Computational Graphics: Lecture 25

The CVD-Lab Team

Thu, May 8, 2014

Outline: NURBS

1. B-splines with `lar-cc`
2. NURBS (Non-Uniform Rational B-Splines) curves
3. Transfinite B-splines
4. Transfinite NURBS

B-splines with `lar-cc`

Summary of previous lecture

B-splines discussed here are called non-uniform because different spline segments may correspond to different intervals in parameter space, unlike uniform B-splines.

Summary of previous lecture

B-splines discussed here are called non-uniform because different spline segments may correspond to different intervals in parameter space, unlike uniform B-splines.

The basis polynomials, and consequently the spline shape and the other properties, are defined by a non-decreasing sequence of real numbers called the knot sequence:

$$t_0 \leq t_1 \leq \cdots \leq t_n,$$

Summary of previous lecture

B-splines discussed here are called non-uniform because different spline segments may correspond to different intervals in parameter space, unlike uniform B-splines.

The basis polynomials, and consequently the spline shape and the other properties, are defined by a non-decreasing sequence of real numbers called the knot sequence:

$$t_0 \leq t_1 \leq \cdots \leq t_n,$$

The knot sequence is used to define the basis polynomials which blend the control points.

Summary of previous lecture

B-splines discussed here are called non-uniform because different spline segments may correspond to different intervals in parameter space, unlike uniform B-splines.

The basis polynomials, and consequently the spline shape and the other properties, are defined by a non-decreasing sequence of real numbers called the knot sequence:

$$t_0 \leq t_1 \leq \cdots \leq t_n,$$

The knot sequence is used to define the basis polynomials which blend the control points.
In particular, each subset of $k + 2$ adjacent knot values is used to compute a basis polynomial of degree k.

Summary of previous lecture

B-splines discussed here are called non-uniform because different spline segments may correspond to different intervals in parameter space, unlike uniform B-splines.

The basis polynomials, and consequently the spline shape and the other properties, are defined by a non-decreasing sequence of real numbers called the knot sequence:

$$t_0 \leq t_1 \leq \cdots \leq t_n,$$

The knot sequence is used to define the basis polynomials which blend the control points.
In particular, each subset of $k+2$ adjacent knot values is used to compute a basis polynomial of degree k.

Notice that some subsequent knots may coincide. In this case we speak of multiplicity of the knots.

Graphs of the basis polynomials

```
from splines import *

knots = [0,0,0,1,1,2,2,3,3,4,4,4]
ncontrols = 9
degree = 2
obj = larMap(BSPLINEBASIS(degree)(knots)(ncontrols))(larDom(knots))
```

Graphs of the basis polynomials

```
from splines import *

knots = [0,0,0,1,1,2,2,3,3,4,4,4]
ncontrols = 9
degree = 2
obj = larMap(BSPLINEBASIS(degree)(knots)(ncontrols))(larDom(knots))

funs = TRANS(obj[0])
var = AA(CAT)(larDom(knots)[0])
cells = larDom(knots)[1]
```

Graphs of the basis polynomials

```
from splines import *

knots = [0,0,0,1,1,2,2,3,3,4,4,4]
ncontrols = 9
degree = 2
obj = larMap(BSPLINEBASIS(degree)(knots)(ncontrols))(larDom(knots))

funs = TRANS(obj[0])
var = AA(CAT)(larDom(knots)[0])
cells = larDom(knots)[1]

graphs =  [[TRANS([var,fun]),cells] for fun in funs]
graph = STRUCT(CAT(AA(MKPOLS)(graphs)))
VIEW(graph)
VIEW(STRUCT(MKPOLS(graphs[0]) + MKPOLS(graphs[-1])))
```

Graphs of the basis polynomials

```
knots = [0,0,0,1,1,2,2,3,3,4,4,4]
ncontrols = 9
degree = 2
```

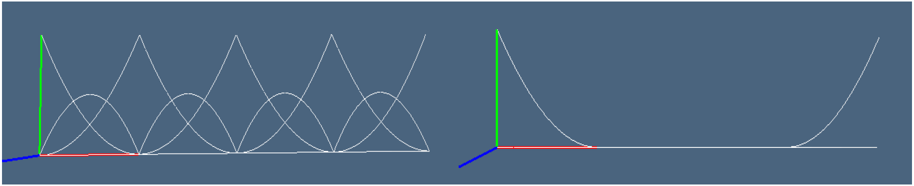


Figure : Graphs of the above B-spline basis

```
picture = STRUCT(CAT(AA(MKPOLS)(graphs)))    # each graph is a LAR model
VIEW(picture)
VIEW(STRUCT(MKPOLS(graphs[0]) + MKPOLS(graphs[-1])))
```

