Prof. Reuven Aviv
Department of Computer Science
Tel Hai Academic College

Computer Graphics

Curves And Surfaces

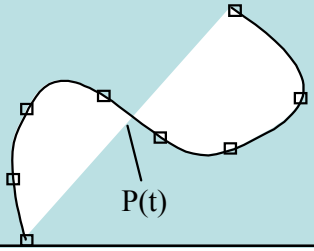
Slides adapted from F. Hill, S. Kelley Computer Graphics

Bezier Curves

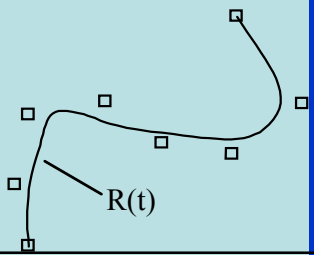
Interactive Curve Design

- 1. Lay down a set of initial *control points*
- 2. Use *an algorithm* to generate curve points
- 3. If the curve is satisfactory, stop
- 4. Else, move some control points
- 5. Go to step 2;

a). The curve interpolates the control points



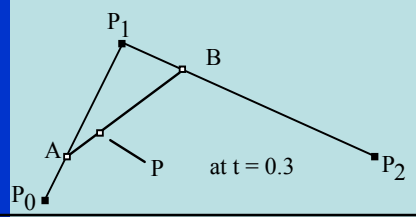
b). The curve approximates the control points



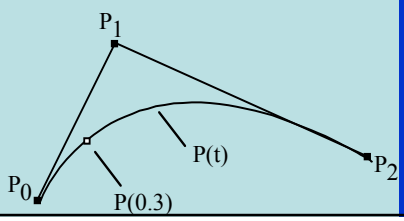
Example: a 3 control point Bezier Curve

- 3 control points: $\boldsymbol{P_0}$, $\boldsymbol{P_1}$, $\boldsymbol{P_2}$
- Let ***A*** be a fraction *t* along P_0P_1 (any t)
  - $\mathbf{A}(t) = (1 - t)\mathbf{P_0} + t\mathbf{P_1}$
- Let ***B*** a fraction *t* along $\mathbf{P_1P_2}$
  - $\mathbf{B}(t) = (1 - t)\mathbf{P_1} + t\mathbf{P_2}$
- Let **P(t)** be a fraction t along **AB**
- $\mathbf{P}(t) = (1-t)\mathbf{A(t)} + t\mathbf{B(t)} = (1-t)^2\mathbf{P_0} + 2t(1-t)\mathbf{P_1} + t^2\mathbf{P_2}$

a).

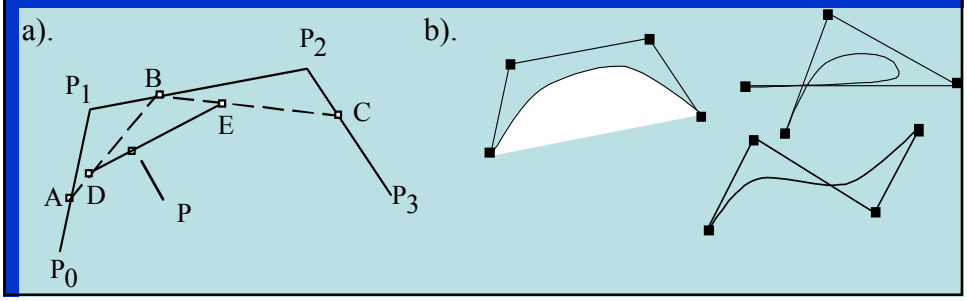


b).



4 control points (P_0, P_1, P_2, P_3) Bezier Curve

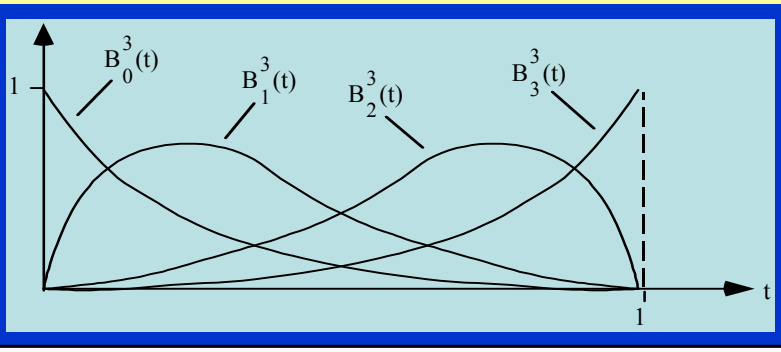
- **A**, **B**, **C** are placed a fraction t along $\mathbf{P_0P_1}$, $\mathbf{P_1P_2}$, $\mathbf{P_2P_3}$
  - then **D**, **E** are placed fraction t along **AB**, **BC**
  - Finally, point **P**(t) is placed fraction t along **DE**
- $\mathbf{P}(t) = \mathbf{P_0}(1-t)^3 + \mathbf{P_1}3(1-t)^2t + \mathbf{P_2}3(1-t)t^2 + \mathbf{P_3}t^3 \quad 0 <= t <= 1$
- $\mathbf{P_i}$ weighted by cubic *Bernstein Polynomials*



Bernstein Polynomials

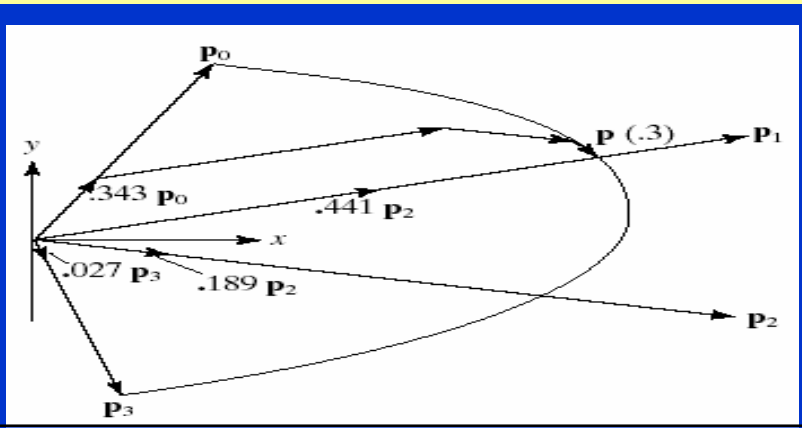
$B_0^3 = (1-t)^3$ $B_1^3 = 3(1-t)^2 t$ $B_2^3 = 3(1-t)t^2$ $B_3^3 = t^3$

- 4 terms of the expansion of $[(1-t)+t]^3$ all ≥ 0
- The 4 Polynomials add to 1
  - **P**(t) is a convex combination of the control points



Blending Points with Bernstein Polynomials

- vectors $\underline{\mathbf{p}_i} = \mathbf{P}_i - \mathbf{O}$ (from the origin to $\mathbf{P}_i$)
- calculate Bernstein polynomials for, say, $t = 0.3$
- $\underline{\mathbf{p}}(0.3) = 0.343\ \underline{\mathbf{p}}_0 + 0.441\ \underline{\mathbf{p}}_1 + 0.189\ \underline{\mathbf{p}}_2 + 0.027\ \underline{\mathbf{p}}_3$.



