
Computer Graphics

- Splines -

Philipp Slusallek

Curves

- **Curve descriptions**

- Explicit functions

- $y(x) = \pm \sqrt{r^2 - x^2}$, restricted domain ($x \in [-1, 1]$)

- Implicit functions

- $x^2 + y^2 = r^2$ unknown solution set

- Parametric functions

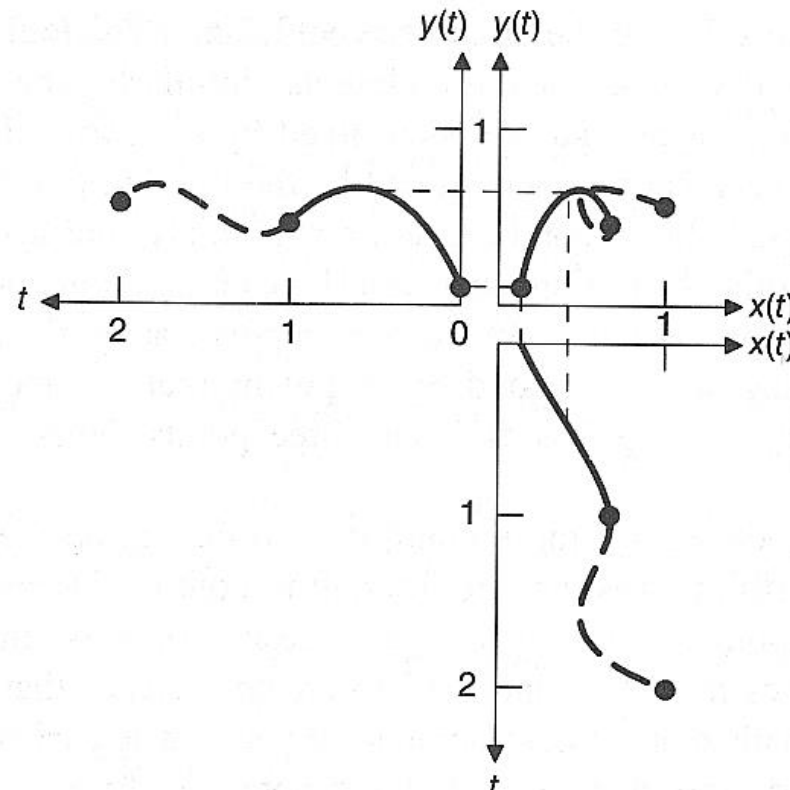
- $x(t) = r \cos(t)$, $y(t) = r \sin(t)$, $t \in [0, 2\pi]$
 - Flexibility and ease of use

- **Typically, use of polynomials**

- Avoids complicated functions (z.B. pow, exp, sin, sqrt)
 - Use simple polynomials, typically of low degree
-

Parametric curves

- **Separate function in each coordinate**
 - 3D: $f(t) = (x(t), y(t), z(t))$



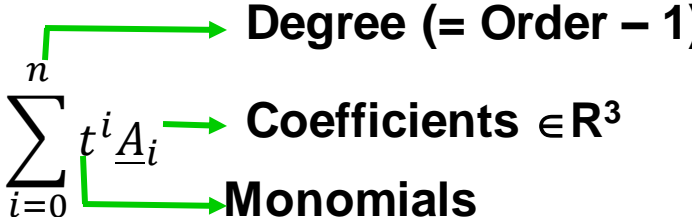
Monomials

- **Monomial basis**

- Simple basis: 1, t, t², ... (t usually in [0 .. 1])

- **Polynomial representation**

$$\underline{P}(t) = \begin{pmatrix} \underline{x}(t) & \underline{y}(t) & \underline{z}(t) \end{pmatrix} = \sum_{i=0}^n t^i \underline{A}_i$$



- Coefficients can be determined from a sufficient number of constraints (e.g. interpolation of given points)
 - Given (n+1) parameter values t_i and points P_i
 - Solution of a linear system in the A_i – possible, but inconvenient

- **Matrix representation**

$$\begin{aligned} \underline{P}(t) &= \begin{pmatrix} x(t) & y(t) & z(t) \end{pmatrix} = T(t) A \\ &= \begin{bmatrix} t^n & t^{n-1} & \dots & 1 \end{bmatrix} \begin{bmatrix} A_{x,n} & A_{y,n} & A_{z,n} \\ A_{x,n-1} & A_{y,n-1} & A_{z,n-1} \\ \vdots & \vdots & \vdots \\ A_{x,0} & A_{y,0} & A_{z,0} \end{bmatrix} \end{aligned}$$

Derivatives

- **Derivative = tangent vector**

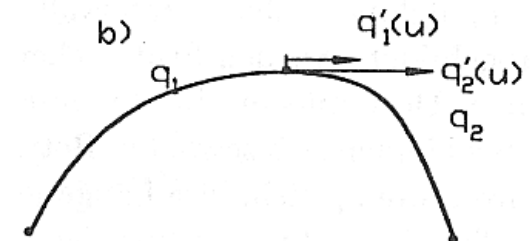
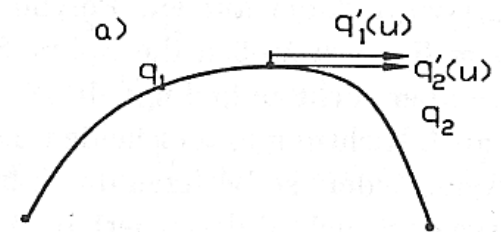
- Polynomial of degree (n-1)

$$P'(t) = (x'(t) \quad y'(t) \quad z'(t)) = T'(t) A$$

$$= [nt^{n-1} \quad (n-1)t^{n-2} \quad \dots \quad 1 \quad 0] \begin{bmatrix} A_{x,n} & A_{y,n} & A_{z,n} \\ A_{x,n-1} & A_{y,n-1} & A_{z,n-1} \\ \vdots & \vdots & \vdots \\ A_{x,0} & A_{y,0} & A_{z,0} \end{bmatrix}$$

- **Continuity and smoothness between parametric curves**

- $C^0 = G^0 =$ same point
- *Parametric* continuity C^1
 - Tangent vectors are identical
- *Geometric* continuity G^1
 - Same direction of tangent vectors
- Similar for higher order derivatives

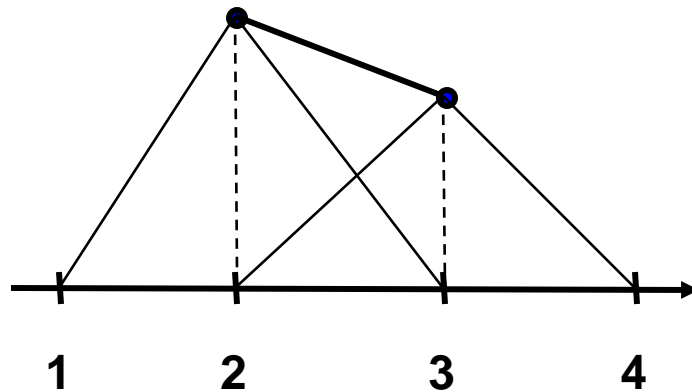
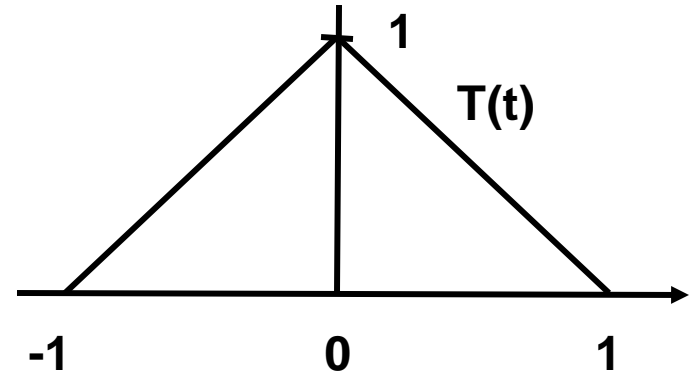
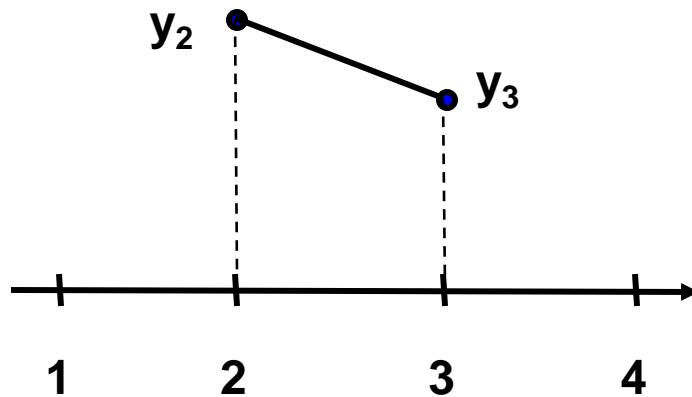


More on Continuity

- **At one point:**
 - **Geometric Continuity:**
 - G0: curves are joined together at that point
 - G1: first derivatives are proportional at joint point
 - Same direction but not necessarily same length
 - G2: first and second derivatives are proportional
 - **Parametric Continuity:**
 - C0: curves are joined
 - C1: first derivative equal
 - C2: first and second derivatives are equal.
 - If t is the time, this implies the acceleration is continuous.
 - Cn: all derivatives up to and including the n th are equal.
-