Graphs of the basis polynomials

It may be interesting to note that the value stored in `obj` is the LAR 2-model of a curve (look at `obj[1]`) embedded in n-dimensional space, with $n = 9$ (the number of control points).

```
[[1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
 [0.9384765625, 0.060546875, 0.0009765625, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
 [0.87890625, 0.1171875, 0.00390625, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
   ...
 [0.0, 0.0, 0.31640625, 0.4921875, 0.19140625, 0.0, 0.0, 0.0, 0.0],
 [0.0, 0.0, 0.2822265625, 0.498046875, 0.2197265625, 0.0, 0.0, 0.0, 0.0],
 [0.0, 0.0, 0.25, 0.5, 0.25, 0.0, 0.0, 0.0, 0.0],
 [0.0, 0.0, 0.2197265625, 0.498046875, 0.2822265625, 0.0, 0.0, 0.0, 0.0],
   ...
 [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.00390625, 0.1171875, 0.87890625],
 [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0009765625, 0.060546875, 0.9384765625],
 [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0]]
```

Every coordinate provides the discretised values of one of the blending functions, i.e.~the values of one B-spline basis function.

Domain partitioning

```python
""" Domain decomposition for 1D bspline maps """
def larDom(knots,tics=32):
    domain = knots[-1]-knots[0]
    return larIntervals([tics*int(domain)])([domain])
```

Domain partitioning

```
""" Domain decomposition for 1D bspline maps """
def larDom(knots,tics=32):
    domain = knots[-1]-knots[0]
    return larIntervals([tics*int(domain)])([domain])

knots = [0,0,0,1,1,2,2,3,3,4,4,4]
print larDom(knots),
>>> [[[0.0], [0.03125], [0.0625], ..., [3.9375], [3.96875], [4.0]],
    [[0, 1], [1, 2], [2, 3], ..., [125, 126], [126, 127], [127, 128]]]
```

Remark

The value returned from `larDom(knots` is a LAR model

NURBS (Non-Uniform Rational B-Splines) curves

NURBS

Rational Non-Uniform B-Splines are normally denoted as NURB splines or simply as NURBS.

These splines are very important for graphics and CAD applications

NURBS properties

- Rational curves and splines are invariant with respect to affine and projective transformations

NURBS properties

- Rational curves and splines are invariant with respect to affine and projective transformations
- NURBS represent exactly the conic sections, i.e.~circles, ellipses, parabolæ, iperbolæ

NURBS properties

- Rational curves and splines are invariant with respect to affine and projective transformations
- NURBS represent exactly the conic sections, i.e.~circles, ellipses, parabolæ, iperbolæ
- Rational B-splines are very flexible: DOFs with degree, control points, knots and weights)

NURBS properties

- Rational curves and splines are invariant with respect to affine and projective transformations
- NURBS represent exactly the conic sections, i.e.~circles, ellipses, parabolæ, iperbolæ
- Rational B-splines are very flexible: DOFs with degree, control points, knots and weights)
- allow for local variation of “parametrization velocity“, via properly modifying the knots

NURBS properties

- Rational curves and splines are invariant with respect to affine and projective transformations
- NURBS represent exactly the conic sections, i.e.˜circles, ellipses, parabolæ, iperbolæ
- Rational B-splines are very flexible: DOFs with degree, control points, knots and weights)
- allow for local variation of "parametrization velocity", via properly modifying the knots
- easy modification of sampling density of spline points along segments with higher or lower curvature

Definition

Rational B-splines of arbitrary degree

A rational B-spline segment $\mathbf{R}_i(t)$ is defined as the projection from the origin on the hyperplane $x_{d+1} = 1$ of a polynomial B-spline segment $\mathbf{P}_i(u)$ in $\mathbb{E}^{d+1}$ homogeneous space.

Definition

Rational B-splines of arbitrary degree

A rational B-spline segment $\mathbf{R}_i(t)$ is defined as the projection from the origin on the hyperplane $x_{d+1} = 1$ of a polynomial B-spline segment $\mathbf{P}_i(u)$ in $\mathbb{E}^{d+1}$ homogeneous space.

$$\mathbf{R}_i(t) = \sum_{\ell=0}^{k} w_{i-\ell}\, \mathbf{p}_{i-\ell} \frac{B_{i-\ell,k+1}(t)}{w(t)} = \sum_{\ell=0}^{k} \mathbf{p}_{i-\ell} N_{i-\ell,k+1}(t)$$

with $k \leq i \leq m$, $t \in [t_i, t_{i+1})$, and

$$w(t) = \sum_{\ell=0}^{k} w_{i-\ell} B_{i-\ell,k+1}(t),$$

where $N_{i,h}(t)$ is the non-uniform rational B-basis function of initial value t_i and order h.

