemented from **ObjectGL**.

3.15.4 NurbsGL* readNurbsObject(const char* filename)

reads a Nurb object from a file.

Reads a NURBS object from a file. The routine can read a Nurbs curve or surface.

Parameters:

filename - the name of the file to read

o - a pointer to a Nurbs Object. If this pointer as any value upon entry, the old entry is deleted first.

Returns:

1 if the Nurbs object was read succesfully, 0 otherwise

Warning:

Calling this function without a proper curve initialized might result in strange results.

Author(s):

Philippe Lavoie

Date:

23 September 1997

The documentation for this class was generated from the following files:

- nurbsGL.hh
- nurbsGL.cc

3.16 NurbsListGL Class Reference

a linked list of **NurbsGL**.

```
#include <nurbs/nurbsGL.hh>
```

Inherits **ObjectListGL**.

Public Members

- **NurbsGL*** **remove** (**NurbsGL*** obj)
 - void **glObject** () const
Calls glObject for all the Nurbs Object in the list.
- void **display** () const
Displays all the NURBS object from the list.
- void **resetDisplayFlags** (int o, int cp, int p, int b, int k, int behavior=NURBS_FLAGS_AFFECT_ALL)
Resets the display flags for the elements.

Detailed Description

a linked list of **NurbsGL**.

Author(s):

Philippe Lavoie

Date:

28 September 1997

Member Function Documentation

3.16.1 void NurbsListGL::glObject() const

Calls glObject for all the Nurbs Object in the list.

Author(s):

Philippe Lavoie

Date:

30 September 1997

Reimplemented from **ObjectListGL**.**3.16.2 void NurbsListGL::display() const**

Displays all the NURBS object from the list.

Author(s):

Philippe Lavoie

Date:

30 September 1997

Reimplemented from **ObjectListGL**.**3.16.3 void NurbsListGL::resetDisplayFlags(int o, int cp, int p, int b, int k, int behavior)**

Resets the display flags for the elements.

Resets the display flags for the elements in the list. It will only resets the flags according to the behavior value.

Parameters:*o* - display flag*cp* - display the control points*p* - display the control polygon*b* - display the bounding box*k* - display the knots*behavior* - specifies which object are affected by the function**Author(s):**

Philippe Lavoie

Date:30 September 1997

The documentation for this class was generated from the following files:

- nurbsGL.hh
- nurbsGL.cc

3.17 NurbsSpolygonGL Class Reference

Holds the control polygon for a surface.

`#include <nurbs/nurbsGL.hh>`

Inherits **ObjectGL**.

Public Members

- **NurbsSpolygonGL** (NurbsSurface& s)
- void **glObject** () const
draws the polygon joining the control points.

Protected Members

- NurbsSurface& **surface**

Detailed Description

Holds the control polygon for a surface.

Author(s):

Philippe Lavoie

Date:

23 September 1997

Member Function Documentation

3.17.1 void NurbsSpolygonGL::glObject() const

draws the polygon joining the control points.

This function calls between a glBegin/glEnd the proper functions to represent the polygon joining all the control points of the NURBS surface.

Author(s):

Philippe Lavoie

Date:

23 September 1997

Reimplemented from **ObjectGL**.

The documentation for this class was generated from the following files:

- nurbsGL.hh
- nurbsGL.cc

3.18 NurbsSurface Class Reference

A class to represent a NURBS surface.

`#include <nurbs/nurbsS.hh>`

Inherits **ParaSurface**.

Inherited by **HNurbsSurface** and **NurbsSurfaceSP**.

Public Members

- **NurbsSurface** ()
Default constructor.
- **NurbsSurface** (const NurbsSurface<T, N>& nS)
the copy constructor.
- **NurbsSurface** (int DegU, int DegV, const Vector<T>& Uk, const Vector<T>& Vk, const Matrix< HPoint_nD<T, N> >& Cp)
constructor with points in homogenous space.
- **NurbsSurface** (int DegU, int DegV, Vector<T>& Uk, Vector<T>& Vk, Matrix< Point_nD<T, N> >& Cp, Matrix<T>& W)
constructor with points in 3D.
- virtual **~NurbsSurface** ()
< Empty desctructor.
- virtual NurbsSurface<T,N>& **operator=** (const NurbsSurface<T, N>&)
NurbsSurface<T,N> assignment.
- void **resize** (int Pu, int Pv, int DegU, int DegV)
Resize the surface.
- virtual void **resizeKeep** (int Pu, int Pv, int DegU, int DegV)
Resize the surface while keeping the old values.
- int **ok** ()
- virtual HPoint_nD<T,N> **operator()** (T u, T v) const
Returns the point on the surface at u,v.
- void **basisFuns** (T u, T v, int spanU, int spanV, Vector<T>& Nu, Vector<T>& Nv) const

Find the non-zero basis functions in the U and V direction.

- void **basisFunsU** (T u, int span, Vector<T>& N) const
Finds the non-zero basis function in the U direction.
- void **basisFunsV** (T u, int span, Vector<T>& N) const
Finds the non-zero basis function in the V direction.
- void **deriveAt** (T u, T v, int d, Matrix< Point_nD<T, N> >& skl) const
Computes the point and the derivatives of degree d and below at (u,v).
- void **deriveAtH** (T u, T v, int d, Matrix< HPoint_nD<T, N> >& skl) const
computes the point and the derivatives of degree d and below at (u,v).
- Point_nD<T,N> **normal** (T u, T v) const
Computes the normal of the surface at (u,v).
- void **globalInterp** (const Matrix< Point_nD<T, N> >& Q, int pU, int pV)
Generates a surface using global interpolation.
- void **globalInterpH** (const Matrix< HPoint_nD<T, N> >& Q, int pU, int pV)
Generates a surface using global interpolation with homogenous points.
- void **leastSquares** (const Matrix< Point_nD<T, N> >& Q, int pU, int pV, int nU, int nV)
generates a surface using global least squares approximation.
- int **skinV** (NurbsCurveArray<T, N>& ca, int degV)
Generates a NURBS surface from skinning.
- int **skinU** (NurbsCurveArray<T, N>& ca, int degU)
Generates a NURBS surface from skinning.
- void **sweep** (const NurbsCurve<T, N>& T, const NurbsCurve<T, N>& C, const NurbsCurve<T, N>& Sv, int K, int useAy=0, int invAz=0)
Generates a surface by sweeping a curve along a trajectory.
- void **sweep** (const NurbsCurve<T, N>& T, const NurbsCurve<T, N>& C, int K, int useAy=0, int invAz=0)
Generates a surface by sweeping a curve along a trajectory.

- void **makeFromRevolution** (const NurbsCurve<T, N>& profile, const Point_nD<T, N>& S, const Point_nD<T, N>& T, double theta=2*M_PI)
Generates a surface of revolution.
- void **makeSphere** (const Point_nD<T, N>& O, T r)
Generates a sphere.
- void **degreeElevate** (int tU, int tV)
Degree elevate the surface in the U and V direction.
- virtual void **degreeElevateU** (int tU)
Degree elevate the surface in the U direction.
- virtual void **degreeElevateV** (int tV)
Degree elevate the surface in the V direction.
- int **decompose** (NurbsSurfaceArray<T, N>& Sa) const
Decompose the surface into Bzier patches.
- void **findSpan** (T u, T v, int& spanU, int& spanV) const
finds the span in the U and V direction.
- int **findSpanU** (T u) const
finds the span in the U direction.
- int **findSpanV** (T v) const
finds the span in the V direction.
- int **findMultU** (int r) const
Finds the multiplicity of a knot in the U knot.
- int **findMultV** (int r) const
finds the multiplicity of a knot in the V knot.
- virtual void **refineKnots** (const Vector<T>& nU, const Vector<T>& nV)
Refine both knot vectors.
- virtual void **refineKnotU** (const Vector<T>& X)
Refines the U knot vector.
- virtual void **refineKnotV** (const Vector<T>& X)
Refines the V knot vector.

- virtual void **mergeKnots** (const Vector<T>& nU, const Vector<T>& nV)
merges a U and V knot vector with the surface knot vectors.
- virtual void **mergeKnotU** (const Vector<T>& X)
merges the U knot vector with another one.
- virtual void **mergeKnotV** (const Vector<T>& X)
merges the V knot vector with another one.
- void **isoCurveU** (T u, NurbsCurve<T, N>& c) const
Generates an iso curve in the U direction.
- void **isoCurveV** (T v, NurbsCurve<T, N>& c) const
Generates an iso curve in the V direction.
- int **read** (const char* filename)
read a surface from a file.
- int **write** (const char* filename) const
Write a surface to a file.
- virtual int **read** (ifstream &fin)
Read a surface from an input stream.
- int **write** (ofstream &fout) const
Write a surface to a file stream.
- int **writeVRML** (const char* filename, const Color& color, int Nu, int Nv, T u_s, T u_e, T v_s, T v_e) const
- ostream& **print** (ostream& os) const
Sends the NURBS Surface to ostream for display.
- int **writeVRML** (const char* filename, T tolerance, const Color& color) const
write the NURBS surface to a VRML file.
- int **writeVRML** (const char* filename, const Color& color=whiteColor, int Nu=20, int Nv=20) const
*< Calls the **ParaSurface** routine with proper values.*
- int **writePOVRAY** (ostream& povray, int patch_type=1, double flatness=0.01, int num_u_steps=8, int num_v_steps=8) const

Writes a set of povray bicubic patches to the ostream.

- int **writePOVRAY** (T, ostream& povray, const Color& color=Color(250, 250, 250), int smooth=0, T ambient=0.2, T diffuse=0.6) const

Writes the surface as a mesh of triangles.

- int **writePOVRAY** (const char *filename, const Color& color, const Point_nD<T, N>& view, const Point_nD<T, N>& up, int patch_type=1, double flatness=0.01, int num_u_steps=8, int num_v_steps=8) const

Writes a set of povray bicubic patches.

- int **writePOVRAY** (T tolerance, const char *filename, const Color& color, const Point_nD<T, N>& view, const Point_nD<T, N>& up, int smooth=0, T ambient=0.2, T diffuse=0.6) const

Writes a set of povray bicubic patches.

- int **writeRIB** (ostream& rib) const

Writes a NuPatch for render man.

- int **writeRIB** (const char* filename, const Color& color, const Point_nD<T, N>& view) const

Writes a NuPatch for render man.

- void **tessellate** (T tolerance, BasicList<Point_nD<T, N> > &points, BasicList<int> &connect, BasicList<Point_nD<T, N> > *normal=0) const

Generates a list of triangles for a surface.

- int **writePS** (const char*, int nu, int nv, const Point_nD<T, N>& camera, const Point_nD<T, N>& lookAt, int cp=0, T magFact=T(-1), T dash=T(5)) const

Writes a post-script file representing the curve.

- int **writePSp** (const char*, int nu, int nv, const Point_nD<T, N>& camera, const Point_nD<T, N>& lookAt, const Vector< Point_nD<T, N> >&, const Vector< Point_nD<T, N> >&, int cp=0, T magFact=0.0, T dash=5.0) const

writes a post-script file representing the curve.

- void **transform** (const MatrixRT<T>& A)

Performs geometrical modifications.

- void **modCP** (int i, int j, const HPoint_nD<T, N>& p)

< Modifies a control point.

- void **modCPby** (int i, int j, const HPoint_nD<T, N>& p)
< Modifies a control point.
- T& **modU** (int i)
- T **modU** (int i) const
- T& **modV** (int i)
< modifies a knot.
- T **modV** (int i) const
< modifies a knot.
- void **modKnotU** (const Vector<T>& uKnot)
< modifies a knot.
- void **modKnotV** (const Vector<T>& vKnot)
< modifies the U knot vector if uKnot is of a proper size.
- int **movePoint** (T u, T v, const Point_nD<T, N>& delta)
< modifies the U knot vector if uKnot is of a proper size.
- int **movePoint** (const Vector<T>& ur, const Vector<T>& vr, const Vector< Point_nD<T, N> >& D, const Vector_INT& Du, const Vector_INT& Dv)
Moves a point with some constraint.
- int **movePoint** (const Vector<T>& ur, const Vector<T>& vr, const Vector< Point_nD<T, N> >& D, const Vector_INT& Du, const Vector_INT& Dv, const Vector_INT& Dk, const Vector_INT& Dl)
Moves a point with some constraint.
- int **movePoint** (const Vector<T>& ur, const Vector<T>& vr, const Vector< Point_nD<T, N> >& D, const Vector_INT& Du, const Vector_INT& Dv, const Vector_INT& Dk, const Vector_INT& Dl, const BasicArray<Coordinate>& fixCP)
Moves a point with some constraint.
- NurbsSurface<T,N>& **transpose** (void)
Transpose the U and V coordinates of a surface.
- const Vector<T>& **KnotU**
A reference to the U knot vector.

- `const Vector<T>& KnotV`
A reference to the V knot vector.
- `const Matrix< HPoint_nD<T,N> >& CtrlP`
A reference to the control points.
- `const int& degreeU`
A reference to the degree in U of the surface.
- `const int& degreeV`
A reference to the degree in V of the surface.

Protected Members

- `Vector<T> U`
the U knot vector.
- `Vector<T> V`
the V knot vector.
- `Matrix< HPoint_nD<T,N> > P`
The matrix of control points.
- `int degU`
the degree of the surface in U.
- `int degV`
the degree of the surface in V.

Related Functions

(Note that these are not member functions.)

- `int surfMeshParams (const Matrix< Point_nD<T,N> >& Q, Vector<T>& uk, Vector<T>& vl)`
Computes the parameters for global surface interpolation.
- `int surfMeshParams (const Matrix< HPoint_nD<T,N> >& Q, Vector<T>& uk, Vector<T>& vl)`
Computes the parameters for global surface interpolation.

- void **globalSurfInterpXY** (const Matrix< Point_nD<T,N> >& Q, int pU, int pV, NurbsSurface<T,N>& S)
Generates a surface using global interpolation.
 - void **globalSurfInterpXY** (const Matrix< Point_nD<T,N> >& Q, int pU, int pV, NurbsSurface<T,N>& S, const Vector<T>& uk, const Vector<T>& vk)
generates a surface using global interpolation.
 - void **globalSurfApprox** (const Matrix< Point_nD<T,N> >& Q, int pU, int pV, NurbsSurface<T,N>& S, double error)
Generates a surface using global approximation.
 - void **gordonSurface** (NurbsCurveArray<T,N>& IU, NurbsCurveArray<T,N>& IV, const Matrix< Point_nD<T,N> >& intersections, NurbsSurface<T,N>& gS)
Interpolation of a surface from 2 sets of orthogonal curves.
-

Detailed Description

A class to represent a NURBS surface.

The NURBS surface is composed of points in homogenous space. It can have any degree in both the *u* and the *v* direction.

Author(s):

Philippe Lavoie

Date:

4 Oct. 1996

Member Function Documentation

3.18.1 NurbsSurface::NurbsSurface()

Default constructor.

Warning:

The surface is initialized to invalid values. Use a reset or a read function to set them to correct values.

Author(s):

Philippe Lavoie

Date:

24 January, 1997

3.18.2 NurbsSurface::NurbsSurface(const NurbsSurface<T,N>& s)

the copy constructor.

Parameters:*s* - the NurbsSurface<T,N> to copy**Author(s):**

Philippe Lavoie

Date:

24 January, 1997

3.18.3 NurbsSurface::NurbsSurface(int DegU, int DegV, const Vector<T>& Uk, const Vector<T>& Vk, const Matrix< HPoint_nD<T,N> >& Cp)

constructor with points in homogenous space.

Parameters:*DegU* - the degree in the *u* direction*DegV* - the degree in the *v* direction*Uk* - the *u* knot vector*Vk* - the *v* knot vector*Cp* - the matrix of control points in 4D**Author(s):**

Philippe Lavoie

Date:

24 January, 1997

3.18.4 NurbsSurface::NurbsSurface(int DegU, int DegV, Vector<T>& Uk, Vector<T>& Vk, Matrix<Point_nD<T,N> >& Cp, Matrix<T>& W)

constructor with points in 3D.

Parameters:

DegU - the degree of the surface in the U direction
DegV - the degree of the surface in the V direction
Uk - the U knot vector
Vk - the V knot vector
Cp - the matrix of 3D control points
W - the weight value for each control points

Author(s):

Philippe Lavoie

Date:

24 January, 1997

3.18.5 virtual NurbsSurface::~~NurbsSurface() [virtual]

< Empty desctructor.

3.18.6 NurbsSurface<T,N>& NurbsSurface::operator=(const NurbsSurface<T,N>& nS) [virtual]

NurbsSurface<T,N> assignment.

Parameters:

nS - the NURBS surface to copy

Author(s):

Philippe Lavoie

Date:

24 January, 1997

Reimplemented in **NurbsSurfaceSP** and **NurbsSurfaceSP**.

3.18.7 void NurbsSurface::resize(int Pu, int Pv, int DegU, int DegV)

Resize the surface.

Resize the surface. Proper values must be assigned once this function has been called since the resize operator is destructive.

Parameters:

Pu - the number of control points in the U direction

Pv - the number of control points in the V direction

DegU - the degree of the surface in the U direction

DegV - the degree of the surface in the V direction

Author(s):

Philippe Lavoie

Date:

24 January, 1997

3.18.8 void NurbsSurface::resizeKeep(int Pu, int Pv, int DegU, int DegV) [virtual]

Resize the surface while keeping the old values.

Parameters:

Pu - the number of control points in the U direction

Pv - the number of control points in the V direction

DegU - the degree of the surface in the U direction

DegV - the degree of the surface in the V direction

Author(s):

Philippe Lavoie

Date:

24 January, 1997

Reimplemented in **NurbsSurfaceSP** and **HNurbsSurfaceSP**.

3.18.9 HPoint_nD<T,N> NurbsSurface::operator()(T u, T v) const [virtual]

Returns the point on the surface at u,v .

Returns the point on the surface at u,v

Parameters:

u - the u parametric value
v - the v parametric value

Returns:

The homogenous point at *u,v*

Author(s):

Philippe Lavoie

Date:

24 January, 1997

Reimplemented from **ParaSurface**.

Reimplemented in **HNurbsSurface**.

3.18.10 `void NurbsSurface::basisFuns(T u, T v, int spanU, int spanV, Vector<T>& Nu, Vector<T>&Nv) const`

Find the non-zero basis functions in the U and V direction.

Parameters:

u - the u parametric value
v - the v parametric value
spanU - the span of u
spanV - the span of v
Nu - the vector containing the U non-zero basis functions
Nv - the vector containing the V non-zero basis functions

Author(s):

Philippe Lavoie

Date:

24 January, 1997

3.18.11 `void NurbsSurface::basisFunsU(T u, int span, Vector<T>& N) const`

Finds the non-zero basis function in the U direction.

Parameters:

u - the *u* parametric value
span - the span of *u*
N - the vector containing the basis functions

Author(s):

Philippe Lavoie

Date:

24 January, 1997

3.18.12 void NurbsSurface::basisFunsV(T v, int span, Vector<T>& N) const

Finds the non-zero basis function in the *V* direction.

Parameters:

v - the *v* parametric value
span - the span of *v*
N - the vector containing the basis functions

Author(s):

Philippe Lavoie

Date:

24 January, 1997

3.18.13 void NurbsSurface::deriveAt(T u, T v, int d, Matrix< Point_nD<T,N> > &skl) const

Computes the point and the derivatives of degree *d* and below at (u,v) .

Computes the matrix of derivatives at u,v . The value of skl(*k*,*l*) represents the derivative of the surface $S(u,v)$ with respect to *u*, *k* times and to *v*, *l* times.