Bezier Curves: de Casteljau construction

- L+1 control points: $\mathbf{P}_0, \mathbf{P}_1, \mathbf{P}_2, \mathbf{P}_3, \ldots \mathbf{P}_L$
- L iterations of replacing points by affine combinations
  - (initial points are the control points)
  - new point is affine combination of previous pair of points
  - Number of points decreases by 1
- Repeat for every t value

$$P_i^4(t) = (1-t)P_i^3(t) + tP_{i+1}^3(t)$$

…

$$P_i^L(t) = (1-t)P_i^{L-1}(t) + tP_{i+1}^{L-1}(t)$$

Upper Index K: Iteration Number. K = 1, …L
Lower index i: point number: i = 0, .. L-K

Result: General Bezier Curves

$$P(t) = \sum_{k=0}^{L} P_k \, B_k^L(t) \qquad B_k^L(t) = \binom{L}{k}(1-t)^{L-k} t^k$$

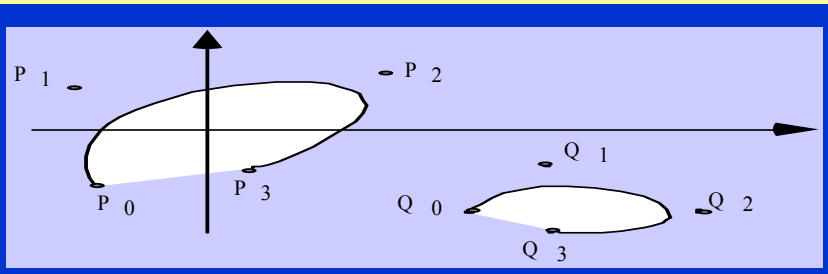
$$\binom{L}{k} = \frac{L!}{k!(L-k)!} \qquad \text{for L > K}$$

- $\boldsymbol{P}(t)$ based on $\boldsymbol{P_0}, \boldsymbol{P_1}, \ldots, \boldsymbol{P_L}$ always interpolates $\boldsymbol{P_0}$, $\boldsymbol{P_L}$
- To apply affine transformation on **P**(t): **Q**(t) = T**P**(t)
  - First transform control points: $\mathbf{P}_k \rightarrow T(\mathbf{P}_k)$

$$Q(t) = \sum_{k=0}^{L} T(P_k) B_k^L(t)$$

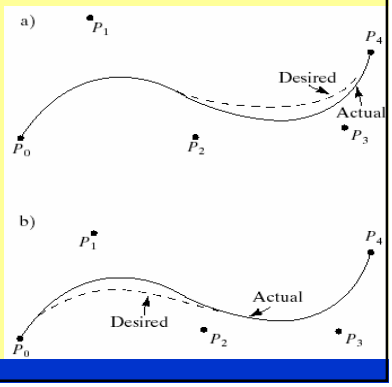
Example transforming Bezier Curve

- **P(t)** is controlled by four control points $\mathbf{P_0}$,..., $\mathbf{P_3}$.
- $\mathbf{P_k}$ points are rotated, scaled, & translated to $\mathbf{Q_k}$ = T($\mathbf{P_k}$)
- The Bezier curve for $\mathbf{Q_k}$ is drawn.
  - It is identical to the result of transforming the original Bezier curve **P**(t) to **Q**(t) = T**P**(t) for each t



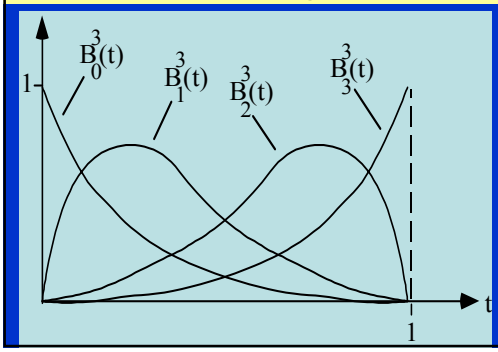
Weakness of Bezier Curves

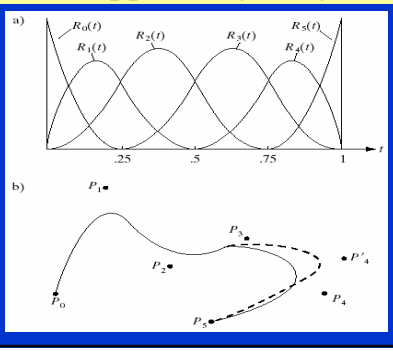
- 1. A Bezier curve with L+1 control points is a combination of L-degree polynomials.
  - designer might need many control points e.g. 40 or more
  - High degree polynomials are expensive to compute, and are vulnerable to numerical round-off errors.
- 2. We need local control of curve
  - Example:
  - To fix fig. a move $\mathbf{P}_2$, $\mathbf{P}_3$, up
  - This changes first half of curve
- Moving P_i changes entire curve



Analysis of the problem

- Bernstein polynomials are ≠ 0 over the entire interval [0, 1].
- Their *support* is over the entire interval [0, 1] (left figure)
- Bezier curve is a blend of these functions
- each control point has an effect on the curve at all t-values
  - Hence no local control
- We need blending functions with local support (right figure)

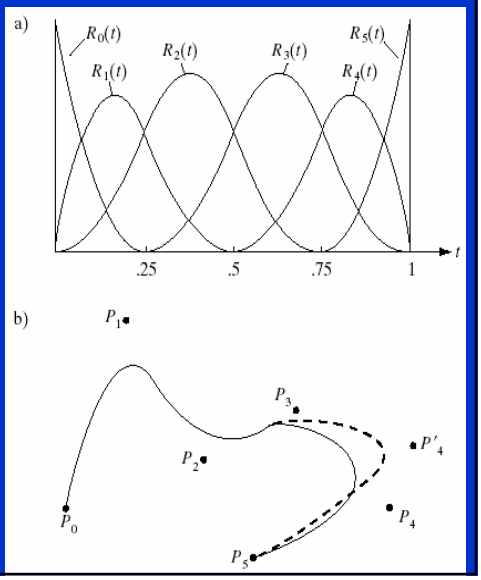




Spline Functions

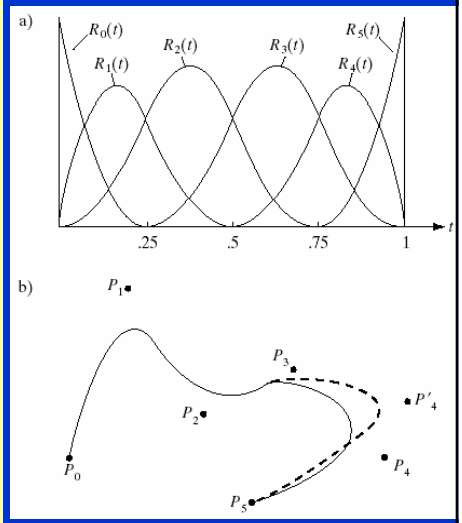
Better Blending Functions

- support of $R_0(t)$ is [0, .25]
- Support of $R_3(t)$ is [.25, 1.0]
- Consider a *t value*
  - no more than three of the blending functions are active ($\neq 0$)
- Curve is given by:
- $\mathbf{S}(t) = \Sigma R_k(t)\mathbf{P}_k$



Better Blending Functions (2)

- *For any t, S*(*t*) depends on no more than three P_k
- For all *t* in [0.75, 1.0] only P_3, P_4, P_5 control the shape of the curve.
- If P_4 is moved to P'_4, only the dashed portion of the curve will change.
- These blending functions give some local control to the control points.



