Bézier Curves

Prof. Dr. Markus Gross
Overview

• Coordinate Systems
• Bernstein Polynomials
• Bézier Curves – Properties
• Derivatives
• Piecewise Curves
Literature

- Christoph Hoffmann: *Geometric and Solid Modeling. An Introduction*. Morgan Kaufmann, 1989
Local Coordinate Systems

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

\[
x = \begin{bmatrix} x \\ y \\ z \end{bmatrix}; \quad u = \begin{bmatrix} u \\ v \end{bmatrix}
\]

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

\[
x(u) = (x(u), y(u), z(u))^T
\]
Local Coordinate Systems

- Surface $x(u,v)$ as a map of a subregion of $(u,v)$ into $E^2$ or $E^3$
  
  $$x(u,v) = (x(u,v), y(u,v), z(u,v))^T$$

- Subdivision of parameter space into disjoint segments (knots):
  
  $$u_0 < u_1 < \cdots < u_p$$

- Surfaces are subdivided by so-called knotlines:
  
  $$u_0 < u_1 < \cdots < u_p \text{ and } v_0 < v_1 < \cdots < v_q$$
Bézier Curves

• $\mathbf{x}(t) = \mathbf{p}(t)$ given by a Bernstein basis expansion:
  
  $$\mathbf{x}(t) = b_0 B_0^n(t) + \ldots + b_n B_n^n(t)$$

• Bernstein polynomial of degree $n$:
  
  $$B_i^n(t) = \binom{n}{i} t^i (1-t)^{n-i}$$

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

• Binomial coefficients:
  
  $$\binom{n}{i} = \begin{cases} \frac{n!}{i!(n-i)!} & 0 \leq i \leq n \\ 0 & \text{else} \end{cases}$$
JAVA-Applet

• Bernstein polynomial:
  – Global support
  – Positive definite
  – Partition of unity
  – Different degrees
Construction and Properties

• Cubic curve ($n = 3$):
  \[ x(t) = b_0 (1-t)^3 + 3b_1 t (1-t)^2 + 3b_2 t^2 (1-t) + b_3 t^3 \]
  – 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
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.

\[
\text{conv}(P) := \{ \sum_{i=1}^{n} \lambda_i p_i \mid \lambda_i \geq 0, \sum_{i=1}^{n} \lambda_i = 1 \}
\]
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 \( b_0 \) and \( b_n \).
  – *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 $b_0$, $b_1$, $b_2$ be 3 control points:
  
  $b_0^1(t) = (1 - t)b_0 + t b_1$
  $b_1^1(t) = (1 - t)b_1 + t b_2$
  $b_2^1(t) = (1 - t)b_1^1(t) + t b_1^1(t)$

- We obtain: $b_0^2(0) = b_0$, $b_0^2(1) = b_2$

- Insert:
  $b_0^2(t) = (1 - t)^2 b_0 + 2t (1 - t) b_1 + t^2 b_2$
deCasteljau Algorithm

- Recursive computation of a point on the curve using a systolic array:
  - Given: \( n+1 \) control points \( b_0, b_1, \ldots, b_n \)
  - Recursion:
    \[
    b'_i(t) = (1 - t)b'_i(t) + t b'_{i+1}(t)
    \]
    \[
    b^0_i(t) = b_i
    \]
    \[
    r = 1, \ldots, n \quad i = 0, \ldots, n - r
    \]

\( \Rightarrow \) **Point on the Bézier curve with**
deCasteljau Algorithm

- Algorithm computes a triangular representation:

\[
\begin{array}{cccc}
 b_0 & b_1^1 & b_2^2 & b_3^3 \\
 b_1 & b_0 & & \\
 b_2 & b_1 & b_0 & \\
 b_3 & b_2 & b_1 & b_0 \\
\end{array}
\]

\[O(n^2)\] costs

⇒ successive linear interpolation, “corner cutting”
deCasteljau Algorithm

• A planar cubic Bézier curve at $t = \frac{1}{2}$:

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

\[
\begin{bmatrix}
0 \\
0 \\
0 \\
2 \\
8 \\
2 \\
4 \\
0
\end{bmatrix}
\begin{bmatrix}
0 \\
1 \\
2 \\
3/2 \\
5 \\
3/2 \\
7/2
\end{bmatrix}
\]
deCasteljau Algorithm
JAVA-Applet

• deCasteljau algorithm:
  – Successive linear interpolation
  – Curve segments
  – Endpoint interpolation
  – Tangency
  – Different degrees
Derivatives of Bézier Curves

• Computation of derivatives:

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

• Recurrence relation of Bernstein polynomials:

\[
\frac{d}{dt} b^n(t) = n \sum_{j=0}^{n} \left( B_{j-1}^{n-1}(t) - B_j^{n-1}(t) \right) b_j
\]

• Forward differencing operator  \( \Delta \) : 

\[
\Delta b_j = b_{j+1} - b_j, \quad \Delta b_j \in \mathbb{R}^3
\]

\[
\frac{d}{dt} b^n(t) = n \sum_{j=0}^{n-1} \Delta b_j B_j^{n-1}(t)
\]
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 \( \Delta^r \) of degree \( r \):
  \[
  \Delta^r b_j = \Delta^{r-1} b_{j+1} - \Delta^{r-1} b_j
  \]

