Curves

D.A. Forsyth, with slides from John Hart

Central issues in modelling

- Construct families of curves, surfaces and volumes that
  - can represent common objects usefully;
  - are easy to interact with; interaction includes:
    - manual modelling;
    - fitting to measurements;
  - support geometric computations
    - intersection
    - collision

Main topics

- Simple curves
- Simple surfaces
- Continuity and splines
- Bezier surfaces and spline surfaces
- Volume models
- Meshes
- Animation

Parametric forms

- A parametric curve is
  - a mapping of one parameter into
    - 2D
    - 3D
  - Examples
    - circle as (cos t, sin t)
    - twisted cubic as (t, t*t, t*t*t)
    - circle as (1-t^2, 2 t, 0)/(1+t^2)
  - domain of the parametrization MATTERS
    - (cos t, sin t), 0<=t<= pi is a semicircle

Curves - basic ideas

- Important cases on the plane
  - Monge (or explicit)
    - given as a function, $y(x)$
    - Examples:
      - many lines, bits of circle, sines, etc
  - Implicit curve
    - $F(x, y)=0$
    - Examples:
      - all lines, circles, ellipses
      - any explicit curve; any parametric algebraic curve; lots of others
      - Important special case: F polynomial
  - Parametric curve
    - $(x(s), y(s))$ for s in some range
    - Examples
      - all lines, circles, ellipses
      - Important special cases: x, y polynomials, rational

Parametric forms

- A parametric surface is
  - a mapping of two parameters into 3D
  - Examples:
    - sphere as (cos s cos t, sin s cos t, sin t)
  - Again, domain matters
- Very common forms
  - Curve

$$\mathbf{x}(\mathrm{s})=\sum_i \mathbf{v}_i \phi_i(s)$$

  - Surface

$$\mathbf{x}(\mathrm{s},\ \mathrm{t})=\sum_{ij} \mathbf{v}_{ij} \phi_{ij}(s,t)$$

Functions phi are known as “blending functions”

Parametric vs Implicit

- Some computations are easier in one form
  - Implicit
    - ray tracing
  - Parametric
    - meshing
- Implicit surfaces bound volumes
  - “hold water”
  - but there might be extra bits
- Parametric surfaces/curves often admit implicit form
- Control
  - implicit: fundamentally global, rigid objects
  - parametric: can have local control

Interpolation

- Construct a parametric curve that passes through (interpolates) a set of points.
- Lagrange interpolate:
  - give parameter values associated with each point
  - use Lagrange polynomials (one at the relevant point, zero at all others) to construct curve
  - curve is:

$$\sum_{i \in \mathrm{points}} \mathbf{p}_i \phi_i^{(l)}(t)$$

Lagrange interpolate

- Construct a parametric curve that passes through (interpolates) a set of points.
- Lagrange interpolate:
  - give parameter values associated with each point
  - use Lagrange polynomials (one at the relevant point, zero at all others) to construct curve
  - degree is (#pts-1)
    - e.g. line through two points
    - quadratic through three.
  -

Lagrange polynomials

- Interpolate points at s=s_i, i=1..n
- Blending functions

$$\phi_i(s) = \begin{cases} 1 & s = s_i \\ 0 & s = s_k, k \neq i \end{cases}$$

- Can do this with a polynomial

$$\frac{\prod_{j=1..i-1,i..n}(s - s_j)}{\prod_{j=1..i-1,i..n}(s_j - s_i)}$$

Fig 2.16a. Interpolation by a polynomial of degree 4.

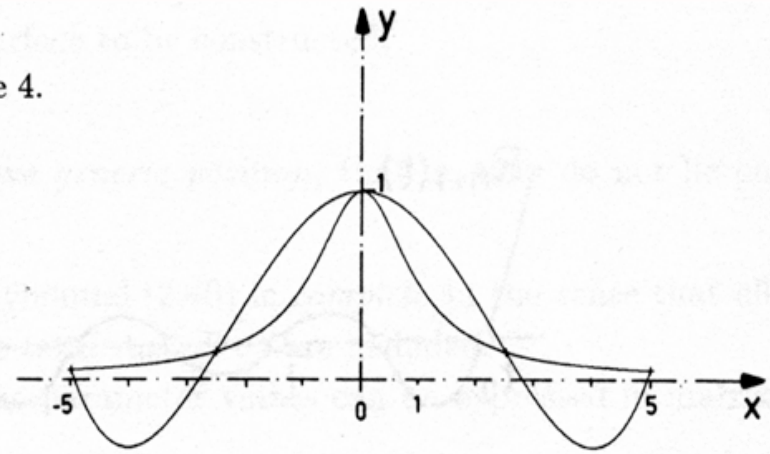
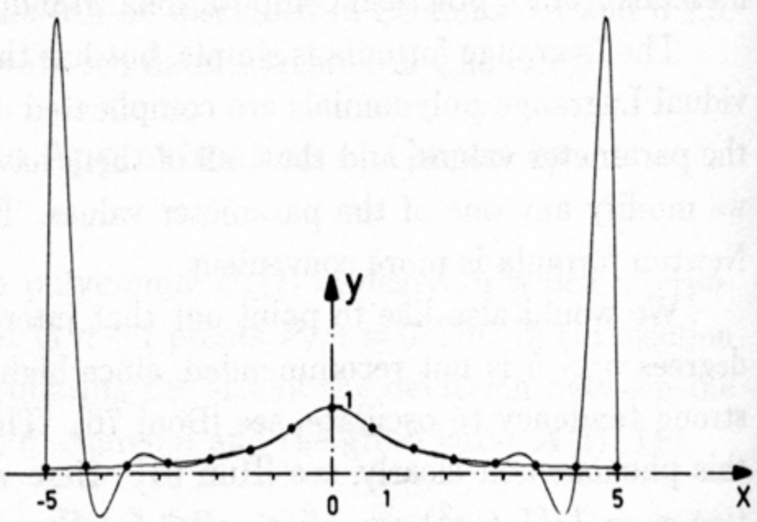


Fig 2.16c. Interpolation by a polynomial of degree 14.



Hermite interpolation

- Hermite interpolate
  - give parameter values and derivatives associated with each point
  - curve passes through given point and the given derivative at that parameter value
  - For two points (most important case) curve is:

$$\mathbf{p}_0\phi_0(t) + \mathbf{p}_1\phi_1(t) + \mathbf{v}_0\phi_2(t) + \mathbf{v}_1\phi_3(t)$$

- use Hermite polynomials to construct curve
  - one at some parameter value and zero at others or
  - derivative one at some parameter value, and zero at others

Hermite curves

- Natural matrix form:
  - solve linear system to get curve coefficients
- Easily "pasted" together

$$\mathbf{p}_0\phi_0(t) + \mathbf{p}_1\phi_1(t) + \mathbf{v}_0\phi_2(t) + \mathbf{v}_1\phi_3(t)$$

Blending functions are cubic polynomials, so we write as:

$$\begin{bmatrix} \phi_0(t) & \phi_1(t) & \phi_2(t) & \phi_3(t) \end{bmatrix} = \begin{bmatrix} 1 & t & t^2 & t^3 \end{bmatrix} \left\{ \begin{array}{cccc} a_0 & a_1 & a_2 & a_3 \\ b_0 & b_1 & b_2 & b_3 \\ c_0 & c_1 & c_2 & c_3 \\ d_0 & d_1 & d_2 & d_3 \end{array} \right\}$$

