Bézier Curves

Prof. Dr. Markus Gross

ETH zürich

Overview

- Coordinate Systems
- Bernstein Polynomials
- Bézier Curves – Properties
- Derivatives
- Piecewise Curves

Literature

- Gerald Farin: ***Curves and Surfaces for Computer Aided Geometric Design***. Third ed., Academic Press, 1992
- Gerald Farin: ***NURB Curves and Surfaces***. A K Peters, 1995
- Wolfgang Böhm et al.: ***A Survey of Curve and Surface Methods in CAGD***. Computer Aided Geometric Design 1, pp. 1-60, 1984
- Carl deBoor: ***A Practical Guide to Splines***. Springer, 1978
- Charles Michelli: ***Mathematical Aspects of Geometric Modeling***. SIAM, Philadelphia, 1995
- Christoph Hoffmann: ***Geometric and Solid Modeling. An Introduction***. Morgan Kaufmann, 1989
- R. Barthels, J. Beatty, A. Barsky: ***An Introduction to Splines for Use in Computer Graphics and Geometric Modeling***. Morgan Kaufmann, 1987
- A. Rockwood, P. Chambers: ***Interactive Curves and Surfaces***. Morgan Kaufmann, 1996

Local Coordinate Systems

- Vectors and Points **bold**: e.g. ***x***, ***y***

$$\boldsymbol{x} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}; \quad \boldsymbol{u} = \begin{bmatrix} u \\ v \end{bmatrix}$$

- Curve ***x***(*u*) as a map of the 1D parameter space *u* into 2D or 3D

$$\boldsymbol{x}(u) = (x(u), y(u), z(u))^T$$

Local Coordinate Systems

- Surface $\mathbf{x}(u,v)$ as a map of a subregion of (u,v) into $\mathbf{E}^2$ or $\mathbf{E}^3$

$$\mathbf{x}(u,v) = (x(u,v), y(u,v), z(u,v))^T$$

- Subdivision of parameter space into disjoint segments (knots):

$$u_0 < u_1 < \cdots < u_p$$

- Surfaces are subdivided by so-called ***knotlines***:

$$u_0 < u_1 < \cdots < u_p \quad \text{and} \quad v_0 < v_1 < \cdots < v_q$$

Bézier Curves

- $\boldsymbol{x}(t)$ = $\boldsymbol{p}(t)$ given by a Bernstein basis expansion:

$$\boldsymbol{x}(t) = \boldsymbol{b}_0 B_0^n(t) + \ldots + \boldsymbol{b}_n B_n^n(t)$$

- Bernstein polynomial of degree n:

$$B_i^n(t) = \binom{n}{i} t^i (1-t)^{n-i}$$

$$i < 0, i > n: \quad B_i^n(t) \equiv 0$$

- Binomial coefficients:

$$\binom{n}{i} = \begin{cases} \dfrac{n!}{i!(n-i)!} & 0 \leq i \leq n \\ 0 & else \end{cases}$$

JAVA-Applet

- Bernstein polynomial:
  - Global support
  - Positive definite
  - Partition of unity
  - Different degrees

Construction and Properties

- Cubic curve (n = 3):

$$\mathbf{x}(t) = \mathbf{b}_0 (1-t)^3 + 3\mathbf{b}_1 t(1-t)^2 + 3\mathbf{b}_2 t^2 (1-t) + \mathbf{b}_3 t^3$$

  - Coefficients $b_0,\ldots,b_n$ are called Bézier-points or control points.
  - Set of control points defines the so-called ***control polygon***
- Properties of Bernstein polynomials:
  - Partition of unity
  - Positivity (positive definite)
  - Recursion
  - Symmetry

JAVA-Applet

- The parametric Bézier Curve:
  - Cubic curves
  - piecewise definitions
  - continuity
  - design property

Construction and Properties

Distinguish between degree (highest order of the polynomial) and order=degree + 1

- Properties of Bézier-Curves:
  - ***affine invariance***: affine transform of all points on the curve is accomplished by the affine transform of its control points.
  - ***convex hull property***: the curve lies in the convex hull of its control polygon.