- In a non-recursive form:
  \[
  \Delta^r b_i = \sum_{j=0}^{r} \binom{r}{j} (-1)^{r-j} b_{j+i}
  \]
  \[
  \frac{d^r}{dt^r} b^n(t) = \frac{n!}{(n-r)!} \sum_{j=0}^{n-r} \Delta^r b_j B_j^{n-r}(t)
  \]
Derivatives of Bézier Curves

• Derivatives at $t = 0$ and $t = 1$:
  \[
  \frac{d^r}{dt^r} b^n(0) = \frac{n!}{(n-r)!} \Delta^r b_0
  \]
  \[
  \frac{d^r}{dt^r} b^n(1) = \frac{n!}{(n-r)!} \Delta^r b_{n-r}
  \]

• $\Delta b_0$ and $\Delta b_1$ 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()`

```c
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

```c
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) $u_0 < \ldots < u_L$ 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 \(b_0, \ldots, b_n\) in \([u_0, u_1]\) and \(b_n, \ldots, b_{2n}\) in \([u_1, u_2]\)
- Enforce \(C^r\)-Continuity at segment boundaries by the following conditions:

\[
b_{n+i} = b^{i}_{n-i}(t) \quad i = 0, \ldots, r
\]

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 $b_{n-1}$, $b_n$ and $b_{n+1}$ influence first derivative in $b_n$
  \[ \Rightarrow \text{co-linearity at ratio } (u_1 - u_0) / (u_2 - u_1) = \Delta_0 / \Delta_1 \]
- Since
  \[ \Delta_1 \Delta b_{n-1} = \Delta_0 \Delta b_n \]
- $C^1$-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:

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

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

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

• For Bernstein polynomials we obtain

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

• For $n = 3$:

\[ M = \begin{bmatrix}
1 & -3 & 3 & -1 \\
0 & 3 & -6 & 3 \\
0 & 0 & 3 & -3 \\
0 & 0 & 0 & 1
\end{bmatrix} \]

• Matrix $M$ is the key to the forward-differencing method.
Spline Interpolation

• Goal: Interpolate a set of points \( p_0, \ldots, p_n \) using basis functions

• Interpolation with Monomials:
  – Canonical form of polynomial interpolation

\[
x(t) = \sum_{j=0}^{n} a_j t^j
\]

with \( x(t_i) = p_i \) and \( t^j \): Monomial of degree \( j \).
Spline Interpolation

- Solution is given by a system of linear equations
  \[ p_i = x(t_i) = \sum_{j=0}^{n} a_j (t_i)^j, \quad i \in [0, n] \]

- Matrix form: (Vandermonde)
  \[
  \begin{bmatrix}
    1 & t_0 & \cdots & t_0^n \\
    \vdots & \vdots & \ddots & \vdots \\
    1 & t_n & \cdots & t_n^n \\
  \end{bmatrix}
  \begin{bmatrix}
    a_0 \\
    \vdots \\
    a_n \\
  \end{bmatrix}
  =
  \begin{bmatrix}
    p_0 \\
    \vdots \\
    p_n \\
  \end{bmatrix}
  \]

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 $\mathbb{R}^3$:

\[
\begin{align*}
&b_0 \\
&b_1 \quad b_0^1[t_1] \\
&b_2 \quad b_1^1[t_1] \quad b_0^1[t_1,t_2] \\
&b_3 \quad b_2^1[t_1] \quad b_1^2[t_1,t_2] \quad b_0^3[t_1,t_2,t_3]
\end{align*}
\]
deCasteljau Algorithm

• The trivariate function \( f(t_1, t_2, t_3) \) is called \textit{blossom} of the curve \( \mathbf{b}^3(t) \)
  
  We obtain \( \mathbf{b}[0,0,0] = \mathbf{b}_0 \) and \( \mathbf{b}[1,1,1] = \mathbf{b}_3 \)

• Evaluation of \([t_1, t_2, t_3] = [0,0,1]\):

\[
\begin{array}{ccc}
  b_0 \\
  b_1 & b_0 \\
  b_2 & b_1 & b_0 \\
  b_3 & b_2 & b_1 & b_1 &= \mathbf{b}[0,0,1] \\
\end{array}
\]

  to get \( \mathbf{b}_2 := \mathbf{b}[0,1,1] \)
deCasteljau Algorithm

• To get the curve: Set \( t_1 = t_2 = t_3 = t \):

\[
\begin{align*}
    b_0 &= b[0,0,0] \\
    b_1 &= b[0,0,1] \quad b[0,0,t] \\
    b_2 &= b[0,1,1] \quad b[0,t,1] \quad b[0,t,t] \\
    b_3 &= b[1,1,1] \quad b[t,1,1] \quad b[t,t,1] \quad b[t,t,t]
\end{align*}
\]

\( t^{<r>} \): \( t \) \( r \)-times as argument

• Bézier control points in blossom notation:

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