This allows us to write the curve as:

$$\begin{bmatrix} 1 & t & t^2 & t^3 \end{bmatrix} \left\{ \begin{array}{cccc} a_0 & a_1 & a_2 & a_3 \\ b_0 & b_1 & b_2 & b_3 \\ c_0 & c_1 & c_2 & c_3 \\ d_0 & d_1 & d_2 & d_3 \end{array} \right\} \left\{ \begin{array}{c} \mathbf{p}_0 \\ \mathbf{p}_1 \\ \mathbf{v}_0 \\ \mathbf{v}_1 \end{array} \right\}$$

Basis matrix

Geometry matrix

Hermite polynomials

$$\begin{bmatrix} \phi_0(t) & \phi_1(t) & \phi_2(t) & \phi_3(t) \end{bmatrix} = \begin{bmatrix} 1 & t & t^2 & t^3 \end{bmatrix} \begin{Bmatrix} a_0 & a_1 & a_2 & a_3 \\ b_0 & b_1 & b_2 & b_3 \\ c_0 & c_1 & c_2 & c_3 \\ d_0 & d_1 & d_2 & d_3 \end{Bmatrix}$$

$$\frac{d}{dt}\begin{bmatrix} \phi_0(t) & \phi_1(t) & \phi_2(t) & \phi_3(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 & 2t & 3t^2 \end{bmatrix} \begin{Bmatrix} a_0 & a_1 & a_2 & a_3 \\ b_0 & b_1 & b_2 & b_3 \\ c_0 & c_1 & c_2 & c_3 \\ d_0 & d_1 & d_2 & d_3 \end{Bmatrix}$$

Constraints

$$\begin{bmatrix} \phi_0(0) & \phi_1(0) & \phi_2(0) & \phi_3(0) \\ \phi_0(1) & \phi_1(1) & \phi_2(1) & \phi_3(1) \\ \frac{d\phi_0}{dt}(0) & \frac{d\phi_1}{dt}(0) & \frac{d\phi_2}{dt}(0) & \frac{d\phi_3}{dt}(0) \\ \frac{d\phi_0}{dt}(1) & \frac{d\phi_1}{dt}(1) & \frac{d\phi_2}{dt}(1) & \frac{d\phi_3}{dt}(1) \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

These constraints give:

Interpolate each endpoint
Have correct derivatives at each endpoint

We can write individual constraints like:

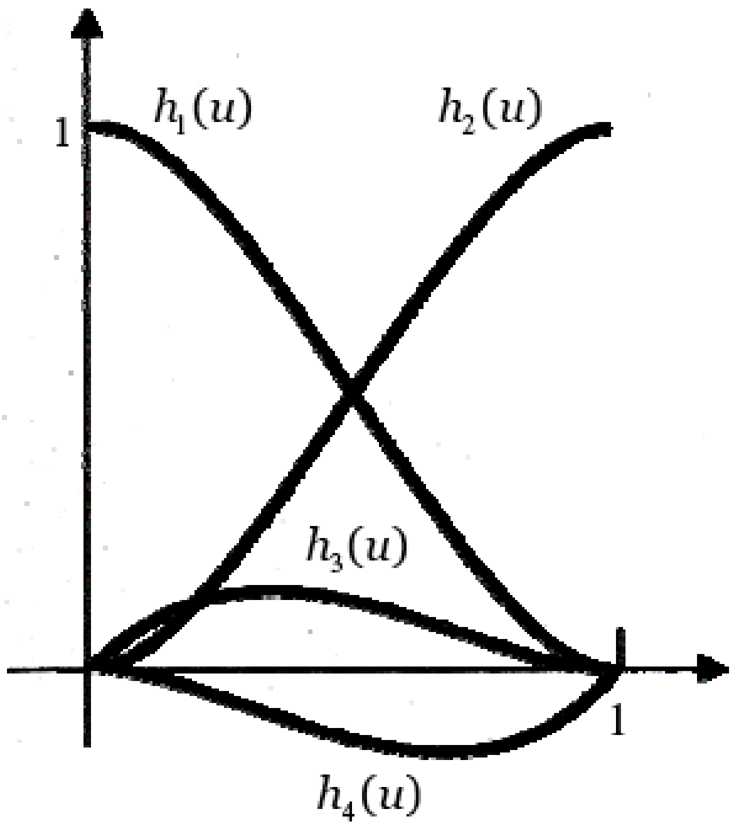
$$\left[\begin{array}{cccc} \phi_0(0) & \phi_1(0) & \phi_2(0) & \phi_3(0) \end{array}\right] = \left[\begin{array}{cccc} 1 & 0 & 0^2 & 0^3 \end{array}\right] \left\{\begin{array}{cccc} a_0 & a_1 & a_2 & a_3 \\ b_0 & b_1 & b_2 & b_3 \\ c_0 & c_1 & c_2 & c_3 \\ d_0 & d_1 & d_2 & d_3 \end{array}\right\}$$

To get:

$$\left[\begin{array}{cccc} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 2 & 3 \end{array}\right] \left\{\begin{array}{cccc} a_0 & a_1 & a_2 & a_3 \\ b_0 & b_1 & b_2 & b_3 \\ c_0 & c_1 & c_2 & c_3 \\ d_0 & d_1 & d_2 & d_3 \end{array}\right\} = \left[\begin{array}{cccc} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{array}\right]$$

Hermite blending functions

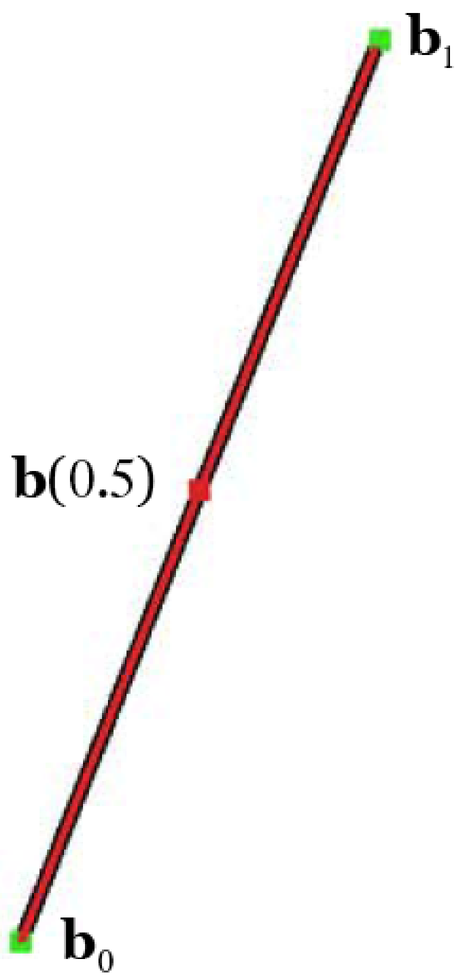
Hermite Blending Polynomials



$$h_1(u) = 2u^3 - 3u^2 + 1$$
$$h_2(u) = -2u^3 + 3u^2$$
$$h_3(u) = u^3 - 2u^2 + u$$
$$h_4(u) = u^3 - u^2$$

