Curves and Surfaces

CS 432 Interactive Computer Graphics

Prof. David E. Breen

Department of Computer Science

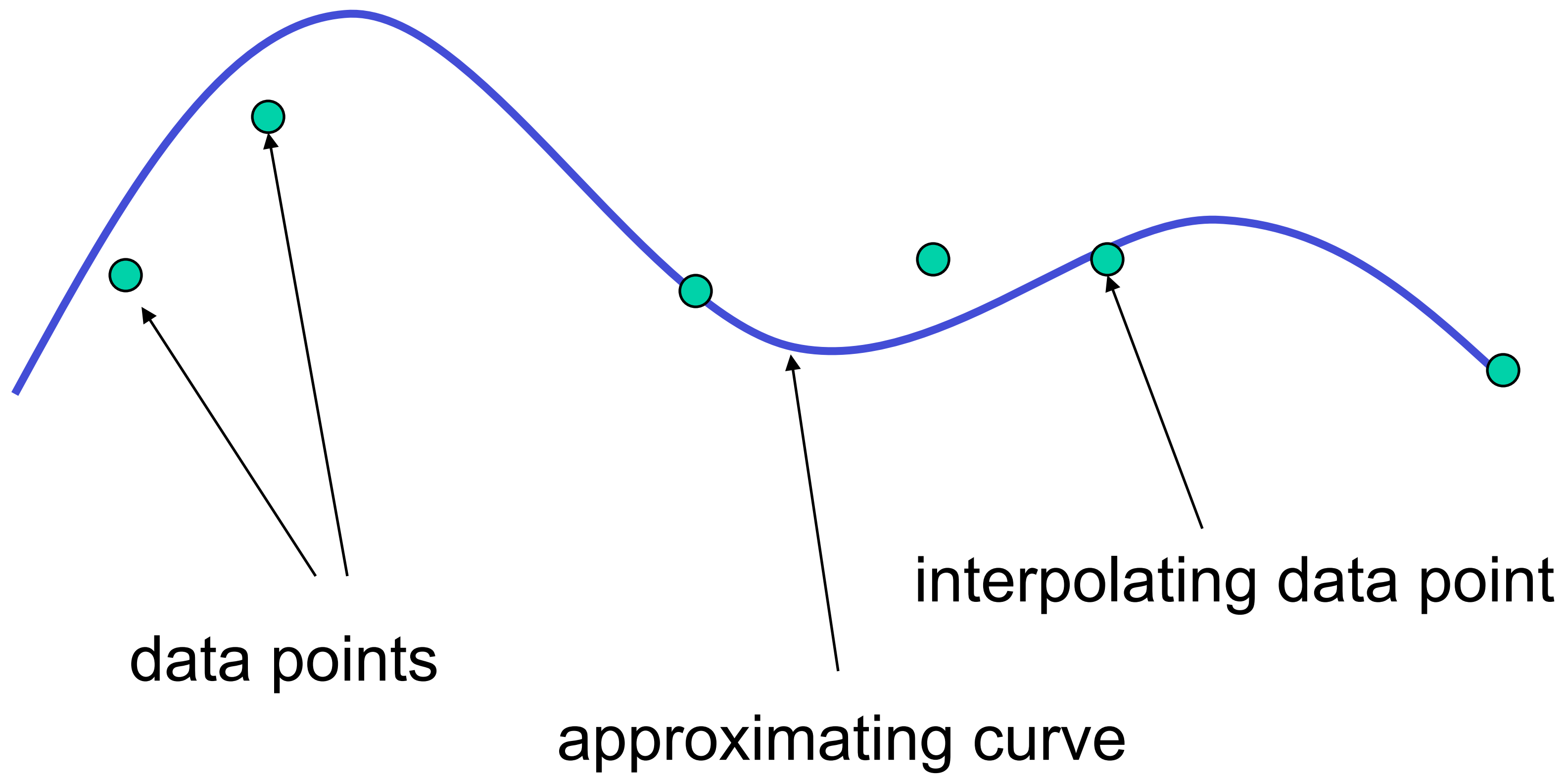
Objectives

- Introduce types of curves and surfaces
  - Explicit
  - Implicit
  - Parametric
  - Strengths and weaknesses
- Discuss Modeling and Approximations
  - Conditions
  - Stability

Escaping Flatland

- Until now we have worked with flat entities such as lines and flat polygons
  - Fit well with graphics hardware
  - Mathematically simple
- But the world is not composed of flat entities
  - Need curves and curved surfaces
  - May only have need at the application level
  - Implementation can render them approximately with flat primitives

Modeling with Curves



What Makes a Good Representation?

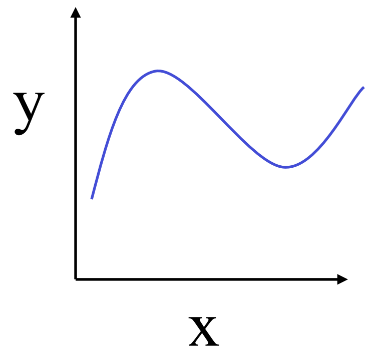
- There are many ways to represent curves and surfaces
- Want a representation that is
  - Stable
  - Smooth
  - Easy to evaluate
  - Must we interpolate or can we just come close to data?
  - Do we need derivatives?

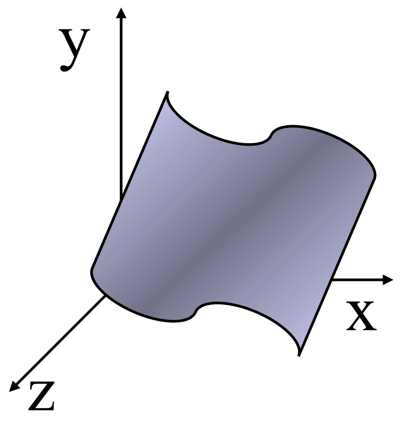
Explicit Representation

- Most familiar form of curve in 2D

  $y=f(x)$

- Cannot represent all curves
  - Vertical lines
  - Circles
- Extension to 3D
  - $y=f(x)$, $z=g(x)$
  - The form $z = f(x,y)$ defines a surface





Implicit Representation

- Two dimensional curve(s)

  $$g(x,y)=0$$

- Much more robust
  - All lines $ax+by+c=0$
  - Circles $x^2+y^2-r^2=0$
- Three dimensions $g(x,y,z)=0$ defines a surface
  - Intersect two surface to get a curve
- In general, we cannot exactly solve for points that satisfy the equation

Algebraic Surface

$$\sum_i \sum_j \sum_k x^i y^j z^k = 0$$

•Quadric surface $2 \geq i+j+k$

•At most 10 terms

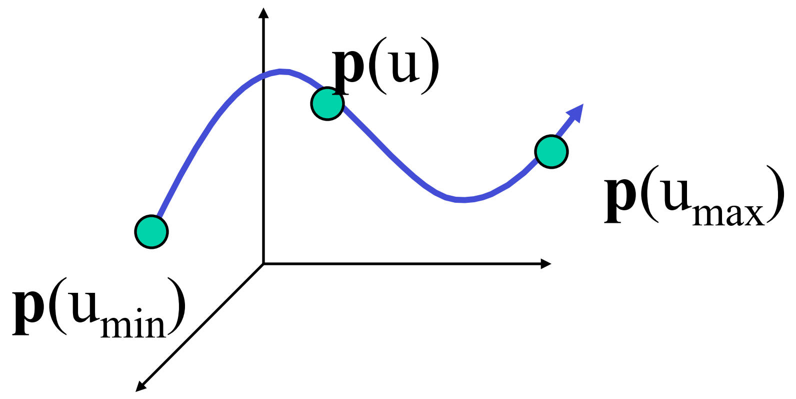
•Can solve intersection with a ray by reducing problem to solving quadratic equation

Parametric Curves

- Separate equation for each spatial variable

$$x=x(u)$$
$$y=y(u) \qquad \mathbf{p}(u)=[x(u), y(u), z(u)]^T$$
$$z=z(u)$$

- For $u_{max} \geq u \geq u_{min}$ we trace out a curve in two or three dimensions



Selecting Functions

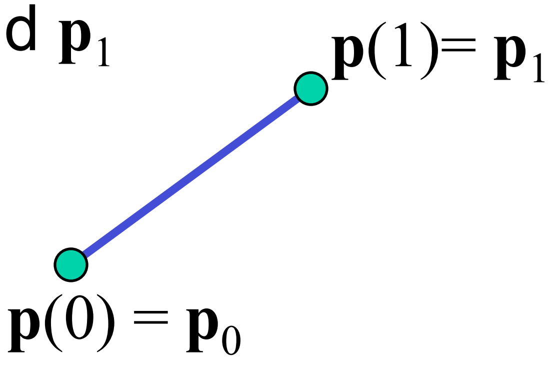
- Usually we can select "good" functions
  - not unique for a given spatial curve
  - Approximate or interpolate known data
  - Want functions which are easy to evaluate
  - Want functions which are easy to differentiate
    - Computation of normals
    - Connecting pieces (segments)
  - Want functions which are smooth

Parametric Lines

We can normalize u to be over the interval (0,1)

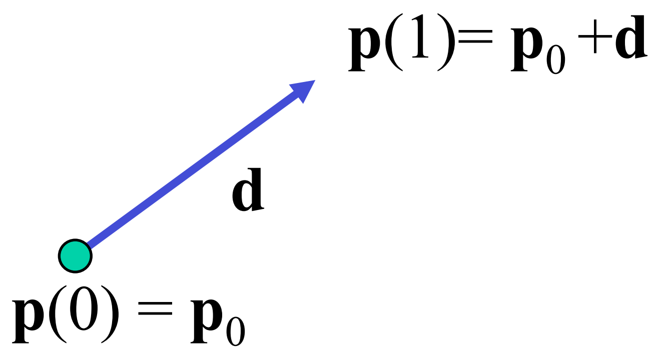
Line connecting two points $\mathbf{p}_0$ and $\mathbf{p}_1$