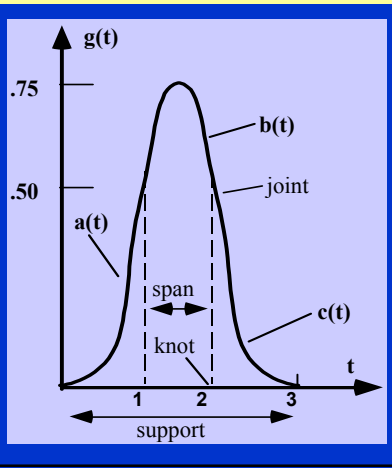
Wish List for Blending Functions

- be easy to compute and numerically stable
  - Polynomials, with small degrees
- sum to one at every *t* in [*a*,*b*]: $\Sigma R_k(t) = 1$
- have small support to offer local control
- interpolate certain control points, chosen by designer
- be smooth enough: Have continuous derivative
  - Derivatives = 0 at t=0, 1 (no jumps of R_k there)
  - Try cubic polynomial: $R(t) = at^3 + bt^2 + ct + d$
  - No solution for $R(0) = R'(0) = R(1) = R'(1)$

Quadratic spline function

- Construct $g(t)$ via three 2nd order polynomial segments: a(t), b(t), c(t)
  - piece-wise polynomial

- g(t) = a(t), span [0,1]
- g(t) = b(t) , span [1,2]
- g(t) = c(t) , span [2,3]
- first derivative continues at knot points t = 0, 1, 2, 3
- Note: interval ≠ [0,1]

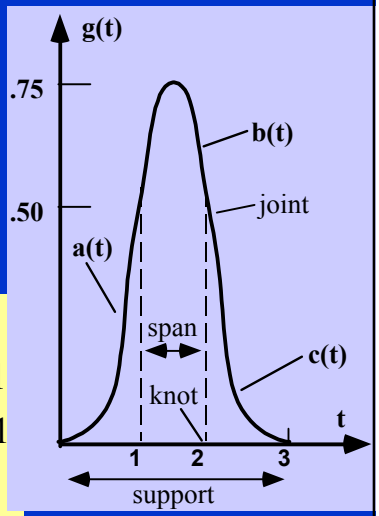


Quadratic spline function (2)

$$a(t) = \frac{1}{2}t^2$$

$$b(t) = \frac{3}{4} - (t - \frac{3}{2})^2$$

$$c(t) = \frac{1}{2}(3 - t)^2$$



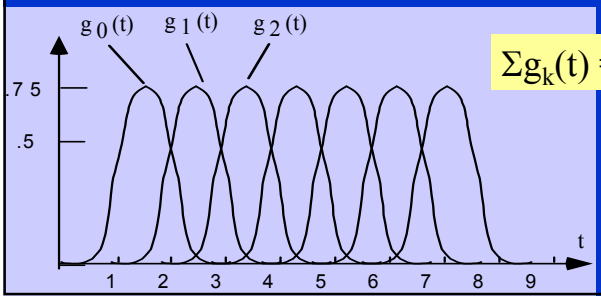
- Knots: t = 0, 1, 2, 3
  - a(1) = b(1) = ½ ; a'(1) = b'(1) = 1
  - b(2) = c(2) = ½; b'(2) = c'(2) = -1
  - a(0) = a'(0) = c(3) = c'(3) = 0
- Need L such functions (1 per P_k)!

Spline Functions

- An n^{th}-degree spline function is a piecewise polynomial of degree n
  - With the (n-1) derivative is continuous at knots
- The example $g(t)$ is a quadratic spline
  - It is a piecewise polynomial of degree 2 and has a continuous first derivative everywhere
- From a spline function $g(t)$ we build the blending functions, $g_k(t)$

Making Blending Functions from Splines

- use translated versions of g(t), $g_k(t)$ as blending functions
- each $g_k(t)$ is formed by translating $g(t)$ by a certain amount
- Example: seven blending functions $g_0(t),..,g_6(t)$
  - formed by translating g(.) by integer amounts:
  - $g_k(t) = g(t-k)$ for k = 0, 1, 2, …

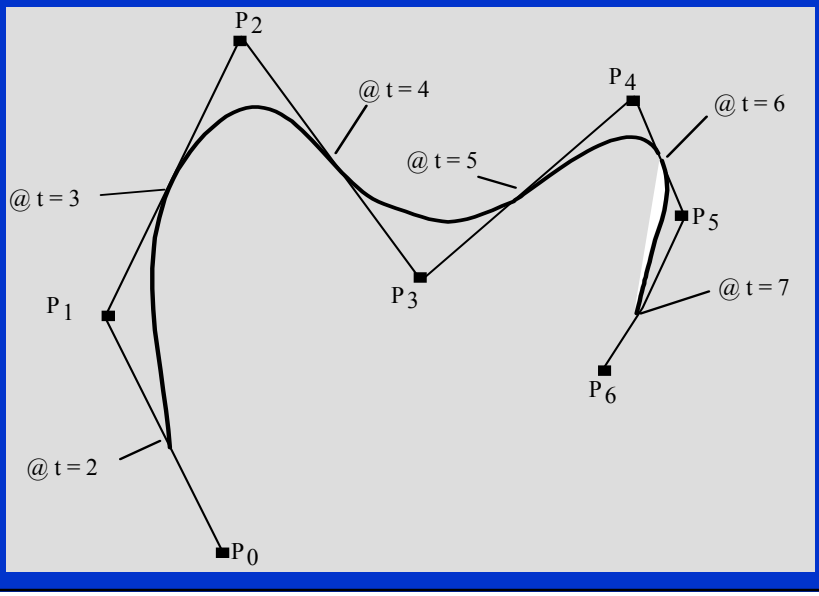


$\Sigma g_k(t) = 1 \quad 2 <= t <= 7$

Generating a curve

- The designer chooses 7 control points $\mathbf{P}_k(t)$
- the curve is generated from $\mathbf{S}(t) = \Sigma\ \mathbf{P}_k g(t-k)$
- Only values of *t* between 2 and 7 can be used.
  - In that range, exactly three of the blending functions are active at any value of *t*
  - good local control of the curve's shape.
- at knots $t = 2, 3, .., 7$ only two of $g_k(t)$ are active, both = 1/2
- $\mathbf{S}(2) = \mathbf{P}_0 g(2-0) + \mathbf{P}_1 g(2-1) = (\mathbf{P}_0 + \mathbf{P}_1)/2$
- $\mathbf{S}(3) = (\mathbf{P}_1 + \mathbf{P}_2)/2$, ….$\mathbf{S}(7) = (\mathbf{P}_5 + \mathbf{P}_6)/2$