Bezier curves

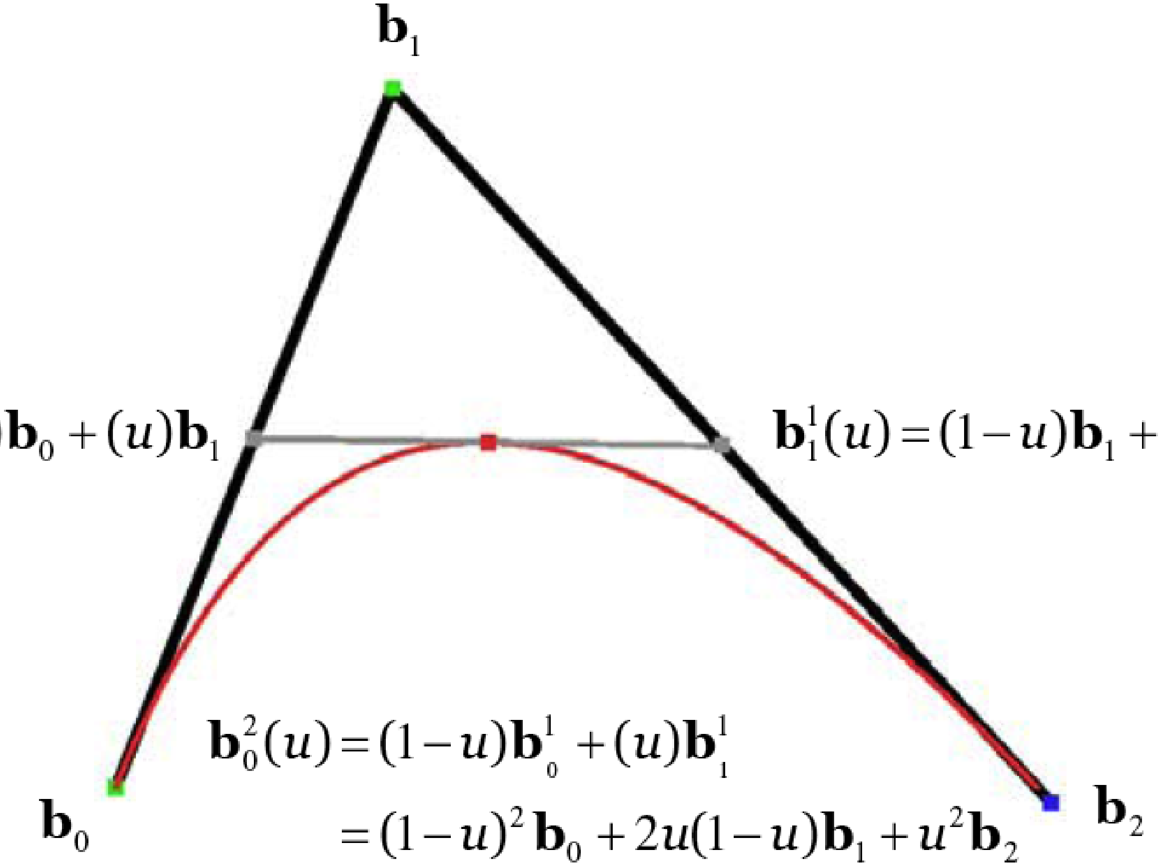
Linear Interpolation



$$\mathbf{b}(u) = (1-u)\mathbf{b}_0 + (u)\mathbf{b}_1 \qquad \text{where } 0 \le u \le 1$$

Bezier curves

“Doubled” Linear Interpolation



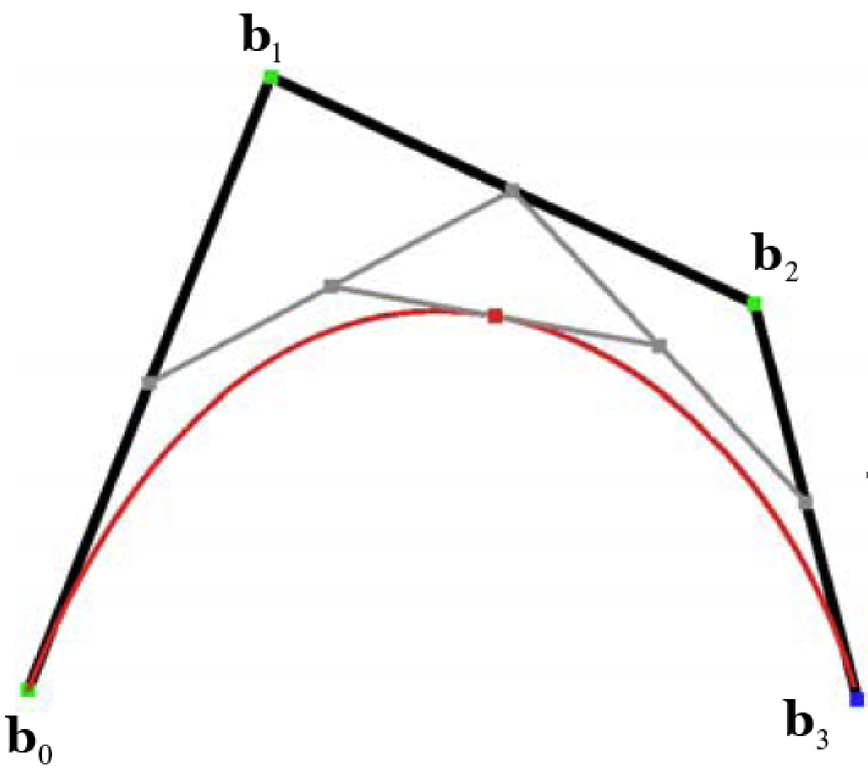
$$\mathbf{b}_0^1(u) = (1-u)\mathbf{b}_0 + (u)\mathbf{b}_1$$

$$\mathbf{b}_1^1(u) = (1-u)\mathbf{b}_1 + (u)\mathbf{b}_2$$

$$\begin{aligned}\mathbf{b}_0^2(u) &= (1-u)\mathbf{b}_0^1 + (u)\mathbf{b}_1^1 \\ &= (1-u)^2\,\mathbf{b}_0 + 2u(1-u)\mathbf{b}_1 + u^2\mathbf{b}_2\end{aligned}$$

Bezier curves

"Tripled" Linear Interpolation



Get a cubic polynomial curve

$$\begin{aligned}\mathbf{b}_0^3(u) = &(1-u)^3\,\mathbf{b}_0\\ &+3(1-u)^2(u)\mathbf{b}_1\\ &+3(1-u)(u)^2\,\mathbf{b}_2\\ &+(u)^3\,\mathbf{b}_3\end{aligned}$$