Linear Interpolation

- Hat Functions and Linear Splines (C0/G0 continuity)



$$T(t) = \begin{cases} 0 & t < -1 \\ 1+t & -1 \leq t < 0 \\ 1-t & 0 \leq t < 1 \\ 0 & t \geq 1 \end{cases}$$
$$T_i(t) = T(t - i)$$

$$P(t) = \sum P_i T_i(t) = y_2 T_2(t) + y_3 T_3(t)$$

Lagrange Interpolation

- **Interpolating basis functions**

- Lagrange polynomials for a set of parameter values $T=\{t_0, \dots, t_n\}$

$$L_i^n(t) = \prod_{\substack{j=0 \\ i \neq j}}^n \frac{t - t_j}{t_i - t_j}, \quad \text{with} \quad L_i^n(t_j) = \delta_{ij} = \begin{cases} 1 & i = j \\ 0 & \text{otherwise} \end{cases}$$

- **Properties**

- Good for interpolation at given parameter values
 - At each t_i : One basis function = 1, all others = 0
- Polynomial of degree n (n factors linear in t)
 - Infinitely continuous derivatives everywhere

- **Lagrange Curves**

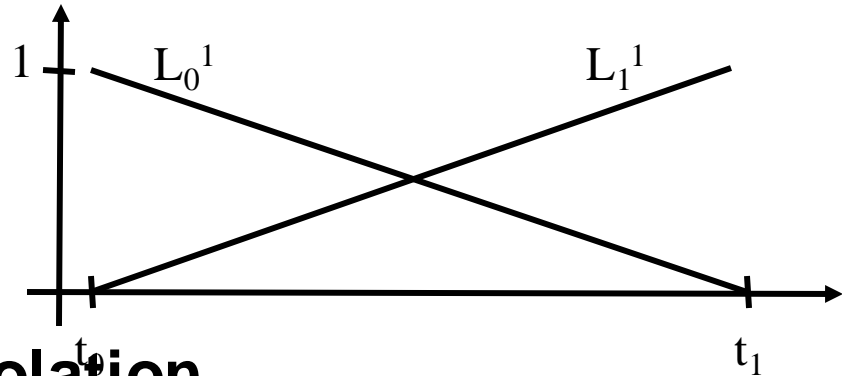
- Use Lagrange Polynomials with point coefficients

$$\underline{P}(t) = \sum_{i=0}^n L_i^n(t) \underline{P}_i$$

Lagrange Interpolation

- **Simple Linear Interpolation**

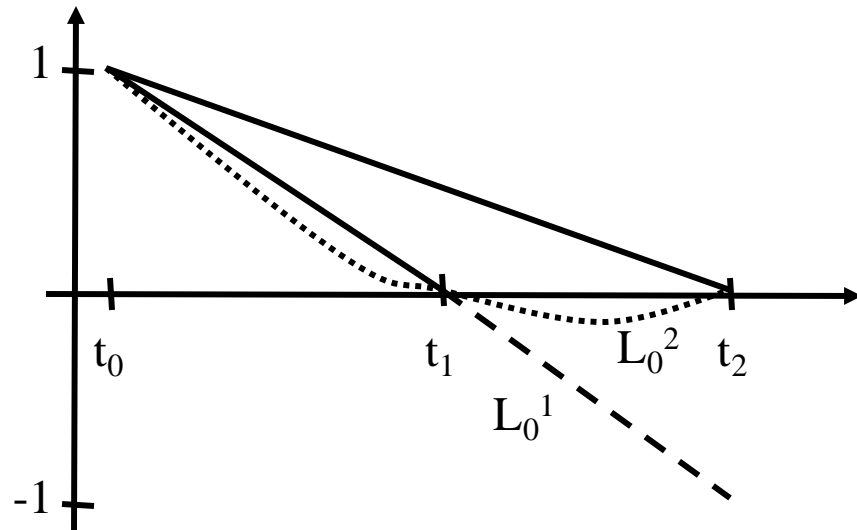
- $T = \{t_0, t_1\}$
$$L_0^1(t) = \frac{t - t_1}{t_0 - t_1}$$
$$L_1^1(t) = \frac{t - t_0}{t_1 - t_0}$$



- **Simple Quadratic Interpolation**

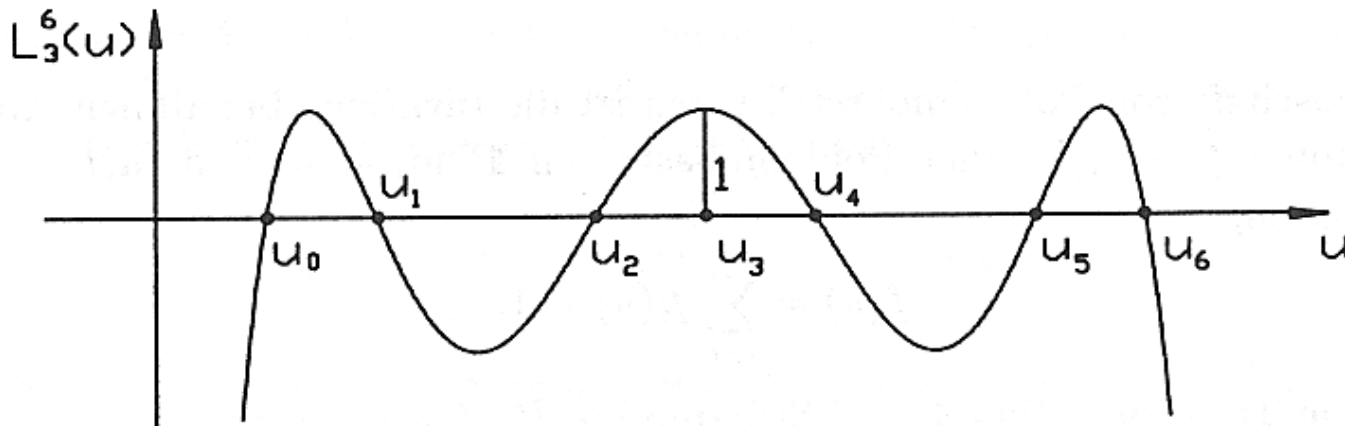
- $T = \{t_0, t_1, t_2\}$

- $$L_0^2(t) = \frac{t - t_1}{t_0 - t_1} \frac{t - t_2}{t_0 - t_2}$$



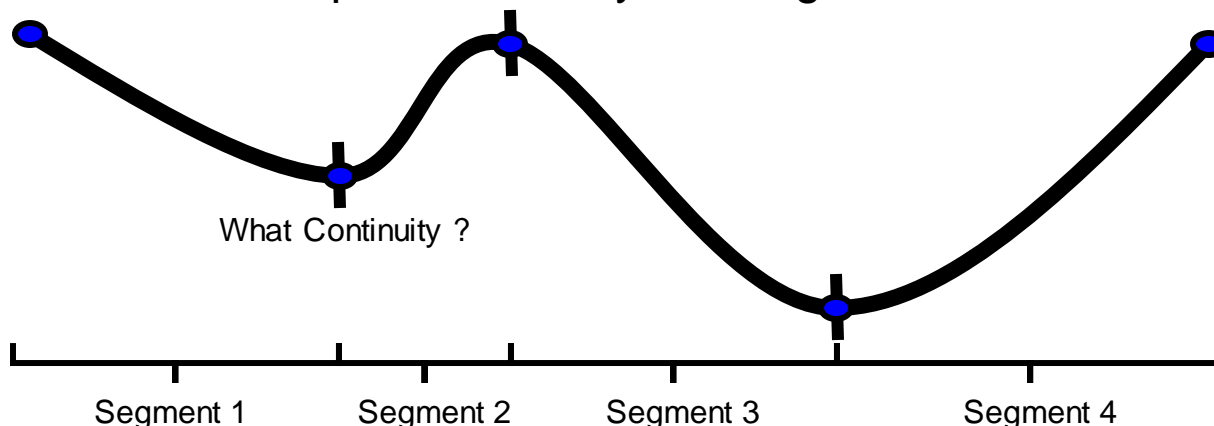
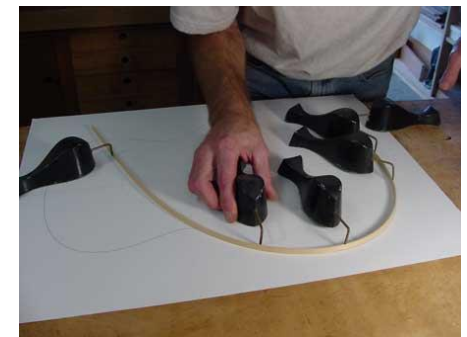
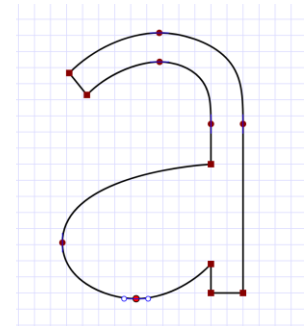
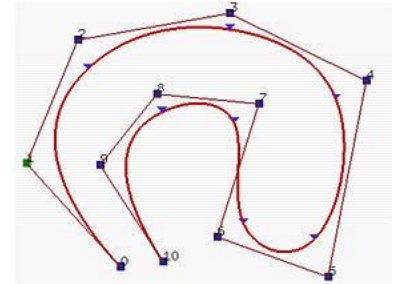
Problems

- **Problems with a single polynomial**
 - Degree depends on the number of interpolation constraints
 - Strong overshooting for high degree ($n > 7$)
 - Problems with smooth joints
 - Numerically unstable
 - No local changes