Controlling the curve



Properties of the Curve

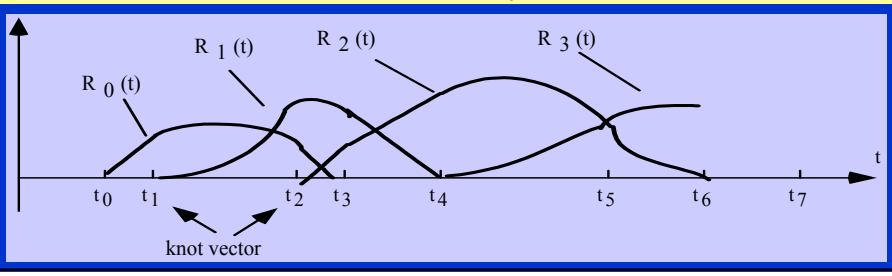
- The designer has some local control of the curve shape
  - since the support of the blending functions is length 3
- the curve pass through midpoints of the polygon edges
  - The curve has intuitive geometric properties.
- each blending function is 1-smooth,
  - whole curve is 1-smooth (1'st derivative is continuous)
- No points on the curve are interpolated
- All polynomials are of degree two,
  - so they are fast and stable to compute.
- The degree of the polynomials does not depend on the number of control points.
  - The technique works for any number of control points

More General Blending Functions

- We need more control of the curve shape
  - it must bend more and be smoother than just 1-smooth.
- This suggests moving to cubic polynomials
- We also want the designer to be able to specify which control points are interpolated.
- And we want a single algorithm that would encompass all of the design techniques described above — including Bezier curves.
- So we want to develop more general families of blending functions that meet all the properties discussed in the earlier wish list.

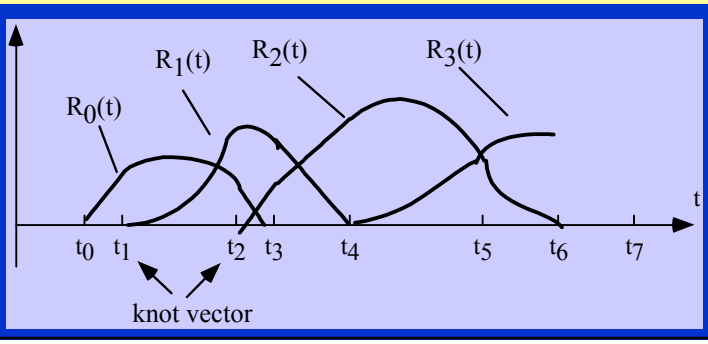
More General Blending Functions (2)

- L+1 control points, P_k, k = 0, 1, 2, …L
- L+1 blending functions $R_0(t)$, ..., $R_L(t)$.
- P(t) an affine sum of the control points: $\mathbf{S}(t) = \Sigma \mathbf{P}_k R_k(t)$
- Blending functions are piecewise polynomials
  - defined on a more general sequence of knots
  - called the knot vector. T = [t_0, t_1, t_2, …], with $t_i \leq t_{i+1}$.



More General Blending Functions (3)

- Each blending function $R_k(t)$ is a piecewise polynomial
  - zero up to time t_k, then non-zero over several spans in the knot vector, and then returns to zero again.
- each $R_k(t)$ is a spline function
- ensures a certain level of smoothness at all t in its support



B-spline (Basis) Functions

- B-spline functions of order n: $b_{i,n}$ i = 0, 1, …
  - pieces of polynomials of degree n-1
- Two most important cases
- n = 3, underlying polynomials of degree 2
  - quadratic B-splines.
- n = 4, underlying polynomials of degree 3
  - cubic B spline
- B-splines of any order can be constructed

B-spline Basis Functions (2)

- Define a knot vector:
- **T** = (t_0, t_1, t_2 . . .t_m) $t_k \leq t_{k+1}$
- A curve is defined by:
  - a set of (m-n) B-spline functions of order n
    - $b_{i,n}(t)$, i = 0, 1, .. m-n-1

– and (m-n) control points $\mathbf{P}_i$, i = 0, 1, …m-n-1

- $\mathbf{P}(t) = \Sigma \mathbf{P}_i b_{i,n}(t)$ $t_n \leq t \leq t_{m-n}$ m>= 2n
- sum over i = 0, 1, …m-n-1

B-spline Basis Functions (3)

- $b_{i,n}(t)$, begins at t_i and ends at t_{i+n+1}
- Its support is $[t_i, t_{i+n+1}]$ "n+1 knot span"
  - $\mathbf{P}_i$ controls $\mathbf{S}(t)$ in the interval $[t_i, t_{i+n+1}]$
- The support of the family of functions, $b_{i,n}(t)$, for $i = 0, \ldots,$ L is the interval $[t_0, t_{n+L}]$.

B-splines: Usage and construction

$$\mathbf{S}(t) = \sum_{i=0}^{m-n-1} \mathbf{P}_i b_{i,n}(t) \ , \ t \in [t_n, t_{m-n}]$$

$$b_{j,0}(t) := \begin{cases} 1 & \text{if} \quad t_j \leq t < t_{j+1} \\ 0 & \text{otherwise} \end{cases}$$

$$b_{j,n}(t) := \frac{t - t_j}{t_{j+n} - t_j} b_{j,n-1}(t) + \frac{t_{j+n+1} - t}{t_{j+n+1} - t_{j+1}} b_{j+1,n-1}(t).$$

Lowest order B-splines

- n = 0 defined on one knot span, constant

$$b_{j,0}(t) = 1_{[t_j, t_{j+1})} = \begin{cases} 1 & \text{if} \quad t_j \le t < t_{j+1} \\ 0 & \text{otherwise} \end{cases}$$

- n = 1 defined on two consecutive knot spans, tent function

$$b_{j,1}(t) = \begin{cases} \frac{t - t_j}{t_{j+1} - t_j} & \text{if} \quad t_j \le t < t_{j+1} \\ \frac{t_{j+2} - t}{t_{j+2} - t_{j+1}} & \text{if} \quad t_{j+1} \le t < t_{j+2} \\ 0 & \text{otherwise} \end{cases}$$

- n = 2 defined on 3 consecutive knot spans, tents & parabola

$$b_{j,2}(t) = \begin{cases} \frac{1}{2}t^2 \\ -t^2 + t + \frac{1}{2} \\ \frac{1}{2}(1 - t)^2 \end{cases}$$

Quadratic and Cubic B-spline curves

- quadratic B-spline curve
  - Each triplet of control points define a segment

$$\mathbf{S}_i(t) = \begin{bmatrix} t^2 & t & 1 \end{bmatrix} \frac{1}{2} \begin{bmatrix} 1 & -2 & 1 \\ -2 & 2 & 0 \\ 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{p}_{i-1} \\ \mathbf{p}_i \\ \mathbf{p}_{i+1} \end{bmatrix}$$

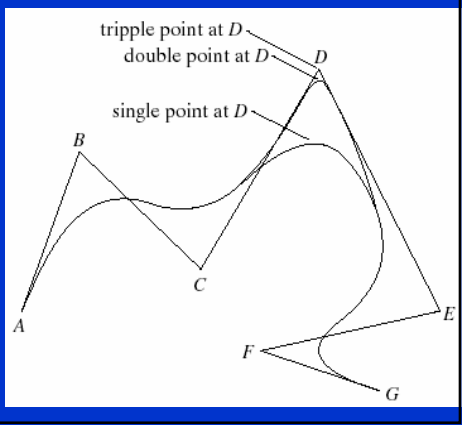
- Cubic B-spline curve
  - Each quartet of control points define a segment

$$\mathbf{S}_i(t) = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \frac{1}{6} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{p}_{i-1} \\ \mathbf{p}_i \\ \mathbf{p}_{i+1} \\ \mathbf{p}_{i+2} \end{bmatrix}$$

Using Multiple Control Points

- cubic B-spline curves, control points A, B, … G
- double control point at D:
  - control polygon is A, B, C, D, D, E, F, G
- the curve is pulled toward D.