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



Ray from $\mathbf{p}_0$ in the direction $\mathbf{d}$

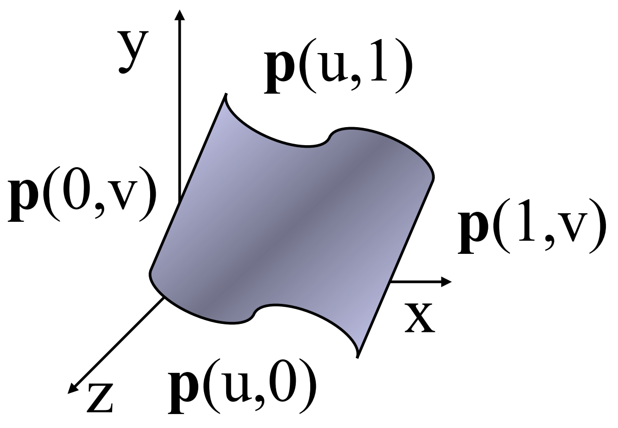
$$\mathbf{p}(u)=\mathbf{p}_0+u\mathbf{d}$$


$\mathbf{p}(1)= \mathbf{p}_0 +\mathbf{d}$
$\mathbf{d}$
$\mathbf{p}(0) = \mathbf{p}_0$

Parametric Surfaces

- Surfaces require 2 parameters

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



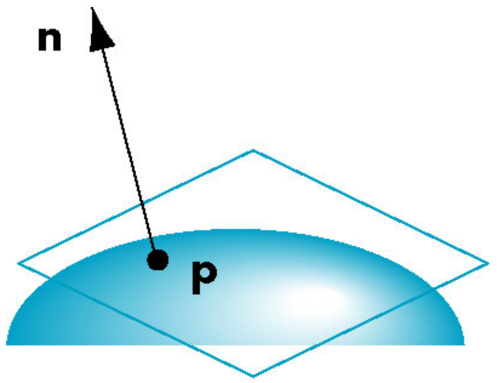
- Want same properties as curves:
  - Smoothness
  - Differentiability
  - Ease of evaluation

Normals

We can differentiate with respect to u and v to obtain the normal at any point **p**

$$\frac{\partial \mathbf{p}(u,v)}{\partial u} = \begin{bmatrix} \partial x(u,v)/\partial u \\ \partial y(u,v)/\partial u \\ \partial z(u,v)/\partial u \end{bmatrix} \qquad \frac{\partial \mathbf{p}(u,v)}{\partial v} = \begin{bmatrix} \partial x(u,v)/\partial v \\ \partial y(u,v)/\partial v \\ \partial z(u,v)/\partial v \end{bmatrix}$$

$$\mathbf{n} = \frac{\partial \mathbf{p}(u,v)}{\partial u} \times \frac{\partial \mathbf{p}(u,v)}{\partial v}$$



Parametric Planes

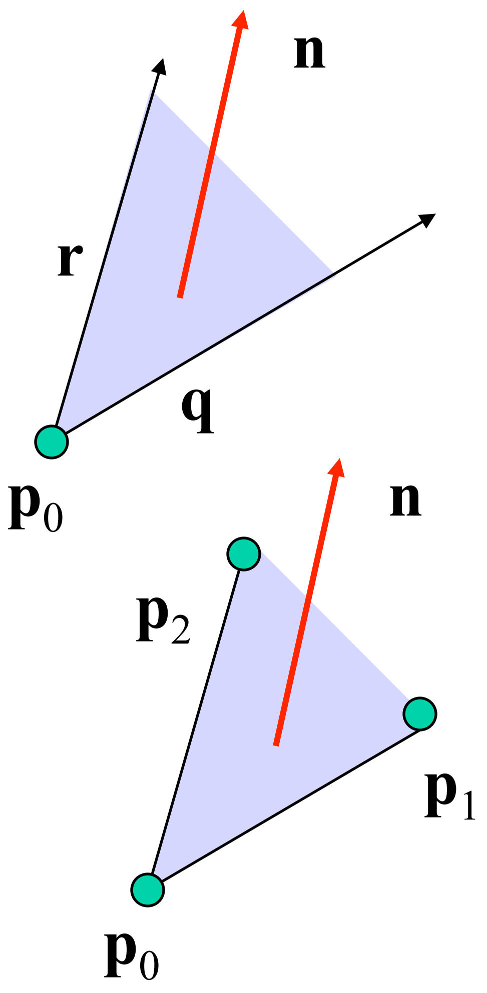
point-vector form

$$\mathbf{p}(u,v)=\mathbf{p}_0+u\mathbf{q}+v\mathbf{r}$$

$$\mathbf{n} = \mathbf{q} \times \mathbf{r}$$

three-point form

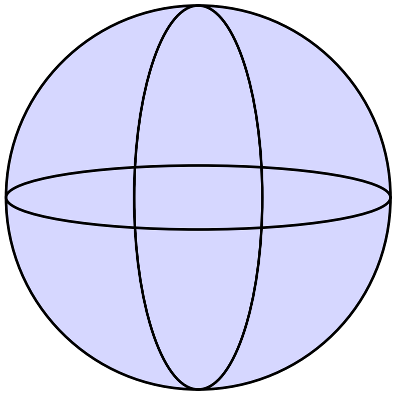
$$\mathbf{q} = \mathbf{p}_1 - \mathbf{p}_0$$
$$\mathbf{r} = \mathbf{p}_2 - \mathbf{p}_0$$



Parametric Sphere

$x(u,v) = r \cos\theta \sin\phi$
$y(u,v) = r \sin\theta \sin\phi$
$z(u,v) = r \cos\phi$

$360 \geq \theta \geq 0$
$180 \geq \phi \geq 0$



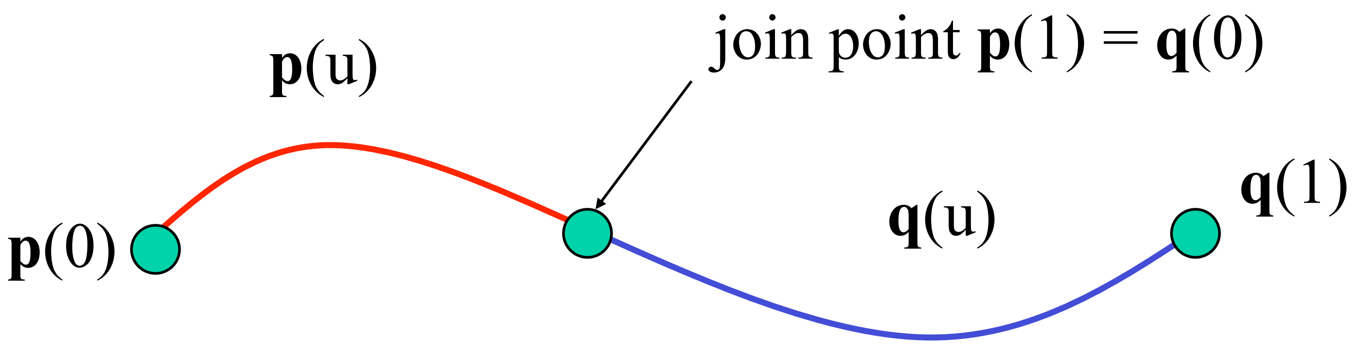
θ constant: circles of constant longitude
ϕ constant: circles of constant latitude

differentiate to show $\mathbf{n} = \mathbf{p}$

Curve Segments

- After normalizing u, each curve is written

  $\mathbf{p}(u)=[x(u), y(u), z(u)]^T, \quad 1 \geq u \geq 0$
- In classical numerical methods, we design a single global curve
- In computer graphics and CAD, it is better to design small connected curve *segments*



Parametric Polynomial Curves

$$x(u) = \sum_{i=0}^{N} c_{xi} u^i \quad y(u) = \sum_{j=0}^{M} c_{yj} u^j \quad z(u) = \sum_{k=0}^{L} c_{zk} u^k$$

•If N=M=K, we need to determine 3(N+1) coefficients

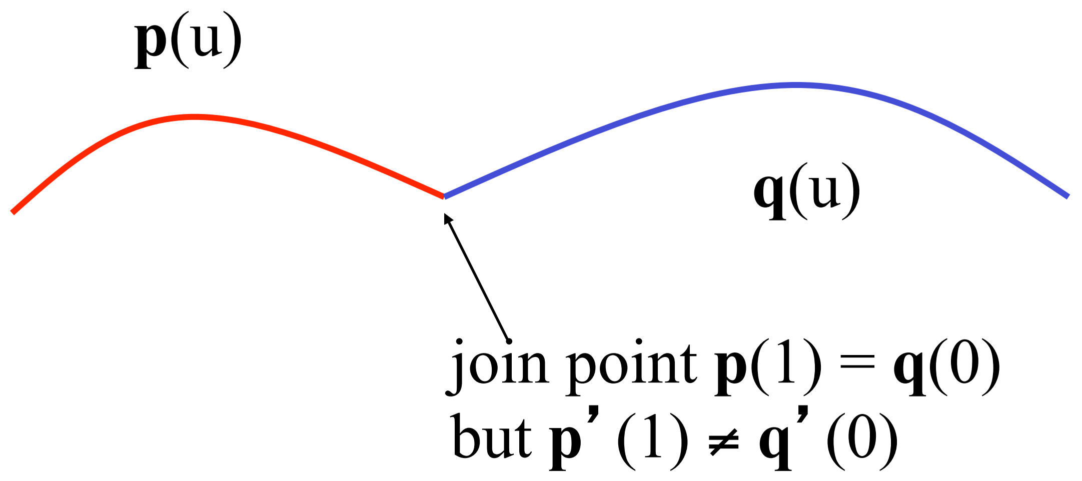
•Equivalently we need 3(N+1) independent conditions

•Noting that the curves for x, y and z are independent, we can define each independently in an identical manner

•We will use the form $\mathrm{p}(u) = \sum_{k=0}^{L} c_k u^k$
where p can be any of x, y, z

Why Polynomials

- Easy to evaluate
- Continuous and differentiable everywhere
  - Must worry about continuity at join points including continuity of derivatives


$\mathbf{p}(u)$
$\mathbf{q}(u)$
join point $\mathbf{p}(1) = \mathbf{q}(0)$
but $\mathbf{p}'(1) \neq \mathbf{q}'(0)$

Cubic Parametric Polynomials

- N=M=L=3, gives balance between ease of evaluation and flexibility in design

$$p(u) = \sum_{k=0}^{3} c_k u^k$$

- Four coefficients to determine for each of x, y and z
- Seek four independent conditions for various values of u resulting in 4 equations in 4 unknowns for each of x, y and z
  - Conditions are a mixture of continuity requirements at the join points and conditions for fitting the data

