Drawing hypergraphs using NURBS curves

Ronny Bergmann

Institute of Mathematics
University of Lübeck

presentation at the
Kolloquium der Institute für Mathematik und Informatik

November 25th, 2009

Content

Introduction

Things done before

As a student of computer science I thought about how to draw graphs.

⇒ student research project: **Gravel**, an editor for graphs

- focus on scientific illustrations and convenient editing
- export especially for TEX (vector graphics)

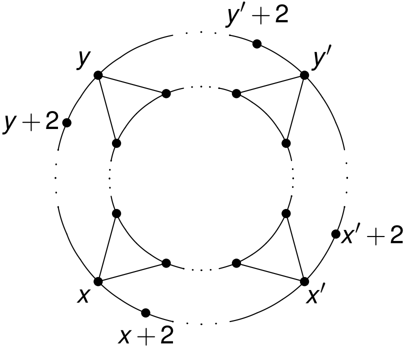


Figure: graph generated with Gravel, from Schiermeyer et al.

Introduction

What am I going to talk about?

working on Gravel I noticed

- some other editors for graphs available
- none supported editing of hypergraphs

$\Rightarrow$ drawing hyperedges with other tools

$\Rightarrow$ no convenient editing

$\Rightarrow$ **diploma thesis** about drawing and editing hypergraphs

Drawing a hypergraph

How do scientists usually draw hypergraphs?

common drawing: **the subset standard**

- nodes are drawn as dots (as in graphs)
- usually nodes are placed first
- hyperedge is a subset of nodes

$\Rightarrow$ drawn as a “cloud” surrounding all its nodes → blackboard

$\Rightarrow$ handle these “smooth” curves (on a computer)

Requirements of curves

...for drawing and editing of a hyperedge

a curve representing a hyperedge should be

- numerically stable
- easy to create
- convenient and interactive editible, e.g. by affin-linear trasnformations
- edited just locally (if most of the curve is already done)
- **periodic**, so that it
  - looks “smooth”, without rough edges
  - outlines all nodes

common in computer graphics: B-splines & NURBS
but they are usually not periodic.

B-splines – definition

a very short introduction of B-splines

a B-spline curve $B(u), u \in [a,b]$ consists of

- a polynomial degree d
  $\Rightarrow$ smoothness
- $n+1$ control points $P_0,\ldots,P_n$
  $\Rightarrow$ form
- a knot vector $(t_i)_{i=0}^m$ with $m = n+d+1$ (nondecreasing)
  $\Rightarrow$ recursive B-spline basis functions $N_{i,d}(u), i = 0,\ldots,n$
  - piecewise nonnegative polynomial functions
  - model distribution of the P_i “along the curve”

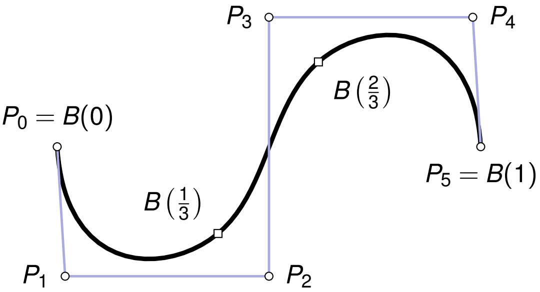
with those we get the **B-spline curve**

$$B(u) = \sum_{i=0}^{n} N_{i,d}(u)P_i, \quad u \in [t_d, t_{m-d}] = [a,b]$$

B-splines – Example

a simple B-spline curve

- degree 3
- $n = 5 \Rightarrow$ six control points
- knot vector $(t_i)_{i=0}^{9} = \begin{pmatrix} 0 & 0 & 0 & 0 & \frac{1}{3} & \frac{2}{3} & 1 & 1 & 1 & 1 \end{pmatrix}$



P_1 has local influence: $[t_1, t_5) = [0, \frac{2}{3})$

B-splines – properties

nice features we obtain by using B-spline curves

- P_0 and P_n are interpolated, if $t_0 = t_d = a$ and $t_{m-d} = t_m = b$
- P_i only has **local** influence to the curve
- Let p be the maximum multiplicity of a knot from t_{d+1} to t_{m-d-1}

$\Rightarrow$ $B(u)$ is "smooth", because it is $d-p$-times differentiable

try to reach $p = 1$!

with $B(u)$ get a curve that

- is numerical stable
- has easy computable derivatives
- is a piecewise polynomial

But we can't form circles!