- triple point at D,
- control polygon A, B, C, D, D, D, E, F, G,
- the curve practically interpolates D.

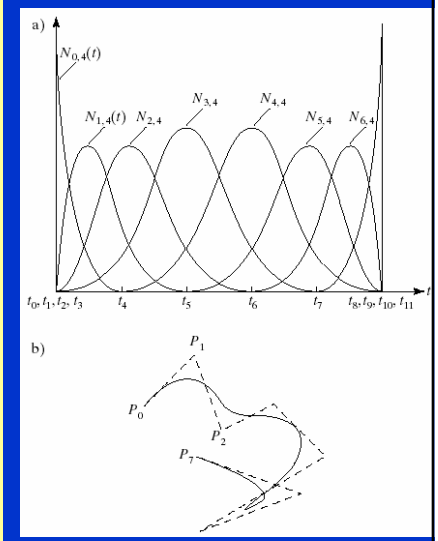


Standard Knot Vector

- The **standard knot vector** for a B-spline of order n begins and ends with a knot of multiplicity n and uses unit spacing for the remaining knots.
- Example
- 8 control points
- and we want to use cubic ($n = 4$) B-splines.
- The standard knot vector turns out to be
- **T** = (0, 0, 0, 0, 1, 2, 3, 4, 5, 5, 5, 5)
  - i.e 6 different knots
- those at the ends are multiple knots

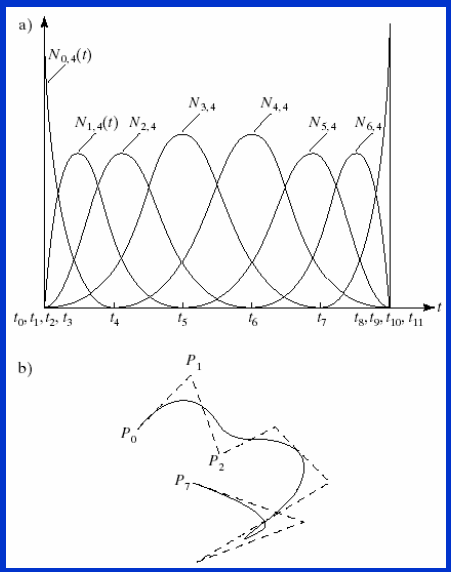
Standard Knot Vector (2)

- $b_{0,4}(t)$ and $b_{7,4}(t)$ discontinuous and have support of one unit span.
- Only $b_{3,4}(t)$ and $b_{4,4}(t)$ have the usual span of four units.
- other functions have 2 or three unit spans
- shapes distorted as they approach the first and last knots.



Standard Knot Vector (3)

- standard knot vector ensures interpolation of the first and last control points.
- direction of the B-spline curve at $t = 0$ is along the first segment of the control polygon, and similarly for the final direction.
- Note that a B-spline curve can cross itself when the control polygon does.



Standard Knot Vector (4)

- Standard knot vector for (L+1) control points and order-n B-splines
- L+n+1 knots, denoted as $t_0, \ldots, t_{L+n}$.
- The first n knots, $t_0, \ldots, t_{n-1}$, all share the value 0.
  - The first n blending functions start at $t = 0$
- Knots $t_n, \ldots, t_L$ increase in increments of 1, from value 1 through value L - n+1.
- The final n knots, $t_{L+1}, \ldots, t_{L+n}$, all equal L - n + 2.

Appendix 1: NURBS

Non Uniform Rational B-splines (NURBS)

$$P(t) = \sum_{k=0}^{L} P_k R_k(t)$$

- blending functions R_k are weighted B-splines
- Weights: $\{w_0, w_1, \ldots w_L\}$

$$P(t) = \frac{\sum_{k=0}^{L} w_k P_k N_{k,m}(t)}{\sum_{k=0}^{L} w_k N_{k,m}(t)}$$

- $N_{k,m}$ are m'th order, non-uniform knots, B-splines
- If all W are the same, we get the usual B-splines

Advantages of NURBS

- NURBS are invariant under **perspective transformation M**
- **M = (m1|m2|m3|m4)**
- drawing a perspective projection of a NURB curve
- find the perspective projection of each of its control points
- then draw the curve using the *same* blending functions
  - The weights must be adjusted as well

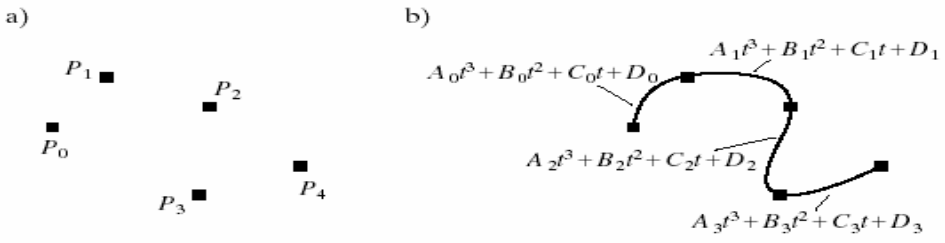
$$T(P(t)) = \frac{\sum_{k=0}^{L} w'_k T(P_k) N_{k,m}(t)}{\sum_{k=0}^{L} w'_k N_{k,m}(t)}$$

$$w'_k = w_k (P'_k \cdot m_4)$$

Appendix 2: Interpolation

Piecewise Cubic Polynomials

- need to draw smooth curves that pass through given set of control points $\mathbf{P}_k = (X_k, Y_k)$ (here $L = 4$)
- use segments of cubic polynomials:
- $\mathbf{R_k(t)} = A_k t^3 + \mathbf{B}_k t^2 + \mathbf{C}_k t + \mathbf{D}_k$ k= 0, 1, …, L-1, t in [0, 1]
- $\boldsymbol{A}_k$, $\mathbf{B}_k$, $\boldsymbol{C}_k$, $\mathbf{D}_k$ have 2 components each. 4L unknowns.
- Given (X_k, Y_k) & required derivatives at $\mathbf{P}_k$, (U_k, V_k)



Hermite Interpolation

- Segment k: $x_k(t) = a_k t^3 + b_k t^2 + c_k t + d_k$ (similar for $y_k(t)$)
- Require segments to pass through X_k X_{k+1} at t = 0, 1
  - $t = 0$: $d_k = X_k$.
  - $t = 1$: $a_k + b_k + c_k + d_k = X_{k+1}$
- Require derivative to equal given values U_k U_{k+1}at $t = 0, 1$
  - $t = 0$: $c_k = U_k$,
  - $t = 1$: $3a_k + 2b_k + c_k = U_{k+1}$
- These conditions hold for $k = 0,.., L\text{-}1$
- $4L$ conditions on the $4L$ unknown coefficients.