Splines

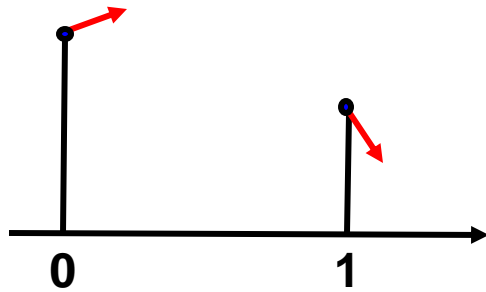
- **Functions for interpolation & approximation**
 - Standard curve and surface primitives in 3D modeling & fonts
 - Key frame and in-betweens in animations
 - Filtering and reconstruction of images
- **Historically**
 - Name for a tool in ship building
 - Flexible metal strip that tries to stay straight
 - Within computer graphics:
 - Piecewise polynomial function
 - Decouples continuity and degree of curve



Hermite Interpolation

- **Hermite Basis (cubic)**

- Interpolation of position P and tangent P' information for $t = \{0, 1\}$
- Very easy to piece together with G1/C1 continuity



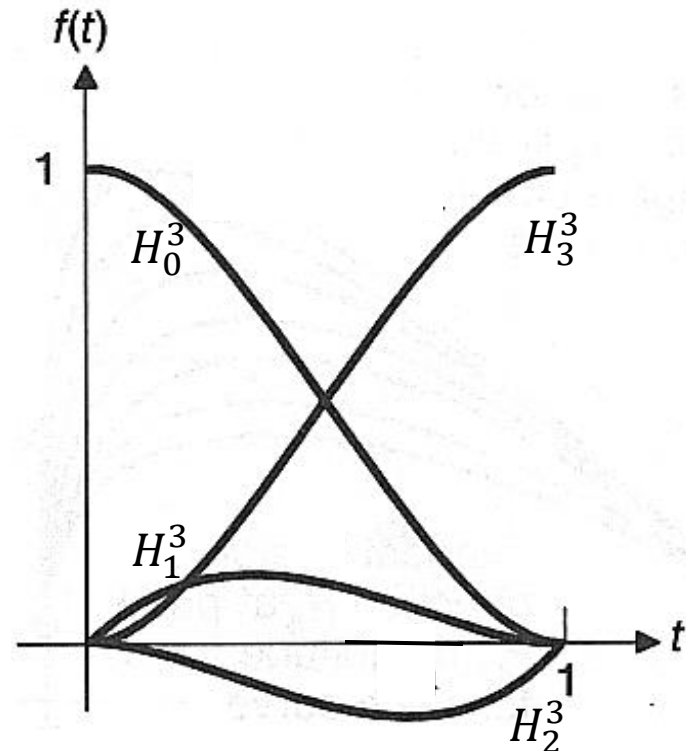
- Basis functions

$$H_0^3(t) = (1 - t)^2(1 + 2t)$$

$$H_1^3(t) = t(1 - t)^2$$

$$H_2^3(t) = -t^2(1 - t)$$

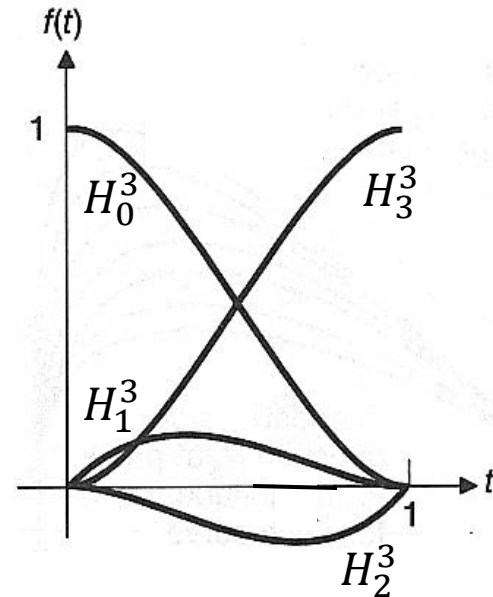
$$H_3^3(t) = (3 - 2t)t^2$$



Hermite Interpolation

- **Properties of Hermite Basis Functions**

- H_0 (H_3) interpolates smoothly from 1 to 0 (1 to 0)
- H_0 and H_3 have zero derivative at $t=0$ and $t=1$
 - No contribution to derivative (H_1, H_2)
- H_1 and H_2 are zero at $t=0$ and $t=1$
 - No contribution to position (H_0, H_3)
- H_1 (H_2) has slope 1 at $t=0$ ($t=1$)
 - Unit factor for specified derivative vector

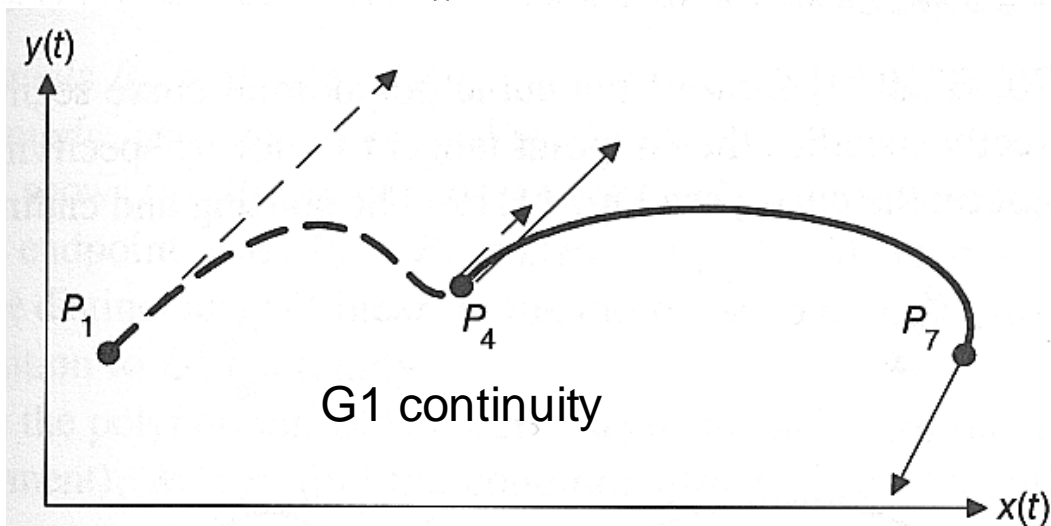
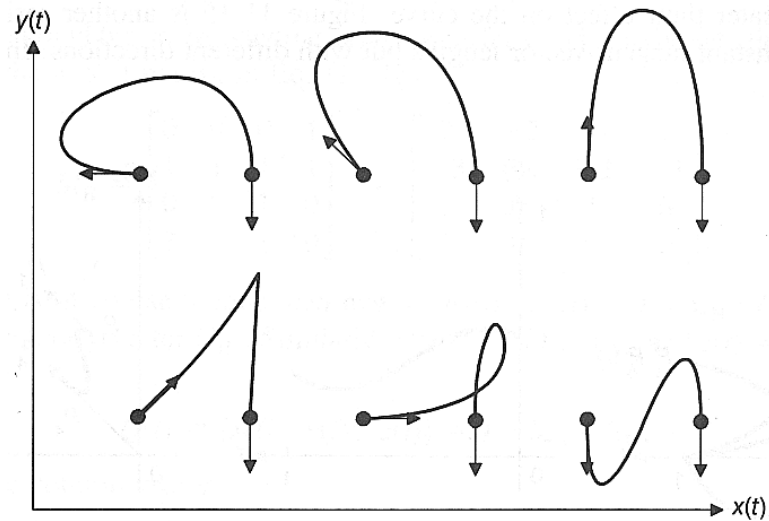
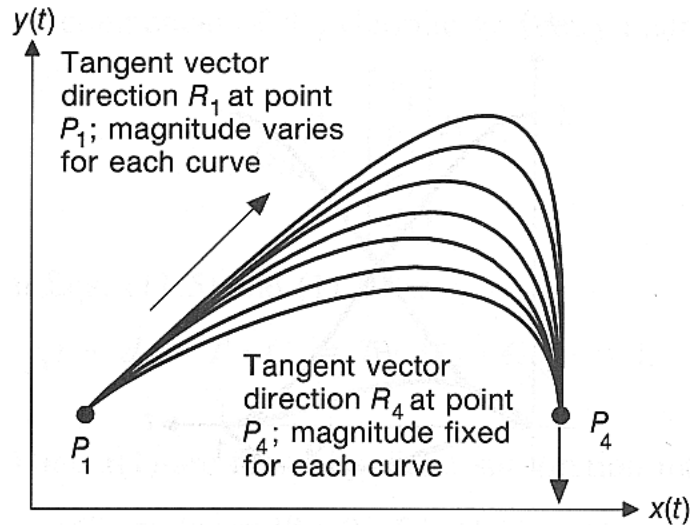


- **Hermite polynomials**

- P_0, P_1 are positions $\in \mathbb{R}^3$
- P'_0, P'_1 are derivatives (tangent vectors) $\in \mathbb{R}^3$

$$\underline{P}(t) = P_0 H_0^3(t) + P'_0 H_1^3(t) + P'_1 H_2^3(t) + P_1 H_3^3(t)$$