$$conv(P) := \{ \sum_{i=1}^{n} \lambda_i \mathbf{p_i} \,|\, \lambda_i \geq 0, \sum_{i=1}^{n} \lambda_i = 1 \}$$

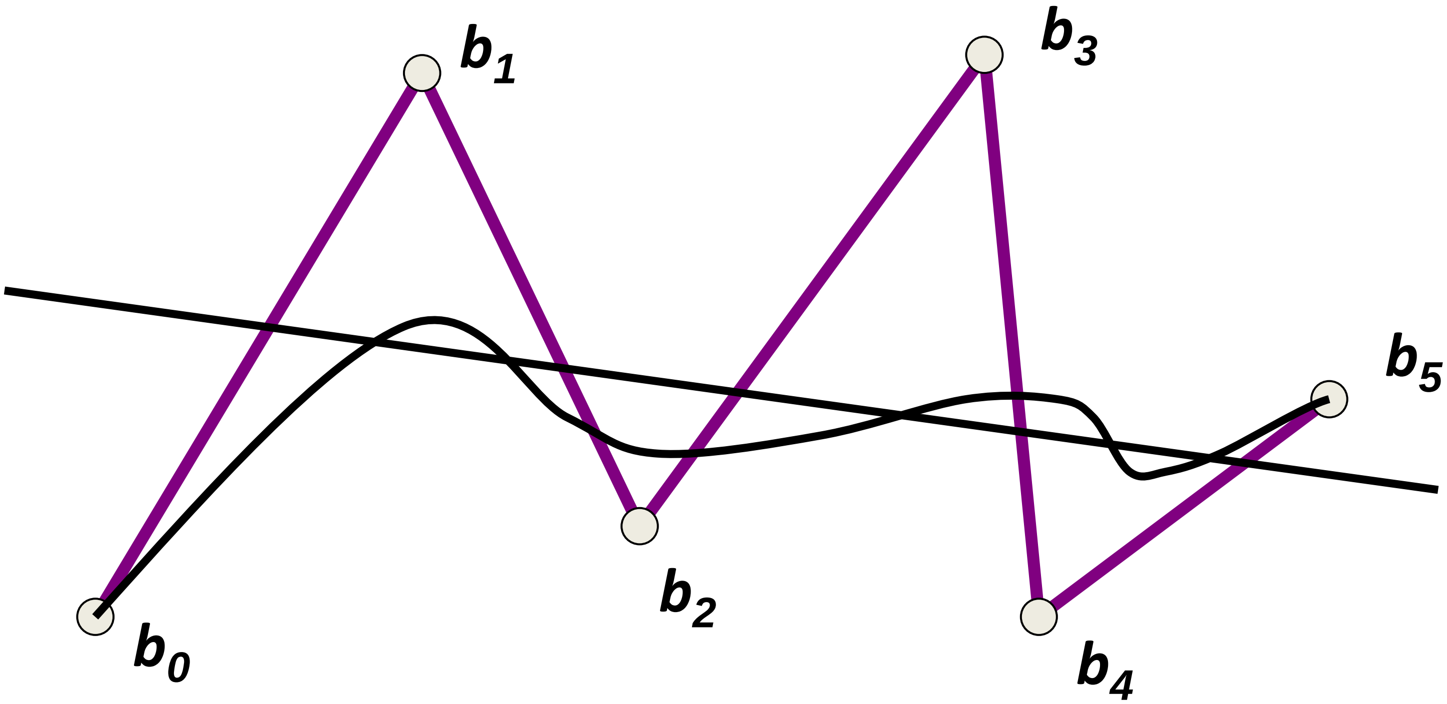
Construction and Properties

- Properties of Bézier-Curves:
  - ***design property***: Control polygon gives a rough sketch of the curve.
  - ***endpoint interpolation***: Since