This is a cubic Bézier curve

Bezier curves as a tableau

“Tripled” Linear Interpolation

Repeated averaging in tableau form:

$$\begin{array}{cccc}
\overbrace{\mathbf{b}_0}^{\text{Input points}} & & & \\
\mathbf{b}_1 & \mathbf{b}_0^1 & & \\
\mathbf{b}_2 & \mathbf{b}_1^1 & \mathbf{b}_0^2 & \\
\mathbf{b}_3 & \mathbf{b}_2^1 & \mathbf{b}_1^2 & \underbrace{\mathbf{b}_0^3}_{\text{Point on curve}}
\end{array}$$

This clearly suggests a recursive procedure …

de Casteljau (formal version)

General Bézier Curves

Given $n+1$ control points

$$\mathbf{b}_0, \mathbf{b}_1, \ldots, \mathbf{b}_n \in R^3$$

We can define a Bézier curve

$$\mathbf{b}(u) = \mathbf{b}^n(u) = \mathbf{b}_0^n(u)$$

via the recursive construction

$$\mathbf{b}_i^r(u) = (1-u)\mathbf{b}_i^{r-1}(u) + (u)\mathbf{b}_{i+1}^{r-1}(u)$$
$$\mathbf{b}_i^0(u) = \mathbf{b}_i$$

This is the de Casteljau Algorithm

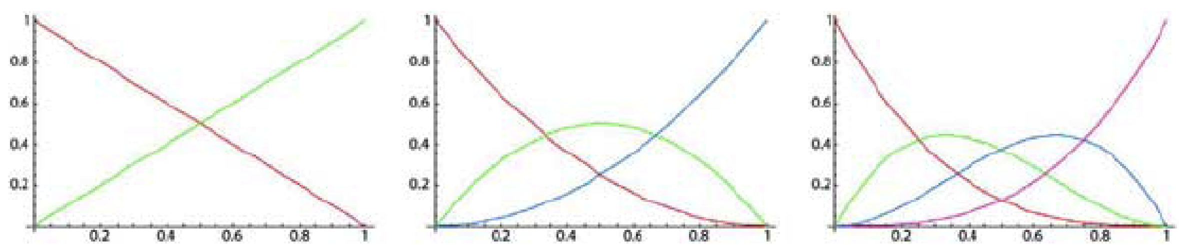
Bezier curve blending functions

Common Bernstein Polynomials

$$B_0^1 = 1 - u$$
$$B_1^1 = u$$

$$B_0^2 = (1-u)^2$$
$$B_1^2 = 2(1-u)(u)$$
$$B_2^2 = u^2$$

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



Curve has the form:

Bezier blending functions

- Bezier-Bernstein polynomials $B_i^n(u) = C(n,i)(1-u)^i u^{n-1}$
    - here C(n, i) is the number of combinations of n items, taken i at a time
    -

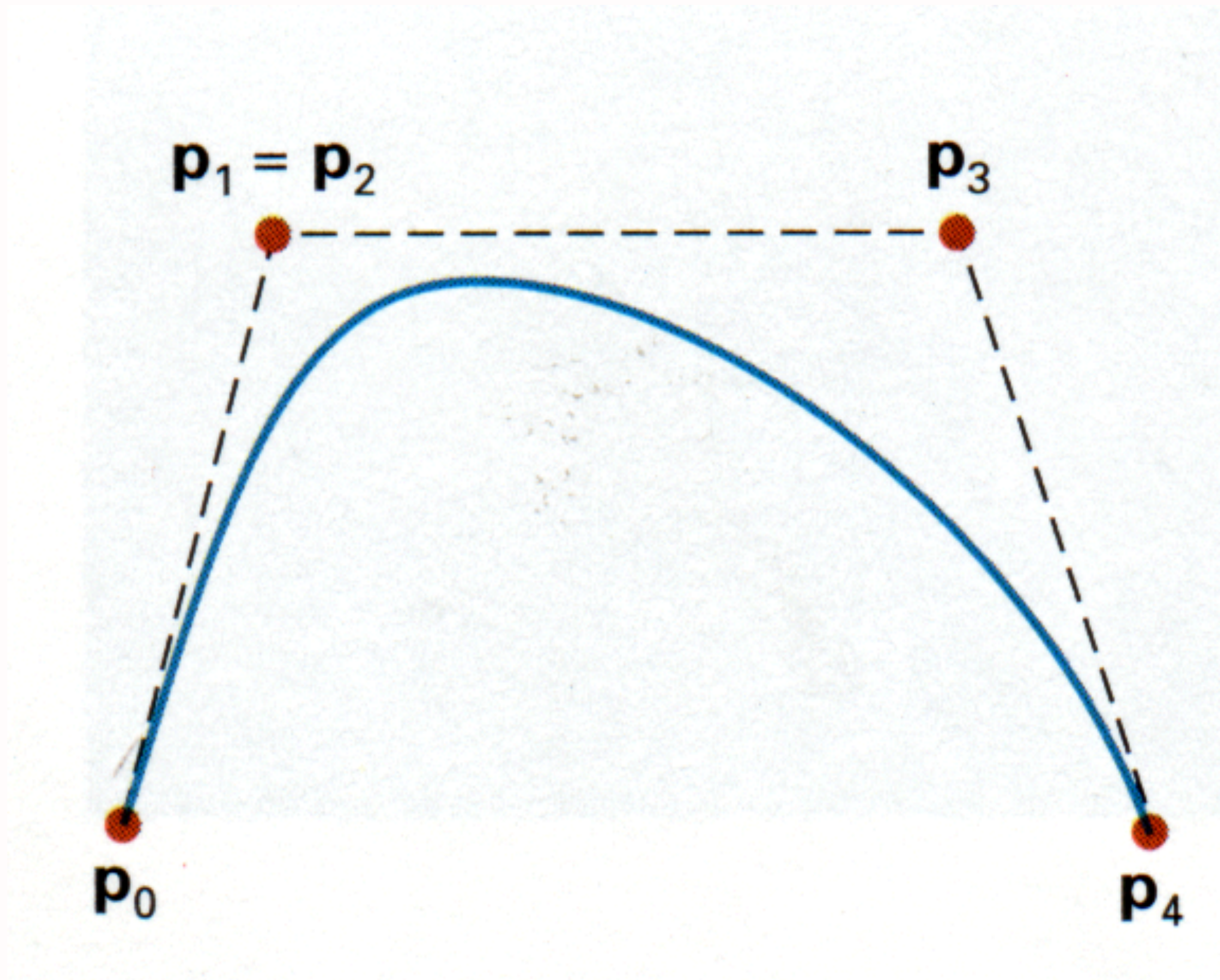
$$C(n,i) = \frac{n!}{(n-i)!i!}$$

Bezier curve properties

- Pass through first, last points
- Tangent to initial, final segments of control polygon
- Lie within convex hull of control polygon
- Subdivide