NURBS

Non **U**niform **R**ational **B**-**S**plines

additional weight $w_i > 0$ for each control point P_i

$\Rightarrow$ different influence of each P_i to the curve

$\Rightarrow$ new NURBS basis functions $R_{i,d}(u)$, piecewise rational polynoms (based on knot vector and weights)

with that modification

- NURBS curve $C(u) = \sum_{i=0}^{n} R_{i,d}(u) P_i$ is a piecewise rational polynomial
- circles are possbile

most important

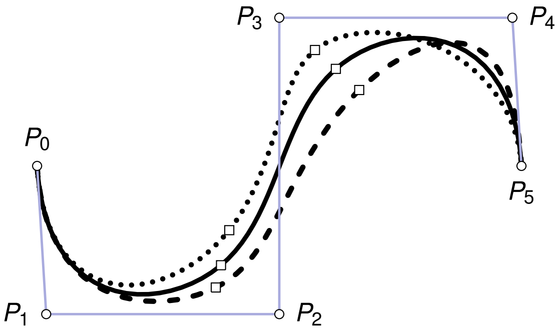
- B-spline properties also apply for NURBS
- using B-spline algorithms by calculation in **homogeneous coordinates**

NURBS – example

showing different influences of a control point to the curve

variation of weight w_3 from the last example with

- $w_3 = 2$ (dotted line)
- $w_3 = 1$ (solid line) $\Rightarrow$ B-spline curve
- $w_3 = \frac{1}{2}$ (dashed line)



Periodic NURBS

How do we model a shape with NURBS?

idea: close the curve to a periodic one
$\Rightarrow C^{(k)}(a) = C^{(k)}(b),\ k = 0, \dots, d-p$

change definition of

- $N_{i,d}(u)$ repeat themselves periodically (shifted)
- change $P_i \Rightarrow$ periodic sequence

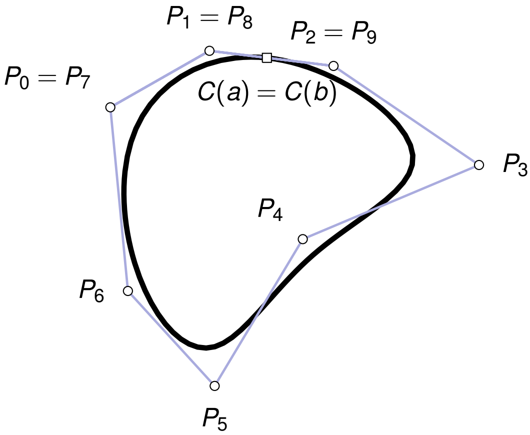
$\Rightarrow$ periodic curve

- no endpoint interpolation!
- **all** other properties remain.
- adapt algorithms, so they keep the curve periodic!

Periodic NURBS – example

What does s shape look like?

- smooth curve
- surrounds exactely one area iff it is injective.

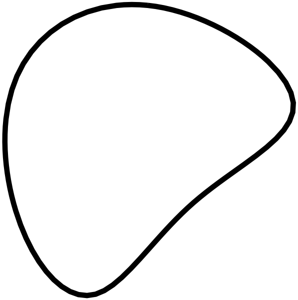


- degree 3
- $C^{(k)}(a) = C^{(k)}(b)$, $k = 0, 1, 2$

Periodic NURBS – example

What does s shape look like?

- smooth curve
- surrounds exactely one area iff it is injective.



- degree 3
- $C^{(k)}(a) = C^{(k)}(b)$, $k = 0, 1, 2$
- start/end could be moved anywhere
- if any point on the curve may be moved

$\Rightarrow$ just the curve is needed in the GUI

Algorithms for NURBS

All your NURBS need are... these.

with these (periodic) NURBS the following algorithms were implemented

- calculate $C^{(k)}(u), k = 0, \ldots, m$ ($k = 0$ for drawing)
- extract arbitraty subcurve (ignoring start/end)
- affin linear transformations (affecting subcurve or whole curve)
- projection onto the curve (essential for interactive editing)
- moving arbitrary point on curve anywhere

except for projection, all these algorithms are well known.
$\Rightarrow$ small adjustments to fit periodic NURBS
$\Rightarrow$ for projection: circular clipping algorithm (Chen et al., 2008)

Drawing hypergraphs using NURBS curves

Ronny Bergmann

Institute of Mathematics
University of Lübeck

Excursion: Distance betwenn a point and a NURBS Curve

Distance and projection on the NURBS Curve

Projection means, for a NURBS curve $C(u)$ (with control Points and weight P_i, w_i) and a point p
Compute the point $C(\hat{u})$ with shortest distance to p