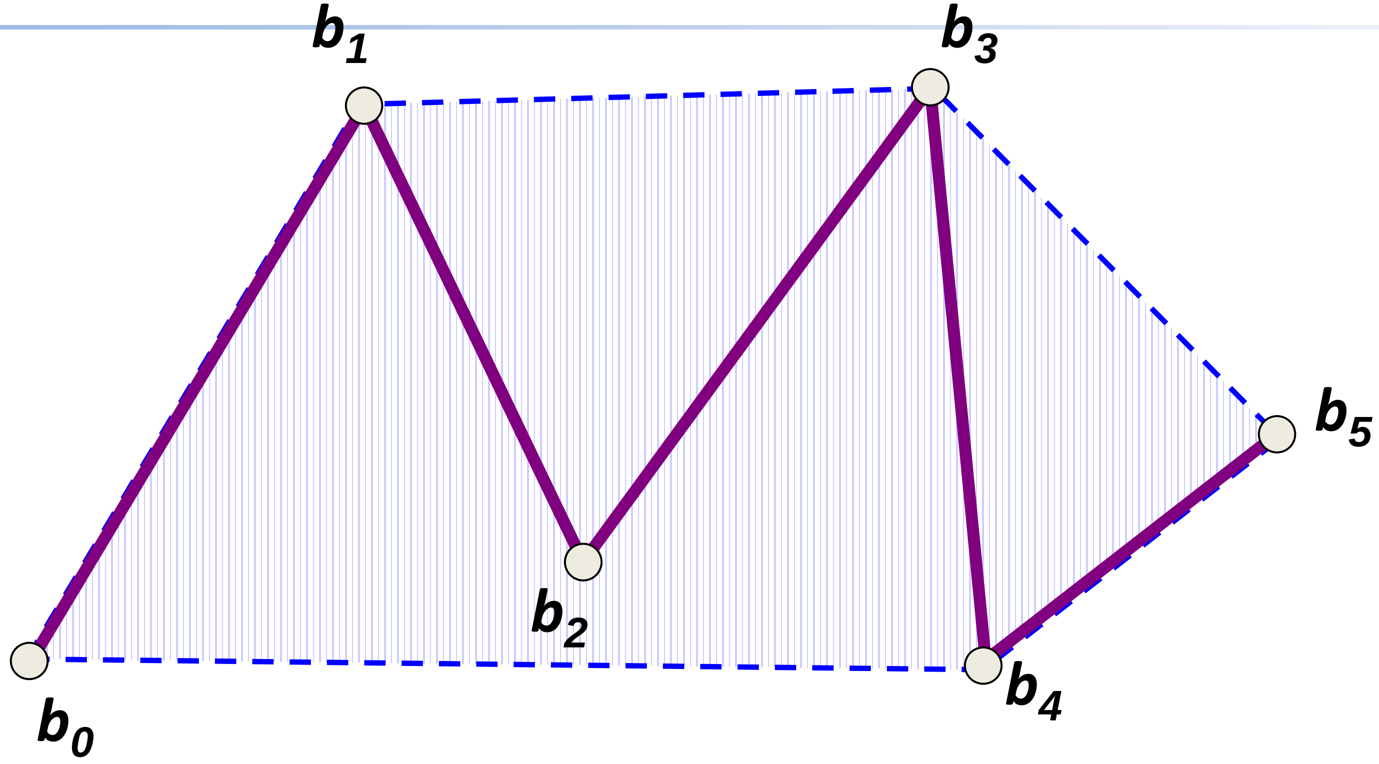
    $$B_0^n(0) = B_n^n(1) = 1$$

    the curve interpolates the endpoints $\boldsymbol{b}_0$ and $\boldsymbol{b}_n$.
  - ***variation diminishing property***: The maximum number of intersections of a line with the curve is less or equal to the number of intersections with its control polygon.