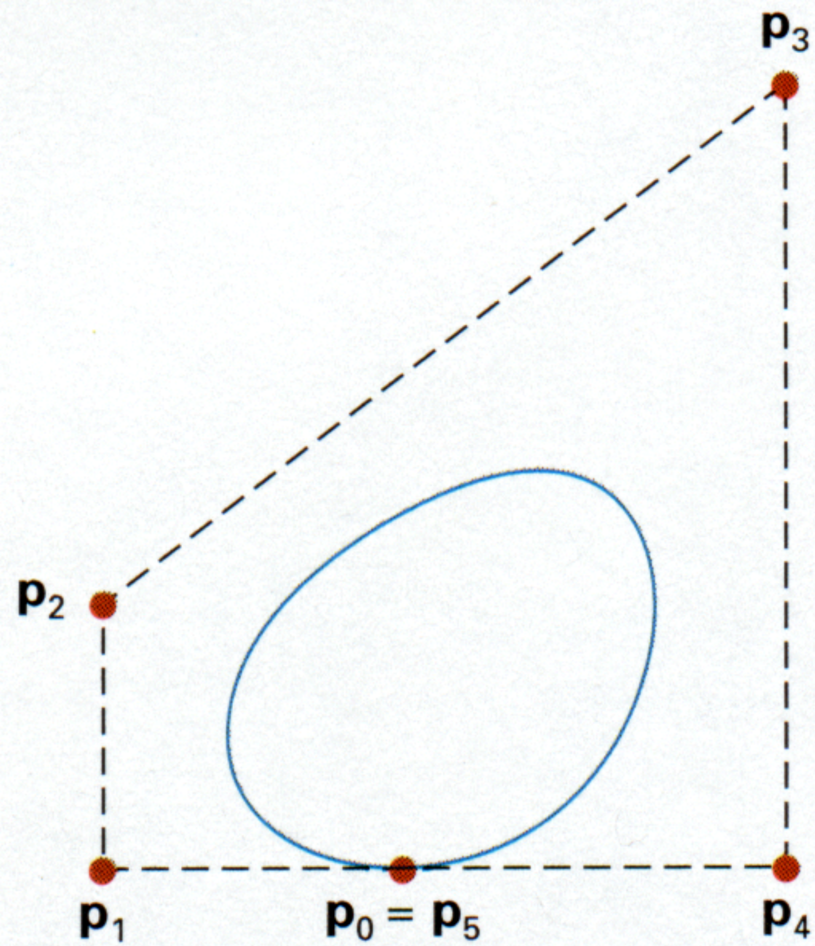
Bezier curve tricks - I

- Pull a curve towards a point by placing two control points on top of one another



Bezier curve tricks - II

- Close a curve by making endpoints the same point
  - clean join by making segments line up



Subdivision for Bezier curves

- Use De Casteljau (repeate linear interpolation) to identify points.
- Points as marked in figure give two control polygons for two Bezier curves, which lie on top of the original.
- Repeated subdivision lead to a polygon that lies very close to the curve
- Limit of subdivision process is a curve

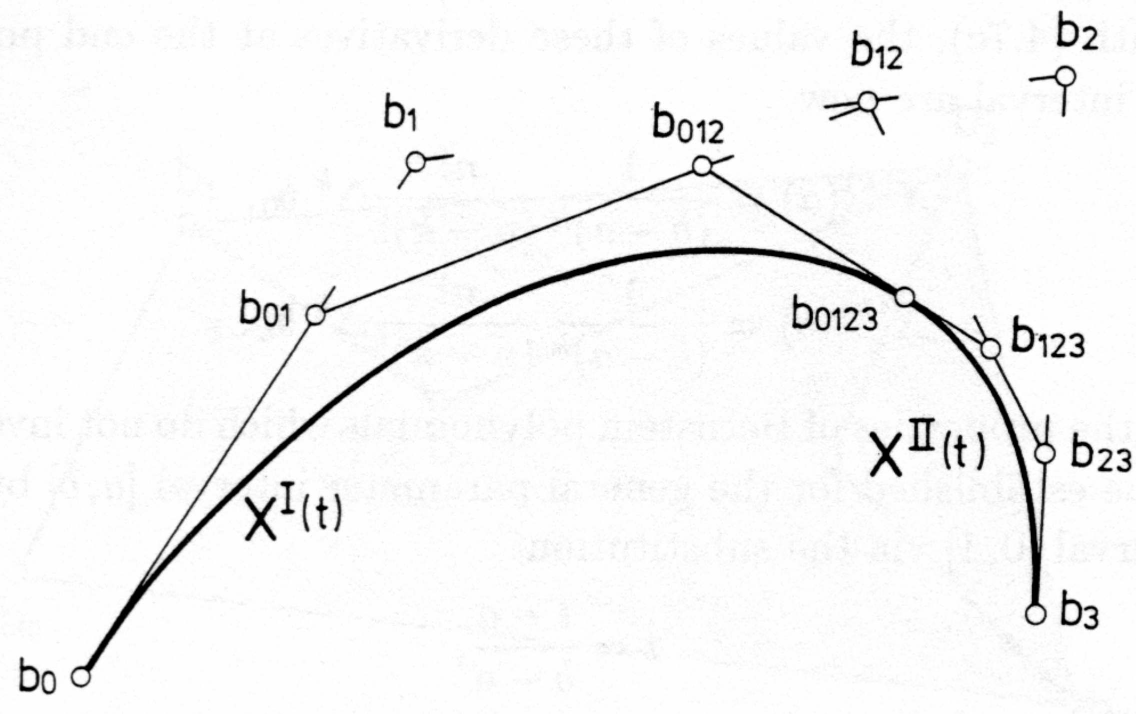


Fig. 4.5. Decomposition of a Bézier curve into two C^3 continuous curve segments (cf. Fig. 4.4).

Degree raising for Bezier curves

- Idea: add a control point without changing curve
- Procedure:
  - curve with k control points is p(t), k+1 is q(t)
  - multiply p(t) by (1-t+t), line up monomials
  - gives relation

$$\mathbf{p}(\mathrm{t})=\sum_{i=0}^{k} \mathbf{p}_i \begin{pmatrix} k \\ i \end{pmatrix} (1-t)^{k-i} t^i \qquad (1\text{-t}+\text{t})\ \mathbf{p}(\mathrm{t})= \mathbf{q}(\mathrm{t})$$

$$\mathbf{q}(\mathrm{t})=\sum_{i=0}^{k+1} \mathbf{q}_i \begin{pmatrix} k \\ i \end{pmatrix} (1-t)^{k+1-i} t^i$$

Degree raising for Bezier curves

$$\binom{k+1}{i}\mathbf{q}_i = \binom{k}{i}\mathbf{p}_i + \binom{k}{i-1}\mathbf{p}_{i-1}$$

Equivalences

- 4 control point Bezier curve is a cubic curve
- so is an Hermite curve
- so we can transform from one to the other
- Easy way:
  - notice that 4-point Bezier curve
    - interpolates endpoints
    - has tangents 3(b_1-b_0), 3(b_3-b_2)
    - this gives Hermite->Bezier, Bezier->Hermite
- Hard way:
  - do the linear algebra

4-point Bezier curve:

$$\begin{bmatrix} 1 & t & t^2 & t^3 \end{bmatrix} \left\{ \begin{matrix} 1 & 0 & 0 & 0 \\ -3 & 3 & 0 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{matrix} \right\} \begin{bmatrix} \mathbf{p}_0 \\ \mathbf{p}_1 \\ \mathbf{p}_2 \\ \mathbf{p}_3 \end{bmatrix}$$

$$\begin{bmatrix} 1 & t & t^2 & t^3 \end{bmatrix} \mathcal{B}_b \mathcal{G}_b$$

Hermite curve:

$$\begin{bmatrix} 1 & t & t^2 & t^3 \end{bmatrix} \left\{ \begin{matrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -3 & 3 & -2 & -1 \\ 2 & -2 & 1 & 1 \end{matrix} \right\} \begin{bmatrix} \mathbf{p}_0 \\ \mathbf{p}_1 \\ \mathbf{v}_0 \\ \mathbf{v}_1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & t & t^2 & t^3 \end{bmatrix} \mathcal{B}_h \mathcal{G}_h$$