The algorithm is important for interactive editing!
(point inversion)

It's idea is based on clipping using a circles around p, that get smaller and smaller

Main idea of the algorithm

divide and conquer with Newton iteration

main idea: circles around p of small radii, cut everything outside

1. Split the Curve into small parts (knot insertion $\Rightarrow$ beziér curves)
2. init circle around p running through $C(a)$ or $C(b)$
3. For each part decide whether
   - the part is outside the circle $\Rightarrow$ discard
   - the part is inside the circle, then
     - a) it has exactly one minimum $\Rightarrow$ Newton iteration $\Rightarrow$ new circle
     - b) it has more than one $\Rightarrow$ split in the middle, use endpoints as new circle radii
4. among all local minima from the newton iteration is the global minimum

Looking at the distance

calculating a function for the distance

The product

$$f(u) = (C(u) - p)(C(u) - p)^T$$

is the **objective squared distance function**.
$f(u)$ is a beziér curve iff $C(u)$ is a beziér curve (K. Mørken,1991)
degree of $f(u)$: $2d$

$f(u)$ has real valued control points!

$\Rightarrow$ use $f(u)$ to determine whether a curve is outise a circle around p (convex hull property)

$\Rightarrow$ if the control points have one "turning point" (variation diminishing property)
$\Rightarrow$ exactly one minimum

Projection – example

finding the shortest distance in a few slides

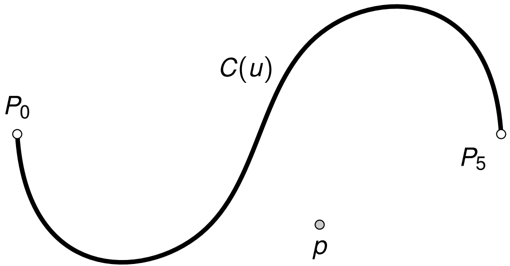


Figure: projection

At first: split
Initial circle K_1

1. $B_1 \Rightarrow$ Circle K_2
   $\Rightarrow$ discard B_1
2. B_2 is inside K_2
   $\Rightarrow$ newton $\Rightarrow p_C$
3. new radius with p_C
   $\Rightarrow K_3$
4. discard B_3, it is outside K_3

only one newton iteration
result is p_C

Projection – example

finding the shortest distance in a few slides

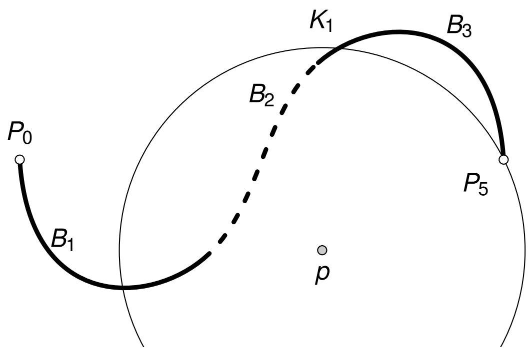


Figure: projection

At first: split
Initial circle K_1

1. $B_1 \Rightarrow$ Circle K_2 $\Rightarrow$ discard B_1
2. B_2 is inside K_2 $\Rightarrow$ newton $\Rightarrow p_C$
3. new radius with p_C $\Rightarrow K_3$
4. discard B_3, it is outside K_3

only one newton iteration
result is p_C

Projection – example

finding the shortest distance in a few slides

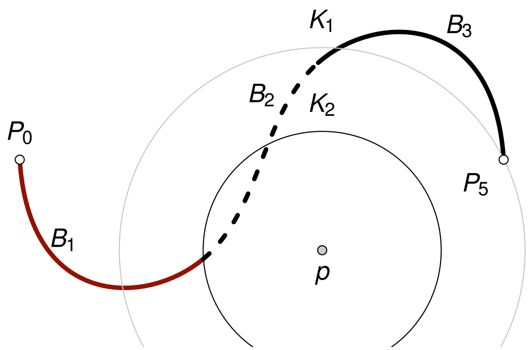


Figure: projection

At first: split
Initial circle K_1

1. $B_1 \Rightarrow$ Circle K_2
   $\Rightarrow$ discard B_1
2. B_2 is inside K_2
   $\Rightarrow$ newton $\Rightarrow p_C$
3. new radius with p_C
   $\Rightarrow K_3$
4. discard B_3, it is outside K_3