Variation Diminishing Property



Convex Hull Property



deCasteljau Algorithm

- Let $\boldsymbol{b_0}$, $\boldsymbol{b_1}$, $\boldsymbol{b_2}$ be 3 control points:

$$\boldsymbol{b}_0^1(t) = (1-t)\boldsymbol{b}_0 + t\,\boldsymbol{b}_1$$
$$\boldsymbol{b}_1^1(t) = (1-t)\boldsymbol{b}_1 + t\,\boldsymbol{b}_2$$
$$\boldsymbol{b}_0^2(t) = (1-t)\boldsymbol{b}_0^1(t) + t\,\boldsymbol{b}_1^1(t)$$

- We obtain: $\boldsymbol{b}_0^2(0) = \boldsymbol{b}_0\,,\ \boldsymbol{b}_0^2(1) = \boldsymbol{b}_2$

- Insert: $$\boldsymbol{b}_0^2(t) = (1-t)^2\,\boldsymbol{b}_0 + 2t(1-t)\boldsymbol{b}_1 + t^2\,\boldsymbol{b}_2$$

deCasteljau Algorithm

- Recursive computation of a point on the curve using a systolic array:
  - Given: $n+1$ control points $\boldsymbol{b_0}, \boldsymbol{b_1}, \ldots, \boldsymbol{b_n}$
  - Recursion:

$$\boldsymbol{b}_i^r(t) = (1-t)\boldsymbol{b}_i^{r-1}(t) + t\,\boldsymbol{b}_{i+1}^{r-1}(t)$$
$$\boldsymbol{b}_i^0(t) = \boldsymbol{b}_i$$
$$r = 1,\ldots,n \quad i = 0,\ldots,n-r$$

$\Rightarrow$ ***Point on the Bézier curve with***

deCasteljau Algorithm

- Algorithm computes a triangular representation:

$$
\begin{array}{llll}
\mathbf{b}_0 & & & \\
\mathbf{b}_1 & \mathbf{b}_0^1 & & \\
\mathbf{b}_2 & \mathbf{b}_1^1 & \mathbf{b}_0^2 & \\
\mathbf{b}_3 & \mathbf{b}_2^1 & \mathbf{b}_1^2 & \mathbf{b}_0^3
\end{array}
$$

$O(n^2)$ costs

$\Rightarrow$ successive linear interpolation, “corner cutting”

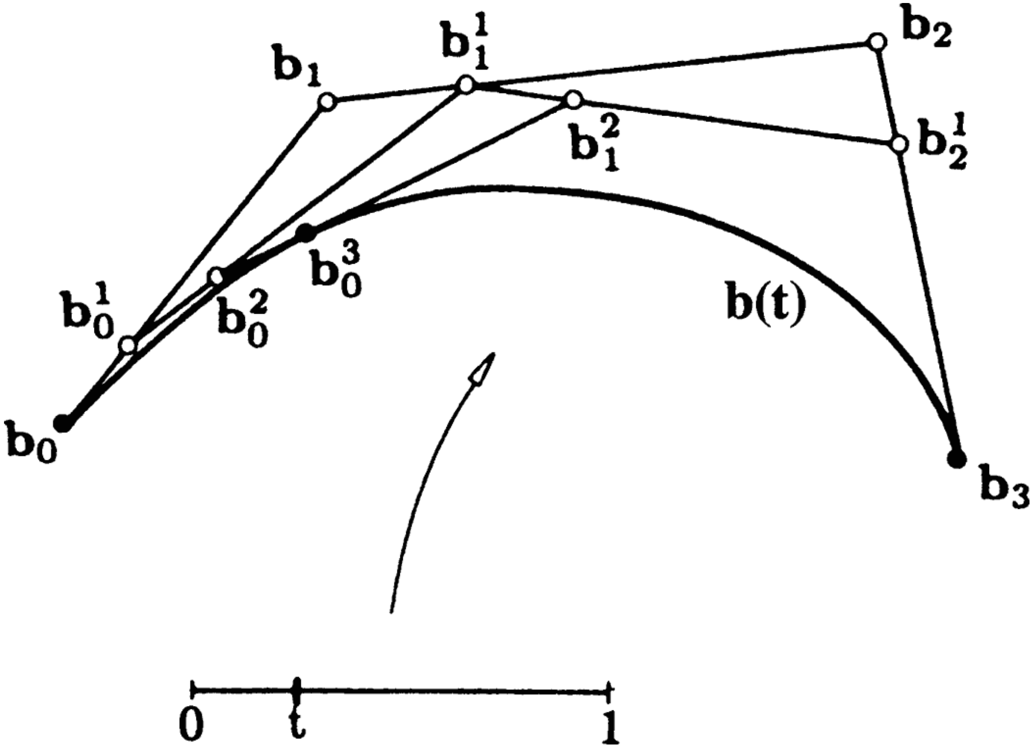
deCasteljau Algorithm

- A planar cubic Bézier curve at $t = ½$:

The student might reflect the situation, where control points are given with $\mathbf{b}_i = (b_{ix}, b_{iy}, b_{iz})$

$$
\begin{array}{cccc}
\begin{bmatrix} 0 \\ 0 \end{bmatrix} & & & \\
\begin{bmatrix} 0 \\ 2 \end{bmatrix} & \begin{bmatrix} 0 \\ 1 \end{bmatrix} & & \\
\begin{bmatrix} 8 \\ 2 \end{bmatrix} & \begin{bmatrix} 4 \\ 2 \end{bmatrix} & \begin{bmatrix} 2 \\ 3/2 \end{bmatrix} & \\
\begin{bmatrix} 4 \\ 0 \end{bmatrix} & \begin{bmatrix} 6 \\ 1 \end{bmatrix} & \begin{bmatrix} 5 \\ 3/2 \end{bmatrix} & \begin{bmatrix} 7/2 \\ 3/2 \end{bmatrix}
\end{array}
$$

deCasteljau Algorithm



JAVA-Applet

- deCasteljau algorithm:
  - Successive linear interpolation
  - Curve segments
  - Endpoint interpolation
  - Tangency
  - Different degrees

Derivatives of Bézier Curves

- Computation of derivatives:
- Recurrence relation of Bernstein polynomials:

$$\frac{d}{dt} B_i^n(t) = n\left(B_{i-1}^{n-1}(t) - B_i^{n-1}(t)\right)$$

- For the curve:

$$\frac{d}{dt} \boldsymbol{b}^{\boldsymbol{n}}(t) = n \sum_{j=0}^{n} \left(B_{j-1}^{n-1}(t) - B_j^{n-1}(t)\right) \boldsymbol{b}_{\boldsymbol{j}}$$

- Forward differencing operator Δ :

$$\Delta \boldsymbol{b}_{\boldsymbol{j}} = \boldsymbol{b}_{\boldsymbol{j+1}} - \boldsymbol{b}_{\boldsymbol{j}} \qquad \Delta \boldsymbol{b}_{\boldsymbol{j}} \in R^3$$

$$\frac{d}{dt} \boldsymbol{b}^{\boldsymbol{n}}(t) = n \sum_{j=0}^{n-1} \Delta \boldsymbol{b}_{\boldsymbol{j}} B_j^{n-1}(t)$$

Derivatives of Bézier Curves

The derivative of a Bézier curve is a Bézier curve of degree n-1

- Generalization to higher order derivatives using a recursive forward difference operator Δ^r of degree r:

$$\Delta^r \mathbf{b}_j = \Delta^{r-1}\mathbf{b}_{j+1} - \Delta^{r-1}\mathbf{b}_j$$

- In a non-recursive form:

$$\Delta^r \mathbf{b}_i = \sum_{j=0}^{r} \binom{r}{j} (-1)^{r-j}\, \mathbf{b}_{j+i}$$

$$\frac{d^r}{dt^r}\mathbf{b}^n(t) = \frac{n!}{(n-r)!}\sum_{j=0}^{n-r} \Delta^r \mathbf{b}_j B_j^{n-r}(t)$$