Cubic Polynomial Surfaces

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

where

$$p(u,v) = \sum_{i=0}^{3}\sum_{j=0}^{3} c_{ij} u^i v^j$$

p is any of x, y or z

Need 48 coefficients (3 independent sets of 16) to determine a surface patch

Designing Parametric Cubic Curves

CS 432 Interactive Computer Graphics

Prof. David E. Breen

Department of Computer Science

Objectives

- Introduce the types of curves
  - Interpolating
  - Hermite
  - Bezier
  - B-spline
- Analyze their performance

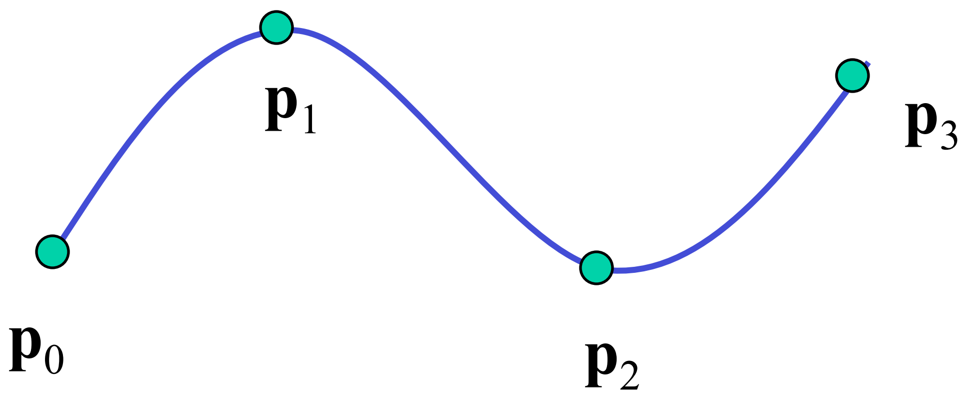
Matrix-Vector Form

$$\mathrm{p}(u) = \sum_{k=0}^{3} c_k u^k$$

define $\mathbf{c} = \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$ $\mathbf{u} = \begin{bmatrix} 1 \\ u \\ u^2 \\ u^3 \end{bmatrix}$

then $\mathrm{p}(u) = \mathbf{u}^T \mathbf{c} = \mathbf{c}^T \mathbf{u}$

Interpolating Curve



Given four data (control) points $\mathbf{p}_0$, $\mathbf{p}_1$,$\mathbf{p}_2$, $\mathbf{p}_3$
determine cubic $\mathbf{p}(u)$ which passes through them

Must find $\mathbf{c}_0$,$\mathbf{c}_1$,$\mathbf{c}_2$, $\mathbf{c}_3$

Interpolation Equations

apply the interpolating conditions at u=0, 1/3, 2/3, 1

$p_0 = p(0) = c_0$
$p_1 = p(1/3) = c_0 + (1/3)c_1 + (1/3)^2 c_2 + (1/3)^3 c_2$
$p_2 = p(2/3) = c_0 + (2/3)c_1 + (2/3)^2 c_2 + (2/3)^3 c_2$
$p_3 = p(1) = c_0 + c_1 + c_2 + c_2$

or in matrix form with **p** = $[p_0\ p_1\ p_2\ p_3]^T$

$$\mathbf{p} = \mathbf{Ac} \qquad \mathbf{A} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & \left(\frac{1}{3}\right) & \left(\frac{1}{3}\right)^2 & \left(\frac{1}{3}\right)^3 \\ 1 & \left(\frac{2}{3}\right) & \left(\frac{2}{3}\right)^2 & \left(\frac{2}{3}\right)^3 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

Interpolation Matrix

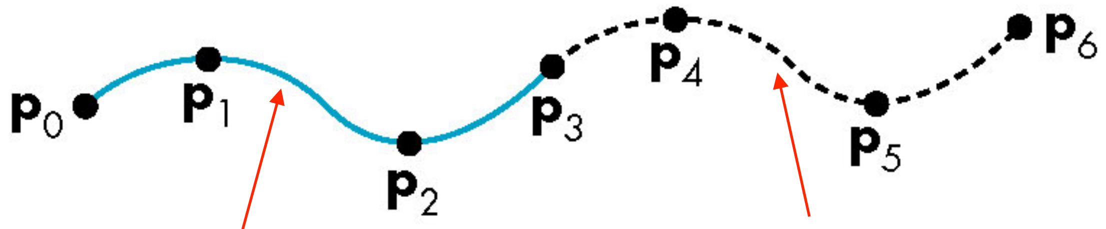
Solving for $\mathbf{c}$ we find the *interpolation matrix*

$$\mathbf{M}_I = \mathbf{A}^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -5.5 & 9 & -4.5 & 1 \\ 9 & -22.5 & 18 & -4.5 \\ -4.5 & 13.5 & -13.5 & 4.5 \end{bmatrix}$$

$$\mathbf{c} = \mathbf{M}_I \mathbf{p}$$

Note that $\mathbf{M}_I$ does not depend on input data and can be used for each segment in x, y, and z

Interpolating Multiple Segments



use $\mathbf{p} = [p_0\ p_1\ p_2\ p_3]^T$

use $\mathbf{p} = [p_3\ p_4\ p_5\ p_6]^T$

Get continuity at join points but not continuity of derivatives

Blending Functions

Rewriting the equation for p(u)

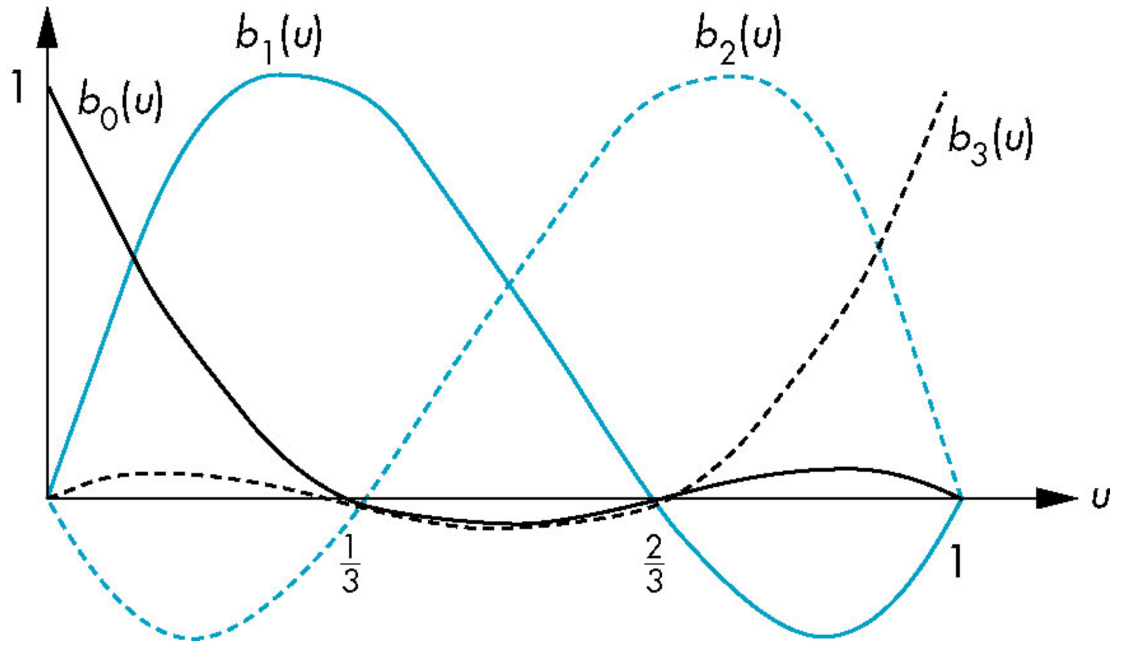
$$p(u)=\mathbf{u}^T\mathbf{c}=\mathbf{u}^T\mathbf{M}_I\mathbf{p} = \mathbf{b}(u)^T\mathbf{p}$$

where $b(u) = [b_0(u)\ b_1(u)\ b_2(u)\ b_3(u)]^T$ is an array of *blending polynomials* such that $p(u) = b_0(u)p_0 + b_1(u)p_1 + b_2(u)p_2 + b_3(u)p_3$

$$b_0(u) = -4.5(u-1/3)(u-2/3)(u-1)$$
$$b_1(u) = 13.5u\,(u-2/3)(u-1)$$
$$b_2(u) = -13.5u\,(u-1/3)(u-1)$$
$$b_3(u) = 4.5u\,(u-1/3)(u-2/3)$$

Blending Functions

- These functions are not smooth
  - Hence the interpolation polynomial is not smooth

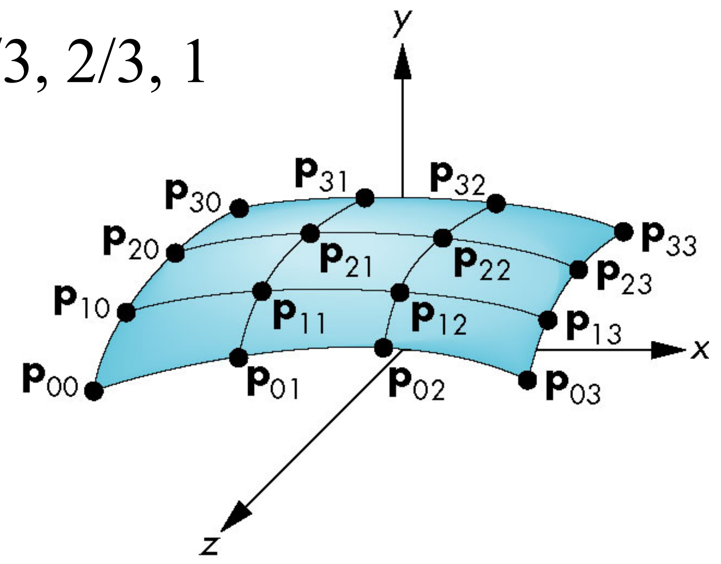


Interpolating Patch

$$p(u,v) = \sum_{i=o}^{3} \sum_{j=0}^{3} c_{ij} u^i v^j$$

Need 16 conditions to determine the 16 coefficients c_{ij}

Choose at u,v = 0, 1/3, 2/3, 1



Matrix Form

Define $\mathbf{v} = [1\ v\ v^2\ v^3]^T$

$$\mathbf{C} = [c_{ij}] \qquad \mathbf{P} = [p_{ij}]$$

$$p(u,v) = \mathbf{u}^T C \mathbf{v}$$

If we observe that for constant u (v), we obtain interpolating curve in v (u), we can show