Examples: Hermite Interpolation



Matrix Representation

- Matrix representation

$$\begin{aligned}
 P(t) &= [t^3 \quad t^2 \quad \dots \quad 1] \begin{bmatrix} A_{x,n} & A'_{y,n} & A_{z,n} \\ A_{x,n-1} & A_{y,n-1} & A_{z,n-1} \\ \vdots & & \\ A_{x,0} & A_{y,0} & A_{z,0} \end{bmatrix} = \\
 &\underbrace{[t^3 \quad t^2 \quad \dots \quad 1]}_T \underbrace{\begin{bmatrix} M_{11} & M_{12} & M_{13} \\ M_{21} & \ddots & \end{bmatrix}}_{\text{Basis Matrix } M \text{ (4x4)}} \underbrace{\begin{bmatrix} G_{x,3} & G_{y,3} & G_{z,3} \\ G_{x,2} & G_{y,2} & G_{z,2} \\ G_{x,1} & G_{y,1} & G_{y,1} \\ G_{x,0} & G_{y,0} & G_{z,0} \end{bmatrix}}_{\text{Geometry Matrix } G \text{ (4x3)}} = \\
 &\underbrace{[t^3 \quad t^2 \quad \dots \quad 1]}_{\text{Basis Functions}} \underbrace{\begin{bmatrix} M_{11} & M_{12} & M_{13} \\ M_{21} & \ddots & \end{bmatrix}}_{M_H} \underbrace{\begin{bmatrix} P_0^T \\ P_1^T \\ P_0'^T \\ P_1'^T \end{bmatrix}}_{G_H}
 \end{aligned}$$

Matrix Representation

- **For cubic Hermite interpolation we obtain:**

$$\begin{aligned} P_0^T &= (0 \quad 0 \quad 0 \quad 1)M_H G_H \\ P_1^T &= (1 \quad 1 \quad 1 \quad 1)M_H G_H \\ P_0'^T &= (0 \quad 0 \quad 1 \quad 0)M_H G_H \\ P_1'^T &= (3 \quad 2 \quad 1 \quad 0)M_H G_H \end{aligned} \quad \text{or} \quad \begin{pmatrix} P_0^T \\ P_1^T \\ P_0' \\ P_1' \end{pmatrix} = G_H = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{pmatrix} M_H G_H$$

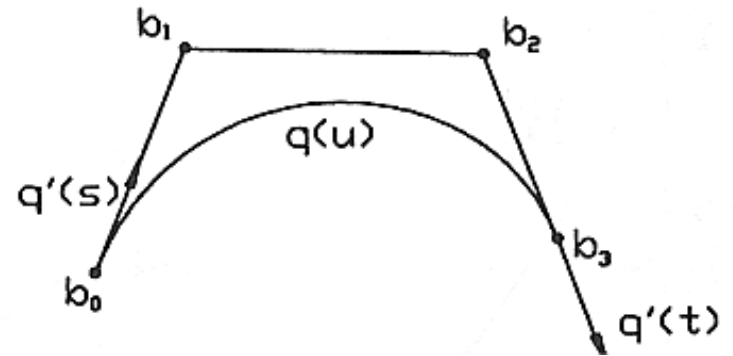
- **Solution:**
 - Two matrices must multiply to unit matrix

$$M_H = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{pmatrix}^{-1} = \begin{pmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

Bézier

- **Bézier Basis [deCasteljau '59, Bézier '62]**

- Different curve representation
- Start and end point
- 2 point that are approximated by the curve (cubics)
- $P'_0 = 3(b_1 - b_0)$ and $P'_1 = 3(b_3 - b_2)$
 - Factor 3 due to derivative of t^3



$$G_H = \begin{bmatrix} P_{0^T} \\ P_{1^T} \\ P'_{0^T} \\ P'_{1^T} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ -3 & 3 & 0 & 0 \\ 0 & 0 & -3 & 3 \end{bmatrix} \begin{bmatrix} b_0^T \\ b_1^T \\ b_2^T \\ b_3^T \end{bmatrix} = M_{HB} G_B$$

Basis transformation

- **Transformation**

- $P(t) = T M_H G_H = T M_H (M_{HB} G_B) = T (M_H M_{HB}) G_B = T M_B G_B$

$$M_B = M_H M_{HB} = \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

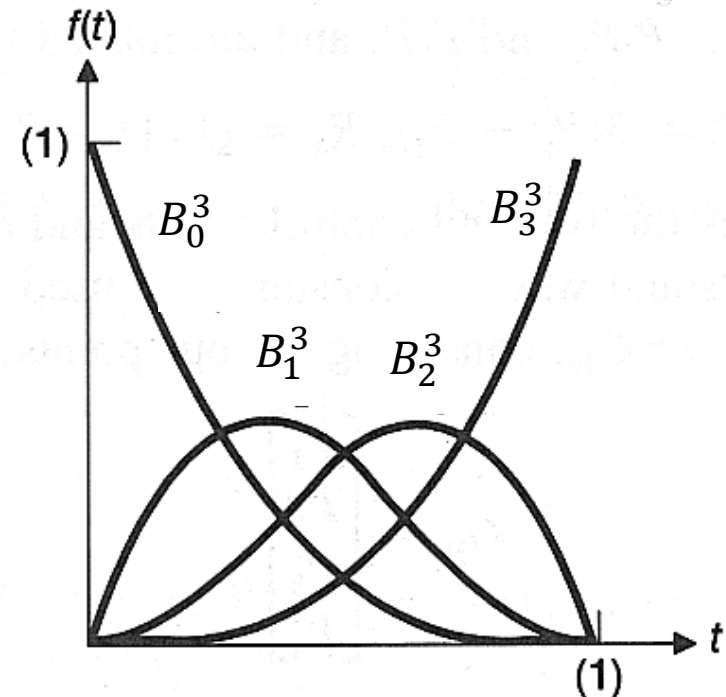
$$P(t) = \sum B_i^3(t) b_i =$$

$$(1-t)^3 b_0 + 3t(1-t)^2 b_1 + 3t^2(1-t) b_2 + t^3 b_3$$

- **Bézier Curves & Basis Functions**

$$P(t) = \sum B_i^n(t) b_i$$

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



**Bernstein-
Polynomials**

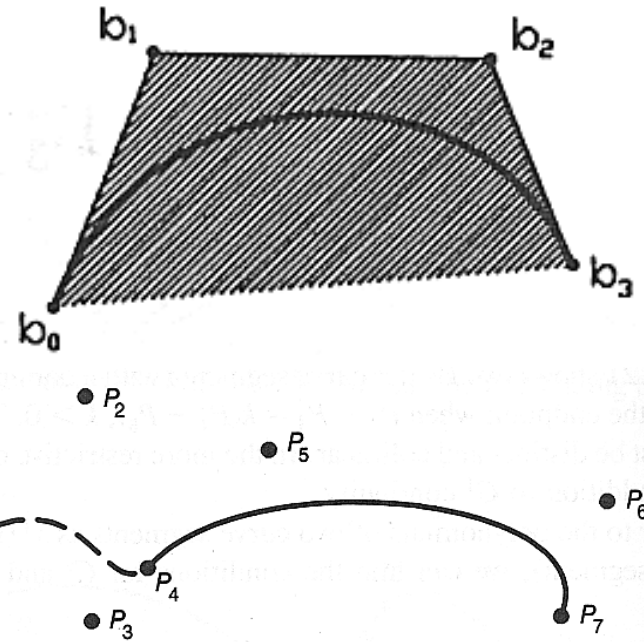
Properties: Bézier

- **Advantages:**

- End point interpolation
- Tangents explicitly specified
- Smooth joints are simple
 - P_3, P_4, P_5 collinear $\rightarrow G^1$ continuous
- Geometric meaning of control points
- Affine invariance
 - $\forall \sum B_i(t) = 1$
- Convex hull property
 - For $0 < t < 1$: $B_i(t) \geq 0$
- Symmetry: $B_i(t) = B_{n-i}(1-t)$

- **Disadvantages**

- Smooth joints need to be maintained explicitly
 - Automatic in B-Splines (and NURBS)



DeCasteljau Algorithm

- **Direct evaluation of the basis functions**
 - Simple but expensive
- **Use recursion**

- Recursive definition of the basis functions

$$B_i^n(t) = tB_{i-1}^{n-1}(t) + (1-t)B_i^{n-1}(t)$$

- Inserting this once yields:

$$P(t) = \sum_{i=0}^n b_i^0 B_i^n(t) = \sum_{i=0}^{n-1} b_i^1(t) B_i^{n-1}(t)$$

- with the new Bézier points given by the recursion

$$b_i^k(t) = tb_{i+1}^{k-1}(t) + (1-t)b_i^{k-1}(t) \quad \text{and} \quad b_i^0(t) = b_i$$