Derivatives of Bézier Curves

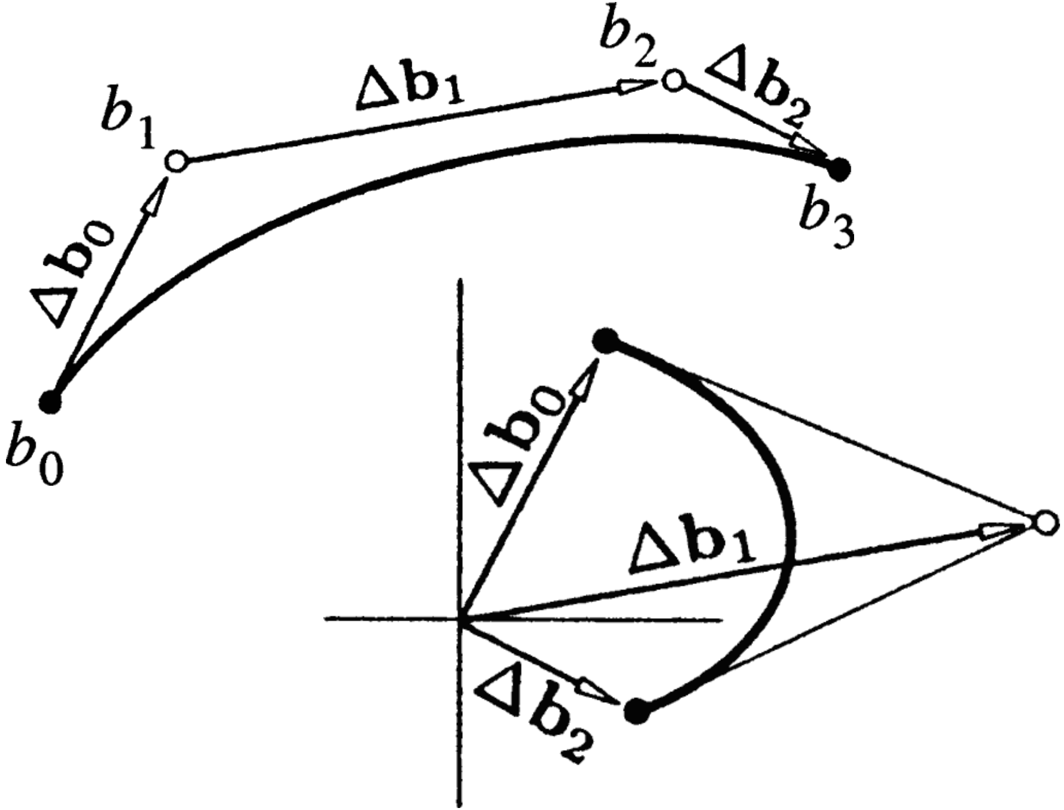
- Derivatives at *t = 0* and *t = 1*:

$$\frac{d^r}{dt^r}\boldsymbol{b}^n(0) = \frac{n!}{(n-r)!}\Delta^r \boldsymbol{b}_0$$

$$\frac{d^r}{dt^r}\boldsymbol{b}^n(1) = \frac{n!}{(n-r)!}\Delta^r \boldsymbol{b}_{n-r}$$

- $\Delta \boldsymbol{b}_0$ and $\Delta \boldsymbol{b}_1$ define the tangent in $t = 0$
- Computation using the deCasteljau algorithm
- Related issues: Subdivision and degree elevation of a curve

Bézier Curve and Derivative



OpenGL Curves

- Define a so-called Evaluator (`glMap`)
- Enable it (`glEnable`)
- `GL_MAP_VERTEX_3`: 3D control points and vertices
- `glEvalCoord1(u)` replaces `glVertex*()`
- Works for geometry, texture, color, normals

OpenGL Curves

- Using glMap1f() and glEvalCoord1f()

```
GLfloat ctrlpoints[4][3] = {
       {-4.0, -4.0, 0.0}, {-2.0, 4.0, 0.0},
       { 2.0, -4.0, 0.0}, { 4.0, 4.0, 0.0}};

void myinit(void)
{
    glClearColor(0.0, 0.0, 0.0, 1.0);
    glMap1f(GL_MAP1_VERTEX_3, 0.0, 1.0, 3, 4,
      &ctrlpoints[0][0]); /* u0, u1, res, order */
    glEnable(GL_MAP1_VERTEX_3);
    glShadeModel(GL_FLAT);
}
```

OpenGL Curves

```
void display(void)
{
  int i;
  glClear(GL_COLOR_BUFFER_BIT);
  glColor3f(1.0, 1.0, 1.0);
  glBegin(GL_LINE_STRIP);
    for (i = 0; i <= 30; i++)
      glEvalCoord1f((GLfloat) i/30.0);
  glEnd();
  /* The following code displays the control points as dots. */
  glPointSize(5.0);
  glColor3f(1.0, 1.0, 0.0);
  glBegin(GL_POINTS);
    for (i = 0; i < 4; i++)
      glVertex3fv(&ctrlpoints[i][0]);
  glEnd();
  glFlush();
}
```