$$\mathbf{C} = M_I P M_I$$

$$p(u,v) = u^T M_I P M_I^T v$$

Blending Patches

$$p(u,v) = \sum_{i=o}^{3} \sum_{j=0}^{3} b_i(u)\, b_j(v)\, p_{ij}$$

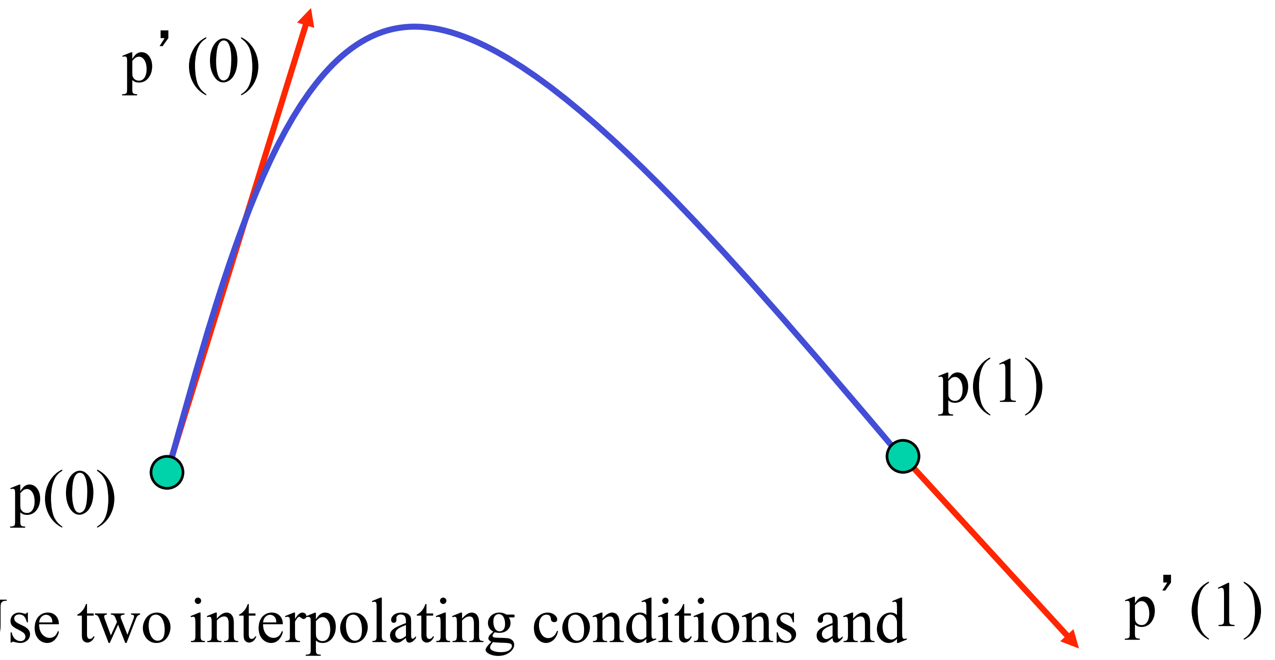
Each $b_i(u)b_j(v)$ is a blending patch

Shows that we can build and analyze surfaces from our knowledge of curves

Other Types of Curves and Surfaces

- How can we get around the limitations of the interpolating form
  - Lack of smoothness
  - Discontinuous derivatives at join points
- We have four conditions (for cubics) that we can apply to each segment
  - Use them other than for interpolation
  - Need only come close to the data

Hermite Form



Use two interpolating conditions and two derivative conditions per segment

Ensures continuity and first derivative continuity between segments

Equations

Interpolating conditions are the same at ends

$$p(0) = p_0 = c_0$$
$$p(1) = p_3 = c_0+c_1+c_2+c_3$$

Differentiating we find $p'(u) = c_1+2uc_2+3u^2c_3$

Evaluating at end points

$$p'(0) = p'_0 = c_1$$
$$p'(1) = p'_3 = c_1+2c_2+3c_3$$

Matrix Form

$$\mathbf{q} = \begin{bmatrix} p_0 \\ p_3 \\ p'_0 \\ p'_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 2 & 3 \end{bmatrix} \mathbf{c}$$

Solving, we find $\mathbf{c}=\mathbf{M}_H\mathbf{q}$ where $\mathbf{M}_H$ is the Hermite matrix

$$\mathbf{M}_H = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -3 & 3 & -2 & -1 \\ 2 & -2 & 1 & 1 \end{bmatrix}$$

Blending Polynomials

$$p(u) = \mathbf{b}(u)^T\mathbf{q}$$

$$\mathbf{b}(u) = \begin{bmatrix} 2u^3 - 3u^2 + 1 \\ -2u^3 + 3u^2 \\ u^3 - 2u^2 + u \\ u^3 - u^2 \end{bmatrix}$$

Although these functions are smooth, the Hermite form is not used directly in Computer Graphics and CAD because we usually have control points but not derivatives

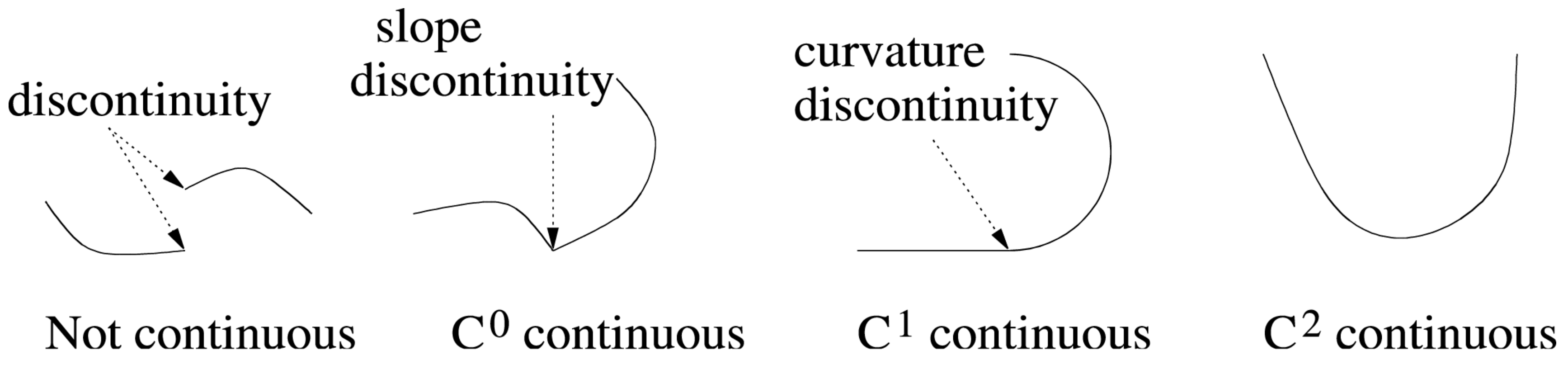
However, the Hermite form is the basis of the Bezier form

Parametric and Geometric Continuity

- We can require the derivatives of x, y,and z to each be continuous at join points (*parametric continuity*)
- Alternately, we can only require that the tangents of the resulting curve be continuous (*geometry continuity*)
- The latter gives more flexibility as we have need satisfy only two conditions rather than three at each join point

Parametric Continuity

- ***Continuity*** (recall from the calculus):
  - Two curves are C^i continuous at a point p iff the i-th derivatives of the curves are equal at p



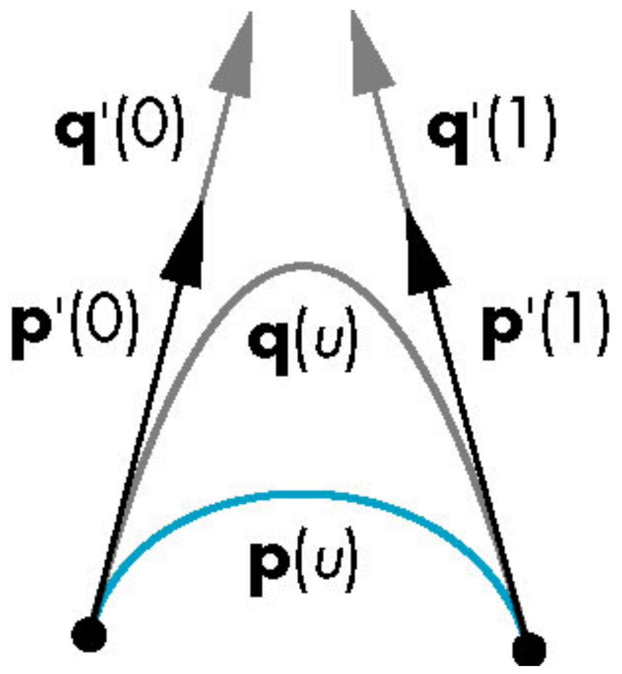
Pics/Math courtesy of Dave Mount @ UMD-CP

Example

- Here the p and q have the same tangents at the ends of the segment but different derivatives
- Generate different

  Hermite curves
- This techniques is used

in drawing applications



Higher Dimensional Approximations

- The techniques for both interpolating and Hermite curves can be used with higher dimensional parametric polynomials
- For interpolating form, the resulting matrix becomes increasingly more ill-conditioned and the resulting curves less smooth and more prone to numerical errors
- In both cases, there is more work in rendering the resulting polynomial curves and surfaces