Parameters:

u - the *u* parametric value
v - the *v* parametric value
d - the derivative is computed up to and including to this value
skl - the matrix containing the derivatives

Author(s):

Philippe Lavoie

Date:

24 January, 1997

Reimplemented from **ParaSurface**.**3.18.14 void NurbsSurface::deriveAtH(T u, T v, int d, Matrix< HPoint_nD<T,N> > &skl) const**computes the point and the derivatives of degree d and below at (u,v) .Computes the matrix of derivatives at u,v . The value of skl(k,l) represents the derivative of the surface $S(u,v)$ with respect to u k times and to v l times.**Parameters:** u - the u parametric value v - the v parametric value d - the derivative is computed up to and including to this value skl - the matrix containing the derivatives**Author(s):**

Philippe Lavoie

Date:

24 January, 1997

Reimplemented from **ParaSurface**.**3.18.15 Point_nD<T,N> NurbsSurface::normal(T u, T v) const**Computes the normal of the surface at (u,v) .**Parameters:** u - the u parametric value v - the v parametric value**Returns:**the normal at (u,v) .**Author(s):**

Philippe Lavoie

Date:

24 January, 1997

3.18.16 void NurbsSurface::globalInterp(const Matrix< Point_nD<T,N> >& Q, int pU, int pV)

Generates a surface using global interpolation.

Parameters:

Q - a matrix of 3D points
pU - the degree of interpolation in the U direction
pV - the degree of interpolation in the V direction

Author(s):

Philippe Lavoie

Date:

24 January, 1997

3.18.17 void NurbsSurface::globalInterpH(const Matrix< HPoint_nD<T,N> >& Q, int pU, int pV)

Generates a surface using global interpolation with homogenous points.

Parameters:

Q - a matrix of 4D points
pU - the degree of interpolation in the U direction
pV - the degree of interpolation in the V direction

Author(s):

Philippe Lavoie

Date:

24 January, 1997

3.18.18 void NurbsSurface::leastSquares(const Matrix< Point_nD<T,N> >& Q, int pU, int pV, int nU, int nV)

generates a surface using global least squares approximation.

Parameters:

Q - a matrix of 3D points
pU - the degree of interpolation in the U direction
pV - the degree of interpolation in the V direction
nU - the number of points in the U direction
nV - the number of points in the V direction

Author(s):

Philippe Lavoie

Date:

24 January, 1997

**3.18.19 int
NurbsSurface::skinV(NurbsCurveArray<T,N>&
ca, int dV)**

Generates a NURBS surface from skinning.

The NURBS surface is generated from skinning. The skinning is performed in the V direction.

Parameters:*ca* - an array of NURBS curves*degV* - the degree to skin in the V direction*surf* - the skinned surface**Returns:**

0 if an error occurs, 1 otherwise

Warning:

The number of curves to skin from must be greater than degV

Author(s):

Philippe Lavoie

Date:

24 January, 1997

**3.18.20 int
NurbsSurface::skinU(NurbsCurveArray<T,N>&
ca, int dU)**

Generates a NURBS surface from skinning.

The NURBS surface is generated from skinning. The skinning is performed in the U direction.

Parameters:*ca* - an array of NURBS curves*degU* - the degree to skin in the U direction

Returns:

0 if an error occurs, 1 otherwise

Warning:

The number of curves to skin from must be greater than degU

Author(s):

Philippe Lavoie

Date:

24 January, 1997

```
3.18.21 void NurbsSurface::sweep(const
NurbsCurve<T,N>& Trj, const
NurbsCurve<T,N>& C, const
NurbsCurve<T,N>& Sv, int K, int useAy, int
invAz)
```

Generates a surface by sweeping a curve along a trajectory.

Sweeping consists of creating a surface by moving a curve profile through a trajectory. The method uses here consists of using skinning of K instances of the curve $C(u)$ along $T(u)$. The K value should be viewed as the minimum number of sections required.

The profile curve $C(u)$ should lie on the xz-plane. It follows the trajectory curve $T(u)$ along its y-axis.

The scaling function is used to modify the shape of the curve profile while it's being swept. It can scale in any of the 4 dimensions to obtain the desired effects on the profile.

See A10.1 on page 476 of the NURBS book for more details about the implementation.

You might have to play with the useAy and invAz variables to obtain a satisfactory result for the sweep operation. This is either because there is an error in the code or because it is the way it is supposed to work.

Parameters:

T - the trajectory of the curve

C - the curve profile to sweep

Sv - a scaling function

K - the minimum number of insertion

$useAy$ - a 0 indicates that rotation angle around the y -axis is not computed, otherwise it is computed. Results are usually better with a value of 0.

invAz - a 1 indicates that the computed rotation around the z -axis should be inversed, a 0 indicates it stays as it is.

Returns:

the NURBS surface representing the sweep of C along T

Warning:

This will not yield a correct value for a closed trajectory curve.

Author(s):

Philippe Lavoie

Date:

25 July, 1997

3.18.22 `void NurbsSurface::sweep(const NurbsCurve<T,N>& Trj, const NurbsCurve<T,N>& C, int K, int useAy,int invAz)`

Generates a surface by sweeping a curve along a trajectory.

Sweeping consists of creating a surface by moving a curve through a trajectory. The method uses here consists of using skinning of K instances of the curve $C(u)$ along $T(u)$. The K value should be viewed as a minimum.

The cross-sectional curve $C(u)$ should lie on the xz -plane. It follows the trajectory curve $T(u)$ along its y -axis.

You might have to play with the *useAy* and *invAz* variables to obtain a satisfactory result for the sweep operation. This is either because there is an error in the code or because it is the way it is supposed to work.

Parameters:

T - the trajectory of the curve

C - the curve to sweep

K - the minimum number of insertion

useAy - a 0 indicates that rotation angle around the y -axis is not computed, otherwise it is computed.

invAz - a 1 indicates that the computed rotation around the z -axis should be inversed, a 0 indicates it stays as it is.

Returns:

the NURBS surface representing the sweep of C along T

Warning:

This will not yield a correct value for a closed trajectory curve.

Author(s):

Philippe Lavoie

Date:

25 July, 1997

3.18.23 `void NurbsSurface::makeFromRevolution(const NurbsCurve<T,N>& profile, const Point_nD<T,N>& S, const Point_nD<T,N>& Tvec, double theta)`

Generates a surface of revolution.

Generates a surface of revolution of a profile around an arbitrary axis (specified by a starting point S and a tangent T) with a certain angle.

The angle is specified in radians.

Parameters:

profile - the curves to rotate around the axis
S - a point on the axis
T - the tangent vector of the rotation axis
theta - the angle of the revolution (in radians)

Warning:

If a point is within a distance of 1e-7 from the axis, it will be assumed to be on the axis.

Author(s):

Philippe Lavoie

Date:

7 October, 1997

3.18.24 `void NurbsSurface::makeSphere(const Point_nD<T,N>& O, T r)`

Generates a sphere.

The NURBS surface is now a sphere of radius r located at O .

Parameters:

O - the location of the center of the sphere
 r - the radius of the sphere

Author(s):

Philippe Lavoie

Date:

8 May, 1998

3.18.25 void NurbsSurface::degreeElevate(int tU, int tV)

Degree elevate the surface in the U and V direction.

Parameters:

tU - elevate the degree of the surface in the u direction by this amount.
 tV - elevate the degree of the surface in the v direction by this amount.

Author(s):

Philippe Lavoie

Date:

24 January, 1997

**3.18.26 void NurbsSurface::degreeElevateU(int t)
[virtual]**

Degree elevate the surface in the U direction.

Parameters:

t - elevate the degree in the u direction by this amount.

Author(s):

Philippe Lavoie

Date:

24 January, 1997

Reimplemented in **NurbsSurfaceSP**.

3.18.27 void NurbsSurface::degreeElevateV(int t) [virtual]

Degree elevate the surface in the V direction.

Parameters:

t - elevate the degree in the v direction by this amount.

Author(s):

Philippe Lavoie

Date:

24 January, 1997

Reimplemented in **NurbsSurfaceSP**.

3.18.28 int NurbsSurface::decompose(NurbsSurfaceArray<T,N>& S) const

Decompose the surface into Bzier patches.

This function decomposes the curve into an array of homogenous Bzier patches.

Parameters:

S - an array of Bzier segments

Returns:

The number of Bzier strips in the u direction.

Author(s):

Philippe Lavoie

Date:

8 October, 1997

3.18.29 void NurbsSurface::findSpan(T u, T v, int& spanU, int& spanV) const

finds the span in the U and V direction.

Finds the span in the U and V direction. The spanU is the index *k* for which the parameter *u* is valid in the $[u_k, u_{k+1}]$ range. The spanV is the index *k* for which the parameter *v* is valid in the $[v_k, v_{k+1}]$ range.

Parameters:

u - find the U span for this parametric value
v - find the V span for this parametric value
spanU - the U span
spanV - the V span

Author(s):

Philippe Lavoie

Date:

24 January, 1997

3.18.30 int NurbsSurface::findSpanU(T u) const

finds the span in the U direction.

Finds the span in the U direction. The span is the index k for which the parameter u is valid in the $[u_k, u_{k+1}]$ range.

Parameters:

u - \rightarrow find the span for this parametric value

Returns:

the span for u

Author(s):

Philippe Lavoie

Date:

24 January, 1997

3.18.31 int NurbsSurface::findSpanV(T v) const

finds the span in the V direction.

Finds the span in the V direction. The span is the index k for which the parameter v is valid in the $[v_k, v_{k+1}]$ range.

Parameters:

v - find the span for this parametric value

Returns:

the span for v

Author(s):

Philippe Lavoie

Date:

24 January, 1997

3.18.32 int NurbsSurface::findMultU(int r) const

Finds the multiplicity of a knot in the U knot.

Parameters:

r - the knot to observe

Returns:

the multiplicity of the knot

Warning:

r must be a valid U knot index

Author(s):

Philippe Lavoie

Date:

24 January, 1997

3.18.33 int NurbsSurface::findMultV(int r) const

finds the multiplicity of a knot in the V knot.

Parameters:

r - the knot to observe

Returns:

the multiplicity of the V knot

Warning:

r must be a valid knot index

Author(s):

Philippe Lavoie

Date:

24 January, 1997

3.18.34 `void NurbsSurface::refineKnots(const
 Vector<T>& nU, const Vector<T>& nV)
 [virtual]`

Refine both knot vectors.

Parameters:

nU - the U knot vector to refine from

nV - the V knot vector to refine from

Author(s):

Philippe Lavoie

Date:

24 January, 1997

Reimplemented in **NurbsSurfaceSP**, **HNurbsSurfaceSP** and **HNurbsSurface**.

3.18.35 `void NurbsSurface::refineKnotU(const
 Vector<T>& X) [virtual]`

Refines the U knot vector.

Parameters:

X - the knot vector to refine from

Author(s):

Philippe Lavoie

Date:

24 January, 1997

Reimplemented in **NurbsSurfaceSP**, **HNurbsSurfaceSP** and **HNurbsSurface**.

3.18.36 `void NurbsSurface::refineKnotV(const
 Vector<T>& X) [virtual]`

Refines the V knot vector.

Parameters:

X - the knot vector to refine from

Author(s):

Philippe Lavoie

Date:

24 January, 1997

Reimplemented in **NurbsSurfaceSP**, **HNurbsSurfaceSP** and **HNurbsSurface**.

```
3.18.37 void NurbsSurface::mergeKnots(const
Vector<T>& nU, const Vector<T>& nV)
[virtual]
```

merges a U and V knot vector with the surface knot vectors.

Parameters:

nU - the U knot vector to merge with

nV - the V knot vector to merge with

Warning:

The nU and nV knot vectors must be compatible with the current vectors

Author(s):

Philippe Lavoie

Date:

24 January, 1997

Reimplemented in **NurbsSurfaceSP** and **HNurbsSurfaceSP**.

```
3.18.38 void NurbsSurface::mergeKnotU(const
      Vector<T>& X) [virtual]
```

merges the U knot vector with another one.

Parameters:

X - a knot vector

Warning:

The knot vector must be compatible with the U knot vector

Author(s):

Philippe Lavoie

Date:

24 January, 1997

Reimplemented in **NurbsSurfaceSP** and **HNurbsSurfaceSP**.

3.18.39 void NurbsSurface::mergeKnotV(const Vector<T>& X) [virtual]

merges the V knot vector with another one.

Parameters:

X - a knot vector

Warning:

The knot vector must be compatible with the V knot vector

Author(s):

Philippe Lavoie

Date:

24 January, 1997

Reimplemented in **NurbsSurfaceSP** and **HNurbsSurfaceSP**.

3.18.40 void NurbsSurface::isoCurveU(T u, NurbsCurve<T,N>& c) const

Generates an iso curve in the U direction.

Generates an iso-parametric curve which goes through the parametric value *u* along the U direction.

Parameters:

u - the U parametric value

c - the iso-parametric curve

Warning:

the parametric value *u* must be in a valid range

Author(s):

Philippe Lavoie

Date:

7 October, 1997

3.18.41 void NurbsSurface::isoCurveV(T v, NurbsCurve<T,N>& c) const

Generates an iso curve in the V direction.

Generates an iso-parametric curve which goes through the parametric value v along the V direction.

Parameters:

v - the V parametric value
c - the iso-parametric curve

Warning:

the parametric value *v* must be in a valid range

Author(s):

Philippe Lavoie

Date:

7 October, 1997

3.18.42 int NurbsSurface::read(const char* filename)

read a surface from a file.

Parameters:

filename - the filename to read from

Returns:

0 if an error occurs, 1 otherwise

Author(s):

Philippe Lavoie

Date:

24 January, 1997

Reimplemented in **HNurbsSurface**.

3.18.43 int NurbsSurface::write(const char* filename) const

Write a surface to a file.

Parameters:

filename - the filename to write to

Returns:

1 on success, 0 on failure

Author(s):

Philippe Lavoie

Date:

24 January, 1997

Reimplemented in **HNurbsSurface**.

3.18.44 int NurbsSurface::read(istream &fin) [virtual]

Read a surface from an input stream.

Parameters:

fin - the input file stream

Returns:

0 if an error occurs, 1 otherwise

Author(s):

Philippe Lavoie

Date:

24 January, 1997

Reimplemented in **NurbsSurfaceSP**, **HNurbsSurfaceSP** and **HNurbsSurface**.

3.18.45 int NurbsSurface::write(ofstream &fout) const

Write a surface to a file stream.

Parameters:

fout - the output filestream to write to.

Returns:

1 on success, 0 on failure

Author(s):

Philippe Lavoie

Date:

24 January, 1997

Reimplemented in **HNurbsSurface**.**3.18.46 ostream& NurbsSurface::print(ostream& o)
const**

Sends the NURBS Surface to ostream for display.

Returns:

the ostream

Author(s):

Philippe Lavoie

Date:

9 November 1998

**3.18.47 int NurbsSurface::writeVRML(const char*
filename, T tolerance, const Color& color) const**

write the NURBS surface to a VRML file.

Writes a VRML file which represents the surface for the parametric space $[uS, uE]$ and $[vS, vE]$. It does not optimize the number of points required to represent the surface.

Parameters:*filename* - the file name for the output VRML file*tolerance* - the tolerance for the tessellation.*color* - the color of the object**Returns:**

1 on success, 0 otherwise

Warning:

The parametric surface must be valid

Author(s):

Philippe Lavoie

Date:

24 January, 1997

3.18.48 `int NurbsSurface::writeVRML(const char*
 filename, const Color& color=whiteColor, int
 Nu=20, int Nv=20) const`

< Calls the **ParaSurface** routine with proper values.

Reimplemented from **ParaSurface**.

3.18.49 `int NurbsSurface::writePOVRAY(ostream&
 povray, int patch_type=1, double flatness=0.01,
 int num_u_steps=8, int num_v_steps=8) const`

Writes a set of povray bicubic patches to the ostream.

Parameters:

patch_type - the type of the patch

flatness - the flatness level

num_u_steps - the minimum number of triangles in the U direction

num_v_steps - the minimum number of triangles in the V direction

povray - the output stream

Returns:

1 on success, 0 on failure.

Warning:

POVRAY only accepts rational spline patches. Thus you can't have a value other than 1.0 for the weights of your surface. A warning message will be generated if this is the case.

POVRAY only deals with surfaces of degree 3. If the surface as a degree below 3 either in the U or V direction it will be elevated to be at 3 in both directions. If the surface as a degree higher than 3, then the function aborts.

Author(s):

Philippe Lavoie

Date:

8 October, 1997

3.18.50 `int NurbsSurface::writePOVRAY(T,
ostream& povray, const Color&
color=Color(250,250,250),int smooth=0 , T
ambient=0.2, T diffuse=0.6) const`

Writes the surface as a mesh of triangles.

Writes the surface as a mesh of triangles. You might have to change the values for the tolerance to get exactly what you're looking for.

Parameters:

povray - the output stream
tolerance - the tolerance when performing the tessellation
color - the color of the object
diffuse - the diffuse factor
ambient - the ambient factor
smooth - flags which indicates if we generate smooth triangles

Returns:

1 on success, 0 on failure

Warning:

It doesn't work very well.

Author(s):

Philippe Lavoie

Date:

8 October, 1997

3.18.51 `int NurbsSurface::writePOVRAY(const
char *filename, const Color& color, const
Point_nD<T,N>& view, const Point_nD<T,N>&
up, int patch_type=1, double flatness=0.01, int
num_u_steps=8, int num_v_steps=8) const`

Writes a set of povray bicubic patches.

Parameters:

patch_type - the type of the patch
flatness - the flatness level
num_u_steps - the minimum number of triangles in the U direction
num_v_steps - the minimum number of triangles in the V direction

Returns:

an ostream containing the definition of the surface

Warning:

POVRAY only accepts rational spline patches. Thus you can't have a value other than 1.0 for the weights of your surface. A warning message will be generated if this is the case.

POVRAY only deals with surfaces of degree 3. If the surface as a degree below 3 either in the U or V direction it will be elevated to be at 3 in both directions. If the surface as a degree higher than 3, then the function aborts.

Author(s):

Philippe Lavoie

Date:

8 October, 1997

3.18.52 `int NurbsSurface::writePOVRAY(T tolerance,
const char *filename, const Color& color, const
Point_nD<T,N>& view, const Point_nD<T,N>&
up, int smooth=0, T ambient=0.2, T
diffuse=0.6) const`

Writes a set of povray bicubic patches.

Parameters:

tolerance - the tolerance when performing the tessellation
filename - the file to write to
color - the color of the object
diffuse - the diffuse factor
ambient - the ambient factor
smooth - flags which indicates if we generate smooth triangles

Returns:

an ostream containing the definition of the surface

Warning:

POVRAY only accepts rational spline patches. Thus you can't have a value other than 1.0 for the weights of your surface. A warning message will be generated if this is the case.

POVRAY only deals with surfaces of degree 3. If the surface as a degree below 3 either in the U or V direction it will be elevated to be at 3 in both directions. If the surface as a degree higher than 3, then the function aborts.

Author(s):

Philippe Lavoie

Date:

8 October, 1997

3.18.53 int NurbsSurface::writeRIB(ostream& rib) const

Writes a NuPatch for render man.

Writes a stream which is compatible with Render Man specifications of a NURBS surface.

The RenderMan ® Interface Procedures and RIB Protocol are: Copyright 1988, 1989, Pixar. All rights reserved. RenderMan ® is a registered trademark of Pixar.