Piecewise Bézier Curves

- Polynomial degree aligned to number of control points
- Variant: Piecewise smooth curve definitions: ***Splines (piecewise curves)***
- Problem: Continuity at the curve boundaries
- Global parameter *u* to describe curve

Piecewise Bézier Curves

- Segment boundaries (knots) $u_0 < \dots < u_L$ define Intervals $[u_i, u_{i+1}]$.
- Local Parameter t to describe the curve in each interval

$$t = \frac{u - u_i}{u_{i+1} - u_i} = \frac{u - u_i}{\Delta_i}$$

- Segmental definition: $s(u) = s_i(t)$.
- Computation of the curve derivatives

$$\frac{ds(u)}{du} = \frac{ds_i(t)}{dt}\frac{dt}{du} = \frac{1}{\Delta_i}\frac{ds_i(t)}{dt}$$

Piecewise Bézier Curves

- Curve in $[u_0, u_2]$, decomposed into 2 Bézier-Segments $\boldsymbol{b}_0,\ldots,\boldsymbol{b}_{\boldsymbol{n}}$ in $[u_0, u_1]$ and $\boldsymbol{b}_{\boldsymbol{n}},\ldots,\boldsymbol{b}_{2\boldsymbol{n}}$ in $[u_1, u_2]$
- Enforce C^r-Continuity at segment boundaries by the following conditions:

$$\boldsymbol{b}_{n+i} = \boldsymbol{b}_{n-i}^{i}(t) \qquad i = 0,\ldots,r$$

  where $t = (u - u_0) / (u_1 - u_0)$ stands for the local Coordinate of u_2 relative to $[u_0, u_1]$

- Control points by extrapolation of the first segment using the deCasteljau-Algorithm

Piecewise Bézier Curves

Example: C^1-Continuity:

- Control points $\boldsymbol{b}_{n-1}$, $\boldsymbol{b}_n$ and $\boldsymbol{b}_{n+1}$ influence first derivative in $\boldsymbol{b}_n$
  $\Rightarrow$ ***co-linearity at ratio*** $(u_1 - u_0) / (u_2 - u_1) = \Delta_0 / \Delta_1$
- Since

$$\Delta_1 \, \Delta \boldsymbol{b}_{n-1} = \Delta_0 \, \Delta \boldsymbol{b}_n$$

- ***C^1-Continuity contains the first 2 control points of the following segment***

JAVA-Applet

- Derivatives of a Bézier curve:
  - Co-linearity of control points
  - Relationship between individual curve segments

Matrix Form

- $x(t)$ as a curve of type:

$$x(t) = \sum_{i=0}^{n} c_i C_i(t)$$

- As an inner product:

$$x(t) = \begin{bmatrix} c_0 & \dots & c_n \end{bmatrix} \begin{bmatrix} C_0(t) \\ \vdots \\ C_n(t) \end{bmatrix}$$

Matrix Form

- Basis transform into a monomial representation with $\boldsymbol{M} = \{m_{ij}\}$:

$$\begin{bmatrix} C_0(t) \\ \vdots \\ C_n(t) \end{bmatrix} = \begin{bmatrix} m_{00} & \dots & m_{0n} \\ \vdots & & \vdots \\ m_{n0} & \dots & m_{nn} \end{bmatrix} \begin{bmatrix} t^0 \\ \vdots \\ t^n \end{bmatrix}$$

- For Bernstein polynomials we obtain

$$m_{ij} = (-1)^{j-i} \binom{n}{j} \binom{j}{i}$$

Matrix Form

- For $n = 3$:

$$\boldsymbol{M} = \begin{bmatrix} 1 & -3 & 3 & -1 \\ 0 & 3 & -6 & 3 \\ 0 & 0 & 3 & -3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Matrix ***M*** is the key to the ***forward-differencing*** method.