Bezier and Spline Curves and Surfaces

CS 432 Interactive Computer Graphics

Prof. David E. Breen

Department of Computer Science

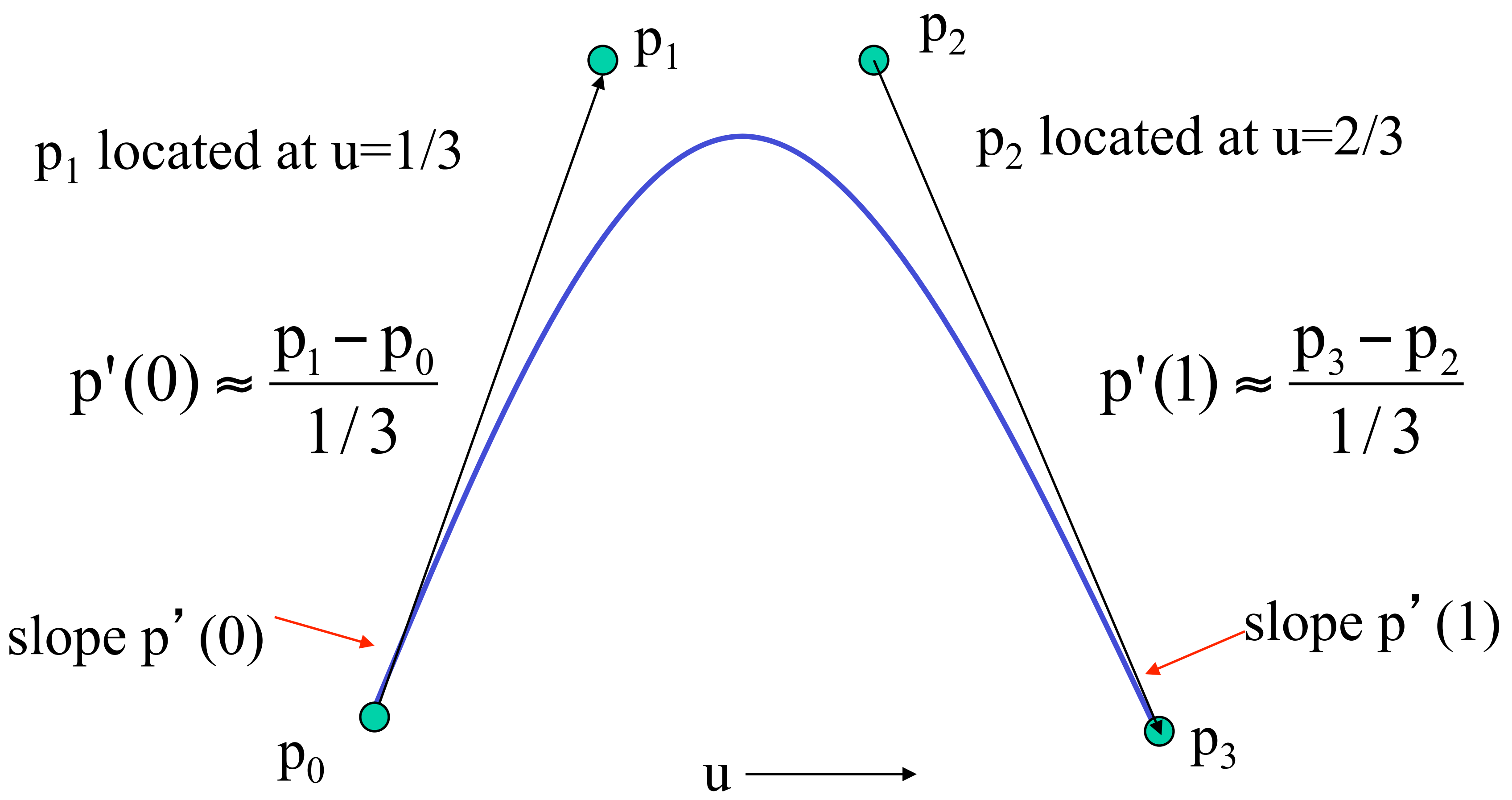
Objectives

- Introduce the Bezier curves and surfaces
- Derive the required matrices
- Introduce the B-spline and compare it to the standard cubic Bezier

Bezier’s Idea

- In graphics and CAD, we do not usually have derivative data
- Bezier suggested using the same 4 data points as with the cubic interpolating curve to approximate the derivatives in the Hermite form

Approximating Derivatives



$$p'(0) \approx \frac{p_1 - p_0}{1/3}$$

$$p'(1) \approx \frac{p_3 - p_2}{1/3}$$

Equations

Interpolating conditions are the same

$$p(0) = p_0 = c_0$$
$$p(1) = p_3 = c_0+c_1+c_2+c_3$$

Approximating derivative conditions

$$p'(0) = 3(p_1 - p_0) = c_0$$
$$p'(1) = 3(p_3 - p_2) = c_1+2c_2+3c_3$$

Solve four linear equations for $\mathbf{c}=\mathbf{M}_B\mathbf{p}$

Bezier Matrix

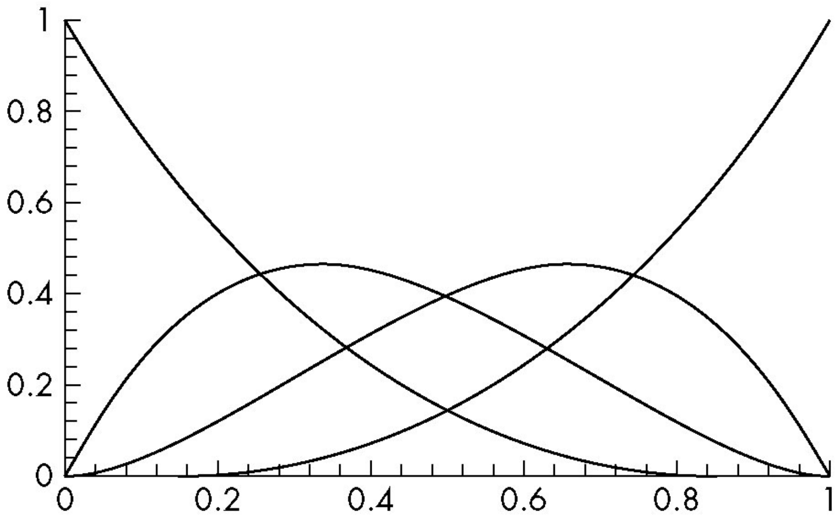
$$\mathbf{M}_B = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -3 & 3 & 0 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{bmatrix}$$

$$p(u) = \mathbf{u}^T \mathbf{M}_B \mathbf{p} = \underbrace{\mathbf{b}(u)^T}_{\text{blending functions}} \mathbf{p}$$

blending functions

Blending Functions

$$\mathbf{b}(u) = \begin{bmatrix} (1-u)^3 \\ 3u(1-u)^2 \\ 3u^2(1-u) \\ u^3 \end{bmatrix}$$



Note that all zeros are at 0 and 1 which forces the functions to be smooth over (0,1)

Cubic Bezier Curve

- Multiplying it all out gives

$$p(u) = (1-u)^3\mathbf{p_0} + 3u(1-u)^2\mathbf{p_1} + 3u^2(1-u)\,\mathbf{p_2} + u^3\mathbf{p_3}$$

Bernstein Polynomials

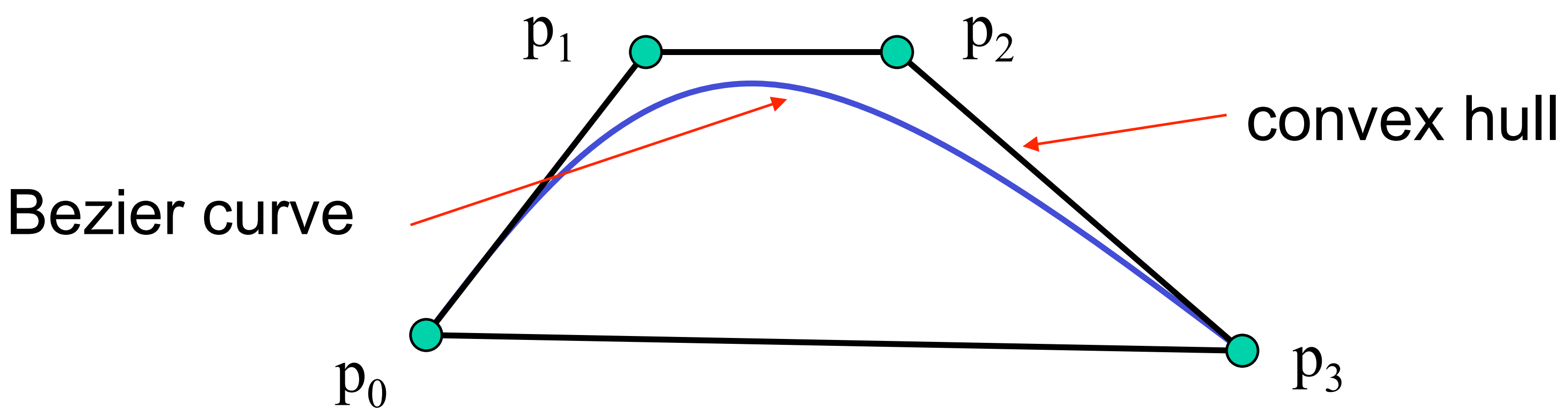
- The blending functions are a special case of the Bernstein polynomials

$$b_{\mathrm{kd}}(u) = \frac{d!}{k!(d-k)!} u^k (1-u)^{d-k}$$

- These polynomials give the blending polynomials for any degree Bezier form
    - All zeros at 0 and 1
    - For any degree they all sum to 1
    - They are all between 0 and 1 inside (0,1)

Convex Hull Property

- The properties of the Bernstein polynomials ensure that all Bezier curves lie in the convex hull of their control points
- Hence, even though we do not interpolate all the data, we cannot be too far away



General Form of Bezier Curve

$$\vec{p}(u) = \sum_{i=0}^{k} \vec{p}_{i+1} \binom{k}{i} (1-u)^{k-i} u^{i}$$

Analysis

- Although the Bezier form is much better than the interpolating form, its derivatives are not continuous at join points
- Can we do better?
  - Go to higher order Bezier
    - More work
    - Derivative continuity still only approximate
    - Supported by OpenGL
  - Apply different conditions
    - Tricky without letting order increase

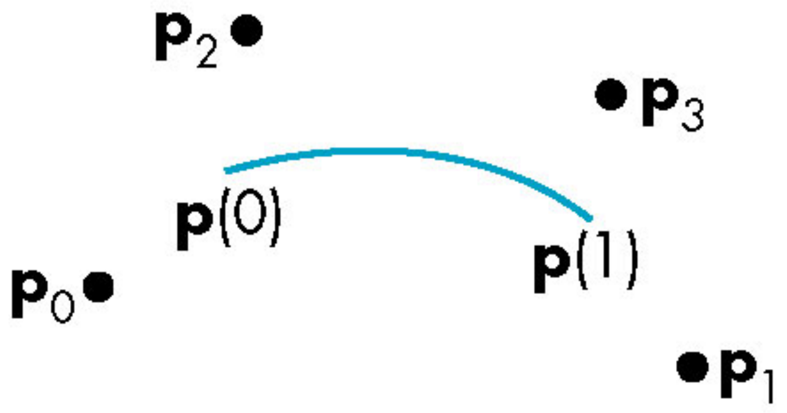
B-Splines

- Basis splines: use the data at $\mathbf{p}=[p_{i-2}\ p_{i-1}\ p_i\ p_{i-1}]^T$ to define curve only between p_{i-1} and p_i
- Allows us to apply more continuity conditions to each segment
- For cubics, we can have continuity of function, first and second derivatives at join points
- Cost is 3 times as much work for curves
  - Add one new point each time rather than three
- For surfaces, we do 9 times as much work