Parameters:

rib - the rib ostream

Returns:

0 if an error occurs, 1 otherwise

Author(s):

Philippe Lavoie

Date:

8 October, 1997

3.18.54 int NurbsSurface::writeRIB(const char* filename, const Color& color, const Point_nD<T,N>& view) const

Writes a NuPatch for render man.

Writes a file which follows the RIB protocol of RenderMan. It generates a file which views the whole object. The material used for rendering is plastic.

The RenderMan ® Interface Procedures and RIB Protocol are: Copyright 1988, 1989, Pixar. All rights reserved. RenderMan ® is a registered trademark of Pixar.

Parameters:

filename - the file to write to
col - the color of the object
view - the view point

Returns:

0 if an error occurs, 1 otherwise

Author(s):

Philippe Lavoie

Date:

8 October, 1997

3.18.55 `void NurbsSurface::tessellate(T tolerance, BasicList<Point_nD<T,N>> &points, BasicList<int> &connect, BasicList<Point_nD<T,N> > *Norm) const`

Generates a list of triangles for a surface.

Generates the list of triangles necessary to represent the surface. This generates the 3D points and the method how to connect them. The connect list is a list of the form ..., j, k, l, -1, ... \$ where the *j*, *k* and *l* value specify the end points of a triangle, the -1 is used to delimit the triangle definitions.

Parameters:

tolerance - the tolerance for the tessellation.
points - the list of points
connect - how the points should be connected

Author(s):

Philippe Lavoie

Date:

8 October, 1997

3.18.56 `int NurbsSurface::writePS(const char*, int nu, int nv, const Point_nD<T,N>& camera, const Point_nD<T,N>& lookAt, int cp=0, T magFact=T(-1), T dash=T(5)) const`

Writes a post-script file representing the curve.

Parameters:

filename - the file to write the postscript file to
nu - the number of lines in the U direction
nv - the number of lines in the V direction
camera - where the camera is
lookAt - where the camera is looking at
plane - where is the projection plane from the camera
cp - a flag indicating if the control points should be drawn, 0 = no and 1 = yes
magFact - a magnification factor, the 2D point of the control points will be magnified by this value. The size is measured in postscript points. If the *magFact* is set to a value smaller or equal to 0, than the program will try to guess a magnification factor such that the curve is large enough to fill the page.
dash - the size of the dash in postscript points . A size smaller or equal to 0 indicates that the line joining the control points is plain.

Returns:

0 if an error occurs, 1 otherwise

Warning:

If the weights of the curve are not all at 1, the result might not be representative of the true NURBS curve.

Author(s):

Philippe Lavoie

Date:

7 October 1998

```

3.18.57  int NurbsSurface::writePSp(const char*,
    int nu, int nv, const Point_nD<T,N>&
    camera, const Point_nD<T,N>& lookAt, const
    Vector< Point_nD<T,N> >&,const Vector<
    Point_nD<T,N> >&, int cp=0,T magFact=0.0,T
    dash=5.0) const
  
```

writes a post-script file representing the curve.

Writes the curve in the postscript format to a file, it also draws the points defined in *points* with their associated vectors if *vector* is used.

Parameters:

filename - the file to write the postscript file to

nu - the number of lines in the U direction
nv - the number of lines in the V direction
camera - where the camera is
lookAt - where the camera is looking at
plane - where is the projection plane from the camera
points - draws these additional points as empty circles
vectors - specify a vector associated with the points (this can be an empty vector)
cp - a flag indicating if the control points should be drawn, 0 = no and 1 = yes
magFact - a magnification factor, the 2D point of the control points will be magnified by this value. The size is measured in postscript points. If the *magFact* is set to a value smaller or equal to 0, than the program will try to guess a magnification factor such that the curve is large enough to fill the page.
dash - the size of the dash in postscript points . A size smaller or equal to 0 indicates that the line joining the control points is plain.

Returns:

0 if an error occurs, 1 otherwise

Warning:

If the weights of the curve are not all at 1, the result might not be representative of the true NURBS curve. If vectors is used, then it must be of the same size as points. If a vector element is (0,0,0) it will not be drawn.

Author(s):

Philippe Lavoie

Date:

7 October 1998

3.18.58 void NurbsSurface::transform(const MatrixRT<T>& A)

Performs geometrical modifications.

Each control points will be modified by a rotation-translation matrix.

Parameters:

A - the rotation-translation matrix

Author(s):

Philippe Lavoie

Date:

22 August 1997

3.18.59 `void NurbsSurface::modCP(int i, int j, const
 HPoint_nD<T, N>& p)`

< Modifies a control point.

3.18.60 `void NurbsSurface::modCPby(int i, int j, const
 HPoint_nD<T, N>& p)`

< Modifies a control point.

3.18.61 `T NurbsSurface::modV(int i) const`

< modifies a knot.

3.18.62 `T NurbsSurface::modV(int i) const`

< modifies a knot.

3.18.63 `void NurbsSurface::modKnotU(const
 Vector<T>& uKnot)`

< modifies a knot.

3.18.64 `void NurbsSurface::modKnotV(const
 Vector<T>& vKnot)`

< modifies the U knot vector if uKnot is of a proper size.

3.18.65 `int NurbsSurface::movePoint(T u, T v, const
 Point_nD<T,N>& delta)`

< modifies the U knot vector if uKnot is of a proper size.

This moves the point $s(u,v)$ by delta.

Parameters:

u - the parameter in the u direction
v - the parameter in the v direction
delta - the displacement of the point at S(u,v)

Returns:

1 if the operation is possible, 0 if the problem is ill defined *i.e.* there isn't enough information to find a unique solution (the system is overdetermined) or that the system has non-independant components.

Author(s):

Philippe Lavoie

Date:

24 January 1997

3.18.66 `int NurbsSurface::movePoint(const Vector<T>& ur, const Vector<T>& vr, const Vector<Point_nD<T,N> >& D, const Vector_INT& Du, const Vector_INT& Dv)`

Moves a point with some constraint.

This will modify the NURBS surface by respecting a certain number of constraints. *ur* and *vr* specifies the parameters on which the constraints should be applied. The constraint are defined by $D_i(u,v)$ which requires 3 vectors to fully qualify. D_i specifies the value of the constraint and *Du* and *Dv* are used to specify on which parameter the constraint is applied.

ur and *vr* should be in an increasing order.

Parameters:

ur - the vector of parameters in the u direction
vr - the vector of parameters in the v direction
D - a vector of the value of $D_i^{(k,l)}(u,v)$
Du - a vector specifying the index of the value of u for D_i
Dv - a vector specifying the index of the value of v for D_i

Returns:

1 if the operation is possible, 0 if the problem is ill defined *i.e.* there isn't enough information to find a unique solution (the system is overdetermined) or that the system has non-independant components.

Warning:

The vectors defining $D_i(u,v)$ should all be of the same size.

Author(s):

Philippe Lavoie

Date:

24 January 1997

3.18.67 `int NurbsSurface::movePoint(const Vector<T>& ur, const Vector<T>& vr, const Vector<Point_nD<T,N> >& D, const Vector_INT& Du, const Vector_INT& Dv, const Vector_INT& Dk, const Vector_INT& Dl)`

Moves a point with some constraint.

This will modify the NURBS surface by respecting a certain number of constraints. u_r and v_r specifies the parameters on which the constraints should be applied. The constraint are defined by $D_i^{\{k,l\}}(u,v)$ which requires 5 vectors to fully qualify. D specifies the value of the constraint and Du and Dv are used to specify on which parameter the constraint is applied and Dk and Dl specify the partial degree of the constraint.

The values in D should be ordered in respect with i, k and l . ur and vr should be in an increasing order.

Parameters:

ur - the vector of parameters in the u direction

vr - the vector of parameters in the v direction

D - a vector of the value of $D_i^{\{k,l\}}(u,v)$

Du - a vector specifying the index of the value of u for D_i

Dv - a vector specifying the index of the value of v for D_i

Dk - a vector specifying the value of k for D_i

Dl - a vector specifying the value of l for D_i

Returns:

1 if the operation is possible, 0 if the problem is ill defined *i.e.* there isn't enough information to find a unique solution (the system is overdetermined) or that the system has non-independant components.

Warning:

The vectors defining $D_i^{\{k,l\}}(u,v)$ should all be of the same size.

Author(s):

Philippe Lavoie

Date:

24 January 1997

3.18.68 `int NurbsSurface::movePoint(const Vector<T>& ur, const Vector<T>& vr, const Vector<Point_nD<T,N> >& D, const Vector_INT& Du, const Vector_INT& Dv, const Vector_INT& Dk, const Vector_INT& Dl, const BasicArray<Coordinate>& fixCP)`

Moves a point with some constraint.

This will modify the NURBS surface by respecting a certain number of constraints. *ur* and *vr* specifies the parameters on which the constraints should be applied. The constraint are defined by $D_i^{(k,l)}(u,v)$ which requires 5 vectors to fully qualify. *D* specifies the value of the constraint and *Du* and *Dv* are used to specify on which parameter the constraint is applied and *Dk* and *Dl* specify the partial degree of the constraint.

A second constraint *fixCP* consists of specifying which control points can not be moved by the routine.

The values in *D* should be ordered in respect with *i*, *k* and *l*. *ur* and *vr* should be in an increasing order.

Parameters:

ur - the vector of parameters in the u direction
vr - the vector of parameters in the v direction
D - a vector of the value of $D_i^{(k,l)}(u,v)$
Du - a vector specifying the index of the value of u for *D_i*
Dv - a vector specifying the index of the value of v for *D_i*
Dk - a vector specifying the value of *k* for *D_i*
Dl - a vector specifying the value of *l* for *D_i*
fixCP - a vector specifying which control points can *not* be modified.

Returns:

1 if the operation is possible, 0 if the problem is ill defined *i.e.* there isn't enough information to find a unique solution (the system is overdetermined) or that the system has non-independant components.

Warning:

The vectors defining $D_i^{(k,l)}(u,v)$ should all be of the same size.

Author(s):

Philippe Lavoie

Date:

24 January 1997

3.18.69 NurbsSurface<T,N>& NurbsSurface::transpose(void)

Transpose the U and V coordinates of a surface.

Transpose the U and V coordinates of a surface. After this operation the (u,v) points correspond to (v,u) .

Returns:

A reference to itself

Warning:

This is not completely tested

Author(s):

Philippe Lavoie

Date:

24 January, 1997

3.18.70 int surfMeshParams(const Matrix<Point_nD<T,N> >& Q, Vector<T>& uk, Vector<T>& vl)

Computes the parameters for global surface interpolation.

Computes the parameters for global surface interpolation. For more information, see A9.3 on p377 on the NURBS book.

Parameters:

Q - the matrix of 3D points

uk - the knot coefficients in the U direction

vl - the knot coefficients in the V direction

Returns:

0 if an error occurs, 1 otherwise

Warning:

Author(s):

Philippe Lavoie

Date:

24 January, 1997

3.18.71 `int surfMeshParams(const Matrix<HPoint_nD<T,N> >& Q, Vector<T>& uk, Vector<T>& vl)`

Computes the parameters for global surface interpolation.

Computes the parameters for global surface interpolation. For more information, see A9.3 on p377 on the NURBS book.

Parameters:

Q - the matrix of 3D points
uk - the knot coefficients in the U direction
vl - the knot coefficients in the V direction

Returns:

0 if an error occurs, 1 otherwise

Author(s):

Philippe Lavoie

Date:

24 January, 1997

3.18.72 `void globalSurfInterpXY(const Matrix<Point_nD<T,N> >& Q, int pU, int pV, NurbsSurface<T,N>& S)`

Generates a surface using global interpolation.

Generates a NURBS surface using global interpolation. The data points are assumed to be part of grided points in x-y. *i.e.* the original data set as points covering the xy plane at regular *x* and *y* intervals with only the *z* being a free variable

Parameters:

Q - a matrix of 3D points
pU - the degree of interpolation in the U direction
pV - the degree of interpolation in the V direction
S - the interpolated surface

Warning:

Q(0,0) in x-y should be the smallest corner and *Q*(*Q*.rows-1,*Q*.cols-1) the biggest.

Author(s):

Philippe Lavoie

Date:

24 January, 1997

3.18.73 void globalSurfInterpXY(const Matrix< Point_nD<T,N> >& Q, int pU, int pV, NurbsSurface<T,N>& S, const Vector<T>& uk, const Vector<T>& vk)

generates a surface using global interpolation.

Generates a NURBS surface using global interpolation. The data points are assumed to be part of grided points in x-y. i.e. the original data set as points covering the xy plane at regular x and y intervals with only the z being a free variable

Parameters:

Q - a matrix of 3D points
 pU - the degree of interpolation in the U direction
 pV - the degree of interpolation in the V direction
 S - the interpolated surface

Warning:

$Q(0,0)$ in x-y should be the smallest corner and $Q(Q.rows-1,Q.cols-1)$ the biggest.

Author(s):

Philippe Lavoie

Date:

24 January, 1997

3.18.74 void globalSurfApprox(const Matrix< Point_nD<T,N> >& Q, int pU, int pV, NurbsSurface<T,N>& S, double error)

Generates a surface using global approximation.

Parameters:

Q - a matrix of 3D points

pU - the degree of interpolation in the U direction
pV - the degree of interpolation in the V direction
S - the interpolated surface

Warning:

This routine is still in a research phase

Author(s):

Philippe Lavoie

Date:

24 January, 1997

3.18.75 `void gordonSurface(NurbsCurveArray<T,N>& lU, NurbsCurveArray<T,N>& lV, const Matrix< Point_nD<T,N> >& intersections, NurbsSurface<T,N>& gS)`

Interpolation of a surface from 2 sets of orthogonal curves.

Interpolation of a surface from 2 sets of orthogonal curves. See A10.3 at page 494 on the NURBS book for more details about the implementation.

Parameters:

lU - an array of curves in the U direction
lV - an array of curves in the V direction
intersections - a matrix of 3D points specifying where the the curves intersect
gS - the Gordon surface

Author(s):

Philippe Lavoie

Date:

24 January, 1997

Member Data Documentation

3.18.76 `const Vector<T>& NurbsSurface::KnotU`

A reference to the U knot vector.

3.18.77 `const Vector<T>& NurbsSurface::KnotV`

A reference to the V knot vector.

3.18.78 `const Matrix< HPoint_nD<T,N> >& NurbsSurface::CtrlP`

A reference to the control points.

3.18.79 `const int& NurbsSurface::degreeU`

A reference to the degree in U of the surface.

3.18.80 `const int& NurbsSurface::degreeV`

A reference to the degree in V of the surface.

3.18.81 `Vector<T> NurbsSurface::U [protected]`

the U knot vector.

3.18.82 `Vector<T> NurbsSurface::V [protected]`

the V knot vector.

3.18.83 `Matrix< HPoint_nD<T,N> > NurbsSurface::P [protected]`

The matrix of control points.

3.18.84 `int NurbsSurface::degU [protected]`

the degree of the surface in U.

3.18.85 `int NurbsSurface::degV [protected]`

the degree of the surface in V.

The documentation for this class was generated from the following files:

- nurbsS.hh
- nurbsS_sp.hh
- nurbsS.cc

3.19 NurbsSurfaceArray Class Reference

An array of `NurbsSurface`.

```
#include <nurbs/nurbsS.hh>
```

Public Members

- **NurbsSurfaceArray** (`NurbsSurface<T, N>* Sa, int size`)
Constructor from a pointer to an array of curves.
- **NurbsSurfaceArray** ()
- virtual `~NurbsSurfaceArray` ()
- virtual `NurbsSurface<T,N>& operator[]` (int i)
< the ith surface.
- virtual `NurbsSurface<T,N> operator[]` (int i) const
< the ith surface.
- virtual void **resize** (int s)
< the ith surface.
- void **init** (`NurbsSurface<T, N>* Sa, int size`)
Initialize the array of curves with a vector of nurbs curve.
- `NurbsSurfaceArray<T,N>& operator=` (const `NurbsSurfaceArray<T, N>& Sa`)
Copy one surface array to another.
- const int& **n**
a reference to the size of the array.

Protected Members

- int **size**
the number of NURBS curves in the array.
- int **rsize**
the number of space allocated for the array.
- `NurbsSurface<T,N>** S`
An array of pointers to NURBS curves.

Detailed Description

An array of **NurbsSurface**.

This class represents an array of **NurbsSurface**.

Author(s):

Philippe Lavoie

Date:

4 Oct. 1996

Member Function Documentation

3.19.1

NurbsSurfaceArray::NurbsSurfaceArray(NurbsSurface<T,N>*
Sa, int s)

Constructor from a pointer to an array of curves.

Author(s):

Philippe Lavoie

Date:

24 January 1997

3.19.2 virtual NurbsSurface<T,N>
NurbsSurfaceArray::operator[](int i) const
[virtual]

< the ith surface.

3.19.3 virtual NurbsSurface<T,N>
NurbsSurfaceArray::operator[](int i) const
[virtual]

< the ith surface.

3.19.4 void NurbsSurfaceArray::resize(int size) [virtual]

< the ith surface.

Parameters:

size - the new size

Author(s):

Philippe Lavoie

Date:

24 January 1997

3.19.5 void NurbsSurfaceArray::init(NurbsSurface<T,N>* Sa,int size)

Initialize the array of curves with a vector of nurbs curve.

Parameters:

ca - a pointer to a vector of NURBS curve

size - the size of the array

Author(s):

Philippe Lavoie

Date:

24 January 1997

3.19.6 NurbsSurfaceArray<T,N>& NurbsSurfaceArray::operator=(const NurbsSurfaceArray<T,N>& Sa)

Copy one surface array to another.

Parameters:

S - the array to copy

Returns:

a reference to itself

24 January 1997

Generated on Wed Dec 16 13:26:23 1998 for NURBS by Doxygen, written by Dimitri van Heesch ©1997-1998

3.20 NurbsSurfaceGL Class Reference

A NURBS surface class for OpenGL.

`#include <nurbs/nurbsGL.hh>`

Inherits **NurbsGL**.

Public Members

- **NurbsSurfaceGL** ()
- **NurbsSurfaceGL** (const NurbsSurface& nS)
- **NurbsSurfaceGL** (const NurbsSurfaceGL& nS)
- **NurbsSurfaceGL** (int DegU, int DegV, const Vector<float>& Uk, const Vector<float>& Vk, const Matrix<HPoint3Df>& Cp)
- **NurbsSurfaceGL** (int DegU, int DegV, Vector<float>& Uk, Vector<float>& Vk, Matrix<Point3Df>& Cp, Matrix<float>& W)
- **~NurbsSurfaceGL** ()
- void **gluNurbs** () const
creates a nurbs surface for OpenGL.
- void **point** (float &u, float &v, int pSize, const Color& colorP, int cp_flag=0) const
draws a point at the location C(u).
- virtual NurbsSurfaceGL& **operator=** (const NurbsSurfaceGL& a)
Copies another Nurbs Curve GL.
- virtual NurbsSurfaceGL& **operator=** (const NurbsSurface& a)
Copies another Nurbs Curve GL.
- void **resetBoundingBox** ()
resets the minP and maxP values of bbox.
- void **resetCPoints** ()
Reset the control point information.
- void **resetPolygon** ()
- void **resetKnots** ()
reset the knots information.
- int **read** (ifstream &fin)
Reads the information from a stream.