Piecewise Cubic Polynomials (4)

- For k = 0, 1, ..L-1

  $a_k = U_{k+1} + U_k - 2(X_{k+1} - X_k)$

  $b_k = 3(X_{k+1} - X_k) - 2\,U_k - U_{k+1}$

  $c_k = U_k$

  $d_k = X_k$
- We need to be able to identify required slopes U_k

Example

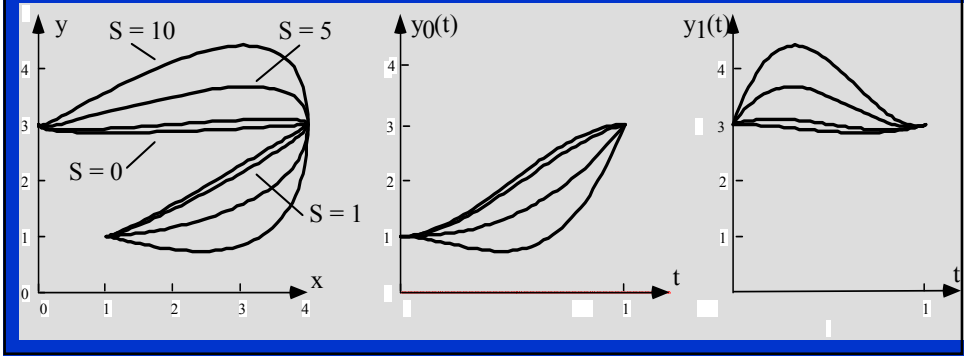
- 3 control points: (1,1), (4, 3), (0,3)
- a curve consisting of two cubic segments, $\boldsymbol{R}_0(t)$ and $\boldsymbol{R}_1(t)$
- Segments join at (4,3)
- $R_0(t)$ passes through (1,1) at $t = 0$ and through (4,3) at $t = 1$.
  - its slope is (1,0) at $t = 0$ and is (0, S) at $t = 1$
- $R_1(t)$ passes through (4, 3) at $t = 0$ and through (0,3) at $t = 1$.
  - Its slope is (0, S) at $t = 0$ and (0, 1) at $t = 1$
- S varies to see the effect of slope changes in the curve

Example (2)

- Segment 0:
- $x_0(t) = -5t^3 + 7t^2 + \mathrm{T} + 1$
- $y_0(t) = (S - 4)t^3 + (6 - S)t^2 + 1$
- Segment 1:
- $x_1(t) = 8t^3 - 12t^2 + 4$
- $y_1(t) = (S + 1)t^3 - (2S + 1)t^2 + St + 3$
- Now change S, look near joint (4,3)
- Only y behavior changes

Example (3)

- magnitude of the slope at the joint is changed
- Direction is not
- S affects the coefficients of $y_0(t)$ and $y_1(t)$
- but has no effect on the x-component functions.



Natural Cubic Splines

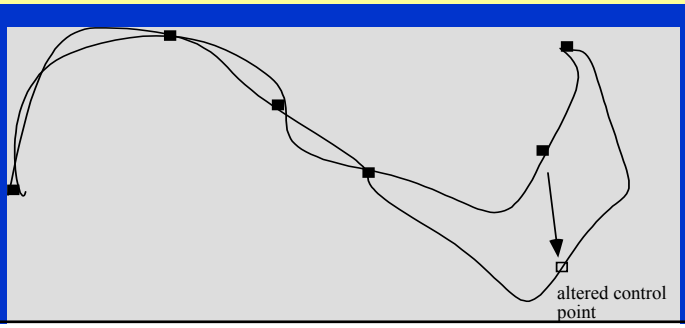
- Setting derivative values U_k, V_k
- Demand $x_k''(t)$, $y_k''(t)$ to be continuous at joints
- Second derivative of segment: $x_k''(t) = 6a_k t + 2b_k$
- Require: $x_{k-1}''(1) = x_k''(0) = 0$

  $-6a_{k-1} + 2b_{k-1} = 2b_k$ k in [0, L-1]

  $-U_{k-1} + 4\,U_k + U_{k+1} = 3(X_{k+1} - X_{k-1})$, k in [1, L-1]
- at endpoints: Impose $x_k''(0) = x_{L-1}''(0) = 0$
- $b_0 = 0$; $3a_{L-1} + b_{L-1} = 0$
- $2U_0 + U_1 = 3(X_1 - X_0)$; $2U_L + U_{L-1} = 3(X_L - X_{L-1})$

Example Natural Cubic spline interpolation

- visually the curve is smooth everywhere;
- at the ends it straightens so the second derivative vanishes.
- effect of moving one of the control points.
- The shape of the curve is affected everywhere. There is no local control with natural splines.

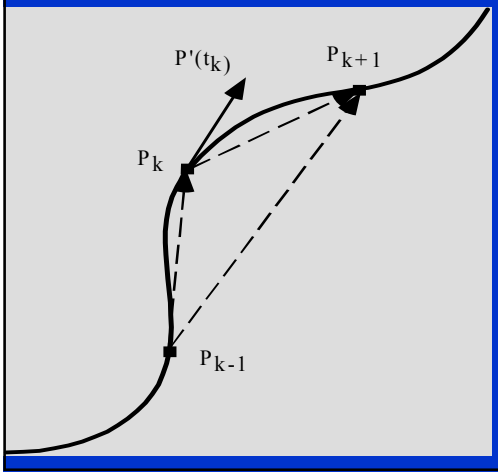


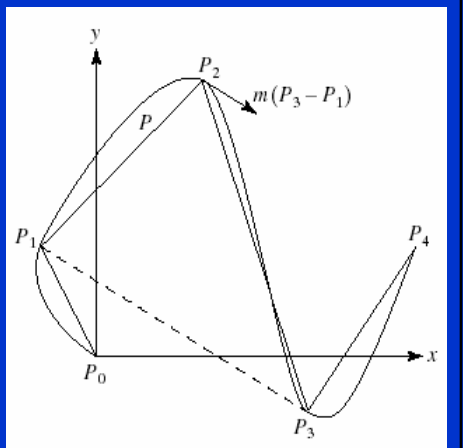
Simple Catmull-Rom Splines

- another method to fix the slopes at the joints.
- 1. require curve to be 1-smooth at the joints (as before)
- 2. do not require that the curve will be 2-smooth there.
  - Hoping to get local control of the curve's shape.
- slopes at joints determined by positions of their neighbors
- Simplest approach: force slope vector at $\boldsymbol{P}_k$, $\underline{\mathbf{P'}}(t_k)$, to be proportional to the vector from $\boldsymbol{P}_{k-1}$ to $\boldsymbol{P}_{k+1}$:
  - $\underline{\mathbf{P'}}(t_k) = m(\mathbf{P}_{k+1} - \mathbf{P}_{k-1})$ usually m = ½
- curve at P_k moves parallel to the direction $\boldsymbol{P}_{k-1}\boldsymbol{P}_{k+1}$