DeCasteljau Algorithm

- **DeCasteljau-Algorithm:**

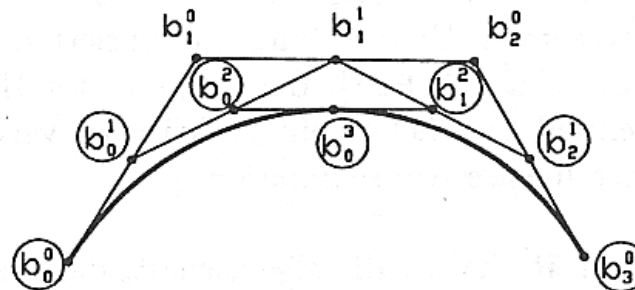
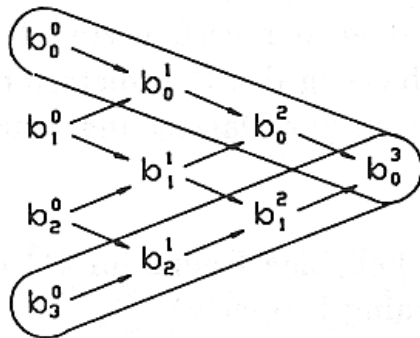
- Recursive degree reduction of the Bezier curve by using the recursion formula for the Bernstein polynomials

$$P(t) = \sum_{i=0}^n b_i^0 B_i^n(t) = \sum_{i=0}^{n-1} b_i^1(t) B_i^{n-1}(t) = \dots = b_i^n(t) \cdot 1$$

$$b_i^k(t) = t b_{i+1}^{k-1}(t) + (1-t) b_i^{k-1}(t)$$

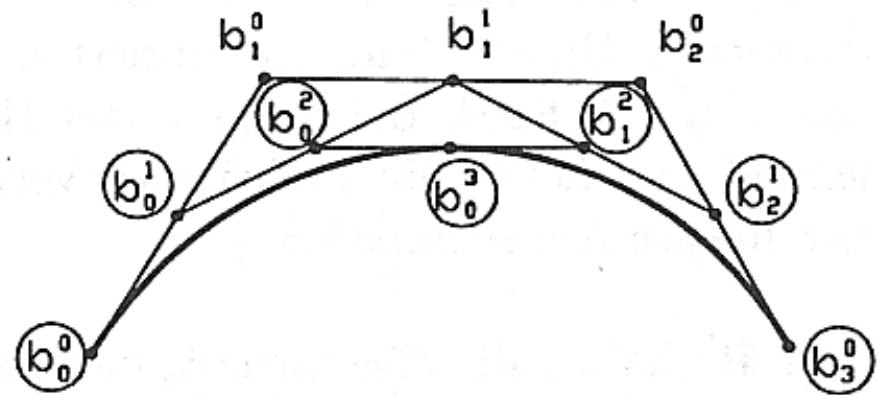
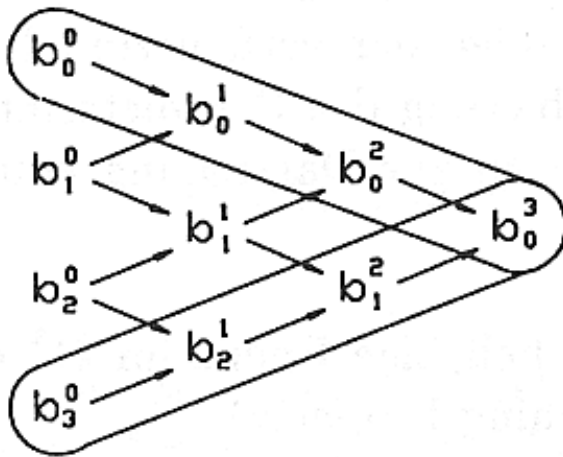
- **Example:**

- $t = 0.5$



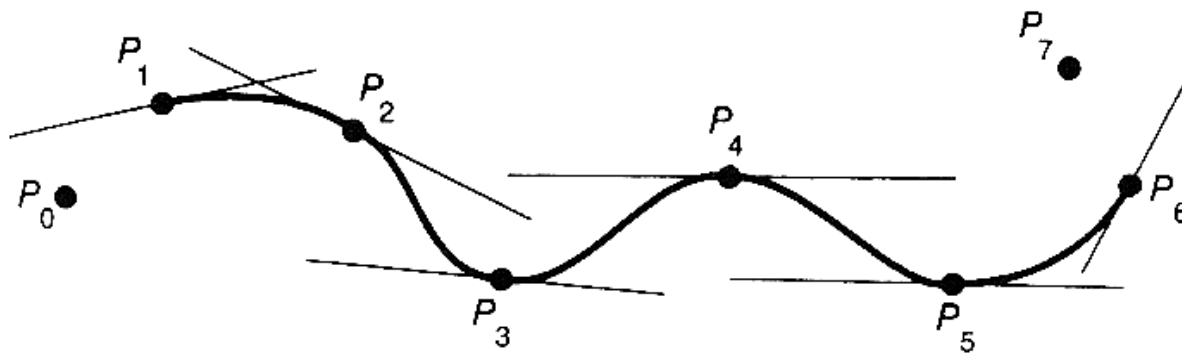
DeCasteljau Algorithm

- **Subdivision using the deCasteljau-Algorithm**
 - Take boundaries of the deCasteljau triangle as new control points for left/right portion of the curve
- **Extrapolation**
 - Backwards subdivision
 - Reconstruct triangle from one side



Catmull-Rom-Splines

- **Goal**
 - Smooth (C^1)-joints between (cubic) spline segments
- **Algorithm**
 - Tangents given by neighboring points P_{i-1} P_{i+1}
 - Construct (cubic) Hermite segments
- **Advantage**
 - Arbitrary number of control points
 - Interpolation without overshooting
 - Local control



Matrix Representation

- **Catmull-Rom-Spline**

- Piecewise polynomial curve
- Four control points per segment
- For n control points we obtain (n-3) polynomial segments

$$\underline{P}^i(t) = TM_{CR}G_{CR} = T\frac{1}{2}\begin{bmatrix}-1 & 3 & -3 & 1 \\ 2 & -5 & 4 & 1 \\ -1 & 0 & 1 & 0 \\ 0 & 2 & 0 & 0\end{bmatrix}\begin{bmatrix}\underline{P}_i^T \\ \underline{P}_{i+1}^T \\ \underline{P}_{i+2}^T \\ \underline{P}_{i+3}^T\end{bmatrix}$$

- **Application**

- Smooth interpolation of a given sequence of points
 - Key frame animation, camera movement, etc.
 - Only G¹-continuity
 - Control points should be equidistant in time
-

Choice of Parameterization

- **Problem**

- Often only the control points are given
- How to obtain a suitable parameterization t_i ?

- **Example: Chord-Length Parameterization**

$$t_0 = 0$$
$$t_i = \sum_{j=1}^i \text{dist}(P_j - P_{j-1})$$

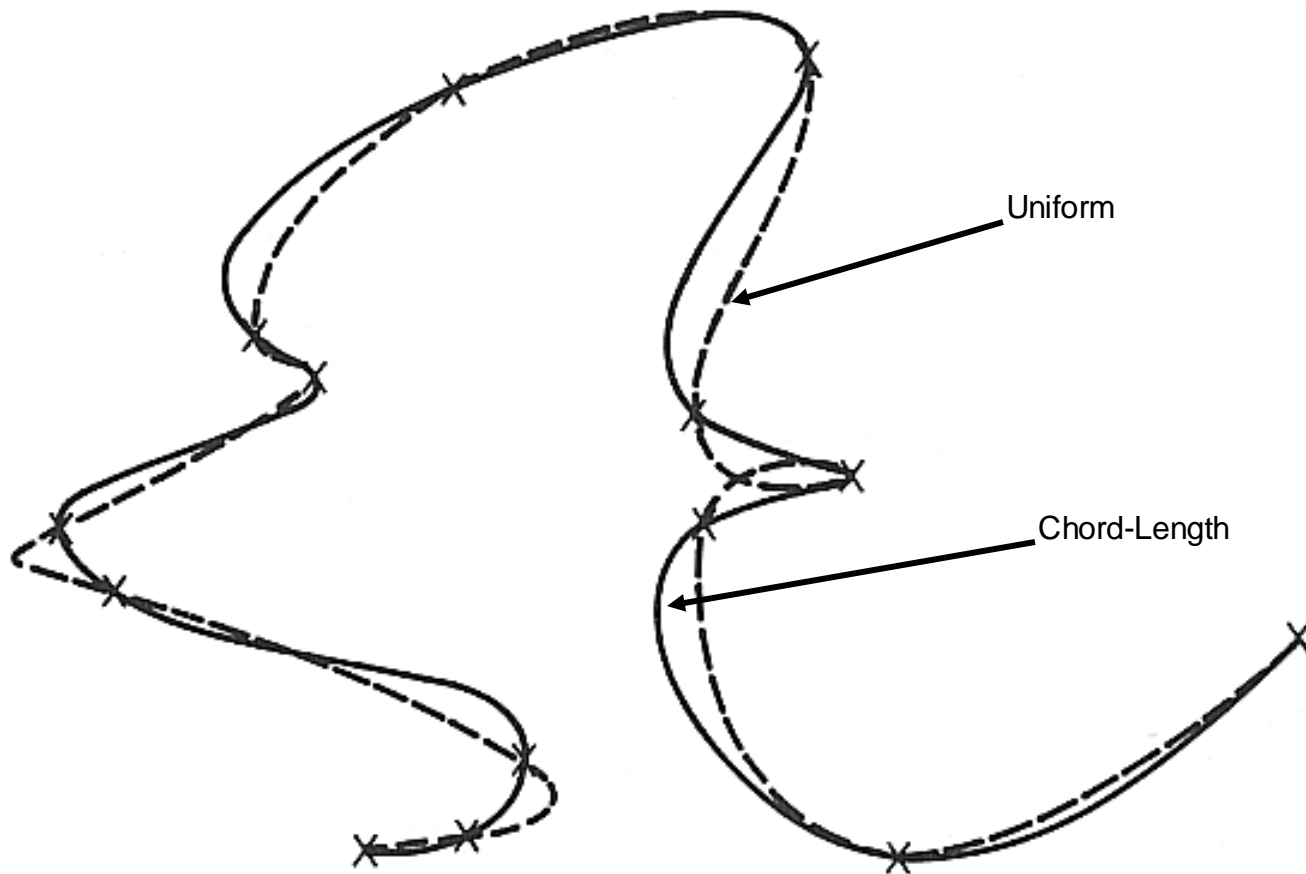
- Arbitrary up to a constant factor

- **Warning**

- Distances are not affine invariant !
 - Shape of curves changes under transformations !!
-

Parameterization

- **Chord-Length versus uniform Parameterization**
 - Analog: Think $P(t)$ as a moving object with mass that may overshoot