Cubic B-spline

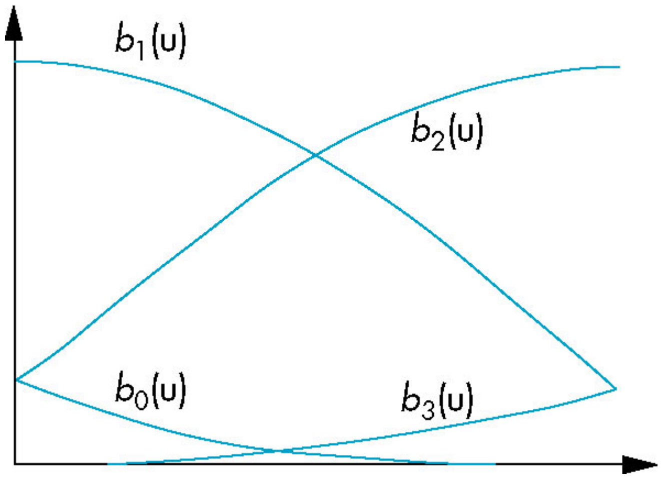
$$p(u) = \mathbf{u}^T\mathbf{M}_S\mathbf{p} = \mathbf{b}(u)^T\mathbf{p}$$

$$\mathbf{M}_S = \begin{bmatrix} 1 & 4 & 1 & 0 \\ -3 & 0 & 3 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{bmatrix}$$

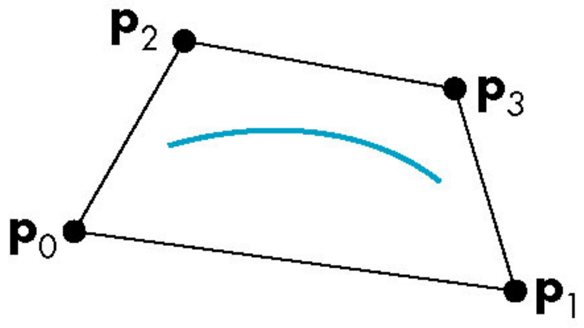


Blending Functions

$$\mathbf{b}(u) = \frac{1}{6}\begin{bmatrix} (1-u)^3 \\ 4-6u^2+3u^3 \\ 1+3u+3u^2-3u^2 \\ u^3 \end{bmatrix}$$



convex hull property



Splines and Basis

- If we examine the cubic B-spline from the perspective of each control (data) point, each interior point contributes (through the blending functions) to four segments
- We can rewrite p(u) in terms of the data points as

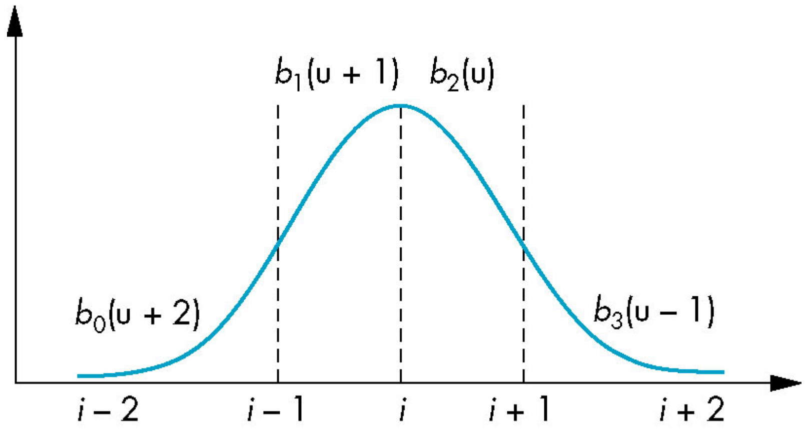
$$p(u) = \sum B_i(u)\, p_i$$

defining the basis functions $\{B_i(u)\}$

Basis Functions

In terms of the blending polynomials

$$B_i(u) = \begin{cases} 0 & u < i-2 \\ b_0(u+2) & i-2 \le u < i-1 \\ b_1(u+1) & i-1 \le u < i \\ b_2(u) & i \le u < i+1 \\ b_3(u-1) & i+1 \le u < i+2 \\ 0 & u \ge i+2 \end{cases}$$



Generalizing Splines

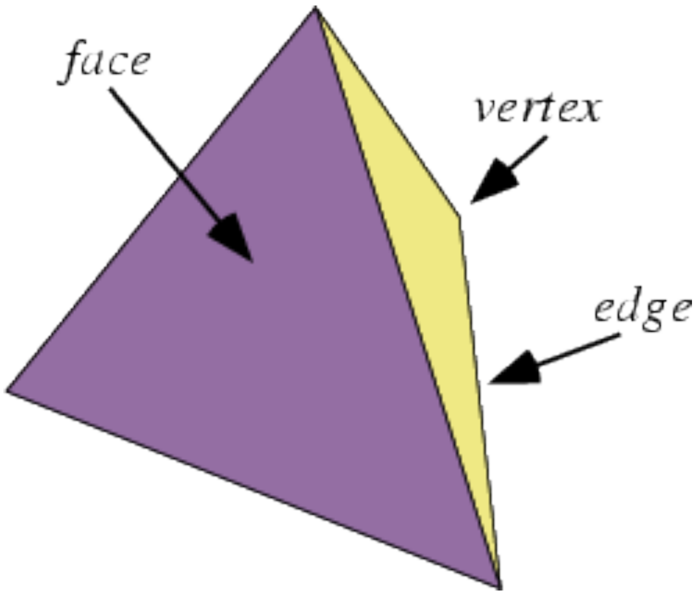
- We can extend to splines of any degree
- Data and conditions to not have to given at equally spaced values (the *knots*)
  - Nonuniform and uniform splines
  - Can have repeated knots
    - Can force spline to interpolate points
- Cox-deBoor recursion gives method of evaluation

NURBS

- <u>N</u>on<u>u</u>niform <u>R</u>ational <u>B</u>-<u>S</u>pline curves and surfaces add a fourth variable w to x,y,z
  - Can interpret as weight to give more importance to some control data
  - Can also interpret as moving to homogeneous coordinate
- Requires a perspective division
  - NURBS act correctly for perspective viewing
- Quadrics are a special case of NURBS

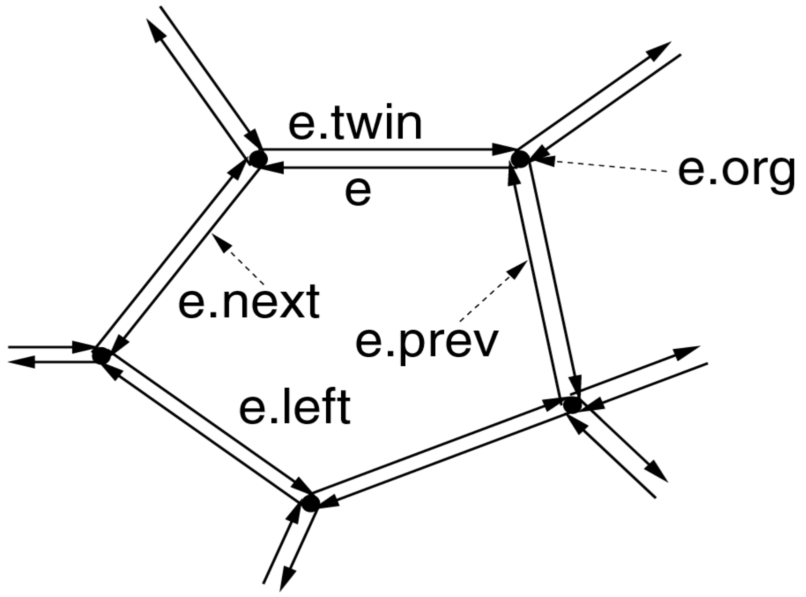
Issues with 3D “mesh” formats

- Easy to acquire
- Easy to render
- Harder to model with
- Error prone
  - split faces, holes, gaps, etc



BRep Data Structures

- Winged-Edge Data Structure (Weiler)
- Vertex
  - n edges
- Edge
  - 2 vertices
  - 2 faces
- Face
  - m edges



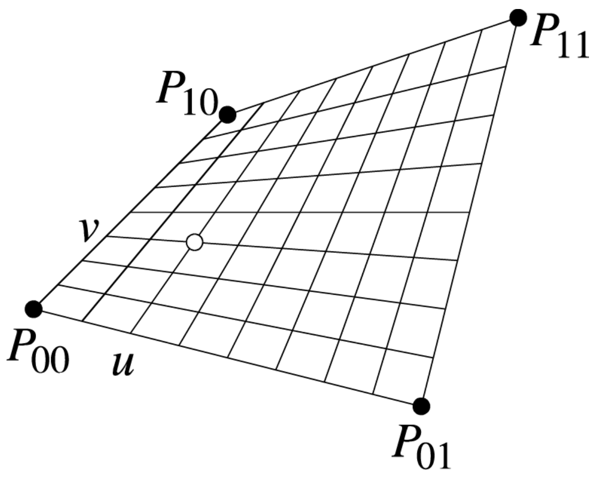
Pics/Math courtesy of Dave Mount @ UMD-CP

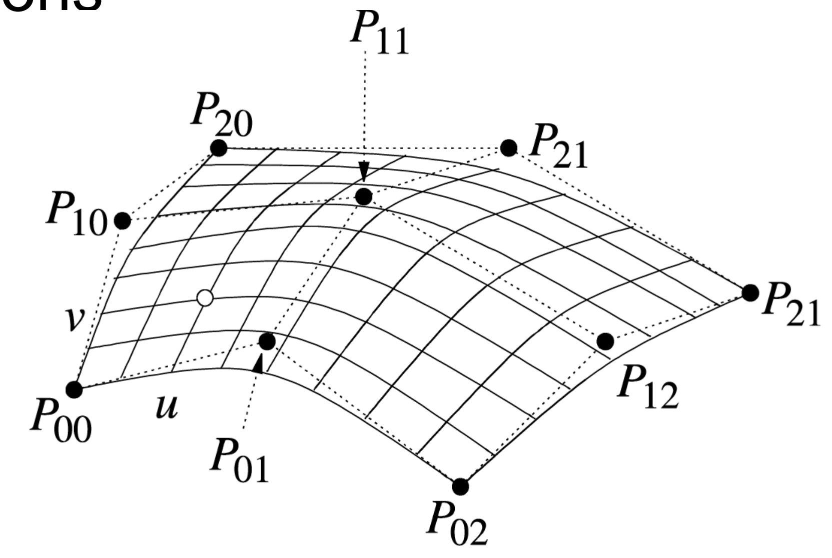
BRep Data Structure

- Vertex structure
  - X,Y,Z point
  - Pointers to *n* coincident edges
- Edge structure
  - 2 pointers to end-point vertices
  - 2 pointers to adjacent faces
  - Pointer to next edge
  - Pointer to previous edge
- Face structure
  - Pointers to *m* edges