Simple Catmull-Rom Splines (2)

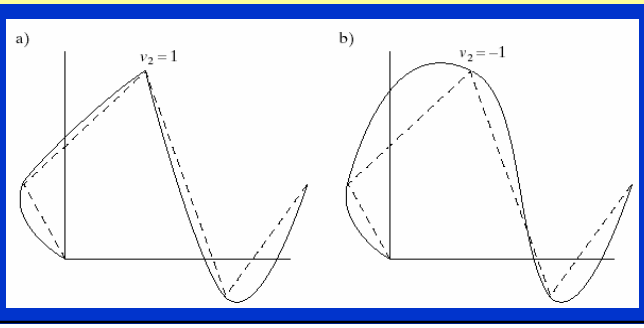
- curve at P_k moves parallel to the direction $\boldsymbol{P_{k-1}P_{k+1}}$
- Endpoints: zero second derivative (as before)





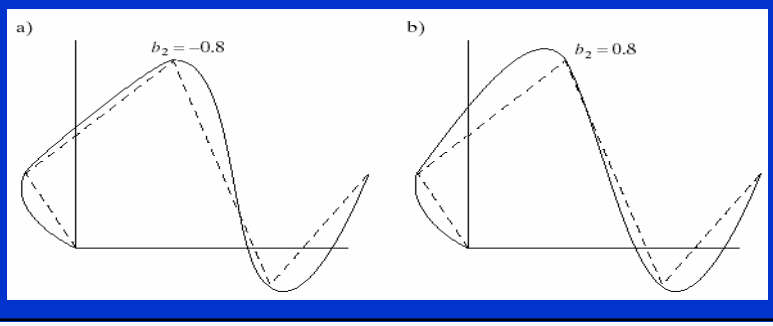
Catmull-Rom tension parameter

- Aim: greater control of the shape at each joint
- By parametrising *m:* $\underline{\mathbf{P}}'(t_k) = \frac{1}{2}(1 - v_k)(\mathbf{P}_{k+1} - \mathbf{P}_{k-1})$
- adjusting magnitude of the slope $\mathbf{P}'(t)$ not its direction.
- Normally $-1 \leq v_k \leq 1$, but it can have any value
- (a) $v_2 = 1$, slope = 0 at the joint, curve straightens at joint
- (b) $v_2 = -1$; the curve is more "slack" at the joint



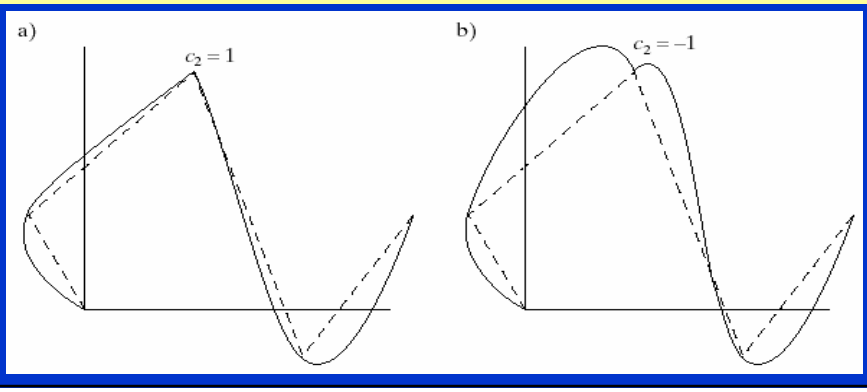
Kochanek-Bartels Splines

- Giving a bias to one side of a joint
- Define the derivative at joint P_k by
  - $P'(t_k) = \frac{1}{2}(1-b_k)(P_k - P_{k-1}) + \frac{1}{2}(1+b_k)(P_{k+1} - P_k)$
  - The bias parameter b_k weighs sides of P_k



Kochanek-Bartels Splines (2)

- Direct Control of continuity, by specifying slopes before and after joints
- $R'_{k-1}(1) = \frac{1}{2}(1-c_k)(P_k - P_{k-1}) + \frac{1}{2}(1+c_k)(P_{k+1} - P_k)$
- $R'_k(0) = \frac{1}{2}(1+c_k)(P_k - P_{k-1}) + \frac{1}{2}(1-c_k)(P_{k+1} - P_k)$
  - If $c_k \neq 0$, the curve is no longer 1-smooth at the joint.



Appendix 3:
Modeling Curved Surfaces

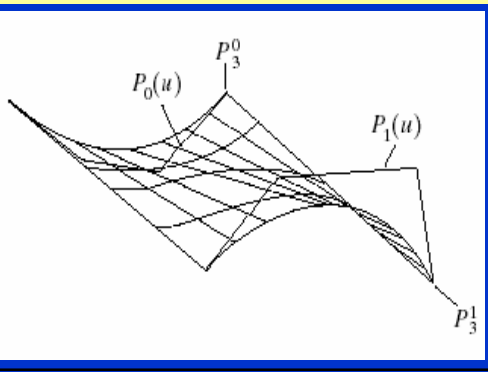
Modeling Curved Surfaces

- Various surfaces can be design by using B-splines
- B-spline ruled surfaces
- B-spline surfaces of revolution
- Bezier surface patches
- B-spline patches
- NURBS surfaces
- Basic idea: Surfaces defined and managed by a set of control points

Ruled Surfaces

- two end curves $P_0(u)$ and $P_1(u)$,
  - Surface points: Linear interpolation of the curves
  - $P(u, v) = (1 - v)\, P_0(u) + v\, P_1(u)$
- Idea: let $P_0(u)$ and $P_1(u)$ be Bezier (or B-spline) curves
  - managed by control points

- 8 control points
- $P_0(u)$ control points
  - $P_0^0, P_1^0, P_2^0, P_3^0,$
- $P_1(u)$ control points
  - $P_0^1, P_1^1, P_2^1, P_3^1.$



Bezier Ruled Surfaces

- $P(u, v) = (1 - v)\, P_0(u) + v\, P_1(u).$
- Recall Bezier curve

$$P(t) = \sum_{k=0}^{L} P_k B_k^L(t)$$

- Apply this to $P_0(u)$, $P_1(u)$, (with L = 3, cubic Bezier curves)