NURBS Implementation

NURB splines can be computed as non-uniform B-splines by using homogeneous control points, and finally by dividing the Cartesian coordinate maps times the homogeneous one.

This approach is used in the NURBS implementation given in Paoluzzi's book, in `pyplasm` and in `lar-cc`

```
""" Alias for the pyplasm definition (too long :o) """
NURBS = RATIONALBSPLINE      # in pyplasm
TNURBS = TRATIONALBSPLINE     # in lar-cc (only)
```

NURBS Implementation

NURB splines can be computed as non-uniform B-splines by using homogeneous control points, and finally by dividing the Cartesian coordinate maps times the homogeneous one.

This approach is used in the NURBS implementation given in Paoluzzi's book, in `pyplasm` and in `lar-cc`

```
""" Alias for the pyplasm definition (too long :o) """
NURBS = RATIONALBSPLINE      # in pyplasm
TNURBS = TRATIONALBSPLINE    # in lar-cc (only)
```

A more efficient and numerically stable variation of the Cox and de Boor formula for the rational case is given by Farin (88), Curves and Surfaces for Computer Aided Geometric Design, p.˜196.

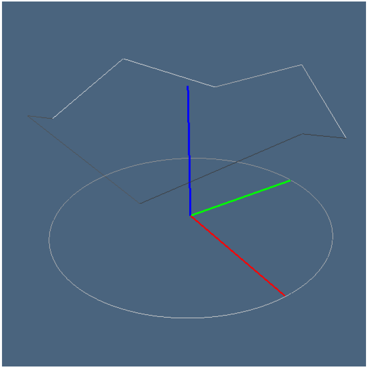
NURBS canonical example

Exact generation of circle as NURBS curve

```
from splines import *

knots = [0,0,0,1,1,2,2,3,3,4,4,4]
_p=math.sqrt(2)/2.0
controls = [[-1,0,1], [-_p,_p,_p], [0,1,1], [_p,_p,_p],[1,0,1], [_p,-_p,_p]
            [0,-1,1], [-_p,-_p,_p], [-1,0,1]]

nurbs = NURBS(2)(knots)(controls)
obj = larMap(nurbs)(larDom(knots))
VIEW(STRUCT( MKPOLS(obj) + [POLYLINE(controls)] ))
```

NURBS canonical example

Circle 2D exactly implemented as a 9-point NURBS curve



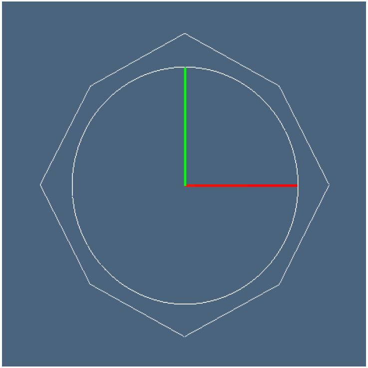


Figure : The curve represents a circle exactly, but it is not exactly parametrized in the circle's arc length

Transfinite B-splines

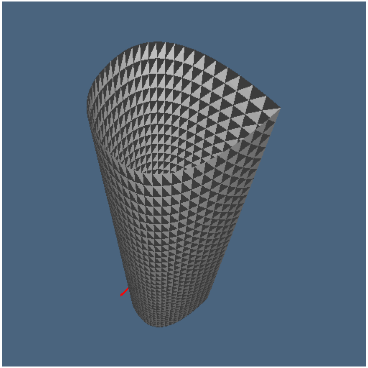
Example: periodic B-spline curve . . .