Spline Surfaces

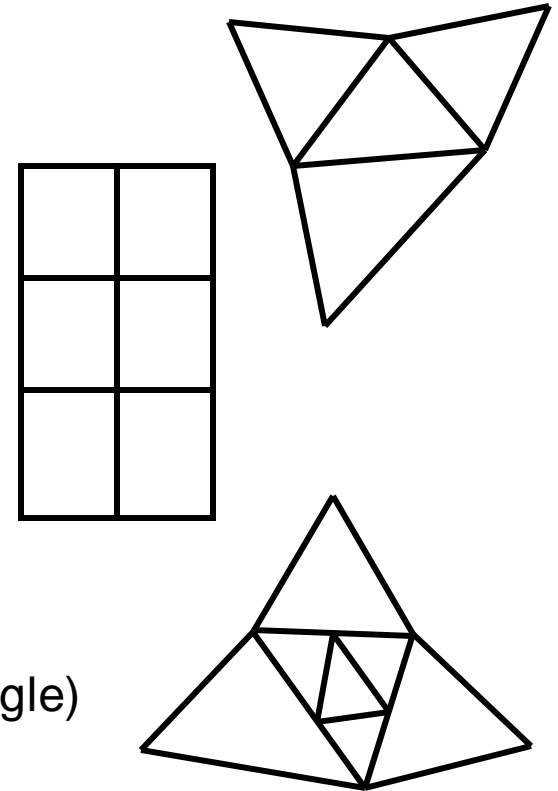
Parametric Surfaces

- **Same Idea as with Curves**

- $\underline{P}: \mathbb{R}^2 \rightarrow \mathbb{R}^3$
- $\underline{P}(u,v) = (x(u,v), y(u,v), z(u,v))^T \in \mathbb{R}^3$ (also $\mathbb{P}(\mathbb{R}^4)$)

- **Different Approaches**

- Triangular Splines
 - Single polynomial in (u,v) via barycentric coordinates with respect to a reference triangle (e.g. B-Patches)
- Tensor Product Surfaces
 - Separation into polynomials in u and in v
- Subdivision Surfaces
 - Start with a triangular mesh in \mathbb{R}^3
 - Subdivide mesh by inserting new vertices
 - Depending on local neighborhood
 - Only piecewise parameterization (in each triangle)



Tensor Product Surfaces

- **Idea**

- Create a “curve of curves”

- **Simplest case: Bilinear Patch**

- Two lines in space

$$\underline{P}^1(v) = (1 - v)\underline{P}_{00} + v\underline{P}_{10}$$

$$\underline{P}^2(v) = (1 - v)\underline{P}_{01} + v\underline{P}_{11}$$

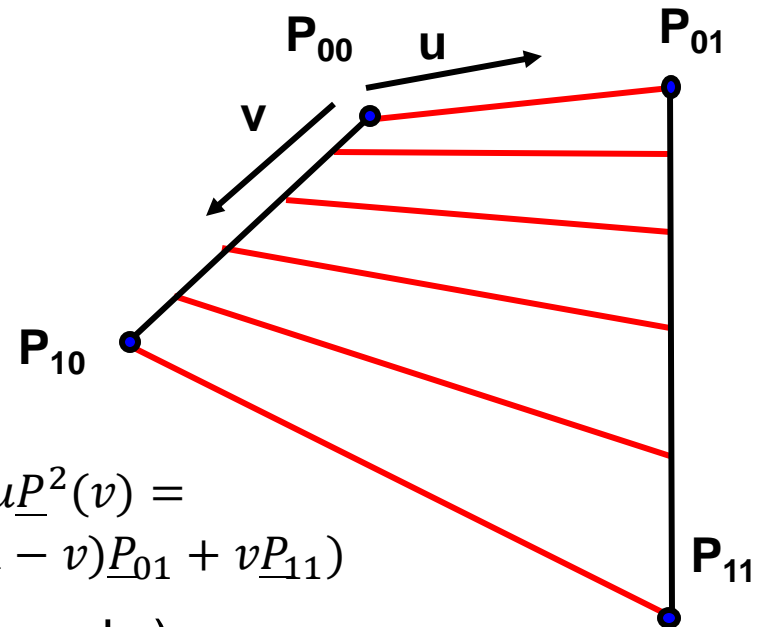
- Connected by lines

$$\begin{aligned}\underline{P}(u, v) &= (1 - u)\underline{P}^1(v) + u\underline{P}^2(v) = \\ &= (1 - u)((1 - v)\underline{P}_{00} + v\underline{P}_{10}) + u((1 - v)\underline{P}_{01} + v\underline{P}_{11})\end{aligned}$$

- Bézier representation (symmetric in u and v)

$$\underline{P}(u, v) = \sum_{i,j=0}^1 B_i^1(u)B_j^1(v)\underline{P}_{ij}$$

- Control mesh \underline{P}_{ij}



Tensor Product Surfaces

- **General Case**

- Arbitrary basis functions in u and v
 - **Tensor Product** of the function space in u and v
- Commonly same basis functions and same degree in u and v

$$\underline{P}(u, v) = \sum_{i=0}^m \sum_{j=0}^n B_i^m(u) B_j^n(v) \underline{P}_{ij}$$

- **Interpretation**

- Curve defined by curves

$$\underline{P}(u, v) = \sum_{i=0}^m B_i(u) \underbrace{\sum_{j=0}^n B_j(v) \underline{P}_{ij}}_{\underline{P}_i'(v)}$$

- Symmetric in u and v
-

Matrix Representation

- **Similar to Curves**

- Geometry now in a „tensor“ (m x n x 3)

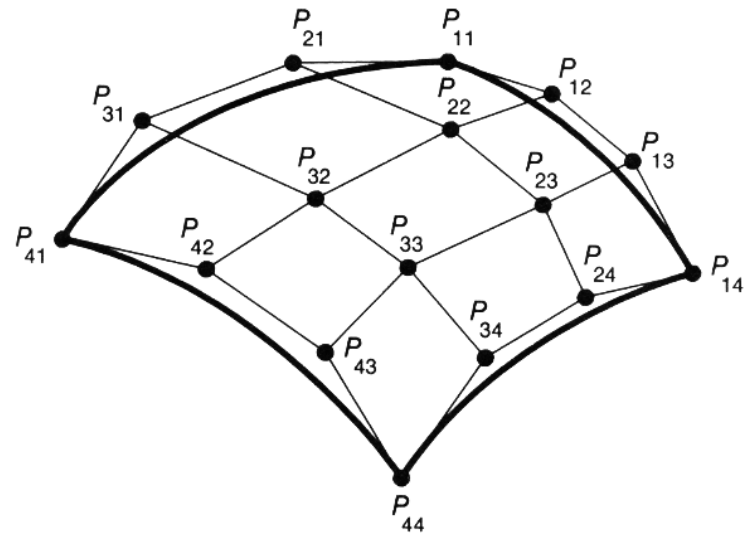
$$\underline{P}(u, v) = U G_{\text{monom}} V^T = (u^m \quad \dots \quad u \quad 1) \begin{pmatrix} G_{nn} & \dots & G_{n0} \\ \vdots & \ddots & \vdots \\ G_{0n} & \dots & G_{00} \end{pmatrix} \begin{pmatrix} v^n \\ \vdots \\ v \\ 1 \end{pmatrix} =$$
$$U B'_U G_{UV} B_V^T V^T$$