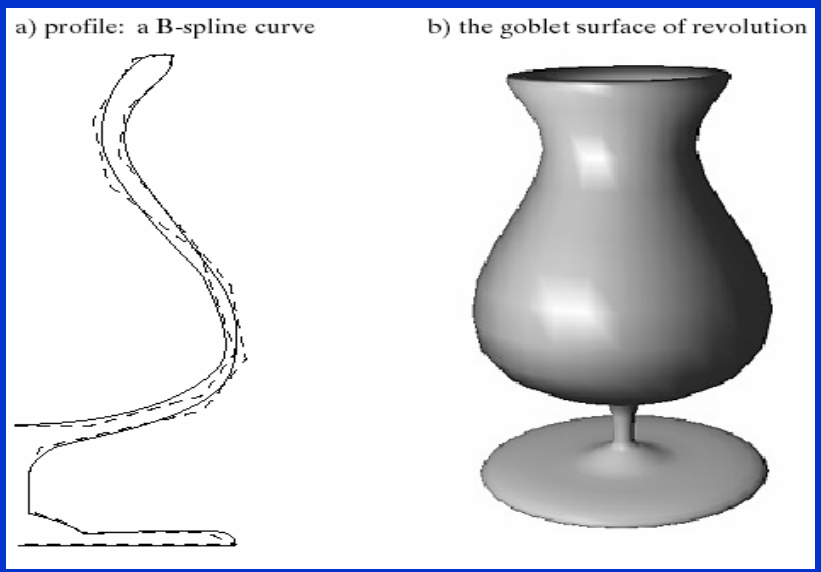
$$P(u,v) = \sum_{k=0}^{3} ((1-v)P_k^0 + vP_k^1)B_k^3(u)$$

- B-spline or NURBS curves could also be used

B-spline Surfaces of Revolution

- a profile $C(v) = (X(v), Z(v))$ is revolved about the z-axis.
- The resulting surface, P(u, v)
  - $\mathbf{P}(u, v) = (X(v)\cos(u), X(v)\sin(u), Z(v))$
- Select $L+1$ control points (X_k, Z_k)
- create the profile curve
  - $(X(v), Z(v)) = \Sigma\ (X_k, Z_k)b_{k,n}(v)$ sum over k = 0,..L
- $\mathbf{P}(u,v) = \Sigma(X_k\cos(u), X_k\sin(u), Z_k)b_{k,n}(v)$ sum over k = 0..L

Example B-spline Surfaces of Revolution



Bezier Surface Patches

- Start with a Bezier curve, managed by control points P_k $k = 0, 1, \ldots L$

$$P(t) = \sum_{k=0}^{L} P_k B_k^L(t)$$

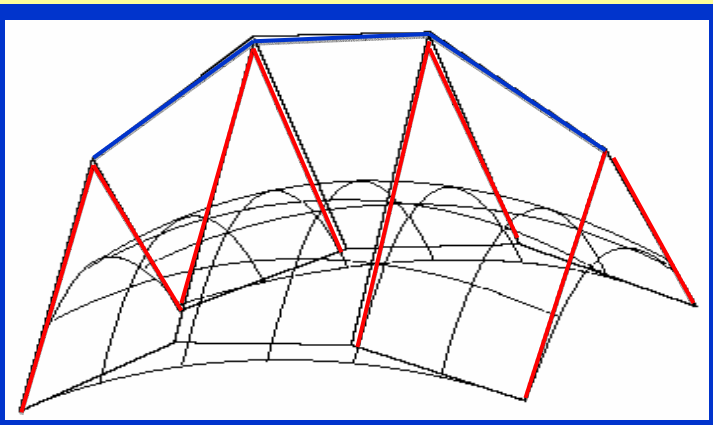
- Now move each $\mathbf{P}_k$ along another Bezier curve, managed by control points $\mathbf{P}_{i,k}$

$$P(u,v) = \sum_{k=0}^{3} P_k(v) B_k^3(u) = \sum_{k=0}^{3} \left(\sum_{i=0}^{2} P_{i,k} B_i^3(v) \right) B_k^3(u)$$

- fixed u ("u contours") are quadratic Bezier curves
- Fixed v ("v contours") are cubic Bezier curves

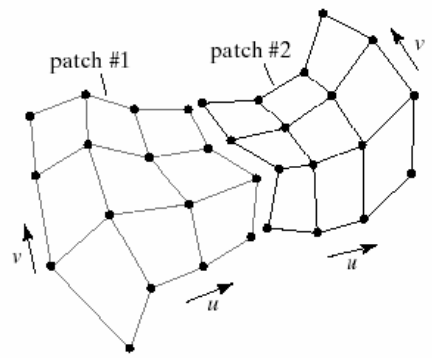
Example

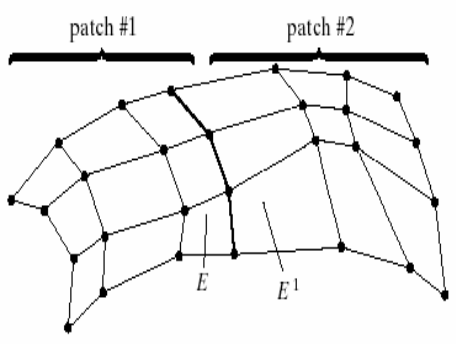
- u contours (fixed u) managed by 3 control points
- v contours (fixed v) managed by 4 control points
- Control Polyhedron



Joining Bezier Patches

- Edges must meet with same control points
- each pair of polyhedron edges that meet at the boundary, such as E and E^1, must be collinear.





B-spline, NURBS Patches

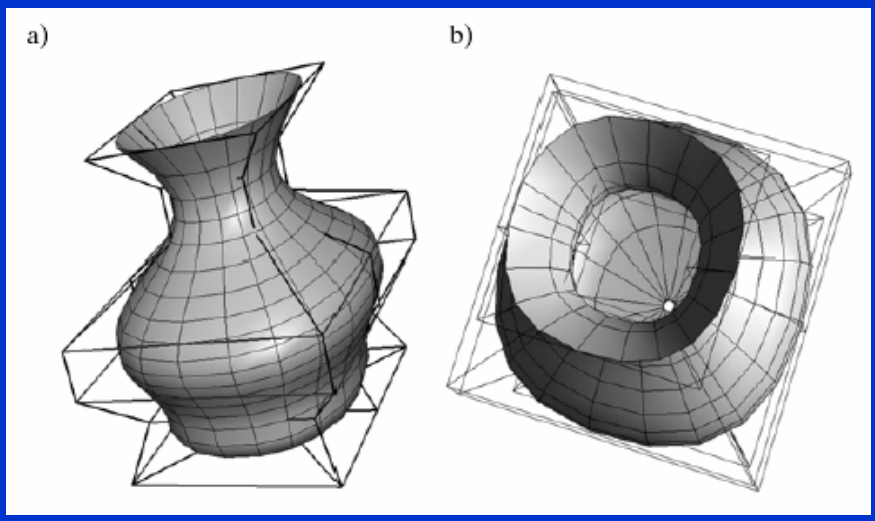
- B-spline patches: greater control

$$P(u,v) = \sum_{i=0}^{M} \sum_{k=0}^{L} P_{i,k} N_{i,m}(u) N_{k,n}(v)$$

- NURBS patches

$$P(u,v) = \frac{\sum_{i=0}^{M} \sum_{k=0}^{L} w_{i,k} P_{i,k} N_{i,m}(u) N_{k,n}(v)}{\sum_{i=0}^{M} \sum_{k=0}^{L} w_{i,k} N_{i,m}(u) N_{k,n}(v)}$$

B-spline Surfaces



Example NURBS Patches