- int **write** (ofstream &fout) const
Writes a NurbsCurveGL to an output stream.
- int **writeRIB** (ofstream &fout) const
- int **writePOVRAY** (ofstream &fout) const
- ObjectGL* **copy** ()
- void **applyTransform** ()
apply the local transformation to the surface.
- void **modifyPoint** (float u, float v, float dx, float dy, float dz)
Modifies a point on the surface.
- void **setImage** (GLubyte *img, GLint w, GLint h)
attach an image to a surface.
- void **setSym** (int set, int uDir, float x, float y, float z, float w)
Sets the symmetry for the control points.
- list<NurbsCurve_2Df*> **trimmedCurves**

Protected Members

- GLubyte* **image**
 - GLint **imgW**
 - GLint **imgH**
-

Detailed Description

A NURBS surface class for OpenGL.

Author(s):

Philippe Lavoie

Date:

23 September 1997

Member Function Documentation

3.20.1 void NurbsSurfaceGL::gluNurbs() const

creates a nurbs surface for OpenGL.

This function calls between a gluBeginSurface/gluEndSurface the proper functions to generate a NURBS surface.

Author(s):

Philippe Lavoie

Date:

23 September 1997

Reimplemented from **NurbsGL**.

3.20.2 void NurbsSurfaceGL::point(float &u, float &v, int pSize,const Color& colorP, int cp_flag) const

draws a point at the location $C(u)$.

This function calls between a glBegin/glEnd the proper functions to represent the point which is at $S(u,v)$ on the surface.

Parameters:

u - the U parametric value

v - the V parametric value

$psize$ - the size of the control points

$colorP$ - the color of the control points

Author(s):

Philippe Lavoie

Date:

23 September 1997

Reimplemented from **NurbsGL**.

3.20.3 NurbsSurfaceGL& NurbsSurfaceGL::operator=(const NurbsSurfaceGL &a) [virtual]

Copies another Nurbs Curve GL.

Parameters:

a - the Nurbs curve to copy

Author(s):

Philippe Lavoie

Date:

6 November 1997

3.20.4 NurbsSurfaceGL& NurbsSurfaceGL::operator=(const NurbsSurface &a) [virtual]

Copies another Nurbs Curve GL.

Parameters:

a - the Nurbs curve to copy

Author(s):

Philippe Lavoie

Date:

6 November 1997

3.20.5 void NurbsSurfaceGL::resetBoundingBox()

resets the minP and maxP values of bbox.

Resets the minP and maxP values for the bounding box.

Warning:

Calling this function without a proper surface initialized might result in strange results.

Author(s):

Philippe Lavoie

Date:

23 September 1997

Reimplemented from **NurbsGL**.

3.20.6 void NurbsSurfaceGL::resetCPoints()

Reset the control point information.

Reset the control point information stored in cpoints.

Author(s):

Philippe Lavoie

Date:

23 September 1997

Reimplemented from **NurbsGL**.

3.20.7 void NurbsSurfaceGL::resetKnots()

reset the knots information.

Reset the knots information stored in knots.

Author(s):

Philippe Lavoie

Date:

23 September 1997

Reimplemented from **NurbsGL**.

3.20.8 int NurbsSurfaceGL::read(istream &fin)

Reads the information from a stream.

Parameters:

fin - the input stream

Returns:

1 on sucess, 0 on failure

Author(s):

Philippe Lavoie

Date:

19 June 1998

Reimplemented from **ObjectGL**.

3.20.9 int NurbsSurfaceGL::write(ofstream &fout) const

Writes a **NurbsCurveGL** to an output stream.

Parameters:

fout - the output stream

Returns:

0 if an error occurs, 1 otherwise

Author(s):

Philippe Lavoie

Date:

19 June 1998

Reimplemented from **ObjectGL**.

3.20.10 void NurbsSurfaceGL::applyTransform()

apply the local transformation to the surface.

Apply the local transformation to the surface. This is necessary if you want to get the proper position for the control points before doing anymore processing on them.

Author(s):

Philippe Lavoie

Date:

23 September 1997

Reimplemented from **ObjectGL**.

3.20.11 void NurbsSurfaceGL::modifyPoint(float u, float v, float dx, float dy, float dz)

Modifies a point on the surface.

Parameters:

u - the u parametric value

v - the v parametric value

dx - the delta value in the *x*-axis direction

dy - the delta value in the *y*-axis direction

dz - the delta value in the *z*-axis direction

Author(s):

Philippe Lavoie

Date:

7 November 1997

Reimplemented from **NurbsGL**.**3.20.12 void NurbsSurfaceGL::setImage(GLubyte *img, GLint w, GLint h)**

attach an image to a surface.

Attach an image to a surface. The image must contain the red, green and blue component in successive GLubyte of data. The image data must be sent row wise.

OpenGL only uses images which are 2^n pixels in width or height. If the image is not of the proper size, it will be padded to be acceptable by OpenGL. If you want to center the image, you have to provide an already centered image.

The image data is reversed so that it shows upside up when mapped to a NURBS surface. The img data is copied into a new vector so it is safe to delete it once it has been passed to this routine.

Parameters:*img* - a pointer to the image data*w* - the width of the image*h* - the height of the image**Author(s):**

Philippe Lavoie

Date:

5 February 1998

3.20.13 void NurbsSurfaceGL::setSym(int set, int uDir, float x, float y, float z, float w)

Sets the symmetry for the control points.

Parameters:*true* - 1 if it should be in symmetrical mode

Author(s):

Philippe Lavoie

Date:

2 July 1998

Reimplemented from **NurbsGL**.

The documentation for this class was generated from the following files:

- nurbsGL.hh
- nurbsGL.cc

3.21 NurbsSurfaceSP Class Reference

A NURBS surface with surface point.

`#include <nurbs/nurbsS_sp.hh>`

Inherits **NurbsSurface**.

Public Members

- **NurbsSurfaceSP** ()
- **NurbsSurfaceSP** (const NurbsSurface<T, N>& nS)
- **NurbsSurfaceSP** (const NurbsSurfaceSP<T, N>& nS)
- **NurbsSurfaceSP** (int DegU, int DegV, const Vector<T>& Uk, const Vector<T>& Vk, const Matrix< HPoint_nD<T, N> >& Cp)
- **NurbsSurfaceSP** (int DegU, int DegV, Vector<T>& Uk, Vector<T>& Vk, Matrix< Point_nD<T, N> >& Cp, Matrix<T>& W)
- virtual NurbsSurface<T,N>& **operator=** (const NurbsSurface<T, N>& a)
- virtual NurbsSurfaceSP<T,N>& **operator=** (const NurbsSurfaceSP<T, N>& a)
- virtual void **resizeKeep** (int Pu, int Pv, int DegU, int DegV)
- virtual void **refineKnots** (const Vector<T>& nU, const Vector<T>& nV)

Refine both knot vectors.

- virtual void **refineKnotU** (const Vector<T>& X)
- virtual void **refineKnotV** (const Vector<T>& X)

Refines the V knot vector.

- virtual void **mergeKnots** (const Vector<T>& nU, const Vector<T>& nV)
- virtual void **mergeKnotU** (const Vector<T>& X)
- virtual void **mergeKnotV** (const Vector<T>& X)
- virtual int **read** (ifstream &fin)
- virtual void **degreeElevateU** (int tU)
- virtual void **degreeElevateV** (int tV)
- NurbsSurfaceSP<T,N> **generateParallel** (T d) const

Generate a parallel surface.

- void **modSurfCPby** (int i, int j, const HPoint_nD<T, N>& a)

< Moves a surface point by a value.

- void **modSurfCP** (int i, int j, const HPoint_nD<T, N>& a)

< Moves a surface point to a value.

- void **modOnlySurfCPby** (int i, int j, const HPoint.nD<T, N>& a)
Move the surface point only.
- void **modOnlySurfCP** (int i, int j, const HPoint.nD<T, N>& a)
< Changes only the the surface point near a control point, the other surface point will not be moved.
- T **maxAtUV** (int i, int j) const
< The maximal basis function for the control point i,j.
- T **maxAtU** (int i) const
< Where is the maximal basis function in U for the control points in row i.
- T **maxAtV** (int i) const
< Where is the maximal basis function in U for the control points in column i.
- HPoint.nD<T,N> **surfP** (int i, int j) const
< the surface point for the control point at i,j.
- void **updateMaxUV** ()
< Updates both the maxU and maxV values.
- void **updateMaxU** ()
Updates the basis value for the U direction.
- void **updateMaxV** ()
Updates the basis value for the V direction.
- int **okMax** ()
< Is the maximal value in a valid range ?.

Protected Members

- Vector<T> **maxU**
The vector of maximal basis function value in U.
- Vector<T> **maxV**
The vector of maximal basis function value in V.
- Vector<T> **maxAtU_**

The vector identifying where is the maximal basis function in U .

- Vector<T> **maxAtV_**

The vector identifying where is the maximal basis function in V .

Detailed Description

A NURBS surface with surface point.

A Nurbs surface with surface point manipulators. This allows someone to modify the point on a surface for which a control point has maximal influence over it. This might provide a more intuitive method to modify a surface.

Author(s):

Philippe Lavoie

Date:

8 May, 1998

Member Function Documentation

3.21.1 virtual void NurbsSurfaceSP::refineKnots(const Vector<T>& nU, const Vector<T>& nV) [virtual]

Refine both knot vectors.

Parameters:

nU - the U knot vector to refine from

nV - the V knot vector to refine from

Author(s):

Philippe Lavoie

Date:

24 January, 1997

Reimplemented from **NurbsSurface**.

3.21.2 virtual void NurbsSurfaceSP::refineKnotV(const Vector<T>& X) [virtual]

Refines the V knot vector.

Parameters:

X - the knot vector to refine from

Author(s):

Philippe Lavoie

Date:

24 January, 1997

Reimplemented from **NurbsSurface**.

3.21.3 NurbsSurfaceSP<T,N> NurbsSurfaceSP::generateParallel(T d) const

Generate a parallel surface.

Generates an offset surface from this surface. An offset surface is a surface which has its surface parallel to an other one. There is a distance of d between the two parallel surfaces.

The algorithm used is very naive. It generates a surface such that a point $s_2(u,v) = s(u,v) + d \cdot n(u,v)$ where $s_2(u,v)$ is the point on the parallel surface at (u,v) , $s(u,v)$ is the point on the original surface at (u,v) , d is the offset between the two and $n(u,v)$ is the normal on the surface at the point (u,v) .

Parameters:

d - the distance between the surface and its offset the parallel surface.

Author(s):

Philippe Lavoie

Date:

8 May, 1998

3.21.4 void NurbsSurfaceSP::modSurfCPby(int i, int j, const HPoint_nD<T, N>& a)

< Moves a surface point by a value.

3.21.5 void NurbsSurfaceSP::modSurfCP(int i, int j, const HPoint_nD<T, N>& a)

< Moves a surface point to a value.

3.21.6 void NurbsSurfaceSP::modOnlySurfCPby(int i, int j, const HPoint_nD<T,N>& a)

Move the surface point only.

Moves only the specified surface point. The other surface points normally affected by moving this point are {\em not} moved.

The point a is in the 4D homogenous space, but only the x,y,z value are used. The weight is not moved by this function.

Parameters:

- i* - the row of the surface point to move
- j* - the column of the surface point to move
- a* - move that surface point by that amount.

Author(s):

Philippe Lavoie

Date:

7 June, 1998

3.21.7 void NurbsSurfaceSP::modOnlySurfCP(int i, int j, const HPoint_nD<T, N>& a)

< Changes only the the surface point near a control point, the other surface point will not be moved.

3.21.8 T NurbsSurfaceSP::maxAtUV(int i, int j) const

< The maximal basis function for the control point i,j.

3.21.9 T NurbsSurfaceSP::maxAtU(int i) const

< Where is the maximal basis function in U for the control points in row i.

3.21.10 `T NurbsSurfaceSP::maxAtV(int i) const`

< Where is the maximal basis function in U for the control points in colum i.

3.21.11 `HPoint_nD<T,N> NurbsSurfaceSP::surfP(int i,
int j) const`

< the surface point for the control point at i,j.

3.21.12 `void NurbsSurfaceSP::updateMaxUV()`

< Updates both the maxU and maxV values.

3.21.13 `void NurbsSurfaceSP::updateMaxU()`

Updates the basis value for the U direction.

Updates the basis value at which a control point has maximal influence. It also finds where the control point has maximal influence.

Warning:

The degree in U of the surface must be of 3 or less.

Author(s):

Philippe Lavoie

Date:

8 May, 1998

3.21.14 `void NurbsSurfaceSP::updateMaxV()`

Updates the basis value for the V direction.

Updates the basis value at which a control point has maximal influence. It also finds where the control point has maximal influence.

Warning:

The degree in V of the surface must be of 3 or less.

Author(s):

Philippe Lavoie

Date:

8 May, 1998

3.21.15 `int NurbsSurfaceSP::okMax()`

< Is the maximal value in a valid range ?.

Member Data Documentation**3.21.16** `Vector<T> NurbsSurfaceSP::maxU` [protected]

The vector of maximal basis function value in U.

3.21.17 `Vector<T> NurbsSurfaceSP::maxV` [protected]

The vector of maximal basis function value in V.

3.21.18 `Vector<T> NurbsSurfaceSP::maxAtU_`
[protected]

The vector identifying where is the maximal basis function in U.

3.21.19 `Vector<T> NurbsSurfaceSP::maxAtV_`
[protected]

The vector identifying where is the maximal basis function in V.

The documentation for this class was generated from the following files:

- nurbsS_sp.hh
- nurbsS_sp.cc

3.22 OpenGL Class Reference

The base class for OpenGL objects.

```
#include <nurbs/nurbsGL.hh>
```

Inherited by **BoundingBoxGL**, **CPointGL**, **KnotGL**, **NurbsCpolygonGL**, **NurbsGL**, **NurbsSpolygonGL**, **ObjectListGL**, **ObjectRefGL**, **PointGL** and **PointListGL**.

Public Members

- enum **ObjectCategory** { **badType** , **nurbsType** , **pointType** , **vectorType** , **listType** }
- enum **ObjectType** { **badObject** , **curveObject** , **surfaceObject** , **pointObject** , **cpointObject** , **cpointPolygonObject** , **bboxObject** , **vectorObject** , **listObject** , **hSurfObject** , **hcpointObject** , **pointListObject** , **spointObject** }
- enum **ObjectCategory** { **badType** , **nurbsType** , **pointType** , **vectorType** , **listType** }
- enum **ObjectType** { **badObject** , **curveObject** , **surfaceObject** , **pointObject** , **cpointObject** , **cpointPolygonObject** , **bboxObject** , **vectorObject** , **listObject** , **hSurfObject** , **hcpointObject** , **pointListObject** , **spointObject** }
- **ObjectGL** ()
The basic constructor.
- virtual **~ObjectGL** ()
The destructor.
- **ObjectGL& operator=** (const **ObjectGL&** a)
*Copies another **ObjectGL**.*
- void **setObjectColor** (const **Color&** c)
- void **setSelectColor** (const **Color&** c)
- void **setCurrentColor** (const **Color&** c)
- void **glSelectColor** () const
- void **glObjectColor** () const
- void **glCurrentColor** () const
- void **hideObject** ()
- void **selectObject** ()
- void **viewObject** ()
- void **currentObject** ()
- virtual void **glColor** () const
- void **glColor** (const **Color&** c) const

- virtual void **glObject** () const = 0
- virtual void **glTransform** () const
Performs the local transformation.
- virtual void **display** () const
- virtual void **displayList** ()
- virtual void **displayName** ()
- virtual void **glNewList** ()
generates a default call list.
- virtual OpenGL*& **previous** ()
- virtual OpenGL*& **next** ()
- virtual OpenGL* **previous** () const
- virtual OpenGL* **next** () const
- OpenGLState **getState** () const
- virtual void **select** ()
- virtual void **deselect** ()
- int **isSelected** () const
- virtual void **activate** ()
- virtual void **deactivate** ()
- int **isActive** () const
- virtual OpenGL* **copy** ()
- virtual void **applyTransform** ()
- virtual int **read** (const char* filename)
Reads the information from a stream.
- virtual int **write** (const char* filename) const
*Writes a **OpenGL** to a file.*
- virtual int **writeRIB** (const char* filename) const
- virtual int **writePOVRAY** (const char* filename) const
- virtual int **read** (ifstream &fin)
Reads the information from a stream.
- virtual int **write** (ofstream &fout) const
*Writes a **OpenGL** to a stream.*
- virtual int **writeRIB** (ofstream &fout) const
- virtual int **writePOVRAY** (ofstream &fout) const
- void **setName** (const char* n)
Sets the name of the object.
- char* **name** () const
- char* **typeName** () const
Returns the name of the type of the class.

- **OpenGL ()**
The basic constructor.
- **virtual ~OpenGL ()**
The destructor.
- **OpenGL& operator= (const OpenGL& a)**
Copies another OpenGL.
- **void setObjectColor (const Color& c)**
- **void setSelectColor (const Color& c)**
- **void setCurrentColor (const Color& c)**
- **void glSelectColor () const**
- **void glObjectColor () const**
- **void glCurrentColor () const**
- **void hideObject ()**
- **void selectObject ()**
- **void viewObject ()**
- **void currentObject ()**
- **virtual void glColor () const**
- **void glColor (const Color& c) const**
- **virtual void glObject () const = 0**
- **virtual void glTransform () const**
Performs the local transformation.
- **virtual void display () const**
- **virtual void displayList ()**
- **virtual void displayName ()**
- **virtual void glNewList ()**
generates a default call list.
- **virtual OpenGL*& previous ()**
- **virtual OpenGL*& next ()**
- **virtual OpenGL* previous () const**
- **virtual OpenGL* next () const**
- **OpenGLState getState () const**
- **virtual void select ()**
- **virtual void deselect ()**
- **int isSelected () const**
- **virtual void activate ()**
- **virtual void deactivate ()**
- **int isActive () const**
- **virtual OpenGL* copy ()**
- **virtual void applyTransform ()**

- virtual int **read** (const char* filename)
Reads the information from a stream.
- virtual int **write** (const char* filename) const
*Writes a **OpenGL** to a file.*
- virtual int **writeRIB** (const char* filename) const
- virtual int **writePOVRAY** (const char* filename) const
- virtual int **read** (ifstream &fin)
Reads the information from a stream.
- virtual int **write** (ofstream &fout) const
*Writes a **OpenGL** to a stream.*
- virtual int **writeRIB** (ofstream &fout) const
- virtual int **writePOVRAY** (ofstream &fout) const
- void **setName** (const char* n)
Sets the name of the object.
- char* **name** () const
- char* **typeName** () const
Returns the name of the type of the class.
- Color **objectColor**
- Color **selectColor**
- Color **currentColor**
- GLfloat* **materialColor**
- ObjectType **type**
- ObjectCategory **category**
- GLfloat **tx**
- GLfloat **ty**
- GLfloat **tz**
- GLfloat **rx**
- GLfloat **ry**
- GLfloat **rz**
- GLfloat **sx**
- GLfloat **sy**
- GLfloat **sz**
- int **callListId**

Protected Members

- int **selected**
 - int **active**
 - ObjectGLState **state**
 - ObjectGL* **prev_**
 - ObjectGL * **next_**
 - char* **name_**
-

Detailed Description

The base class for OpenGL objects.

This is the base virtual class for objects which can be displayed using OpenGL.

The class contains informations regarding the color of the object and the state of the object (selected, active, displayed, hidden). It also associates a color for each state.

Information about local transformations which should be applied to the objects are also given.

Author(s):

Philippe Lavoie

Date:

28 September 1997

Member Function Documentation

3.22.1 ObjectGL::ObjectGL()

The basic constructor.

It sets default values for the state and the colors. The default values are defined by the default global variables: `objectColorDefault`, `selectColorDefault`, `currentColorDefault` and `objectStateDefault`.