Biparametric Surfaces

- Biparametric surfaces
  - A generalization of parametric curves
  - 2 parameters: *s*, *t* (or *u*, *v*)
  - Two parametric functions





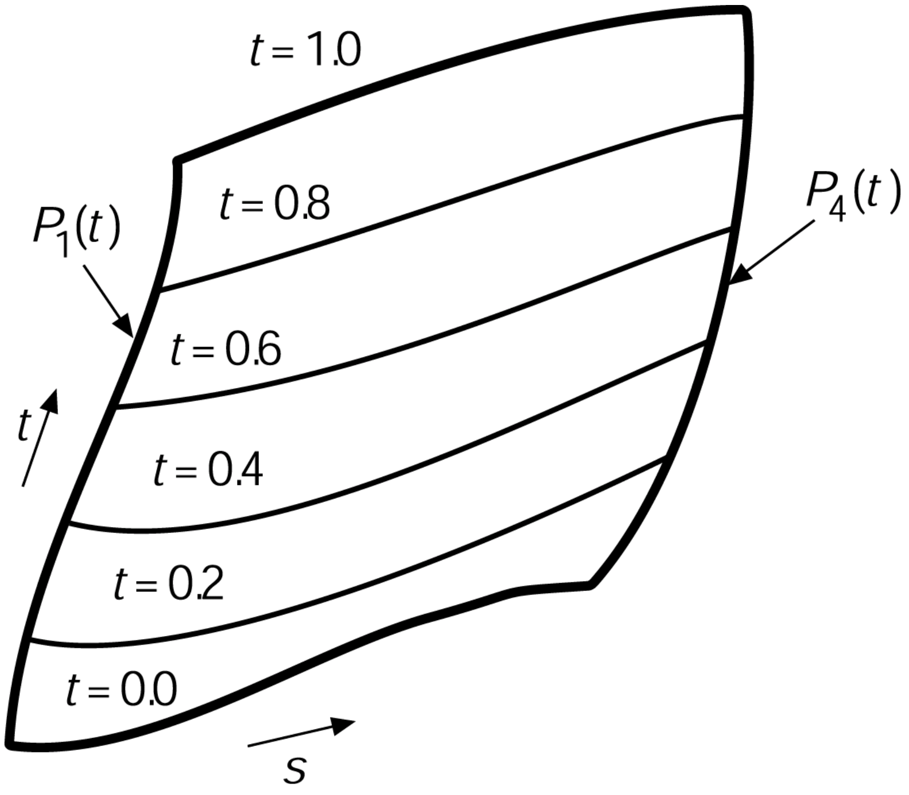
Bicubic Surfaces

- Recall the 2D curve: $Q(s) = G \cdot M \cdot S$
  - *G*: Geometry Matrix
  - *M*: Basis Matrix
  - *S*: Polynomial Terms $[s^3 \;\; s^2 \;\; s \;\; 1]$
- For 3D, we allow the points in G to vary in 3D along *t* as well:

$$Q(s,t) = \begin{bmatrix} G_1(t) & G_2(t) & G_3(t) & G_4(t) \end{bmatrix} \cdot M \cdot S$$

Observations About Bicubic Surfaces

- For a fixed *t_1*, $Q(s, t_1)$ is a curve
- Gradually incrementing *t_1 to t_2*, we get a new curve
- The combination of these curves is a surface
- $G_i(t)$ are 3D curves



Bicubic Surfaces

- Each $G_i(t)$ is $G_i(t) = \mathbf{G_i} \cdot \mathrm{M} \cdot \mathrm{T}$, where

$$\mathbf{G_i} = \begin{bmatrix} g_{i1} & g_{i2} & g_{i3} & g_{i4} \end{bmatrix}$$

- Transposing $G_i(t)$, we get

$$G_i(t) = T^T \cdot M^T \cdot \mathbf{G_i}^{\mathrm{T}}$$

$$= \mathbf{T}^{\mathrm{T}} \cdot \mathbf{M}^{\mathrm{T}} \cdot \begin{bmatrix} g_{i1} & g_{i2} & g_{i3} & g_{i4} \end{bmatrix}^{\mathrm{T}}$$

Bicubic Surfaces

- Substituting $G_i(t)$ into $Q(s) = G \cdot M \cdot S$ we get $Q(s, t)$
- The g_{11}, etc. are the *control points* for the Bicubic surface patch:

$$Q(s,t) = T^T \cdot M^T \cdot \begin{bmatrix} g_{11} & g_{21} & g_{31} & g_{41} \\ g_{12} & g_{22} & g_{32} & g_{42} \\ g_{13} & g_{23} & g_{33} & g_{43} \\ g_{14} & g_{24} & g_{34} & g_{44} \end{bmatrix} \cdot M \cdot S$$

Bicubic Surfaces

- Writing out $Q(s,t) = T^T \cdot M^T \cdot \mathbf{G} \cdot \mathrm{M} \cdot \mathrm{S} \quad 0 \leq \mathrm{s,t} \leq 1$ gives

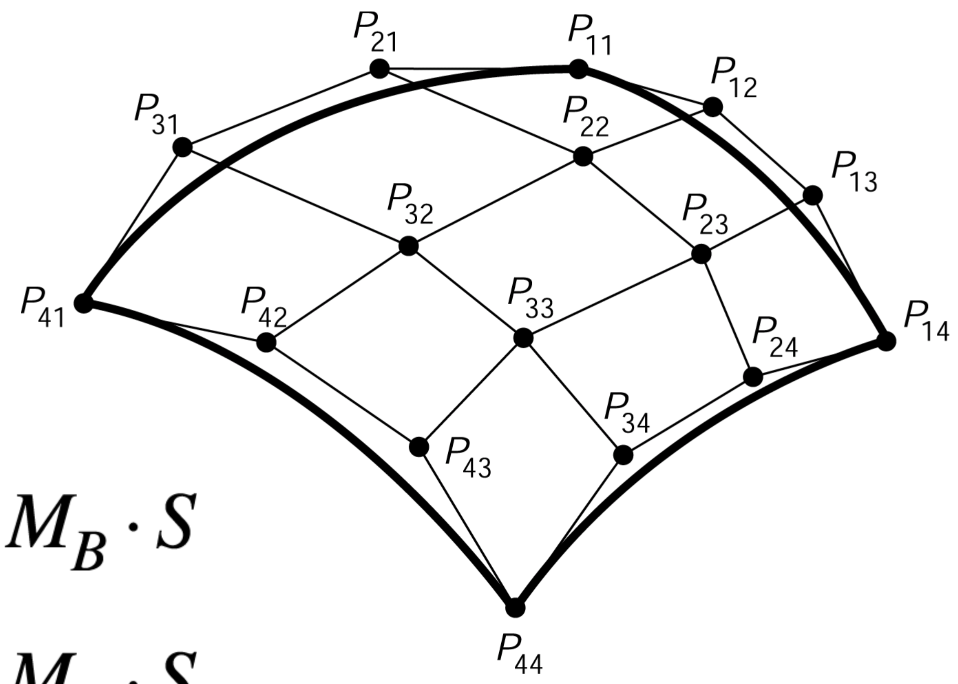
$$x(s,t) = T^T \cdot M^T \cdot G_x \cdot M \cdot S$$

$$y(s,t) = T^T \cdot M^T \cdot G_y \cdot M \cdot S$$

$$z(s,t) = T^T \cdot M^T \cdot G_z \cdot M \cdot S$$

Bézier Patches

- Bézier Surfaces (similar definition)



$$x(s,t) = T^T \cdot M_B^T \cdot G_{B_x} \cdot M_B \cdot S$$

$$y(s,t) = T^T \cdot M_B^T \cdot G_{B_y} \cdot M_B \cdot S$$

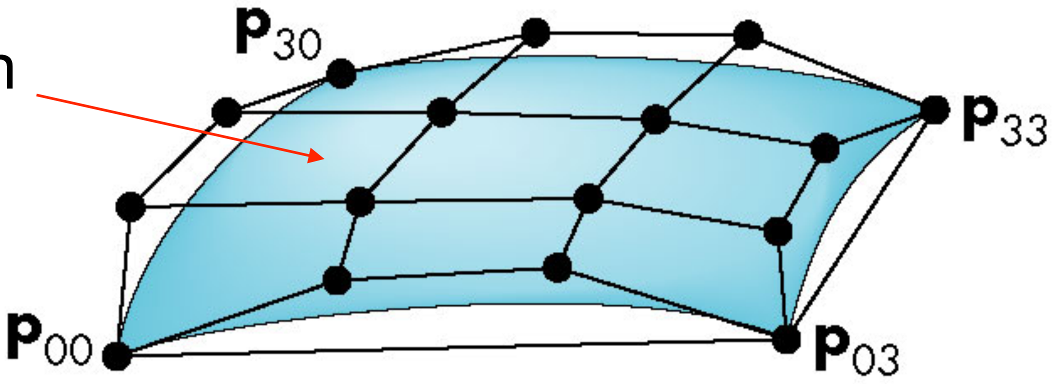
$$z(s,t) = T^T \cdot M_B^T \cdot G_{B_z} \cdot M_B \cdot S$$