- Degree

- u: m
- v: n
- Along the diagonal (u=v): m+n
 - Not nice → „Triangular Splines“

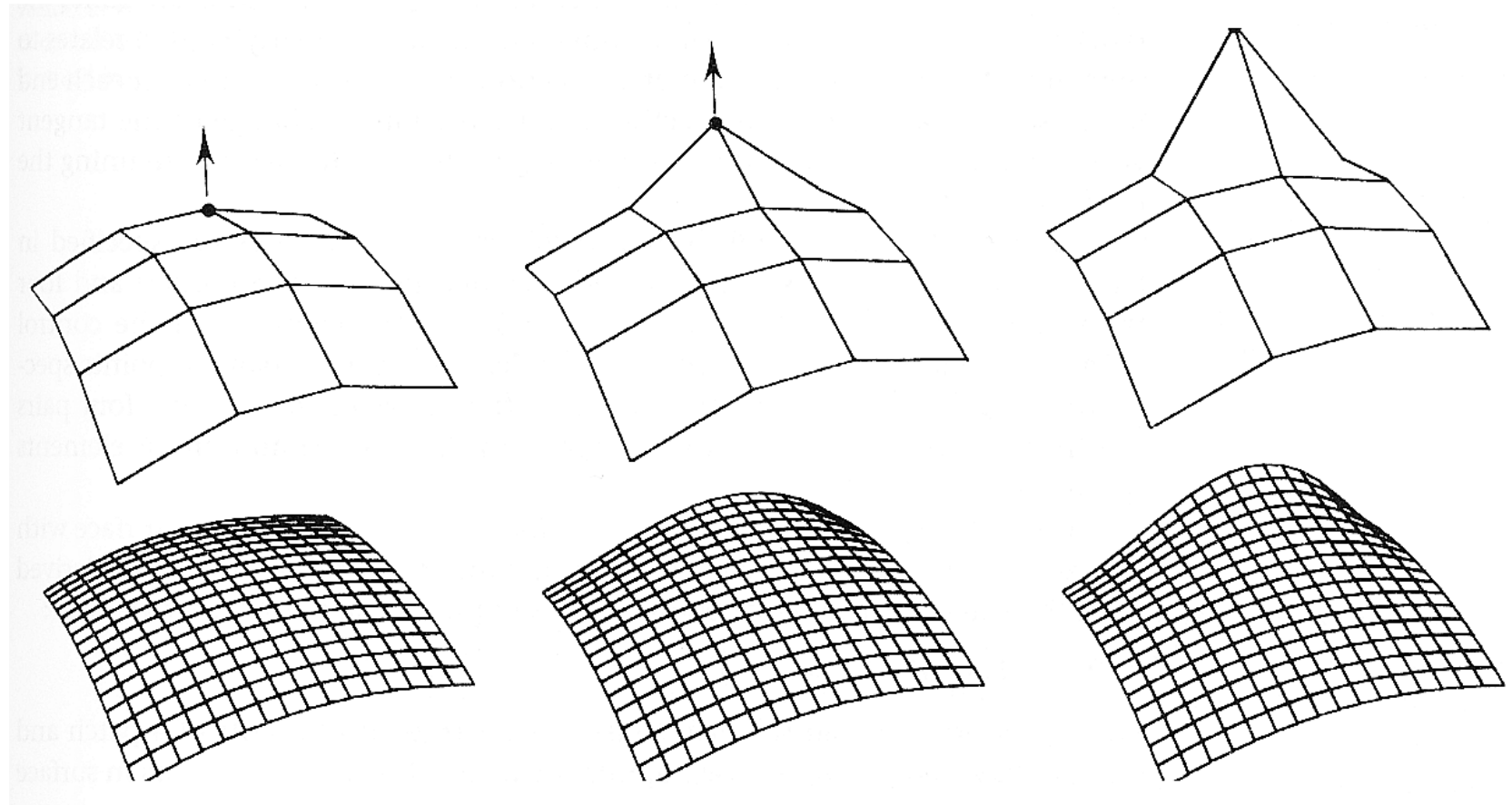
Tensor Product Surfaces

- **Properties Derived Directly From Curves**
- **Bézier Surface:**
 - Surface interpolates corner vertices of mesh
 - Vertices at edges of mesh define boundary curves
 - Convex hull property holds
 - Simple computation of derivatives
 - Direct neighbors of corners vertices define tangent plane
- **Similar for Other Basis Functions**



Tensor Product Surfaces

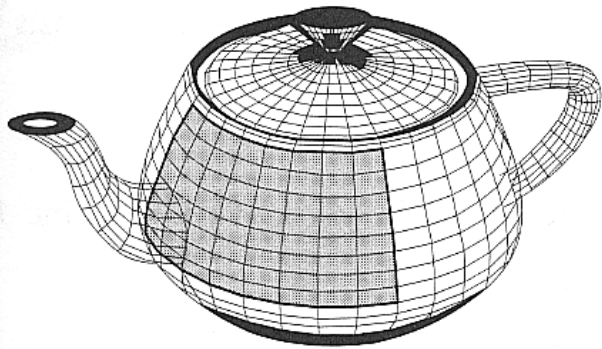
- **Modifying a Bézier Surface**



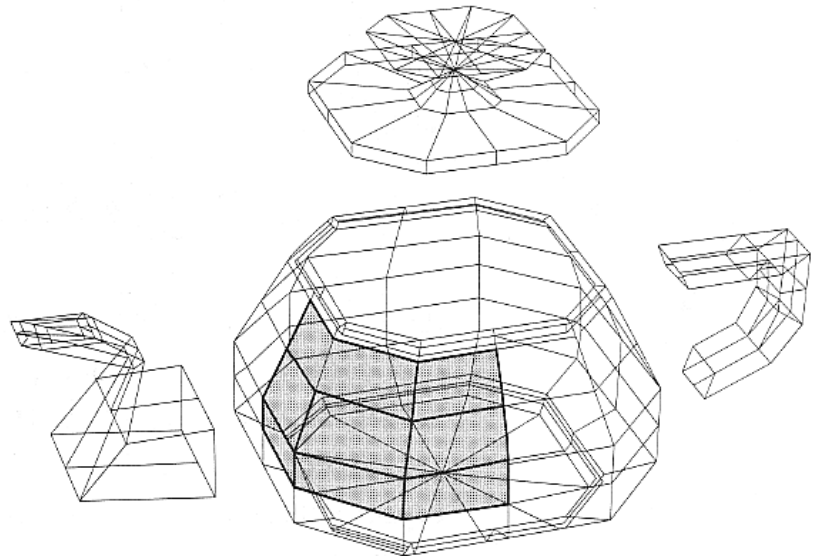
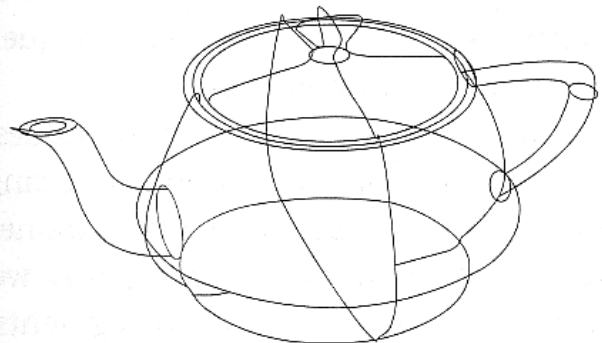
Tensor Product Surfaces

- **Representing the Utah Teapot as a set continuous Bézier patches**

– <http://www.holmes3d.net/graphics/teapot/>



(a)



(b)

Operations on Surfaces

- **deCausteljau/deBoor Algorithm**
 - Once for u in each column
 - Once for v in the resulting row
 - Due to symmetry also in other order
- **Similarly we can derive the related algorithms**
 - Subdivision
 - Extrapolation
 - Display
 - ...

Ray Tracing of Spline Surfaces

- **Several approaches**

- Tessellate into many triangles (using deCasteljau or deBoor)

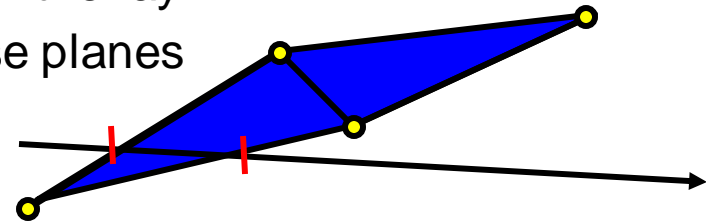
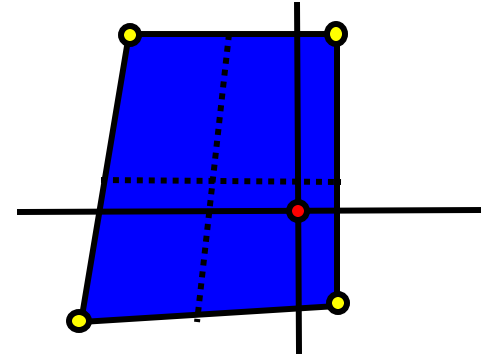
- Often the fastest method
 - May need enormous amounts of memory

- Recursive subdivision

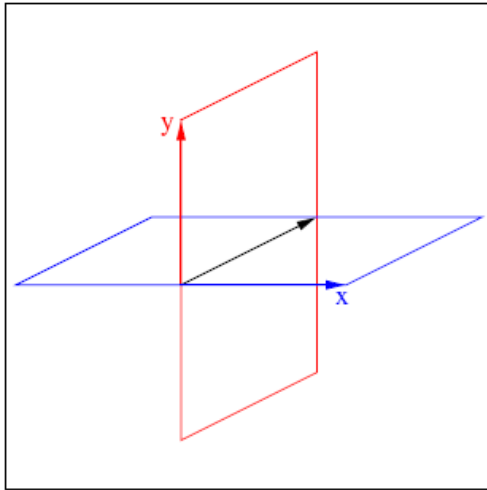
- Simply subdivide patch recursively
 - Delete parts that do not intersect ray (Pruning)
 - Fixed depth ensures crack-free surface
 - May cache intermediate results for next rays

- Bézier Clipping [Sederberg et al.]

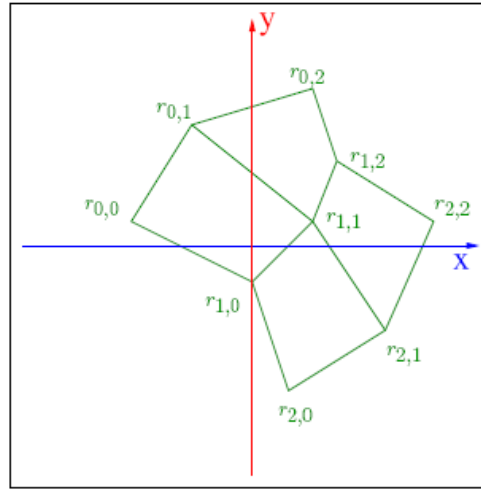
- Find two orthogonal planes that intersect in the ray
 - Project the surface control points into these planes
 - Intersection must have distance zero
 - ➔ Root finding
 - ➔ Can eliminate parts of the surface where convex hull does not intersect ray
 - Must deal with many special cases – rather slow