Author(s):

Philippe Lavoie

Date:

20 September 1997

3.22.2 OpenGL::~~OpenGL() [virtual]

The destructor.

Author(s):

Philippe Lavoie

Date:

30 September 1997

3.22.3 OpenGL& OpenGL::operator=(const OpenGL &a)

Copies another OpenGL.

Copies another OpenGL. This will not copy the prev and next pointers.

Parameters:

a - the object to copy

Author(s):

Philippe Lavoie

Date:

6 November 1997

3.22.4 void OpenGL::glTransform() const [virtual]

Performs the local transformation.

Performs the local transformation in this order: scaling, translation, rotation in x, rotation in y and rotation in z.

Author(s):

Philippe Lavoie

Date:

20 September 1997

3.22.5 void OpenGL::glNewList() [virtual]

generates a default call list.

Generates a default call list. It generates a call list from the glObject() call using \verb.GL_COMPILE..

Returns:

The call list identification number for the object if the call was succesfull, 0 otherwise.

Author(s):

Philippe Lavoie

Date:

23 September 1997

3.22.6 int OpenGL::read(const char* filename) [virtual]

Reads the information from a stream.

Parameters:

filename - the input file

Returns:

1 on sucess, 0 on failure

Author(s):

Philippe Lavoie

Date:

19 June 1998

Reimplemented in **PointListGL**, **ObjectListGL** and **HNurbsSurfaceGL**.

3.22.7 int OpenGL::write(const char* filename) const [virtual]

Writes a OpenGL to a file.

Parameters:

filename - the filename to write to.

Returns:

0 if an error occurs, 1 otherwise

Author(s):

Philippe Lavoie

Date:

19 June 1998

Reimplemented in **PointListGL**, **ObjectListGL** and **HNurbsSurfaceGL**.

3.22.8 int OpenGL::read(istream &fin) [virtual]

Reads the information from a stream.

Parameters:

fin - the input stream

Returns:

1 on success, 0 on failure

Author(s):

Philippe Lavoie

Date:

19 June 1998

Reimplemented in **PointListGL**, **NurbsSurfaceGL**, **NurbsCurveGL** and **HNurbsSurfaceGL**.

3.22.9 int OpenGL::write(ofstream &fout) const [virtual]

Writes a OpenGL to a stream.

Parameters:

fout - the output stream

Returns:

0 if an error occurs, 1 otherwise

Author(s):

Philippe Lavoie

Date:

19 June 1998

Reimplemented in **PointListGL**, **NurbsSurfaceGL**, **NurbsCurveGL** and **HNurbsSurfaceGL**.

3.22.10 void ObjectGL::setName(const char* n)

Sets the name of the object.

Sets the name of the object. The content of n is copied to the name.

Parameters:

n - the name of the object

Author(s):

Philippe Lavoie

Date:

20 September 1997

3.22.11 char* ObjectGL::typeName() const

Returns the name of the type of the class.

Returns:

the name of the type of the class, 0 if the type is unknown

Author(s):

Philippe Lavoie

Date:

20 September 1997

3.22.12 ObjectGL::ObjectGL()

The basic constructor.

It sets default values for the state and the colors. The default values are defined by the default global variables: `objectColorDefault`, `selectColorDefault`, `currentColorDefault` and `objectStateDefault`.

Author(s):

Philippe Lavoie

Date:

20 September 1997

3.22.13 OpenGL::~~OpenGL() [virtual]

The destructor.

Author(s):

Philippe Lavoie

Date:

30 September 1997

**3.22.14 OpenGL& OpenGL::operator=(const
 OpenGL &a)**

Copies another OpenGL.

Copies another OpenGL. This will not copy the prev and next pointers.

Parameters:

a - the object to copy

Author(s):

Philippe Lavoie

Date:

6 November 1997

3.22.15 void OpenGL::glTransform() const [virtual]

Performs the local transformation.

Performs the local transformation in this order: scaling, translation, rotation in x, rotation in y and rotation in z.

Author(s):

Philippe Lavoie

Date:

20 September 1997

3.22.16 void OpenGL::glNewList() [virtual]

generates a default call list.

Generates a default call list. It generates a call list from the glObject() call using \verb{GL_COMPILE}..

Returns:

The call list identification number for the object if the call was succesfull, 0 otherwise.

Author(s):

Philippe Lavoie

Date:

23 September 1997

**3.22.17 int OpenGL::read(const char* filename)
[virtual]**

Reads the information from a stream.

Parameters:

filename - the input file

Returns:

1 on sucess, 0 on failure

Author(s):

Philippe Lavoie

Date:

19 June 1998

Reimplemented in **PointListGL**, **ObjectListGL** and **HNurbsSurfaceGL**.

**3.22.18 int OpenGL::write(const char* filename) const
[virtual]**

Writes a OpenGL to a file.

Parameters:

filename - the filename to write to.

Returns:

0 if an error occurs, 1 otherwise

Author(s):

Philippe Lavoie

Date:

19 June 1998

Reimplemented in **PointListGL**, **ObjectListGL** and **HNurbsSurfaceGL**.

3.22.19 int OpenGL::read(istream &fin) [virtual]

Reads the information from a stream.

Parameters:

fin - the input stream

Returns:

1 on success, 0 on failure

Author(s):

Philippe Lavoie

Date:

19 June 1998

Reimplemented in **PointListGL**, **NurbsSurfaceGL**, **NurbsCurveGL** and **HNurbsSurfaceGL**.

3.22.20 int OpenGL::write(ofstream &fout) const [virtual]

Writes a OpenGL to a stream.

Parameters:

fout - the output stream

Returns:

0 if an error occurs, 1 otherwise

Author(s):

Philippe Lavoie

Date:

19 June 1998

Reimplemented in **PointListGL**, **NurbsSurfaceGL**, **NurbsCurveGL** and **HNurbsSurfaceGL**.

3.22.21 void ObjectGL::setName(const char* n)

Sets the name of the object.

Sets the name of the object. The content of *n* is copied to the name.

Parameters:

n - the name of the object

Author(s):

Philippe Lavoie

Date:

20 September 1997

3.22.22 char* ObjectGL::typeName() const

Returns the name of the type of the class.

Returns:

the name of the type of the class, 0 if the type is unknown

Author(s):

Philippe Lavoie

Date:

20 September 1997

The documentation for this class was generated from the following files:

- nurbsGL.hh
- test.hh
- nurbsGL.cc

3.23 ObjectListGL Class Reference

A link list of **ObjectGL**.

`#include <nurbs/nurbsGL.hh>`

Inherits **ObjectGL**.

Inherited by **NurbsListGL** and **ObjectRefListGL**.

Public Members

- **ObjectListGL** ()
The constructor.
- virtual **~ObjectListGL** ()
The destructor.
- **ObjectGL*** **first** () const
- **ObjectGL*** **last** () const
- **ObjectGL*** **current** () const
- **ObjectGL*&** **first** ()
- **ObjectGL*&** **last** ()
- **ObjectGL*&** **current** ()
- virtual void **glObject** () const
Calls glObject for all the elements from the list.
- virtual void **display** () const
Displays all the elements from the list.
- virtual void **displayName** () const
Displays all the elements from the list.
- void **reset** ()
Deletes all the node of the list.
- void **setResetMode** (int m)
- void **add** (**ObjectGL*** obj)
adds an element to the list.
- **ObjectGL*** **remove** (**ObjectGL*** obj)
finds an element and remove it from the list.
- void **activate** ()
Activate all the objects of the list.

- void **deactivate** ()
Deactivates all the objects of the list.
- void **select** ()
Select all the objects of the list.
- void **deselect** ()
Deselect all the objects of the list.
- void **transformTo** (GLfloat x, GLfloat y, GLfloat z, GLfloat a, GLfloat b, GLfloat c, GLfloat sx, GLfloat sy, GLfloat sz, int behavior=NURBS_FLAGS_AFFECT_ALL)
transforms the elements stored in the list.
- void **transformBy** (GLfloat x, GLfloat y, GLfloat z, GLfloat a, GLfloat b, GLfloat c, GLfloat sx, GLfloat sy, GLfloat sz, int behavior=NURBS_FLAGS_AFFECT_ALL)
Transforms the elements stored in the list.
- ObjectGL* **goTo** (int a)
Moves the current pointer to the n th element.
- ObjectGL* **goToActive** (int a)
moves the current pointer to the n th active element.
- ObjectGL* **goToNext** ()
move the current pointer to the next element.
- ObjectGL* **goToPrevious** ()
move the current pointer to the previous one.
- ObjectGL* **goToNextActive** ()
move the current pointer to the next active element.
- ObjectGL* **goToPreviousActive** ()
move the current pointer to the previous activeElement.
- ObjectGL* **jumpToNext** ()
move the current pointer to the next element.
- ObjectGL* **jumpToPrevious** ()
move the current pointer to the next element.

- void **setJumpSize** (int a)
- int **size** () const
- virtual int **read** (const char* filename)
reads a list of objects.
- virtual int **write** (const char* filename) const
Writes a list of objects.
- virtual int **writeRIB** (const char* filename) const
Write a list of object in the RIB format.
- virtual int **writePOVRAY** (const char* filename) const
Write a list of object in the POVray format.
- void **viewAllObjects** ()
Sets all object inside the list to view mode.
- void **hideAllObjects** ()
Sets all object inside the list to hide mode.

Protected Members

- ObjectGL* **first_**
- ObjectGL * **last_**
- ObjectGL* **current_**
- int **jumpSize**
- int **n**
- int **resetMode**

Detailed Description

A link list of **ObjectGL**.

Author(s):

Philippe Lavoie

Date:

28 September 1997

Member Function Documentation

3.23.1 `ObjectListGL::ObjectListGL()`

The constructor.

By default the reset mode is set to delete all the elements from the list (

`NURBSLIST_DELETE_AT_RESET`

). If you don't want this behavior, call

`setResetMode(NURBS_KEEP_AT_RESET)`

Author(s):

Philippe Lavoie

Date:

30 September 1997

3.23.2 `ObjectListGL::~~ObjectListGL()` [virtual]

The destructor.

Author(s):

Philippe Lavoie

Date:

30 September 1997

3.23.3 `void ObjectListGL::glObject() const` [virtual]

Calls `glObject` for all the elements from the list.

Displays all the elements from the list

Author(s):

Philippe Lavoie

Date:

30 September 1997

Reimplemented from `ObjectGL`.

Reimplemented in `NurbsListGL`.

3.23.4 void ObjectListGL::display() const [virtual]

Displays all the elements from the list.

Author(s):

Philippe Lavoie

Date:

30 September 1997

Reimplemented from **ObjectGL**.

Reimplemented in **NurbsListGL**.

3.23.5 void ObjectListGL::displayName() const [virtual]

Displays all the elements from the list.

Author(s):

Philippe Lavoie

Date:

30 September 1997

Reimplemented from **ObjectGL**.

3.23.6 void ObjectListGL::reset()

Deletes all the node of the list.

Deletes all the node of the list. Depending on the reset mode, it will also delete the elements.

Author(s):

Philippe Lavoie

Date:

30 September 1997

3.23.7 void ObjectListGL::add(ObjectGL* obj)

adds an element to the list.

Adds an element to the list.

Parameters:

obj - the element to add

Author(s):

Philippe Lavoie

Date:

30 September 1997

Reimplemented in **ObjectRefListGL**.

3.23.8 ObjectGL* ObjectListGL::remove(ObjectGL* obj)

finds an element and remove it from the list.

Finds an element and delete it from the list. The element will *not* be deleted. This is up to the calling function.

Parameters:

obj - the element to search

Returns:

a pointer to obj if it was found in the list, 0 otherwise

Author(s):

Philippe Lavoie

Date:

30 September 1997

Reimplemented in **ObjectRefListGL**.

3.23.9 void ObjectListGL::activate()

Activate all the objects of the list.

Author(s):

Philippe Lavoie

Date:

30 September 1997

Reimplemented from **ObjectGL**.

3.23.10 void ObjectListGL::deactivate()

Deactivates all the objects of the list.

Author(s):

Philippe Lavoie

Date:

30 September 1997

Reimplemented from **ObjectGL**.

3.23.11 void ObjectListGL::select()

Select all the objects of the list.

Author(s):

Philippe Lavoie

Date:

30 September 1997

Reimplemented from **ObjectGL**.

3.23.12 void ObjectListGL::deselect()

Deselect all the objects of the list.

Author(s):

Philippe Lavoie

Date:

2 October 1997

Reimplemented from **ObjectGL**.

3.23.13 void ObjectListGL::transformTo(GLfloat x, GLfloat y, GLfloat z, GLfloat a, GLfloat b, GLfloat c, GLfloat sx, GLfloat sy, GLfloat sz, int behavior)

transforms the elements stored in the list.

Transforms the elements stored in the list *to* a certain value if the elements are in a proper state. The behavior variable specifies which objects should be affected by this function.

Parameters:

x - a translation in the *\$x\$* direction
y - a translation in the *\$y\$* direction
z - a translation in the *\$z\$* direction
a - a rotation around the *\$x\$*-axis
b - a rotation around the *\$y\$*-axis
c - a rotation around the *\$z\$*-axis
sx - a scaling in the direction of the *\$x\$*-axis
sy - a scaling in the direction of the *\$x\$*-axis
sz - a scaling in the direction of the *\$x\$*-axis
behavior - specifies which objects are affected by the function

Author(s):

Philippe Lavoie

Date:

30 September 1997

3.23.14 void ObjectListGL::transformBy(GLfloat x, GLfloat y, GLfloat z, GLfloat a, GLfloat b, GLfloat c, GLfloat sx, GLfloat sy, GLfloat sz, int behavior)

Transforms the elements stored in the list.

Transforms the elements stored in the list *by* a certain value if the elements are in a proper state. The behavior variable specifies which objects should be affected by this function.

Parameters:

x - a translation in the *\$x\$* direction
y - a translation in the *\$y\$* direction
z - a translation in the *\$z\$* direction

a - a rotation around the *x*-axis
b - a rotation around the *y*-axis
c - a rotation around the *z*-axis
sx - a scaling in the direction of the *x*-axis
sy - a scaling in the direction of the *y*-axis
sz - a scaling in the direction of the *z*-axis
behavior - specifies which objects are affected by the function

Author(s):

Philippe Lavoie

Date:

30 September 1997

3.23.15 ObjectGL* ObjectListGL::goTo(int a)

Moves the current pointer to the *n* th element.

Moves the current pointer to the *n* th element. This must be in a valid range otherwise 0 is returned.

Parameters:

a - the *a* th element

Returns:

A pointer to the current element or 0 if *a* was out of range.

Author(s):

Philippe Lavoie

Date:

30 October 1997

3.23.16 ObjectGL* ObjectListGL::goToActive(int a)

moves the current pointer to the *n* th active element.

Moves the current pointer to the *n* th active element. This must be in a valid range otherwise 0 is returned.

Parameters:

a - the *a* th active element

Returns:

A pointer to the current element or 0 if a was out of range.

Author(s):

Philippe Lavoie

Date:

29 January 1998

3.23.17 ObjectGL* ObjectListGL::goToNext()

move the current pointer to the next element.

Moves the current pointer to the next element. If there are no next element, it goes to the first element.

Returns:

A pointer to the current element

Author(s):

Philippe Lavoie

Date:

2 October 1997

3.23.18 ObjectGL* ObjectListGL::goToPrevious()

move the current pointer to the previous one.

Moves the current pointer to the previous element. If there are no previous element, it goes to the last element.

Returns:

A pointer to the current element

Author(s):

Philippe Lavoie

Date:

30 September 1997

3.23.19 ObjectGL* ObjectListGL::goToNextActive()

move the current pointer to the next active element.

Moves the current pointer to the next element. If there are no next element, it goes to the first element.

Returns:

A pointer to the current element

Author(s):

Philippe Lavoie

Date:

29 January 1998

3.23.20 ObjectGL* ObjectListGL::goToPreviousActive()

move the current pointer to the previous activeElement.

Moves the current pointer to the previous element. If there are no previous element, it goes to the last element.

Returns:

A pointer to the current element

Author(s):

Philippe Lavoie

Date:

29 January 1998

3.23.21 ObjectGL* ObjectListGL::jumpToNext()

move the current pointer to the next element.

Moves the current pointer to the next element. If there are no next element, it goes to the first element.

Returns:

A pointer to the current element

Author(s):

Philippe Lavoie

Date:

2 October 1997

3.23.22 ObjectGL* ObjectListGL::jumpToPrevious()

move the current pointer to the next element.

Moves the current pointer to the next element. If there are no next element, it goes to the first element.

Returns:

A pointer to the current element

Author(s):

Philippe Lavoie

Date:

2 October 1997

3.23.23 int ObjectListGL::read(const char* filename) [virtual]

reads a list of objects.

Parameters:

filename - the name of the file to read from

Returns:

1 on success, 0 otherwise

Author(s):

Philippe Lavoie

Date:

13 October 1997

Reimplemented from **ObjectGL**.

3.23.24 int ObjectListGL::write(const char* filename) const [virtual]

Writes a list of objects.

Parameters:

filename - the name of the file to read from

Returns:

1 on success, 0 otherwise

Author(s):

Philippe Lavoie

Date:

13 October 1997

Reimplemented from **ObjectGL**.

3.23.25 int ObjectListGL::writeRIB(const char* filename) const [virtual]

Write a list of object in the RIB format.

Parameters:

filename - the name of the file to read from

Returns:

1 on success, 0 otherwise

Author(s):

Philippe Lavoie

Date:

13 October 1997

Reimplemented from **ObjectGL**.

3.23.26 int ObjectListGL::writePOVRAY(const char* filename) const [virtual]

Write a list of object in the POVRAY format.

Parameters:

filename - the name of the file to read from

Returns:

1 on success, 0 otherwise

Author(s):

Philippe Lavoie

Date:

13 October 1997

Reimplemented from **ObjectGL**.

3.23.27 void ObjectListGL::viewAllObjects()

Sets all object inside the list to view mode.

Author(s):

Philippe Lavoie

Date:

29 January 1998

3.23.28 void ObjectListGL::hideAllObjects()

Sets all object inside the list to hide mode.

Author(s):

Philippe Lavoie

Date:

29 January 1998

The documentation for this class was generated from the following files:

- nurbsGL.hh
- nurbsGL.cc

3.24 ObjectRefGL Class Reference

A reference object for OpenGL.

```
#include <nurbs/nurbsGL.hh>
```

Inherits **ObjectGL**.

Public Members

- **ObjectRefGL** (ObjectGL* p)
- void **glObject** () const
- ObjectGL*& **ptr**

Protected Members

- ObjectGL* **ptr_**

Detailed Description

A reference object for OpenGL.

This class reference other OpenGL objects.

Author(s):

Philippe Lavoie

Date:

28 September 1997

The documentation for this class was generated from the following file:

- nurbsGL.hh

3.25 ObjectRefListGL Class Reference

A link list of ObjectRefListGL.

`#include <nurbs/nurbsGL.hh>`

Inherits **ObjectListGL**.

Public Members

- **ObjectRefListGL** ()
 - **~ObjectRefListGL** ()
The destructor.
 - void **add** (ObjectGL* obj)
 - ObjectGL* **remove** (ObjectGL* obj)
finds an element and remove it from the list.
 - void **refList** (const ObjectListGL* list, int addOnce=1)
reference all the elements from a list.
-

Detailed Description

A link list of ObjectRefListGL.

Author(s):

Philippe Lavoie

Date:

28 September 1997

Member Function Documentation

3.25.1 ObjectRefListGL::~~ObjectRefListGL()

The destructor.