Spline Interpolation

- Goal: Interpolate a set of points $\boldsymbol{p}_0,\ldots,\boldsymbol{p}_n$ using basis functions
- Interpolation with Monomials:
  - Canonical form of polynomial interpolation

$$\boldsymbol{x}(t) = \sum_{j=0}^{n} \boldsymbol{a}_j t^j$$

with $\boldsymbol{x}(t_i) = \boldsymbol{p}_i$ and t^j: Monomial of degree j.

Spline Interpolation

- Solution is given by a system of linear equations

$$\boldsymbol{p}_i = \boldsymbol{x}(t_i) = \sum_{j=0}^{n} \boldsymbol{a}_j (t_i)^j , \quad i \in [0, n]$$

- Matrix form: (*Vandermonde*)

$$\begin{bmatrix} 1 & t_0 & \cdots & t_0^n \\ \vdots & \vdots & & \vdots \\ 1 & t_n & \cdots & t_n^n \end{bmatrix} \begin{bmatrix} \boldsymbol{a}_0 \\ \vdots \\ \boldsymbol{a}_n \end{bmatrix} = \begin{bmatrix} \boldsymbol{p}_0 \\ \vdots \\ \boldsymbol{p}_n \end{bmatrix}$$

Vandermonde matrices are notoriously ill-conditioned

deCasteljau Algorithm

- The notion of blossoms:
- ***Blossoming*** as a generalization of the deCasteljau-algorithm
- Increasing popularity
- Interpolation for different parameter values t_1, t_2, t_3 traces out a region in $\mathbf{R}^3$:

$$
\begin{array}{llll}
\boldsymbol{b}_0 & & & \\
\boldsymbol{b}_1 & \boldsymbol{b}_0^1[t_1] & & \\
\boldsymbol{b}_2 & \boldsymbol{b}_1^1[t_1] & \boldsymbol{b}_0^1[t_1,t_2] & \\
\boldsymbol{b}_3 & \boldsymbol{b}_2^1[t_1] & \boldsymbol{b}_1^2[t_1,t_2] & \boldsymbol{b}_0^3[t_1,t_2,t_3]
\end{array}
$$

deCasteljau Algorithm

- The trivariate function $\boldsymbol{f}(t_1, t_2, t_3)$ is called ***blossom*** of the curve $\boldsymbol{b}^3(t)$
  We obtain $\boldsymbol{b}[0,0,0] = \boldsymbol{b}_0$ and $\boldsymbol{b}[1,1,1] = \boldsymbol{b}_3$
- Evaluation of $[t_1, t_2, t_3] = [0,0,1]$:

$$
\begin{array}{llll}
\boldsymbol{b}_0 & & & \\
\boldsymbol{b}_1 & \boldsymbol{b}_0 & & \\
\boldsymbol{b}_2 & \boldsymbol{b}_1 & \boldsymbol{b}_0 & \\
\boldsymbol{b}_3 & \boldsymbol{b}_2 & \boldsymbol{b}_1 & \boldsymbol{b}_1 = \boldsymbol{b}[0,0,1]
\end{array}
$$

to get $\boldsymbol{b}_2 := \boldsymbol{b}[0,1,1]$

deCasteljau Algorithm

- To get the curve: Set $t_1 = t_2 = t_3 = t$:

$$\boldsymbol{b}_0 = \boldsymbol{b}[0,0,0]$$

$$\boldsymbol{b}_1 = \boldsymbol{b}[0,0,1] \quad \boldsymbol{b}[0,0,t]$$

$$\boldsymbol{b}_2 = \boldsymbol{b}[0,1,1] \quad \boldsymbol{b}[0,t,1] \quad \boldsymbol{b}[0,t,t]$$

$$\boldsymbol{b}_3 = \boldsymbol{b}[1,1,1] \quad \boldsymbol{b}[t,1,1] \quad \boldsymbol{b}[t,t,1] \quad \boldsymbol{b}[t,t,t]$$

  $t^{<r>}$: t r-times as argument

- Bézier control points in blossom notation:

$$\boldsymbol{b}_i = \boldsymbol{b}[0^{<n-i>},1^{<i>}]$$