Converting

- Say we know G_b
  - what G_h will give the same curve?

$$\mathcal{B}_h \mathcal{G}_h = \mathcal{B}_b \mathcal{G}_b$$

$$\mathcal{G}_h = \mathcal{B}_h^{-1} \mathcal{B}_b \mathcal{G}_b$$

  - known G_h works similarly

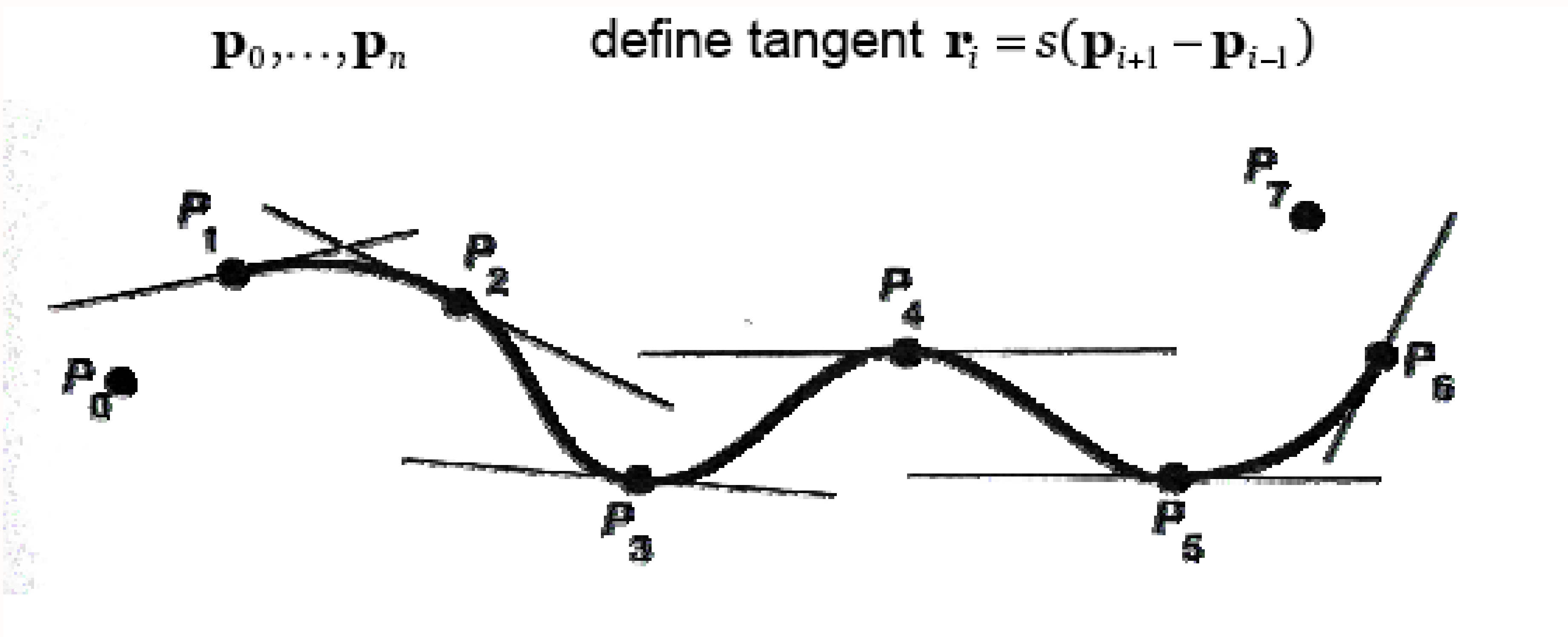
Continuity

- Geometric continuity
  - G^0 - end points join up
  - G^1 - end points join up, tangents are parallel
- Continuity
  - function of parametrization as well as geometry

Achieving geometric continuity

- Bezier curves
  - endpoints on top of each other
  - end tangents parallel
- Hermite curves
  - endpoints on top of each other
  - end tangents parallel
  - Catmull-Rom construction if we don’t have tangents

Catmull-Rom construction (partial)

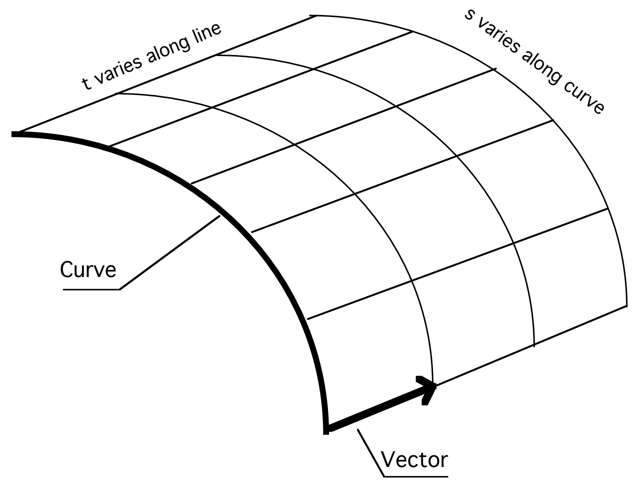


Simple surface constructions

- Surfaces can be:
  - explicit
  - implicit
  - parametric

Extruded surfaces

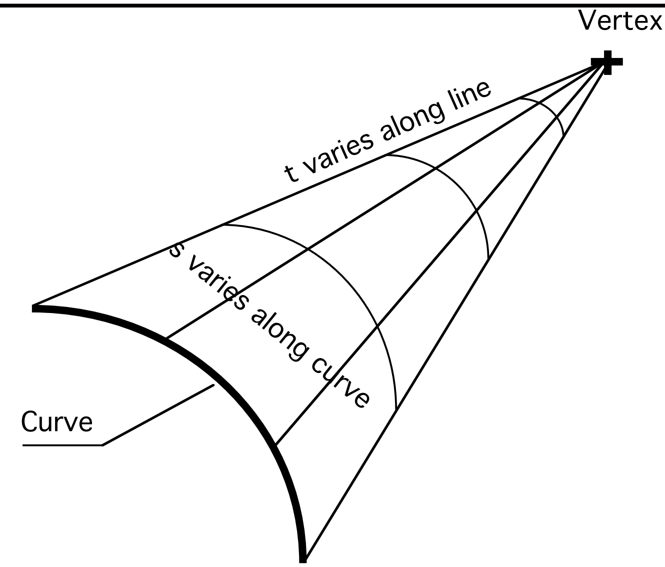
- Geometrical model - Pasta machine
- Take curve and "extrude" surface along vector
- Many human artifacts have this form - rolled steel, etc.