Author(s):

Philippe Lavoie

Date:

3 June 1998

3.25.2 ObjectGL* ObjectRefListGL::remove(ObjectGL* obj)

finds an element and remove it from the list.

Finds an element and delete it from the list. The element will *not* be deleted. This is up to the calling function. Since this is a reference list, we check if the element is referencing obj. If it is, we remove it from the list. If not, we check if obj is one of the element from the list and if so we remove it.

Parameters:

obj - the element to remove

Returns:

the element from the list or 0 if obj wasn't found.

Author(s):

Philippe Lavoie

Date:

30 September 1997

Reimplemented from **ObjectListGL**.

3.25.3 void ObjectRefListGL::refList(const ObjectListGL* list, int addOnce)

reference all the elements from a list.

Reference all the elements from a list. If the addOnce variable is set, it will reference the elements of the list only once. So if the reference list was already referencing one of the elements, it will not be added again. If the addOnce variable is not set, the resulting list might reference an element more than once.

Parameters:

list - the list to reference

Author(s):

Philippe Lavoie

Date:

30 September 1997

The documentation for this class was generated from the following files:

- nurbsGL.hh
- nurbsGL.cc

3.26 ParaCurve Class Reference

An abstract parametric curve class.

`#include <nurbs/curve.hh>`

Inherited by **NurbsCurve**.

Public Members

- **ParaCurve** ()
< Empty constructor.
- virtual **HPoint_nD**<T,N> **operator()** (T u) const = 0
abstract function.
- **HPoint_nD**<T,N> **hpointAt** (T u) const
Wrapper to the operator() function.
- **Point_nD**<T,N> **pointAt** (T u) const
Projects the homogenous point at u into normal space.
- virtual **HPoint_nD**<T,N> **hpointAt** (T u, int span) const = 0
abstract function.
- **Point_nD**<T,N> **pointAt** (T u, int span)
Projects the homogenous point at u into normal space.
- virtual void **deriveAtH** (T u, int, Vector< **HPoint_nD**<T, N> >&) const = 0
abstract function.
- virtual void **deriveAt** (T u, int, Vector< **Point_nD**<T, N> >&) const = 0
abstract function.
- virtual T **minKnot** () const = 0
abstract function.
- virtual T **maxKnot** () const = 0
abstract function.
- virtual T **minDist2** (const **Point_nD**<T, N>& p, T& guess, T error=0.0001, T s=0.2, int sep=9, int maxIter=10, T um=-1, T uM=-1) const

Find the minimal distance between a point and the curve.

- virtual Point_nD<T,N> **minDistY** (T y, T& guessU, T error=0.0001, T s=-1, int sep=9, int maxIter=10, T um=-1, T uM=-1) const

Find the closest point on the curve to the y coordinate.

- virtual Point_nD<T,N> **minDistX** (T y, T& guessU, T error=0.0001, T s=-1, int sep=9, int maxIter=10, T um=-1, T uM=-1) const

Find the closest point on the curve to the x coordinate.

- virtual Point_nD<T,N> **minDistZ** (T y, T& guessU, T error=0.0001, T s=-1, int sep=9, int maxIter=10, T um=-1, T uM=-1) const

Find the closest point on the curve to the z coordinate.

- virtual T **extremum** (int findMin, CoordinateType coord, T minDu=0.0001, int sep=9, int maxIter=10, T um=-1, T uM=-1) const

Finds the minimal or maximal value on the curve of the x,y or z coordinate.

Detailed Description

An abstract parametric curve class.

This is an abstract class used as a basis for NURBS and HNURBS curves.

Author(s):

Philippe Lavoie

Date:

4 Oct. 1996

Member Function Documentation

3.26.1 ParaCurve::ParaCurve()

< Empty constructor.

3.26.2 `virtual HPoint_nD<T,N>
ParaCurve::operator()(T u) const =
0 [pure virtual]`

abstract function.

Reimplemented in `NurbsCurve`.

3.26.3 `HPoint_nD<T,N> ParaCurve::hpointAt(T u)
const`

Wrapper to the `operator()` function.

Reimplemented in `NurbsCurve`.

3.26.4 `Point_nD<T,N> ParaCurve::pointAt(T u) const`

Projects the homogenous point at u into normal space.

3.26.5 `virtual HPoint_nD<T,N> ParaCurve::hpointAt(T
u, int span) const = 0 [pure virtual]`

abstract function.

Reimplemented in `NurbsCurve`.

3.26.6 `Point_nD<T,N> ParaCurve::pointAt(T u, int
span)`

Projects the homogenous point at u into normal space.

3.26.7 `virtual void ParaCurve::deriveAtH(T u, int,
Vector< HPoint_nD<T, N> >&) const = 0 [pure
virtual]`

abstract function.

Reimplemented in `NurbsCurve`.

3.26.8 `virtual void ParaCurve::deriveAt(T u, int, Vector< Point_nD<T, N> >&) const = 0` [pure virtual]

abstract function.

Reimplemented in `NurbsCurve`.

3.26.9 `virtual T ParaCurve::minKnot() const = 0` [pure virtual]

abstract function.

Reimplemented in `NurbsCurve`.

3.26.10 `virtual T ParaCurve::maxKnot() const = 0` [pure virtual]

abstract function.

Reimplemented in `NurbsCurve`.

3.26.11 `T ParaCurve::minDist2(const Point_nD<T,N>& p, T& guess,T error,T s,int sep,int maxIter, T um, T uM) const` [virtual]

Find the minimal distance between a point and the curve.

This is an iterative method to find the closest point to a curve.

Parameters:

- p* - the minimal distance from that point
- guess* - a starting value for the parameter *u*, on exit this will be set to the value of the point on the curve closest to *p*.
- error* - when iterations have an error smaller than this value, the function exits
- s* - the size of the search in the parametric space.
- sep* - the number of points initially looked at to find a minimal distance
- maxiter* - the maximal number of iterations
- um* - the minimal parametric value
- uM* - the maximal parametric value

Returns:

The value of the minimal distance between *p* and the curve. The variable *guess* now holds the parametric value of the curve point closest to *p*.

Warning:

It has not been tested with closed loop curves.

Author(s):

Philippe Lavoie

Date:

24 January 1997

3.26.12 Point_nD<T,N> ParaCurve::minDistY(T y, T& guessU, T error, T s, int sep, int maxIter, T um, T uM) const [virtual]

Find the closest point on the curve to the y coordinate.

This is an iterative method to find the closest point on the curve to the y coordinate.

Parameters:

y - the y coordinate to be close too.

$guess$ - a starting value for the parameter u , on exit this will be set to the value of the point on the curve closest to y .

$error$ - when iterations have an error smaller than this value, the function exits

s - the size of the search in the parametric space.

sep - the number of points initially looked at to find a minimal distance

$maxiter$ - the maximal number of iterations

um - the minimal parametric value

uM - the maximal parametric value

Returns:

The value of the minimal distance between p and the curve. The variable $guess$ now holds the parametric value of the curve point closest to p .

Warning:

It has not been tested with closed loop curves.

Author(s):

Philippe Lavoie

Date:

24 January 1997

3.26.13 `Point_nD<T,N> ParaCurve::minDistX(T x, T& guessU, T error, T s, int sep, int maxIter, T um, T uM) const [virtual]`

Find the closest point on the curve to the x coordinate.

This is an iterative method to find the closest point on the curve to the x coordinate.

Parameters:

x - the x coordinate to be close too.
guess - a starting value for the parameter u , on exit this will be set to the value of the point on the curve closest to x .
error - when iterations have an error smaller than this value, the function exits
s - the size of the search in the parametric space.
sep - the number of points initially looked at to find a minimal distance
maxiter - the maximal number of iterations
um - the minimal parametric value
uM - the maximal parametric value

Returns:

The value of the minimal distance between p and the curve. The variable *guess* now holds the parametric value of the curve point closest to p .

Warning:

It has not been tested with closed loop curves.

Author(s):

Philippe Lavoie

Date:

24 January 1997

3.26.14 `Point_nD<T,N> ParaCurve::minDistZ(T z, T& guessU, T error, T s, int sep, int maxIter, T um, T uM) const [virtual]`

Find the closest point on the curve to the x coordinate.

This is an iterative method to find the closest point on the curve to the x coordinate.

Parameters:

z - the x coordinate to be close too.

guess - a starting value for the parameter u , on exit this will be set to the value of the point on the curve closest to x .
error - when iterations have an error smaller than this value, the function exits
s - the size of the search in the parametric space.
sep - the number of points initially looked at to find a minimal distance
maxIter - the maximal number of iterations
um - the minimal parametric value
uM - the maximal parametric value

Returns:

The value of the minimal distance between p and the curve. The variable *guess* now holds the parametric value of the curve point closest to p .

Warning:

It has not been tested with closed loop curves.

Author(s):

Philippe Lavoie

Date:

24 January 1997

3.26.15 T ParaCurve::extremum(int findMin, CoordinateType coord, T minDu, int sep, int maxIter, T um, T uM) const [virtual]

Finds the minimal or maximal value on the curve of the x,y or z coordinate.

Finds the minimal or maximal value on the curve of the x,y or z coordinate.

Parameters:

findMin - a flag indicating if we're looking for the minimal value or the maximal value.
coord - Which coordinate to find: x,y or z.
minDu - The minimal distance between iterations in the parametric space.
sep - the number of points initially looked at to find a minimal distance
maxIter - the maximal number of iterations
um - the minimal parametric value
uM - the maximal parametric value

Returns:

The minimal value of z along the curve

Warning:

It has not been tested with closed loop curves.

Author(s):

Philippe Lavoie

Date:

24 January 1997

The documentation for this class was generated from the following files:

- curve.hh
- curve.cc

3.27 ParaSurface Class Reference

An abstract parametric surface class.

Inherited by **NurbsSurface**.

Public Members

- **ParaSurface** ()
< Empty constructor.
- virtual **HPoint_nD**<T,N> **operator()** (T u, T v) const = 0
an abstract function.
- **HPoint_nD**<T,N> **hpointAt** (T u, T v) const
< Calls operator().
- **Point_nD**<T,N> **pointAt** (T u, T v) const
< Projects the point in the normal space.
- virtual void **deriveAtH** (T u, T v, int d, Matrix< **HPoint_nD**<T, N> >& skl) const = 0
an abstract function.
- virtual void **deriveAt** (T u, T v, int d, Matrix< **Point_nD**<T, N> >& skl) const = 0
an abstract function.
- virtual T **minDist2** (const **Point_nD**<T, N>& p, T& guessU, T& guessV, T error=0.001, T s=0.2, int sep=9, int maxIter=10, T um=0.0, T uM=1.0, T vm=0.0, T vM=1.0) const
Find the minimal distance between a point and the surface.
- virtual T **minDist2b** (const **Point_nD**<T, N>& p, T& guessU, T& guessV, T error=0.001, T s=0.3, int sep=5, int maxIter=10, T um=0.0, T uM=1.0, T vm=0.0, T vM=1.0) const
Find the minimal distance between a point and the surface.
- virtual T **minDist2xy** (const **Point_nD**<T, N>& p, T& guessU, T& guessV, T error=0.01, T dU=0.0001, T s=0.3, int sepU=5, int sepV=5, int maxIter=10, T um=0.0, T uM=1.0, T vm=0.0, T vM=1.0) const
Find the minimal distance between a point and the surface in the x-y plane.

- **int projectOn** (const Point_nD<T, N>& p, T& u, T& v, int maxI=100, const T um=0.0, const T uM=1.0, const T vm=0.0, const T vM=1.0) const

projects a point onto the surface.

- **T extremum** (int findMin, CoordinateType coord, T minDu=0.0001, int sepU=5, int sepV=5, int maxIter=10, T um=0.0, T uM=1.0, T vm=0.0, T vM=1.0) const

Finds the minimal or maximal value on the curve of the x,y or z coordinate.

- **int intersectWith** (const ParaSurface<T, N> &S, Point_nD<T, N>& p, T& u, T& v, T& s, T& t, int maxI=100, T um=0.0, T uM=1.0, T vm=0.0, T vM=1.0) const

Finds the intersection of two surfaces near a point.

- **int intersectWith** (const ParaSurface<T, N> &S, struct InterPoint<T, N> &iter, int maxI=100, T um=0.0, T uM=1.0, T vm=0.0, T vM=1.0) const

Finds the intersection of two surfaces near a point.

- **virtual int writeVRML** (const char* filename, const Color& color, int Nu, int Nv, T u_s, T u_e, T v_s, T v_e) const

Write the NURBS surface to a VRML file.

- **virtual int writeVRML** (const char* filename, const Color& color=whiteColor, int Nu=20, int Nv=20) const = 0

an abstract function.

Detailed Description

An abstract parametric surface class.

This is an abstract class used as a basis for NURBS and HNURBS surfaces.

Author(s):

Philippe Lavoie

Date:

4 Oct. 1996

Member Function Documentation

3.27.1 ParaSurface::ParaSurface()

< Empty constructor.

3.27.2 virtual HPoint_nD<T,N> ParaSurface::operator()(T u, T v) const = 0 [pure virtual]

an abstract function.

Reimplemented in NurbsSurface and HNurbsSurface.

3.27.3 HPoint_nD<T,N> ParaSurface::hpointAt(T u, T v) const

< Calls operator().

3.27.4 Point_nD<T,N> ParaSurface::pointAt(T u, T v) const

< Projects the point in the normal space.

3.27.5 virtual void ParaSurface::deriveAtH(T u, T v, int d, Matrix< HPoint_nD<T, N> >& skl) const = 0 [pure virtual]

an abstract function.

Reimplemented in NurbsSurface.

3.27.6 virtual void ParaSurface::deriveAt(T u, T v, int d, Matrix< Point_nD<T, N> >& skl) const = 0 [pure virtual]

an abstract function.

Reimplemented in NurbsSurface.


```

3.27.7  T ParaSurface::minDist2(const Point_nD<T,N>&
    p, T& guessU, T& guessV, T error=0.001,T
    s=0.2,int sep=9,int maxIter=10,T um=0.0, T
    uM=1.0, T vm=0.0, T vM=1.0) const  [virtual]

```

Find the minimal distance between a point and the surface.

This is an iterative method to find the closest point to a surface.

Parameters:

p - the minimal distance from that point
guessU - a starting value for the parameter *u*, on exit this will be set to the value of the point on the surface closest to *p*.
guessV - a starting value for the parameter *v*, on exit this will be set to the value of the point on the surface closest to *p*.
error - when iterations have an error smaller than this value, the function exits
s - the size of the search in the parametric space.
sep - the number of points initially looked at to find a minimal distance.
maxiter - the maximal number of iterations
um - the minimal parametric value for *u*
uM - the maximal parametric value for *u*
vm - the minimal parametric value for *v*
vM - the maximal parametric value for *v*

Returns:

The value of the minimal distance between *p* and the surface. The variables *guessU* and *guessV* now holds the parametric value of the surface point closest to *p*.

Warning:

It has not been tested with closed loop surfaces.

Author(s):

Philippe Lavoie

Date:

24 January 1997

```

3.27.8  T ParaSurface::minDist2b(const
        Point_nD<T,N>& p, T& guessU, T& guessV, T
        error=0.001,T s=0.3,int sep=5,int maxIter=10,T
        um=0.0, T uM=1.0, T vm=0.0, T vM=1.0) const
        [virtual]

```

Find the minimal distance between a point and the surface.

This is an iterative method to find the closest point to a surface. The method is slightly different than minDist2.

Parameters:

p - the minimal distance from that point
guessU - a starting value for the parameter *u*, on exit this will be set to the value of the point on the surface closest to *p*.
guessV - a starting value for the parameter *v*, on exit this will be set to the value of the point on the surface closest to *p*.
error - when iterations have an error smaller than this value, the function exits
s - the size of the search in the parametric space.
sep - the number of points initially looked at to find a minimal distance.
maxiter - the maximal number of iterations
um - the minimal parametric value for *u*
uM - the maximal parametric value for *u*
vm - the minimal parametric value for *v*
vM - the maximal parametric value for *v*

Returns:

The value of the minimal distance between *p* and the surface. The variables *guessU* and *guessV* now holds the parametric value of the surface point closest to *p*.

Warning:

It has not been tested with closed loop surfaces.

Author(s):

Philippe Lavoie

Date:

24 January 1997

```

3.27.9  T ParaSurface::minDist2xy(const
        Point_nD<T,N>& p, T& guessU, T& guessV, T
        error=0.01,T dU=0.0001, T s=0.3,int sepU=5, int
        sepV=5, int maxIter=10,T um=0.0, T uM=1.0, T
        vm=0.0, T vM=1.0) const [virtual]

```

Find the minimal distance between a point and the surface in the x-y plane.

This is an iterative method to find the closest point to a surface. The distance is search in the x-y plane. The z component is *not* taken into account for the search.

Parameters:

p - the minimal distance from that point
guessU - a starting value for the parameter *u*, on exit this will be set to the value of the point on the surface closest to *p*.
guessV - a starting value for the parameter *v*, on exit this will be set to the value of the point on the surface closest to *p*.
error - when iterations have an error smaller than this value, the function exits
dU - if a parametric delta is smaller than this value, the function stops.
s - the size of the search in the parametric space.
sepU - the number of points initially looked at to find a minimal distance in the *u* direction
sepV - the number of points initially looked at to find a minimal distance in the *v* direction
maxiter - the maximal number of iterations
um - the minimal parametric value for *u*
uM - the maximal parametric value for *u*
vm - the minimal parametric value for *v*
vM - the maximal parametric value for *v*

Returns:

The value of the minimal distance between *p* and the surface. The variables *guessU* and *guessV* now holds the parametric value of the surface point closest to *p*.

Warning:

It has not been tested with closed loop surfaces.

Author(s):

Philippe Lavoie

Date:

24 January 1997

3.27.10 `int ParaSurface::projectOn(const Point_nD<T,N>& p, T& u, T& v, int maxI, const T um, const T uM, const T vm, const T vM) const`

projects a point onto the surface.

Projects a point using Newton-Raphson's method. We want to solve for

$$X'_u(u_i, v_i) \cdot \delta u + X'_v(u_i, v_i) \cdot \delta v = P - X(u_i, v_i)$$

This is an over-determined system and the least square approximation is taken:

$$\begin{bmatrix} X'_u(u_i, v_i) \cdot X'_u(u_i, v_i) & X'_u(u_i, v_i) \cdot X'_v(u_i, v_i) \\ X'_v(u_i, v_i) \cdot X'_u(u_i, v_i) & X'_v(u_i, v_i) \cdot X'_v(u_i, v_i) \end{bmatrix} \begin{bmatrix} \delta u \\ \delta v \end{bmatrix} = \begin{bmatrix} (P - X(u_i, v_i)) \cdot X'_u(u_i, v_i) \\ (P - X(u_i, v_i)) \cdot X'_v(u_i, v_i) \end{bmatrix}$$

If $\|X'_u\| = 0$ and $\|X'_v\| = 0$, then the equation is singular. In that case the routine returns with 0. Use one of the other iterative method to find a suitable projection, if this happens.

Parameters:

P - the point to project
u - the u parametric value of the result
v - the v parametric value of the result
maxI - the maximal number of iterations
um - the minimal parametric value for *u*
uM - the maximal parametric value for *u*
vm - the minimal parametric value for *v*
vM - the maximal parametric value for *v*

Returns:

1 on sucess, 0 if their was a singularity in computation

Author(s):

Philippe Lavoie

Date:

24 January 1997

3.27.11 `T ParaSurface::extremum(int findMin, CoordinateType coord, T minDu, int sepU, int sepV, int maxIter, T um, T uM, T vm, T vM) const`

Finds the minimal or maximal value on the curve of the x,y or z coordinate.