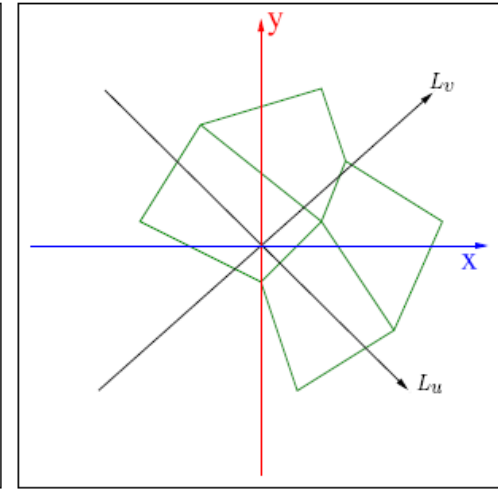
Bézier Clipping



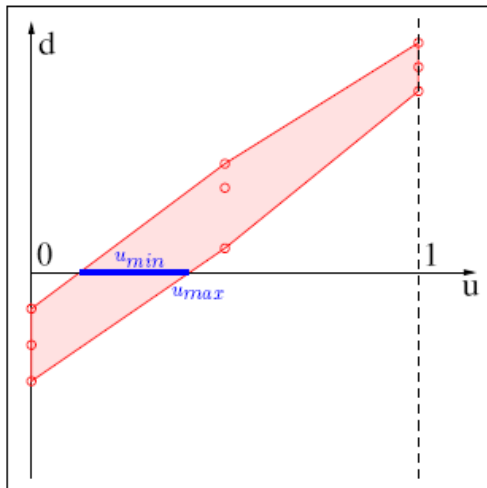
(a)



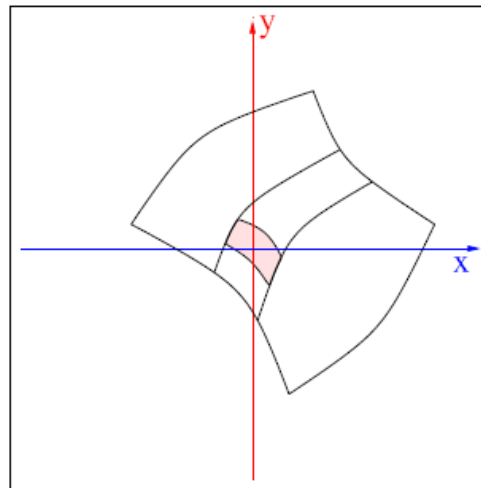
(b)



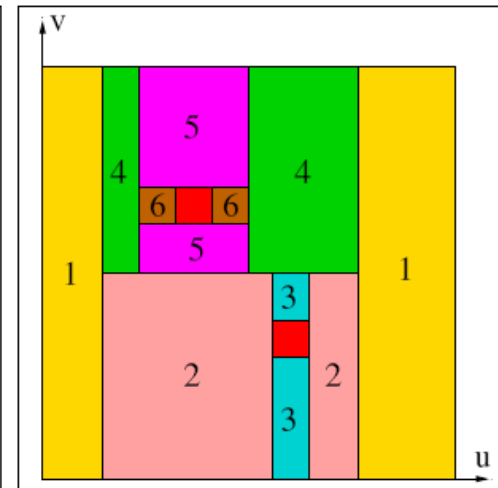
(c)



(d)

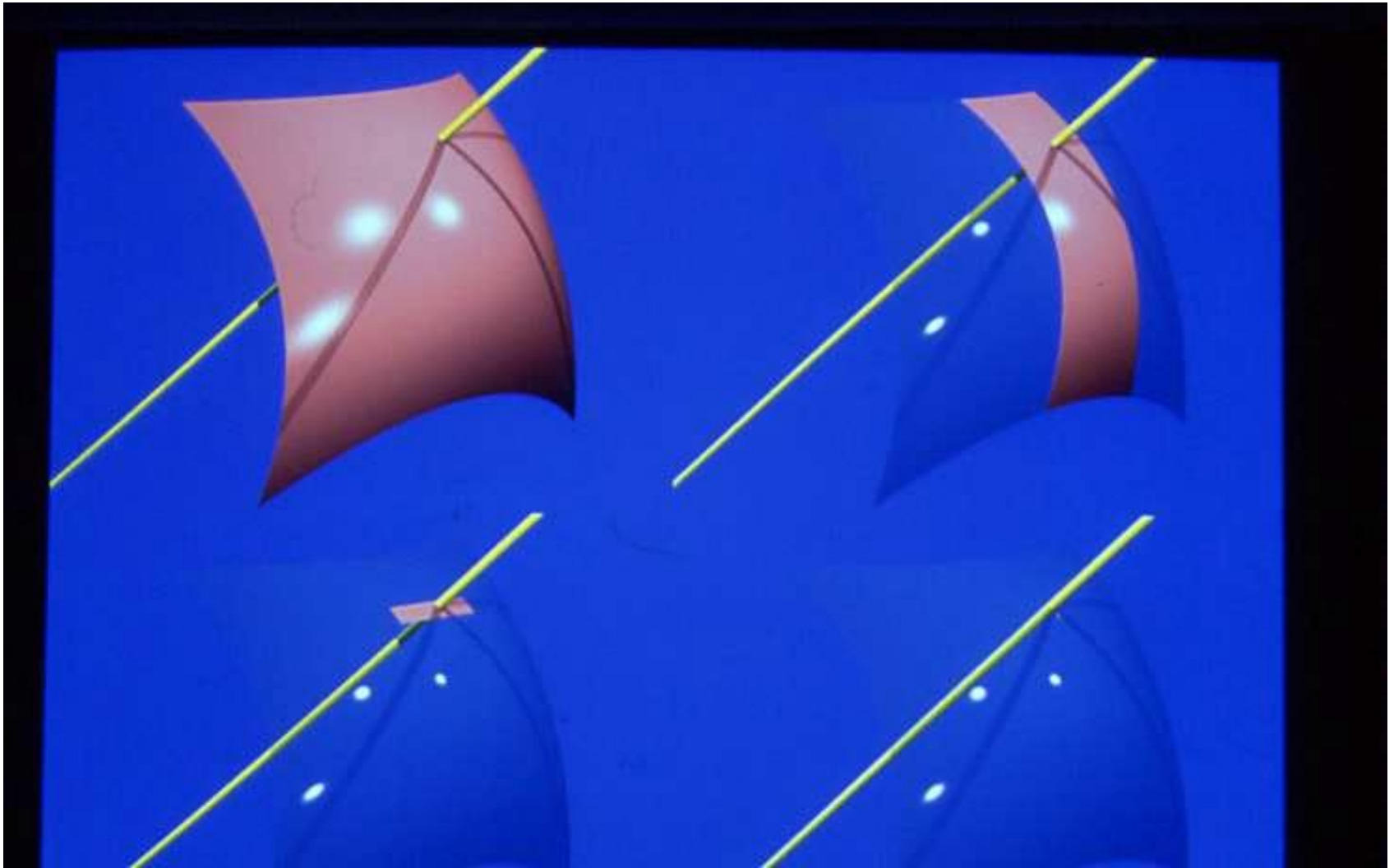


(e)



(f)

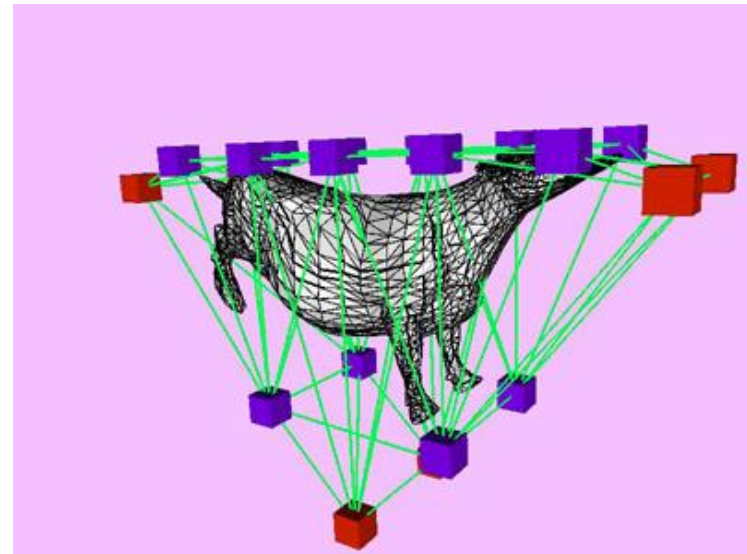
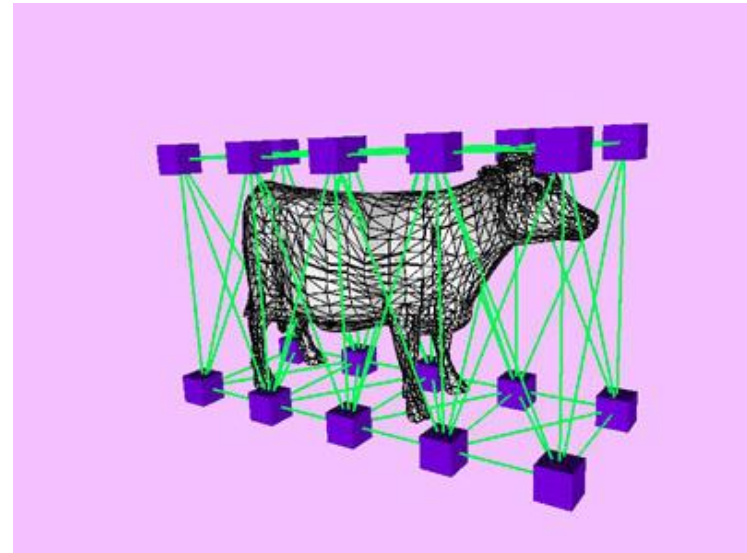
Bézier Clipping



Higher Dimensions

- **Volumes**

- Spline: $\mathbb{R}^3 \rightarrow \mathbb{R}$
 - Volume density
 - Rarely used
- Spline: $\mathbb{R}^3 \rightarrow \mathbb{R}^3$
 - Modifications of points in 3D
 - Displacement mapping
 - Free Form Deformations (FFD)



FFD