only one newton iteration
result is p_C

Projection – example

finding the shortest distance in a few slides

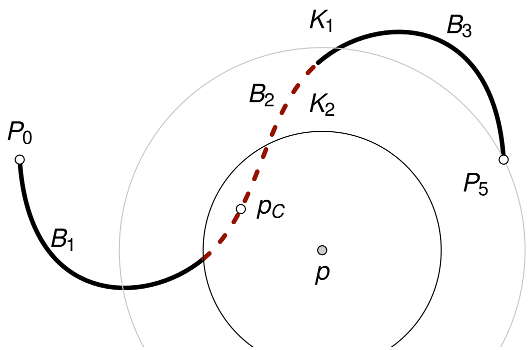


Figure: projection

At first: split
Initial circle K_1

1. $B_1 \Rightarrow$ Circle K_2 $\Rightarrow$ discard B_1
2. B_2 is inside K_2 $\Rightarrow$ newton $\Rightarrow p_C$
3. new radius with p_C $\Rightarrow K_3$
4. discard B_3, it is outside K_3

only one newton iteration
result is p_C

Projection – example

finding the shortest distance in a few slides

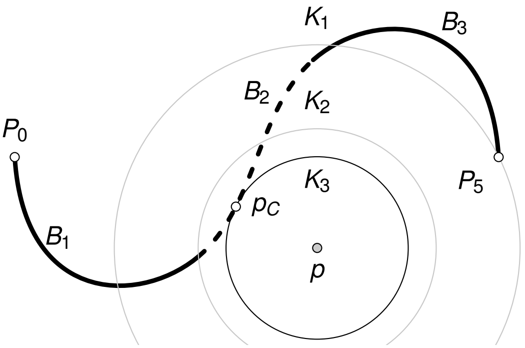


Figure: projection

At first: split
Initial circle K_1

1. $B_1 \Rightarrow$ Circle K_2
   $\Rightarrow$ discard B_1
2. B_2 is inside K_2
   $\Rightarrow$ newton $\Rightarrow p_C$
3. new radius with p_C
   $\Rightarrow K_3$
4. discard B_3, it is outside K_3

only one newton iteration
result is p_C

Projection – example

finding the shortest distance in a few slides

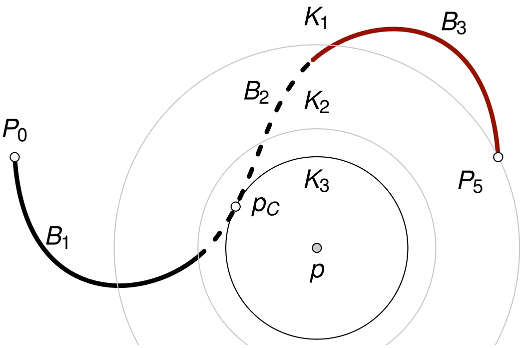


Figure: projection

At first: split
Initial circle K_1

1. $B_1 \Rightarrow$ Circle K_2 $\Rightarrow$ discard B_1
2. B_2 is inside K_2 $\Rightarrow$ newton $\Rightarrow p_C$
3. new radius with p_C $\Rightarrow K_3$
4. discard B_3, it is outside K_3

only one newton iteration
result is p_C

Projection – example

finding the shortest distance in a few slides

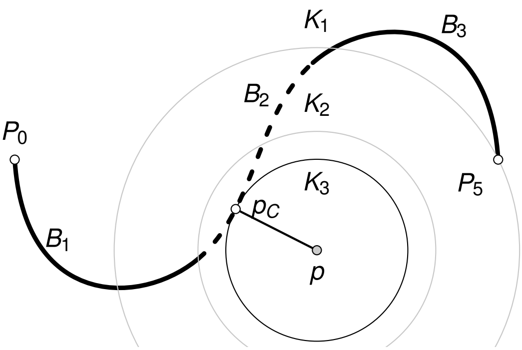


Figure: projection

At first: split
Initial circle K_1

1. $B_1 \Rightarrow$ Circle K_2 $\Rightarrow$ discard B_1
2. B_2 is inside K_2 $\Rightarrow$ newton $\Rightarrow p_C$
3. new radius with p_C $\Rightarrow K_3$
4. discard B_3, it is outside K_3

only one newton iteration
result is p_C

The hyperedge shape

What should a drawing of a hyperedge look like? Part I

hypergraph $\mathscr{H} = (V, \mathscr{E})$, with nodes $v_i \in V$ and hyperedges $E_i \in \mathscr{E} \subset \mathscr{P}(V)\backslash\emptyset$