Parameters:

findMin - a flag indicating if we're looking for the minimal value or the maximal value.

coord - Which coordinate to find: x,y or z.

minDu - The minimal distance between iterations in the parametric space.

sepU - the number of points initially looked at to find a minimal distance in the U direction

sepV - the number of points initially looked at to find a minimal distance in the V direction

maxiter - the maximal number of iterations

um - the minimal parametric value for *u*

uM - the maximal parametric value for *u*

vm - the minimal parametric value for *v*

vM - the maximal parametric value for *v*

Returns:

The minimal value of z along the curve

Warning:

It has not been tested with closed loop curves.

Author(s):

Philippe Lavoie

Date:

24 January 1997

3.27.12 `int ParaSurface::intersectWith(const
ParaSurface<T,N> &S, Point_nD<T,N>& p, T&
u, T& v, T& s, T& t, int maxI=100, T um=0.0,
T uM=1.0, T vm=0.0, T vM=1.0) const`

Finds the intersection of two surfaces near a point.

The method used is similar to the one used to project a point on a surface. It's a modified Newton-Raphson's method.

Parameters:

S - the surface to intersect with

P - the point for the intersection

u - the u parametric value of the intersection

v - the v parametric value of the intersection

s - the u parametric value of the intersection for S

t - the v parametric value of the intersection for S

maxI - the maximal number of iterations
um - the minimal parametric value for *u*
uM - the maximal parametric value for *u*
vm - the minimal parametric value for *v*
vM - the maximal parametric value for *v*

Returns:

1 on sucess, 0 if their was a singularity in computation

Author(s):

Philippe Lavoie

Date:

6 July 1998

3.27.13 `int ParaSurface::intersectWith(const
ParaSurface<T,N> &S, struct InterPoint<T,N>
&iter, int maxI=100, T um=0.0, T uM=1.0, T
vm=0.0, T vM=1.0) const`

Finds the intersection of two surfaces near a point.

Parameters:

S - the surface to intersect with
iter - the iteration point
maxI - the maximal number of iterations
um - the minimal parametric value for *u*
uM - the maximal parametric value for *u*
vm - the minimal parametric value for *v*
vM - the maximal parametric value for *v*

Returns:

1 on sucess, 0 if their was a singularity in computation

Author(s):

Philippe Lavoie

Date:

6 July 1998

3.27.14 `int ParaSurface::writeVRML(const char* filename, const Color& color, int Nu, int Nv, T uS, T uE, T vS, T vE) const` [virtual]

Write the NURBS surface to a VRML file.

Writes a VRML file which represents the surface for the parametric space $[uS, uE]$ and $[vS, vE]$. It does not optimize the number of points required to represent the surface.

Parameters:

filename - the file name for the output VRML file
Nu - the number of points in the *u* direction
Nv - the number of points in the *v* direction
uS - the starting value of *u*
uE - the end value of *u*
vS - the starting value of *v*
vE - the end value of *v*

Returns:

1 on success, 0 otherwise

Warning:

The parametric surface must be valid

Author(s):

Philippe Lavoie

Date:

24 January, 1997

Reimplemented in **NurbsSurface**.

3.27.15 `virtual int ParaSurface::writeVRML(const char* filename, const Color& color=whiteColor, int Nu=20, int Nv=20) const = 0` [pure virtual]

an abstract function.

Reimplemented in **NurbsSurface**.

The documentation for this class was generated from the following files:

- surface.hh
- surface.cc

3.28 PointGL Class Reference

A class to hold a 3D point.

```
#include <nurbs/nurbsGL.hh>
```

Inherits **ObjectGL**.

Public Members

- **PointGL** (const Point3Df& p3d)
- void **glObject** () const
Displays a point.
- void **modify** (const Point3Df& v)
- void **set** (const Point3Df& v)
- void **setSize** (int s)
- float **x** () const
- float **y** () const
- float **z** () const
- const Point3Df& **point** () const
- ObjectGL* **copy** ()

Protected Members

- Point3Df **p**
- int **psize**

Detailed Description

A class to hold a 3D point.

Author(s):

Philippe Lavoie

Date:

23 September 1997

Member Function Documentation

3.28.1 void PointGL::glObject() const

Displays a point.

Author(s):

Philippe Lavoie

Date:

30 September 1997

Reimplemented from **ObjectGL**.

The documentation for this class was generated from the following files:

- nurbsGL.hh
- nurbsGL.cc

3.29 PointListGL Class Reference

A class to hold a list of points.

```
#include <nurbs/nurbsGL.hh>
```

Inherits **ObjectGL**.

Public Members

- **PointListGL** ()
- **PointListGL** (const PointListGL &pl)
Copy constructor.
- **PointListGL** (const BasicList<Point3Df> &l)
Constructor from a list of points.
- virtual void **glObject** () const
Displays a list of points.
- void **setSize** (int s)
- int **read** (const char*f)
- int **write** (const char* f) const
- int **read** (ifstream &fin)
Reads a list of points.
- int **write** (ofstream &fout) const
Writes a list of points.
- void **applyTransform** ()
apply the local transformation to the curve.
- ObjectGL* **copy** ()
- mutable BasicList<Point3Df> **list**

Protected Members

- int **psize**

Detailed Description

A class to hold a list of points.

Author(s):

Philippe Lavoie

Date:

29 Mars 1998

Member Function Documentation

3.29.1 PointListGL::PointListGL(const PointListGL &pl)

Copy constructor.

Parameters:

pl - list to copy

Author(s):

Philippe Lavoie

Date:

30 Mars 1998

3.29.2 PointListGL::PointListGL(const BasicList<Point3Df> &l)

Constructor from a list of points.

Parameters:

l - list of points

Author(s):

Philippe Lavoie

Date:

30 Mars 1998

3.29.3 void PointListGL::glObject() const [virtual]

Displays a list of points.

Author(s):

Philippe Lavoie

Date:

29 Mars 1998

Reimplemented from **ObjectGL**.

3.29.4 int PointListGL::read(istream &fin)

Reads a list of points.

Parameters:

fin - input file stream

Author(s):

Philippe Lavoie

Date:

29 Mars 1998

Reimplemented from **ObjectGL**.

3.29.5 int PointListGL::write(ofstream &fout) const

Writes a list of points.

Parameters:

fout - output file stream

Author(s):

Philippe Lavoie

Date:

29 Mars 1998

Reimplemented from **ObjectGL**.

3.29.6 void PointListGL::applyTransform()

apply the local transformation to the curve.

Apply the local transformation to the curve. This is necessary if you want to get the proper position for the control points before doing anymore processing on them.

Author(s):

Philippe Lavoie

Date:

23 September 1997

Reimplemented from **ObjectGL**.

The documentation for this class was generated from the following files:

- nurbsGL.hh
- nurbsGL.cc

3.30 RGBAf Struct Reference

a class to hold rgba floating point values.

```
#include <nurbs/nurbsGL.hh>
```

Public Members

- **RGBAf** ()
- **RGBAf** (const RGBAf& c)
- **RGBAf** (const Color& c)
- float **r**
- float **g**
- float **b**
- float **a**

Detailed Description

a class to hold rgba floating point values.

This holds the red, green, blue and alpha floating point values. These informations are needed by some renderer.

Author(s):

Philippe Lavoie

Date:

28 September 1997

The documentation for this struct was generated from the following file:

- nurbsGL.hh

3.31 SPointCurveGL Class Reference

A class to hold a control point from a NURBS curve or surface.

`#include <nurbs/nurbsGL.hh>`

Inherits **SPointGL**.

Public Members

- **SPointCurveGL** (int i, NurbsCurveSPf *c, int fix)
The constructor for a curve point object.
- **~SPointCurveGL** ()
- virtual void **glObject** () const
Displays a control point.
- virtual void **modify** (const HPoint3Df& v)
Modifies the surface point.
- void **updateOthers** ()
Updates the other control points.
- void **setStartEnd** (SPointCurveGL* s, int r)

Protected Members

- HPoint3Df **spoint**
 - NurbsCurveSPf* **curve**
 - SPointCurveGL* **start**
 - int **rows**
-

Detailed Description

A class to hold a control point from a NURBS curve or surface.

Author(s):

Philippe Lavoie

Date:

11 May 1998

Member Function Documentation

3.31.1 SPointCurveGL::SPointCurveGL(int i, NurbsCurveSPf *c,int fix)

The constructor for a curve point object.

Parameters:

- i* - the index of the control point
- c* - the pointer to the NURBS curve

Author(s):

Philippe Lavoie

Date:

20 September 1997

3.31.2 void SPointCurveGL::glObject() const [virtual]

Displays a control point.

Displays a control point on the surface of a NURBS curve or a NURBS surface.

Author(s):

Philippe Lavoie

Date:

12 May 1998

Reimplemented from **ObjectGL**.

3.31.3 void SPointCurveGL::modify(const HPoint3Df& v) [virtual]

Modifies the surface point.

Parameters:

- v* - modifies the point by this value

Author(s):

Philippe Lavoie

Date:

12 May 1998

Reimplemented from **CPointGL**.

3.31.4 void SPointCurveGL::updateOthers()

Updates the other control points.

Author(s):

Philippe Lavoie

Date:

12 May 1998

The documentation for this class was generated from the following files:

- nurbsGL.hh
- nurbsGL.cc

3.32 SPointGL Class Reference

A class to hold a control point from a NURBS surface.

`#include <nurbs/nurbsGL.hh>`

Inherits **CPointGL**.

Inherited by **SPointCurveGL**, **SPointHSurfaceGL** and **SPointSurfaceGL**.

Public Members

- virtual `~SPointGL ()`
- void `setFixEdit (int f)`
- int `fixEdit () const`

Protected Members

- **SPointGL** (`HPoint3Df& cp, int i, int j, int fix`)
- int `editFix`

Detailed Description

A class to hold a control point from a NURBS surface.

Author(s):

Philippe Lavoie

Date:

11 May 1998

The documentation for this class was generated from the following file:

- nurbsGL.hh

3.33 SPointHSurfaceGL Class Reference

a class to hold a HNURBS surface point.

```
#include <nurbs/nurbsGL.hh>
```

Inherits **SPointGL**.

Public Members

- **SPointHSurfaceGL** (int i, int j, HNurbsSurfaceSPf *s, ObjectListGL *sp, int fix)

The constructor for a surface point object.

- **~SPointHSurfaceGL** ()
- virtual void **glObject** () const

Displays a control point.

- virtual void **modify** (const HPoint3Df& v)

Modifies the surface point.

- void **updateOthers** ()

Updates the other control points.

- void **setStartEnd** (SPointHSurfaceGL* s, int r, int c=0)

Protected Members

- HPoint3Df **spoint**
- HNurbsSurfaceSPf* **surface**
- SPointHSurfaceGL* **start**
- ObjectListGL* **spoints**
- int **rows**
- int **cols**

Detailed Description

a class to hold a HNURBS surface point.

Author(s):

Philippe Lavoie

Date:11 May 1998

Member Function Documentation**3.33.1 SPointHSurfaceGL::SPointHSurfaceGL(int i, int j, HNurbsSurfaceSPf *s, ObjectListGL *sp,int fix)**

The constructor for a surface point object.

Parameters:

- i* - the row of the control point
- j* - the column of the control point
- c* - the pointer to the NURBS surface

Author(s):

Philippe Lavoie

Date:

20 September 1997

3.33.2 void SPointHSurfaceGL::glObject() const [virtual]

Displays a control point.

Displays a control point on the surface of a NURBS curve or a NURBS surface.

Author(s):

Philippe Lavoie

Date:

12 May 1998

Reimplemented from **ObjectGL**.

3.33.3 void SPointHSurfaceGL::modify(const HPoint3Df& v) [virtual]

Modifies the surface point.

Parameters:

v - modifies the point by this value

Returns:**Warning:****Author(s):**

Philippe Lavoie

Date:

12 May 1998

Reimplemented from **CPointGL**.

3.33.4 void SPointHSurfaceGL::updateOthers()

Updates the other control points.

Author(s):

Philippe Lavoie

Date:

12 May 1998

The documentation for this class was generated from the following files:

- nurbsGL.hh
- nurbsGL.cc

3.34 SPointSurfaceGL Class Reference

A class to hold a surface point.

```
#include <nurbs/nurbsGL.hh>
```

Inherits **SPointGL**.

Public Members

- **SPointSurfaceGL** (int i, int j, NurbsSurfaceSPf *s, ObjectListGL *sp, int fix)
The constructor for a surface point object.
- **~SPointSurfaceGL** ()
- virtual void **gObject** () const
Displays a control point.
- virtual void **modify** (const HPoint3Df& v)
Modifies the surface point.
- void **updateOthers** ()
Updates the other control points.
- void **setStartEnd** (SPointSurfaceGL* s, int r, int c=0)

Protected Members

- HPoint3Df **spoint**
- NurbsSurfaceSPf* **surface**
- SPointSurfaceGL* **start**
- ObjectListGL* **spoints**
- int **rows**
- int **cols**

Detailed Description

A class to hold a surface point.

Author(s):

Philippe Lavoie

Date:11 May 1998

Member Function Documentation**3.34.1 SPointSurfaceGL::SPointSurfaceGL(int i, int j, NurbsSurfaceSPf *s, ObjectListGL *sp,int fix)**

The constructor for a surface point object.

Parameters:

i - the row of the control point
j - the column of the control point
c - the pointer to the NURBS surface

Author(s):

Philippe Lavoie

Date:

20 September 1997

3.34.2 void SPointSurfaceGL::glObject() const [virtual]

Displays a control point.

Displays a control point on the surface of a NURBS curve or a NURBS surface.

Author(s):

Philippe Lavoie

Date:

12 May 1998

Reimplemented from **ObjectGL**.

3.34.3 void SPointSurfaceGL::modify(const HPoint3Df& v) [virtual]

Modifies the surface point.

Parameters:

v - modifies the point by this value

Author(s):

Philippe Lavoie

Date:

12 May 1998

Reimplemented from **CPointGL**.

3.34.4 void SPointSurfaceGL::updateOthers()

Updates the other control points.

Author(s):

Philippe Lavoie

Date:

12 May 1998

The documentation for this class was generated from the following files:

- nurbsGL.hh
- nurbsGL.cc

Index

- ~HNurbsSurface
 - HNurbsSurface, 18
- ~NurbsSurface
 - NurbsSurface, 148
- ~ObjectGL
 - ObjectGL, 208, 213
- ~ObjectListGL
 - ObjectListGL, 220
- ~ObjectRefListGL
 - ObjectRefListGL, 232
- activate
 - ObjectListGL, 222
- add
 - ObjectListGL, 221
- addLevel
 - HNurbsSurface, 19, 20
 - HNurbsSurfaceGL, 37
 - HNurbsSurfaceSP, 42, 43
- applyTransform
 - HNurbsSurfaceGL, 38
 - NurbsCurveGL, 125
 - NurbsSurfaceGL, 194
 - PointListGL, 257
- basisFun
 - NurbsCurve, 76
- basisFuns
 - NurbsCurve, 77
 - NurbsSurface, 150
- basisFunsU
 - NurbsSurface, 150
- basisFunsV
 - NurbsSurface, 151
- BoundingBoxGL, 5
 - BoundingBoxGL, 6
 - glObject, 6
- chordLengthParam
 - NurbsCurve, 114
- copy
 - HNurbsSurface, 20
 - HNurbsSurfaceSP, 43
- CPointGL, 7
 - glObject, 8
 - modifySym, 8
- created
 - MatrixRT, 56
- CtrlP
 - NurbsCurve, 115
 - NurbsSurface, 183
- deactivate
 - ObjectListGL, 223
- decompose
 - NurbsCurve, 101
 - NurbsSurface, 159
- decreaseLevelOfDetail
 - HNurbsSurfaceGL, 33
- degree
 - NurbsCurve, 115
- degreeElevate
 - NurbsSurface, 158
- degreeElevateU
 - NurbsSurface, 158
- degreeElevateV
 - NurbsSurface, 158
- degreeU
 - NurbsSurface, 183
- degreeV
 - NurbsSurface, 183
- degU
 - NurbsSurface, 183
- degV
 - NurbsSurface, 183

-
- derive
 - NurbsCurve, 73
 - derive3D
 - NurbsCurve, 72
 - deriveAt
 - HNurbsSurface, 21
 - NurbsCurve, 71, 72
 - NurbsSurface, 151
 - ParaCurve, 237
 - ParaSurface, 245
 - deriveAtH
 - HNurbsSurface, 21
 - NurbsCurve, 71
 - NurbsSurface, 152
 - ParaCurve, 237
 - ParaSurface, 245
 - dersBasisFuns
 - NurbsCurve, 78
 - deselect
 - ObjectListGL, 223
 - display
 - NurbsListGL, 136
 - ObjectListGL, 220
 - displayName
 - ObjectListGL, 221
 - drawAaImg
 - NurbsCurve, 111–113
 - drawImg
 - NurbsCurve, 111
 - extremum
 - ParaCurve, 241
 - ParaSurface, 249
 - findKnot
 - NurbsCurve, 80
 - findMult
 - NurbsCurve, 79
 - findMultSpan
 - NurbsCurve, 79
 - findMultU
 - NurbsSurface, 161
 - findMultV
 - NurbsSurface, 161
 - findSpan
 - NurbsCurve, 78
 - NurbsSurface, 159
 - findSpanU
 - NurbsSurface, 160
 - findSpanV
 - NurbsSurface, 160
 - firstD
 - NurbsCurve, 74
 - firstDn
 - NurbsCurve, 75
 - generateParallel
 - NurbsSurfaceSP, 200
 - getRemovalBnd
 - NurbsCurve, 80
 - glNewList
 - ObjectGL, 209, 213
 - globalApproxErrBnd
 - NurbsCurve, 86
 - globalInterp
 - NurbsCurve, 95
 - NurbsSurface, 152
 - globalInterpD
 - NurbsCurve, 97
 - globalInterpH
 - NurbsCurve, 96, 97
 - NurbsSurface, 153
 - globalSurfApprox
 - NurbsSurface, 181
 - globalSurfInterpXY
 - NurbsSurface, 180, 181
 - glObject
 - BoundingBoxGL, 6
 - CPointGL, 8
 - HCPPointGL, 11
 - KnotGL, 48
 - NurbsCpolygonGL, 57
 - NurbsGL, 133
 - NurbsListGL, 135
 - NurbsSpolygonGL, 137
 - ObjectListGL, 220
 - PointGL, 254
 - PointListGL, 256
 - SPointCurveGL, 261
 - SPointHSurfaceGL, 265
 - SPointSurfaceGL, 268
 - glTransform
-