$$(x(s,t), y(s,t), z(s,t)) = (x_c(s), y_c(s), z_c(s)) + t(v_0, v_1, v_2)$$

Cones

- From every point on a curve, construct a line segment through a single fixed point in space - the vertex
- Curve can be space or plane curve, but shouldn't pass through the vertex



$$(x(s,t), y(s,t), z(s,t)) = (1-t)(x_c(s), y_c(s), z_c(s)) + t(v_0, v_1, v_2)$$

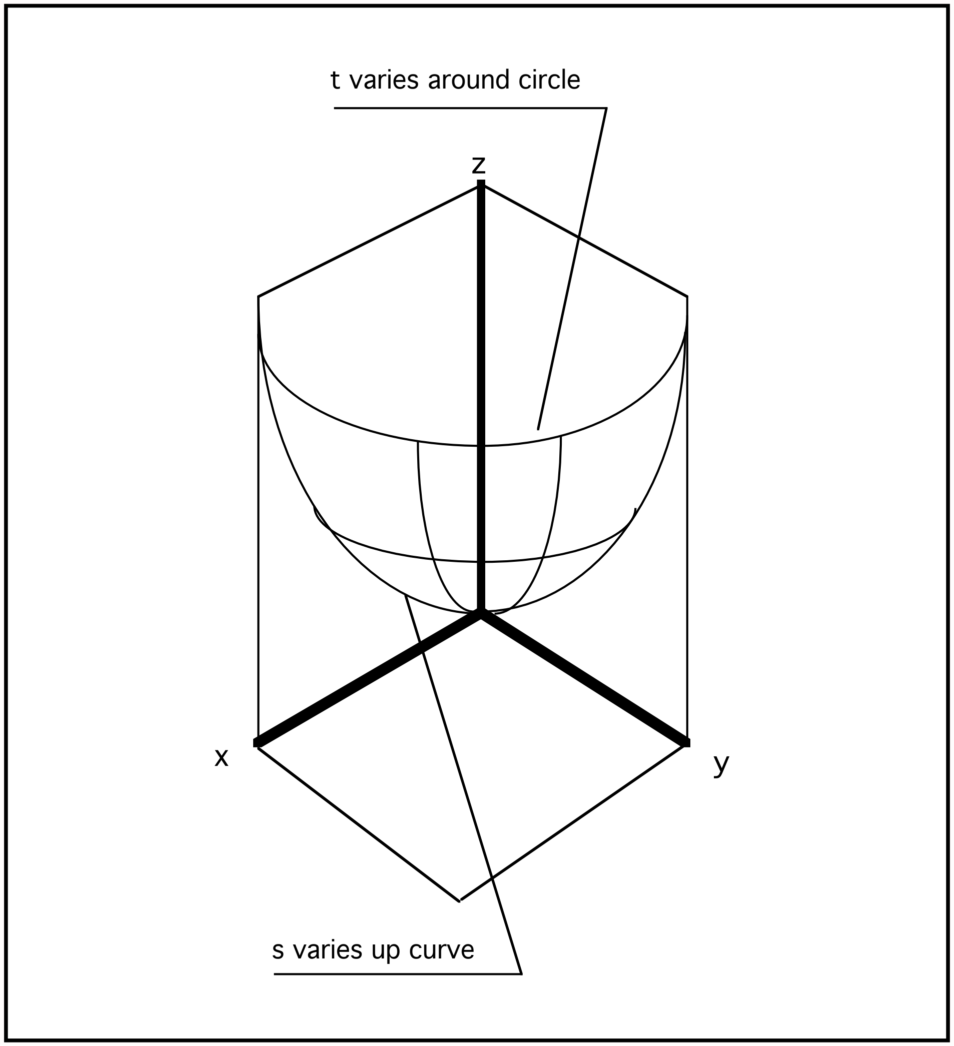
Surfaces of revolution

- Plane curve + axis
- "spin" plane curve around axis to get surface
- Choice of plane is arbitrary, choice of axis affects surface
- In this case, curve is on x-z plane, axis is z axis.

$$\boxed{\begin{aligned}&(x(s,t), y(s,t), z(s,t)) = \\ &\quad (x_c(s)\cos(t), x_c(s)\sin(t), z_c(s))\end{aligned}}$$

SOR-2

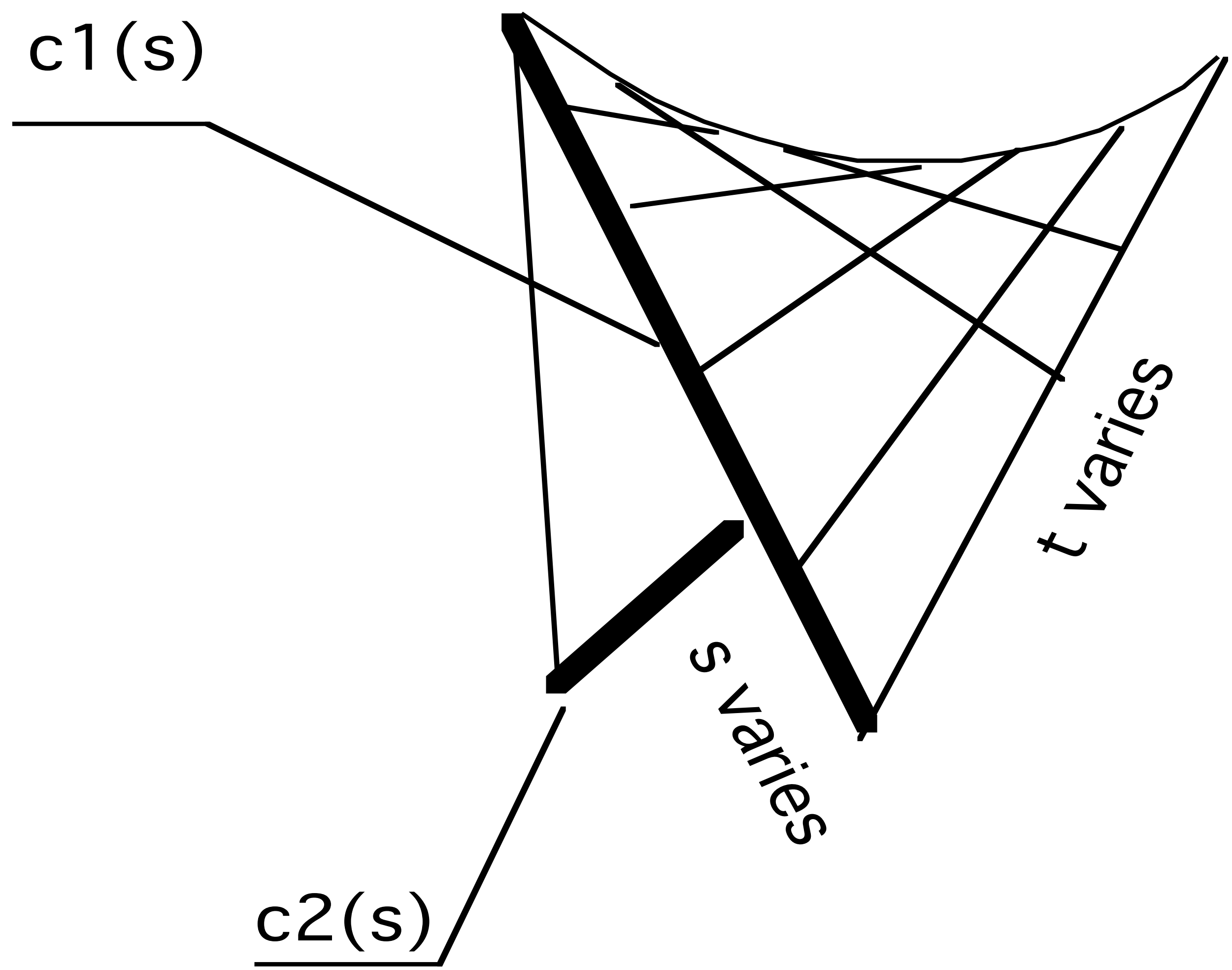
- Many artifacts are SOR's, as they're easy to make on a lathe.
- Controlling is quite easy - concentrate on the cross section.
- Axis crossing cross-section leads to ugly geometry.