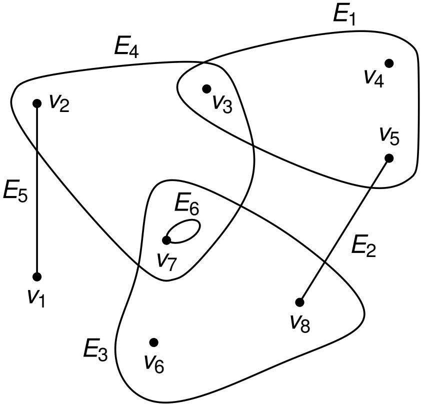


Figure: a Gravel export of a hypergraph by C.Berge

types of shapes

- loops – e.g. E_6
- iff $|E| = 2|$, simple curve joining the nodes, e.g. E_5
- periodic curve enclosing only its nodes, e.g. E_3

The hyperedge shape – decorations

What should a drawing of a hyperedge look like? Part II

additional attributes for the curve of a shape

- solid, dashed, dotted,...
- line width
- color
- label (cf. last slide)

and a margin δ:

shortest distance from node borders of each $v \in E$ to the curve

with $\delta > 0$ no node “touches” the curve
with $\delta > \alpha \in \mathbb{R}^+$ we have a margin inside the shape

blackboard

The hyperedge shape – construction

Creating a shape for an hyperedge.

Using periodic NURBS curves, we can create shapes:

- circles
- interpolation through user defined points
- convex hull based on interpolation and margin

and modify them globally or locally by

- scaling, rotation, translation
- move control points or positions on the curve
- replace parts

The hyperedge shape – validation

Did you forget including a node?

a hyperedge shape can be validated:

- Are all nodes $v \in E$ inside the shape?
- Are all others outside?
- Is the margin big enough?

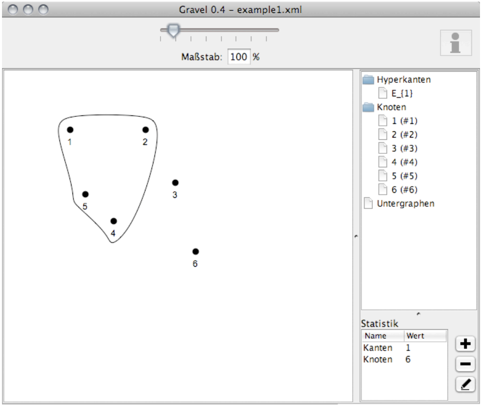
some criteria can't be checked (up to now?)

- minimization of crossings
- simplicity and other aesthetic criteria

Gravel – editing graphs and hypergraphs

So how can you use that now?

all the presented elements are implemented in **Gravel**, an editor for graphs and hypergraphs



Summary

Everything in short again

- hyperedges in the subset standard require periodic curves
- using NURBS and their algorithms
- extended to periodic NURBS

$\Rightarrow$ interactive editing

- hyperedge shape as formal definition of the hyperedge drawing
- easy creation and modification of a shape
- validation of the hyperedge shape (mostly) possible

$\Rightarrow$ a first editor for hypergraphs

future plans

What's next?

Gravel is available at gravel.darkmoonwolf.de (though in german only), the complete application is available as

- jar-file
- Mac OS X Application package
- source files

future plans are

- internationalization (using Java i18n)
- an algorithm API for
  - graph and hypergraph drawing algorithms
  - educative presentations of well known algorithms
  - stepwise execution of algorithms
- more basic shapes for hyperedges
- ...

One final example

TEX-Export using a TikZ picture in LATEX

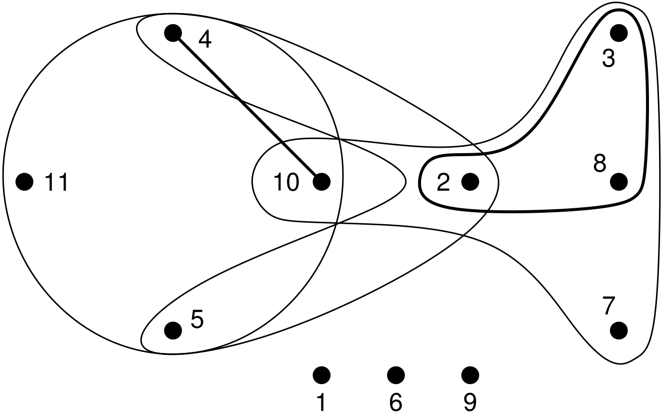


Figure: A competition hypergraph from Sonntag and Teichert

The End

Thanks for your attention.

Are there any questions?