Transfinite surface from Bezier control curves and periodic B-spline curve

```
from splines import *

b1 = BEZIER(S1)([[0,1,0],[0,1,5]])
b2 = BEZIER(S1)([[0,0,0],[0,0,5]])
b3 = BEZIER(S1)([[1,0,0],#[2,-1,2.5],
    [1,0,5]])
b4 = BEZIER(S1)([[1,1,0],[1,1,5]])
b5 = BEZIER(S1)([[0,1,0],[0,1,5]])

controls = [b1,b2,b3,b4,b5]
knots = [0,1,2,3,4,5,6,7]                    # periodic B-spline
knots = [0,0,0,1,2,3,3,3]                    # non-periodic B-spline

tbspline = TBSPLINE(S2)(2)(knots)(controls)
dom = larModelProduct([larDomain([10]),larDom(knots)])
dom = larIntervals([32,48],'simplex')([1,3])
obj = larMap(tbspline)(dom)
VIEW(STRUCT( MKPOLS(obj) ))
VIEW(SKEL_1(STRUCT( MKPOLS(dom) )))
```

Example: periodic B-spline curve ...

Transfinite surface from Bezier control curves and periodic B-spline curve



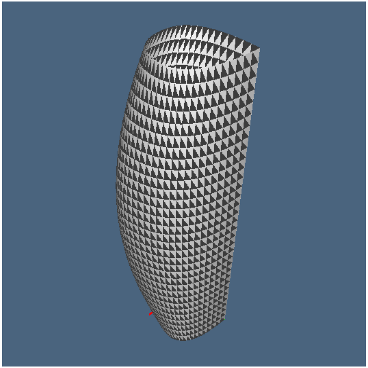


Figure : Try your own variations: e.g. a case handle ...

Transfinite NURBS

Transfinite NURBS interface

The `TNURBS` function, that is by definite an alias to `TRATIONALBSPLINE`, is used to define a NURBS surface by blending 1D curves, or a NURBS solid by blending 2D surfaces, and so on.

For an example of use, just look at the test example `test05.py`, where a cylinder surface with unit radius and height is generated by blending 9 vertical unit segments via the unit 1D circle as NURBS curve.

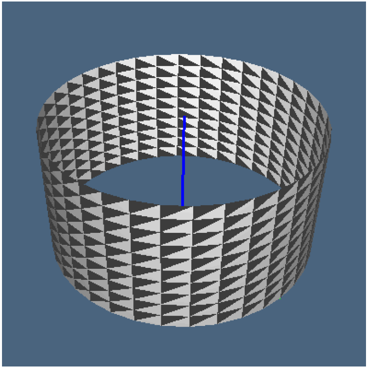
Example: transfinite cylinder surface

generated below has both radius and height equal to 1

```
knots = [0,0,0,1,1,2,2,3,3,4,4,4]
_p=math.sqrt(2)/2.0
controls = [[-1,0,1], [-_p,_p,_p], [0,1,1], [_p,_p,_p],[1,0,1], [_p,-_p,_p],
            [0,-1,1], [-_p,-_p,_p], [-1,0,1]]
c1 = BEZIER(S1)([[-1,0,0,1],[-1,0,1,1]])
c2 = BEZIER(S1)([[-_p,_p,0,_p],[-_p,_p,_p,_p]])
c3 = BEZIER(S1)([[0,1,0,1],[0,1,1,1]])
c4 = BEZIER(S1)([[_p,_p,0,_p],[_p,_p,_p,_p]])
c5 = BEZIER(S1)([[1,0,0,1],[1,0,1,1]])
c6 = BEZIER(S1)([[_p,-_p,0,_p],[_p,-_p,_p,_p]])
c7 = BEZIER(S1)([[0,-1,0,1],[0,-1,1,1]])
c8 = BEZIER(S1)([[-_p,-_p,0,_p],[-_p,-_p,_p,_p]])
c9 = BEZIER(S1)([[-1,0,0,1],[-1,0,1,1]])
controls = [c1,c2,c3,c4,c5,c6,c7,c8,c9]

tnurbs = TNURBS(S2)(2)(knots)(controls)
dom = larModelProduct([larDomain([10]),larDom(knots)])
dom = larIntervals([10,36],'simplex')([1,4])
obj = larMap(tnurbs)(dom)
VIEW(STRUCT( MKPOLS(obj) ))
```

Example: transfinite cylinder surface

generated below has both radius and height equal to 1



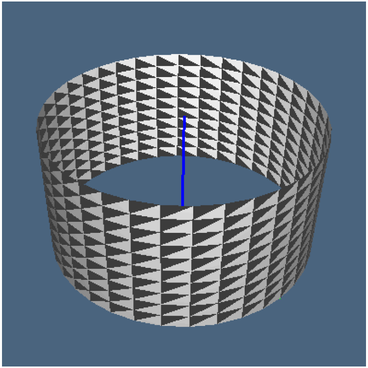


Figure : Try your own variations: e.g. a case handle ...

References

GP4CAD book