Ruled surfaces

- Popular, because it's easy to build a curved surface out of straight segments - eg pavilions, etc.
- Take two space curves, and join corresponding points - same s - with line segment.
- Even if space curves are lines, the surface is usually curved.

$$
\begin{aligned}
(x(s,t), y(s,t), z(s,t)) = \\
(1-t)(x_1(s), y_1(s), z_1(s)) + \\
t(x_2(s), y_2(s), z_2(s))
\end{aligned}
$$

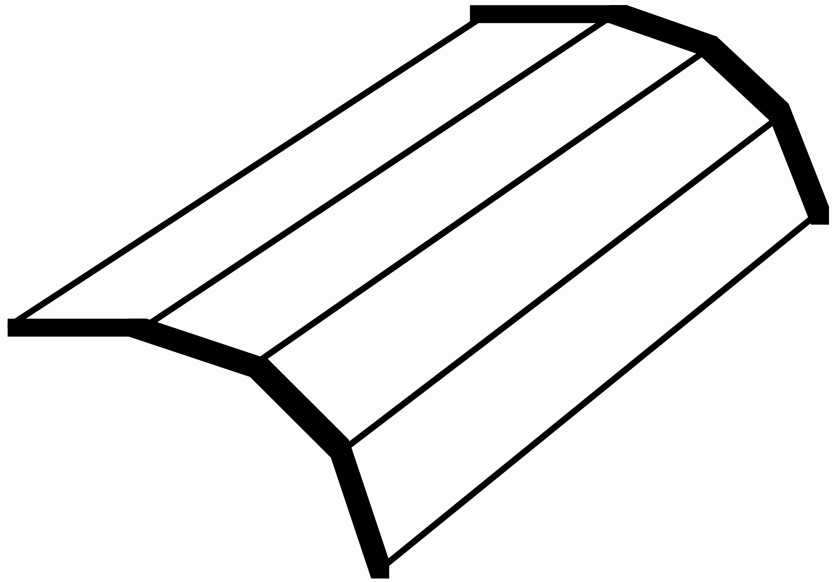

c1(s)
t varies
s varies
c2(s)

Normals

- Recall: normal is cross product of tangent in t direction and s direction.
- Cylinder: normal is cross-product of curve tangent and direction vector
- SOR: take curve normal and spin round axis

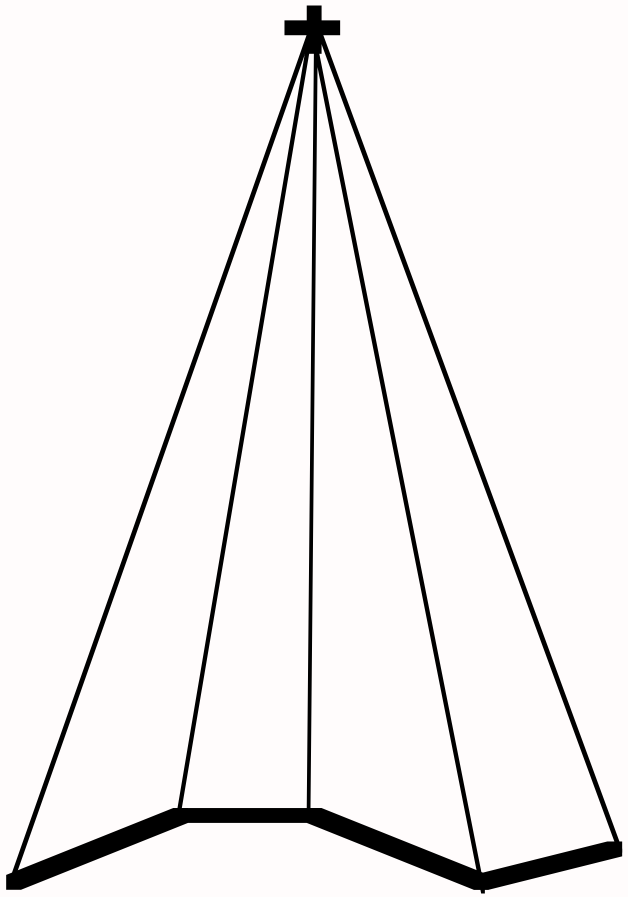
Rendering

- Cylinders: small steps along curve, straight segments along t generate polygons; exact normal is known.



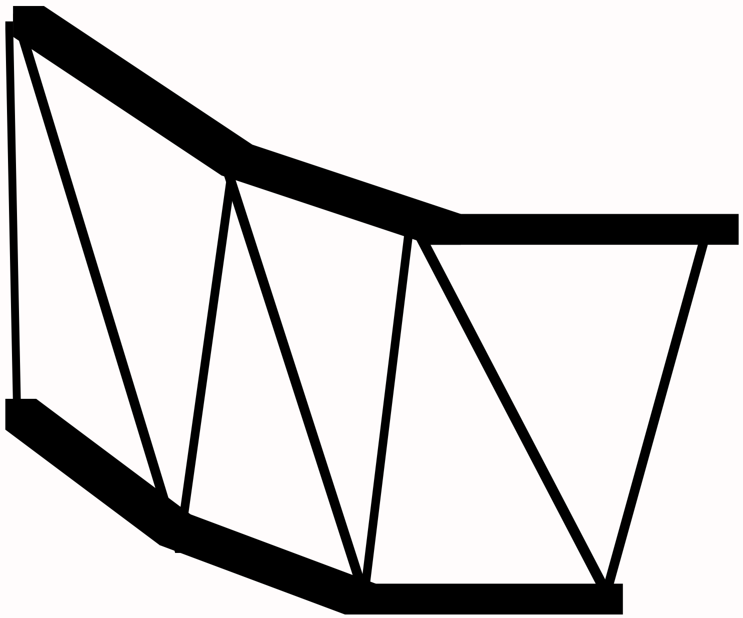
Rendering

- Cone: small steps in s generate straight edges, join with vertex to get triangles, normals known exactly except at vertex.



Rendering

- SOR: small steps in s generate strips, small steps in t along the strip generate edges; join up to form triangles. Normals known exactly.



Rendering

- Ruled surface: steps in s generate polygons, join opposite sides to make triangles - otherwise “non planar polygons” result. Normals known exactly.