- OpenGL, 209, 213
- gluNurbs
 - HNurbsSurfaceGL, 34
 - NurbsCurveGL, 122
 - NurbsSurfaceGL, 191
- gordonSurface
 - NurbsSurface, 182
- goTo
 - ObjectListGL, 225
- goToActive
 - ObjectListGL, 225
- goToNext
 - ObjectListGL, 226
- goToNextActive
 - ObjectListGL, 226
- goToPrevious
 - ObjectListGL, 226
- goToPreviousActive
 - ObjectListGL, 227
- HCPPointGL, 10
 - glObject, 11
 - modify, 11
- hideAllObjects
 - ObjectListGL, 230
- HNurbsSurface, 12
 - ~HNurbsSurface, 18
 - addLevel, 19, 20
 - copy, 20
 - deriveAt, 21
 - deriveAtH, 21
 - HNurbsSurface, 16–18
 - initBase, 23
 - isoCurveU, 24
 - isoCurveV, 25
 - maxLevel, 27
 - modifies, 21
 - movePointOffset, 22
 - read, 25, 26
 - refineKnots, 27
 - refineKnotU, 28
 - refineKnotV, 28
 - scale, 22
 - setFixedOffsetVector, 28
 - setVariableOffsetVector, 29
 - splitUV, 18, 19
 - updateLevels, 24
 - updateSurface, 23
 - write, 26
- HNurbsSurfaceGL, 30
 - addLevel, 37
 - applyTransform, 38
 - decreaseLevelOfDetail, 33
 - gluNurbs, 34
 - HNurbsSurfaceGL, 32, 33
 - modifyPoint, 38
 - point, 34
 - read, 35
 - resetBoundingBox, 34
 - resetCPoints, 35
 - resetPolygon, 35
 - selectHigherLevel, 37
 - selectHighestLevel, 37
 - selectLowerLevel, 37
 - selectNextPatch, 36
 - selectPrevPatch, 36
 - setSym, 38
 - write, 36
- HNurbsSurfaceSP, 40
 - addLevel, 42, 43
 - copy, 43
 - modOnlySurfCPby, 45
 - modSurfCPby, 44
 - read, 44
 - updateLevels, 42
 - updateMaxU, 45
 - updateMaxV, 45
 - updateSurface, 42
- hpointAt
 - NurbsCurve, 70
 - ParaCurve, 237
 - ParaSurface, 245
- init
 - NurbsCurveArray, 117
 - NurbsSurfaceArray, 187
- initBase
 - HNurbsSurface, 23
- intersectWith
 - ParaSurface, 250, 251
- isoCurveU
 - HNurbsSurface, 24

- NurbsSurface, 164
- isoCurveV
 - HNurbsSurface, 25
 - NurbsSurface, 164
- jumpToNext
 - ObjectListGL, 227
- jumpToPrevious
 - ObjectListGL, 227
- Knot
 - NurbsCurve, 115
- KnotGL, 47
 - glObject, 48
- KnotU
 - NurbsSurface, 182
- KnotV
 - NurbsSurface, 182
- leastSquares
 - NurbsCurve, 83, 85
 - NurbsSurface, 153
- leastSquaresH
 - NurbsCurve, 84
- length
 - NurbsCurve, 98
- lengthF
 - NurbsCurve, 100
- lengthIn
 - NurbsCurve, 99
- makeCircle
 - NurbsCurve, 100, 101
- makeFromRevolution
 - NurbsSurface, 157
- makeLine
 - NurbsCurve, 101
- makeSphere
 - NurbsSurface, 157
- Material, 49
 - Material, 50
- MatrixRT, 51
 - created, 56
 - MatrixRT, 52, 53
 - operator=, 55
 - rotate, 53
- rotateDeg, 55
- rotateDegXYZ, 55
- rotateXYZ, 53
- scale, 54
- translate, 54
- maxAtU
 - NurbsSurfaceSP, 201
- maxAtU_
 - NurbsSurfaceSP, 203
- maxAtUV
 - NurbsSurfaceSP, 201
- maxAtV
 - NurbsSurfaceSP, 201
- maxAtV_
 - NurbsSurfaceSP, 203
- maxKnot
 - NurbsCurve, 78
 - ParaCurve, 238
- maxLevel
 - HNurbsSurface, 27
- maxU
 - NurbsSurfaceSP, 203
- maxV
 - NurbsSurfaceSP, 203
- mergeKnots
 - NurbsSurface, 163
- mergeKnotU
 - NurbsSurface, 163
- mergeKnotV
 - NurbsSurface, 164
- mergeKnotVector
 - NurbsCurve, 82
- mergeOf
 - NurbsCurve, 102
- minDist2
 - ParaCurve, 238
 - ParaSurface, 245
- minDist2b
 - ParaSurface, 246
- minDist2xy
 - ParaSurface, 247
- minDistX
 - ParaCurve, 239
- minDistY
 - ParaCurve, 239
- minDistZ

- ParaCurve, 240
- minKnot
 - NurbsCurve, 78
 - ParaCurve, 238
- modCP
 - NurbsSurface, 175
- modCPby
 - NurbsSurface, 175
- modifies
 - HNurbsSurface, 21
- modify
 - HCPPointGL, 11
 - SPointCurveGL, 261
 - SPointHSurfaceGL, 265
 - SPointSurfaceGL, 268
- modifyPoint
 - HNurbsSurfaceGL, 38
 - NurbsCurveGL, 125
 - NurbsSurfaceGL, 194
- modifySym
 - CPointGL, 8
- modKnotU
 - NurbsSurface, 175
- modKnotV
 - NurbsSurface, 175
- modOnlySurfCP
 - NurbsSurfaceSP, 201
- modOnlySurfCPby
 - HNurbsSurfaceSP, 45
 - NurbsCurveSP, 128
 - NurbsSurfaceSP, 201
- modSurfCP
 - NurbsSurfaceSP, 200
- modSurfCPby
 - HNurbsSurfaceSP, 44
 - NurbsSurfaceSP, 200
- modV
 - NurbsSurface, 175
- movePoint
 - NurbsCurve, 103–106
 - NurbsSurface, 175–177
- movePointOffset
 - HNurbsSurface, 22
- n
 - NurbsSurfaceArray, 188
- normal
 - NurbsCurve, 73
 - NurbsSurface, 152
- NurbsCpolygonGL, 57
 - glObject, 57
- NurbsCurve, 59
 - basisFun, 76
 - basisFuns, 77
 - chordLengthParam, 114
 - CtrlP, 115
 - decompose, 101
 - degree, 115
 - derive, 73
 - derive3D, 72
 - deriveAt, 71, 72
 - deriveAtH, 71
 - dersBasisFuns, 78
 - drawAaImg, 111–113
 - drawImg, 111
 - findKnot, 80
 - findMult, 79
 - findMultSpan, 79
 - findSpan, 78
 - firstD, 74
 - firstDn, 75
 - getRemovalBnd, 80
 - globalApproxErrBnd, 86
 - globalInterp, 95
 - globalInterpD, 97
 - globalInterpH, 96, 97
 - hpointAt, 70
 - Knot, 115
 - leastSquares, 83, 85
 - leastSquaresH, 84
 - length, 98
 - lengthF, 100
 - lengthIn, 99
 - makeCircle, 100, 101
 - makeLine, 101
 - maxKnot, 78
 - mergeKnotVector, 82
 - mergeOf, 102
 - minKnot, 78
 - movePoint, 103–106
 - normal, 73
 - NurbsCurve, 66, 67

- operator(), 69
- operator=, 68
- read, 107, 108
- refineKnotVector, 82
- removeKnot, 81
- removeKnotsBound, 81
- reset, 68
- resize, 68
- splitAt, 102
- tessellate, 113
- transform, 103
- write, 108
- writePS, 109
- writePSp, 109
- writeVRML, 110
- NurbsCurveArray, 116
 - init, 117
 - NurbsCurveArray, 117
 - read, 118
 - resize, 117
 - write, 118
 - writePS, 119
 - writePSp, 119
- NurbsCurveGL, 121
 - applyTransform, 125
 - gluNurbs, 122
 - modifyPoint, 125
 - operator=, 123
 - point, 122
 - read, 124
 - resetBoundingBox, 123
 - resetCPoints, 124
 - resetKnots, 124
 - write, 125
- NurbsCurveSP, 127
 - modOnlySurfCPby, 128
 - updateMaxU, 128
- NurbsGL, 130
 - glObject, 133
 - NurbsGL, 132
 - operator=, 132
 - readNurbsObject, 133
- NurbsListGL, 135
 - display, 136
 - glObject, 135
 - resetDisplayFlags, 136
- NurbsSpolygonGL, 137
 - glObject, 137
- NurbsSurface, 139
 - ~NurbsSurface, 148
 - basisFuns, 150
 - basisFunsU, 150
 - basisFunsV, 151
 - CtrlP, 183
 - decompose, 159
 - degreeElevate, 158
 - degreeElevateU, 158
 - degreeElevateV, 158
 - degreeU, 183
 - degreeV, 183
 - degU, 183
 - degV, 183
 - deriveAt, 151
 - deriveAtH, 152
 - findMultU, 161
 - findMultV, 161
 - findSpan, 159
 - findSpanU, 160
 - findSpanV, 160
 - globalInterp, 152
 - globalInterpH, 153
 - globalSurfApprox, 181
 - globalSurfInterpXY, 180, 181
 - gordonSurface, 182
 - isoCurveU, 164
 - isoCurveV, 164
 - KnotU, 182
 - KnotV, 182
 - leastSquares, 153
 - makeFromRevolution, 157
 - makeSphere, 157
 - mergeKnots, 163
 - mergeKnotU, 163
 - mergeKnotV, 164
 - modCP, 175
 - modCPby, 175
 - modKnotU, 175
 - modKnotV, 175
 - modV, 175
 - movePoint, 175–177
 - normal, 152
 - NurbsSurface, 146, 147

- operator(), 149
- operator=, 148
- P, 183
- print, 167
- read, 165, 166
- refineKnots, 161
- refineKnotU, 162
- refineKnotV, 162
- resize, 148
- resizeKeep, 149
- skinU, 154
- skinV, 154
- surfMeshParams, 179
- sweep, 155, 156
- tessellate, 172
- transform, 174
- transpose, 178
- U, 183
- V, 183
- write, 165, 166
- writePOVRAY, 168–170
- writePS, 172
- writePSp, 173
- writeRIB, 171
- writeVRML, 167, 168
- NurbsSurfaceArray, 185
 - init, 187
 - n, 188
 - NurbsSurfaceArray, 186
 - operator=, 187
 - operator[], 186
 - resize, 186
 - rsize, 188
 - S, 188
 - size, 188
- NurbsSurfaceGL, 189
 - applyTransform, 194
 - gluNurbs, 191
 - modifyPoint, 194
 - operator=, 191, 192
 - point, 191
 - read, 193
 - resetBoundingBox, 192
 - resetCPoints, 192
 - resetKnots, 193
 - setImage, 195
 - setSym, 195
 - write, 193
- NurbsSurfaceSP, 197
 - generateParallel, 200
 - maxAtU, 201
 - maxAtU_, 203
 - maxAtUV, 201
 - maxAtV, 201
 - maxAtV_, 203
 - maxU, 203
 - maxV, 203
 - modOnlySurfCP, 201
 - modOnlySurfCPby, 201
 - modSurfCP, 200
 - modSurfCPby, 200
 - okMax, 202
 - refineKnots, 199
 - refineKnotV, 199
 - surfP, 202
 - updateMaxU, 202
 - updateMaxUV, 202
 - updateMaxV, 202
- OpenGL, 204
 - ~OpenGL, 208, 213
 - glNewList, 209, 213
 - glTransform, 209, 213
 - OpenGL, 208, 212
 - operator=, 209, 213
 - read, 210, 211, 214, 215
 - setName, 212, 216
 - typeName, 212, 216
 - write, 210, 211, 214, 215
- ObjectListGL, 217
 - ~ObjectListGL, 220
 - activate, 222
 - add, 221
 - deactivate, 223
 - deselect, 223
 - display, 220
 - displayName, 221
 - glObject, 220
 - goTo, 225
 - goToActive, 225
 - goToNext, 226
 - goToNextActive, 226

- goToPrevious, 226
- goToPreviousActive, 227
- hideAllObjects, 230
- jumpToNext, 227
- jumpToPrevious, 227
- ObjectListGL, 220
- read, 228
- remove, 222
- reset, 221
- select, 223
- transformBy, 224
- transformTo, 223
- viewAllObjects, 230
- write, 228
- writePOVRAY, 229
- writeRIB, 229
- ObjectRefGL, 231
- ObjectRefListGL, 232
- ~ObjectRefListGL, 232
- refList, 233
- remove, 233
- okMax
 - NurbsSurfaceSP, 202
- operator()
 - NurbsCurve, 69
 - NurbsSurface, 149
 - ParaCurve, 236
 - ParaSurface, 245
- operator=
 - MatrixRT, 55
 - NurbsCurve, 68
 - NurbsCurveGL, 123
 - NurbsGL, 132
 - NurbsSurface, 148
 - NurbsSurfaceArray, 187
 - NurbsSurfaceGL, 191, 192
 - ObjectGL, 209, 213
- operator[]
 - NurbsSurfaceArray, 186
- P
 - NurbsSurface, 183
- ParaCurve, 235
 - deriveAt, 237
 - deriveAtH, 237
 - extremum, 241
 - hpointAt, 237
 - maxKnot, 238
 - minDist2, 238
 - minDistX, 239
 - minDistY, 239
 - minDistZ, 240
 - minKnot, 238
 - operator(), 236
 - ParaCurve, 236
 - pointAt, 237
- ParaSurface, 243
 - deriveAt, 245
 - deriveAtH, 245
 - extremum, 249
 - hpointAt, 245
 - intersectWith, 250, 251
 - minDist2, 245
 - minDist2b, 246
 - minDist2xy, 247
 - operator(), 245
 - ParaSurface, 245
 - pointAt, 245
 - projectOn, 248
 - writeVRML, 251, 252
- point
 - HNurbsSurfaceGL, 34
 - NurbsCurveGL, 122
 - NurbsSurfaceGL, 191
- pointAt
 - ParaCurve, 237
 - ParaSurface, 245
- PointGL, 253
 - glObject, 254
- PointListGL, 255
 - applyTransform, 257
 - glObject, 256
 - PointListGL, 256
 - read, 257
 - write, 257
- print
 - NurbsSurface, 167
- projectOn
 - ParaSurface, 248
- read
 - HNurbsSurface, 25, 26

- HNurbsSurfaceGL, 35
- HNurbsSurfaceSP, 44
- NurbsCurve, 107, 108
- NurbsCurveArray, 118
- NurbsCurveGL, 124
- NurbsSurface, 165, 166
- NurbsSurfaceGL, 193
- ObjectGL, 210, 211, 214, 215
- ObjectListGL, 228
- PointListGL, 257
- readNurbsObject
 - NurbsGL, 133
- refineKnots
 - HNurbsSurface, 27
 - NurbsSurface, 161
 - NurbsSurfaceSP, 199
- refineKnotU
 - HNurbsSurface, 28
 - NurbsSurface, 162
- refineKnotV
 - HNurbsSurface, 28
 - NurbsSurface, 162
 - NurbsSurfaceSP, 199
- refineKnotVector
 - NurbsCurve, 82
- refList
 - ObjectRefListGL, 233
- remove
 - ObjectListGL, 222
 - ObjectRefListGL, 233
- removeKnot
 - NurbsCurve, 81
- removeKnotsBound
 - NurbsCurve, 81
- reset
 - NurbsCurve, 68
 - ObjectListGL, 221
- resetBoundingBox
 - HNurbsSurfaceGL, 34
 - NurbsCurveGL, 123
 - NurbsSurfaceGL, 192
- resetCPoints
 - HNurbsSurfaceGL, 35
 - NurbsCurveGL, 124
 - NurbsSurfaceGL, 192
- resetDisplayFlags
 - NurbsListGL, 136
- resetKnots
 - NurbsCurveGL, 124
 - NurbsSurfaceGL, 193
- resetPolygon
 - HNurbsSurfaceGL, 35
- resize
 - NurbsCurve, 68
 - NurbsCurveArray, 117
 - NurbsSurface, 148
 - NurbsSurfaceArray, 186
- resizeKeep
 - NurbsSurface, 149
- RGBAf, 259
- rotate
 - MatrixRT, 53
- rotateDeg
 - MatrixRT, 55
- rotateDegXYZ
 - MatrixRT, 55
- rotateXYZ
 - MatrixRT, 53
- rsize
 - NurbsSurfaceArray, 188
- S
 - NurbsSurfaceArray, 188
- scale
 - HNurbsSurface, 22
 - MatrixRT, 54
- select
 - ObjectListGL, 223
- selectHigherLevel
 - HNurbsSurfaceGL, 37
- selectHighestLevel
 - HNurbsSurfaceGL, 37
- selectLowerLevel
 - HNurbsSurfaceGL, 37
- selectNextPatch
 - HNurbsSurfaceGL, 36
- selectPrevPatch
 - HNurbsSurfaceGL, 36
- setFixedOffsetVector
 - HNurbsSurface, 28
- setImage
 - NurbsSurfaceGL, 195

- setName
 - ObjectGL, 212, 216
- setSym
 - HNurbsSurfaceGL, 38
 - NurbsSurfaceGL, 195
- setVariableOffsetVector
 - HNurbsSurface, 29
- skinU
 - NurbsSurface, 154
- skinV
 - NurbsSurface, 154
- splitAt
 - NurbsCurve, 102
- splitUV
 - HNurbsSurface, 18, 19
- SPointCurveGL, 260
 - glObject, 261
 - modify, 261
 - SPointCurveGL, 261
 - updateOthers, 261
- SPointGL, 263
- SPointHSurfaceGL, 264
 - glObject, 265
 - modify, 265
 - SPointHSurfaceGL, 265
 - updateOthers, 266
- SPointSurfaceGL, 267
 - glObject, 268
 - modify, 268
 - SPointSurfaceGL, 268
 - updateOthers, 269
- surfMeshParams
 - NurbsSurface, 179
- surfP
 - NurbsSurfaceSP, 202
- sweep
 - NurbsSurface, 155, 156
- size
 - NurbsSurfaceArray, 188
- tessellate
 - NurbsCurve, 113
 - NurbsSurface, 172
- transform
 - NurbsCurve, 103
 - NurbsSurface, 174
- transformBy
 - ObjectListGL, 224
- transformTo
 - ObjectListGL, 223
- translate
 - MatrixRT, 54
- transpose
 - NurbsSurface, 178
- typeName
 - ObjectGL, 212, 216
- U
 - NurbsSurface, 183
- updateLevels
 - HNurbsSurface, 24
 - HNurbsSurfaceSP, 42
- updateMaxU
 - HNurbsSurfaceSP, 45
 - NurbsCurveSP, 128
 - NurbsSurfaceSP, 202
- updateMaxUV
 - NurbsSurfaceSP, 202
- updateMaxV
 - HNurbsSurfaceSP, 45
 - NurbsSurfaceSP, 202
- updateOthers
 - SPointCurveGL, 261
 - SPointHSurfaceGL, 266
 - SPointSurfaceGL, 269
- updateSurface
 - HNurbsSurface, 23
 - HNurbsSurfaceSP, 42
- V
 - NurbsSurface, 183
- viewAllObjects
 - ObjectListGL, 230
- write
 - HNurbsSurface, 26
 - HNurbsSurfaceGL, 36
 - NurbsCurve, 108
 - NurbsCurveArray, 118
 - NurbsCurveGL, 125
 - NurbsSurface, 165, 166
 - NurbsSurfaceGL, 193

- ObjectGL, 210, 211, 214, 215
- ObjectListGL, 228
- PointListGL, 257
- writePOVRAY
 - NurbsSurface, 168–170
 - ObjectListGL, 229
- writePS
 - NurbsCurve, 109
 - NurbsCurveArray, 119
 - NurbsSurface, 172
- writePSP
 - NurbsCurve, 109
 - NurbsCurveArray, 119
 - NurbsSurface, 173
- writeRIB
 - NurbsSurface, 171
 - ObjectListGL, 229
- writeVRML
 - NurbsCurve, 110
 - NurbsSurface, 167, 168
 - ParaSurface, 251, 252