Bezier Patches

Using same data array **P**=[p_{ij}] as with interpolating form

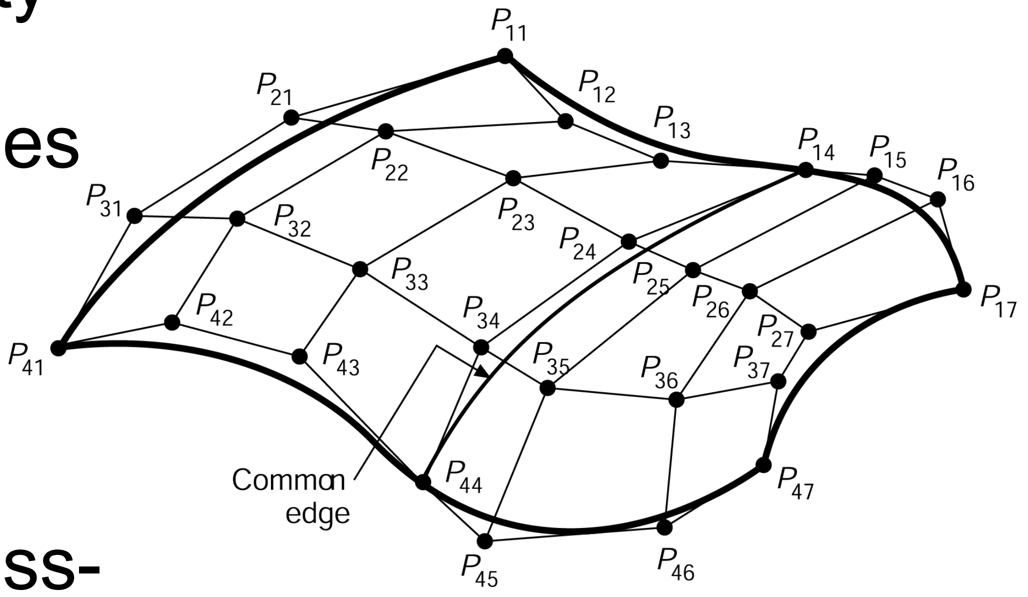
$$p(u,v) = \sum_{i=0}^{3}\sum_{j=0}^{3} b_i(u)\, b_j(v)\, p_{ij} = u^T \mathbf{M}_B \mathbf{P} \mathbf{M}_B^T v$$

Patch lies in convex hull



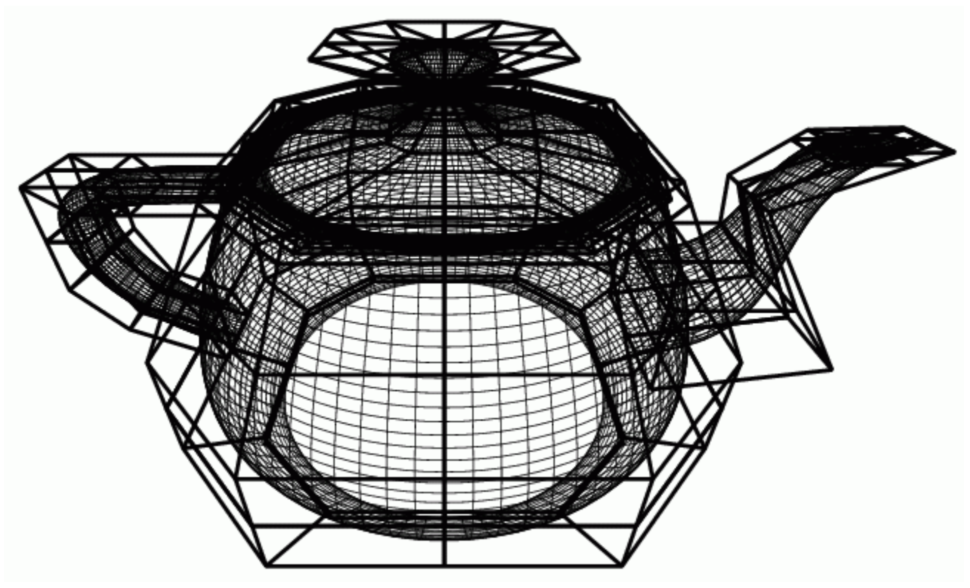
Bézier Surfaces

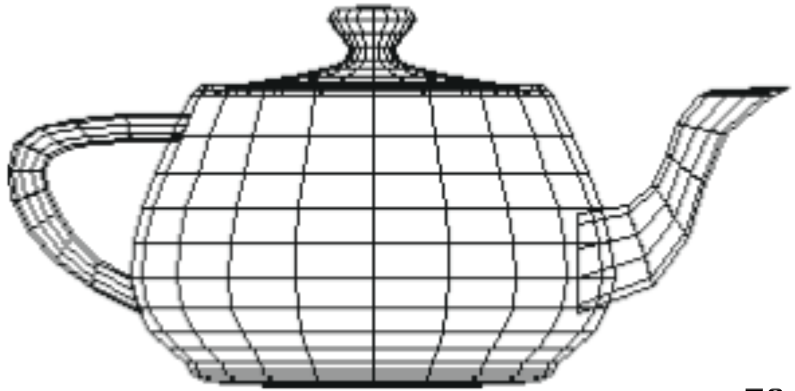
- C^0 and G^0 continuity can be achieved between two patches by setting the 4 boundary control points to be equal
- G^1 continuity achieved when cross-wise CPs are co-linear



Bézier Surfaces: Example
Utah Teapot

- Utah Teapot modeled with 306 3D control points that define 32 Bézier patches with *G*^{1} continuity





Rendering Curves and Surfaces

CS 432 Interactive Computer Graphics

Prof. David E. Breen

Department of Computer Science

Objectives

- Introduce methods to draw curves
  - Approximate with lines
  - Finite Differences
- Derive the recursive method for evaluation of Bezier curves and surfaces
- Learn how to convert all polynomial data to data for Bezier polynomials

Every Curve is a Bezier Curve

- We can render a given polynomial using the recursive method if we find control points for its representation as a Bezier curve
- Suppose that p(u) is given as an interpolating curve with control points **q**

$$p(u)=\mathbf{u}^T\mathbf{M}_I\mathbf{q}$$

- There exist Bezier control points **p** such that

$$p(u)=\mathbf{u}^T\mathbf{M}_B\mathbf{p}$$

- Equating and solving, we find $\mathbf{p}=\mathbf{M}_B^{-1}\mathbf{M}_I$

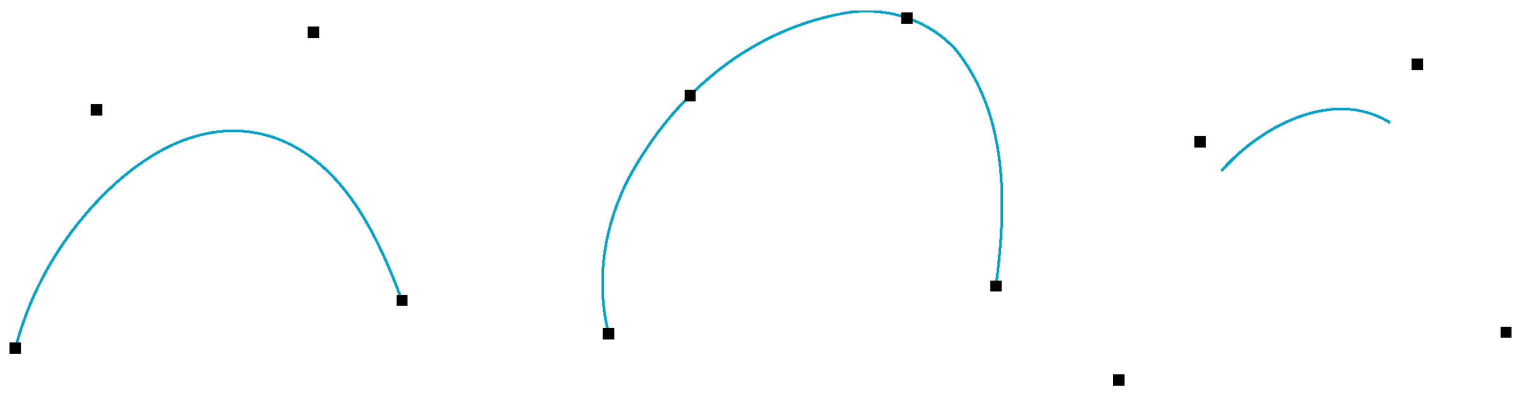
Matrices

Interpolating to Bezier $\mathbf{M}_B^{-1}\mathbf{M}_I = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -\frac{5}{6} & 3 & -\frac{3}{2} & \frac{1}{3} \\ \frac{1}{3} & -\frac{3}{2} & 3 & -\frac{5}{6} \\ 0 & 0 & 0 & 1 \end{bmatrix}$

B-Spline to Bezier $\mathbf{M}_B^{-1}\mathbf{M}_S = \begin{bmatrix} 1 & 4 & 1 & 0 \\ 0 & 4 & 2 & 0 \\ 0 & 2 & 4 & 0 \\ 0 & 1 & 4 & 1 \end{bmatrix}$

Example

These three curves were all generated from the same original data using Bezier recursion by converting all control point data to Bezier control points



Evaluating Polynomials

- Simplest method to render a polynomial curve is to evaluate the polynomial at many points and form an approximating polyline
- For surfaces we can form an approximating mesh of triangles or quadrilaterals
- Use Horner's method to evaluate polynomials

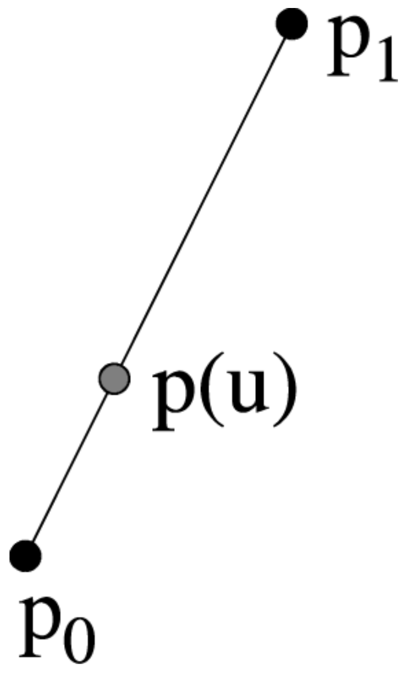
$$p(u)=c_0+u(c_1+u(c_2+uc_3))$$

- 3 multiplications/evaluation for cubic

The de Casteljau Algorithm

Basic case, with two points:

- Plotting a curve via *repeated linear interpolation*
  - Given $\langle \mathbf{p}_0, \mathbf{p}_1, \dots \rangle$ a sequence of control points
  - Simple case: Mapping a parameter u to the line $\overline{\mathbf{p}_0, \mathbf{p}_1}$



$$\mathbf{p}(u) = (1-u)\mathbf{p}_0 + u\mathbf{p}_1 \qquad \text{for } 0 \leq u \leq 1.$$

Pics/Math courtesy of Dave Mount @ UMD-CP

The de Casteljau Algorithm

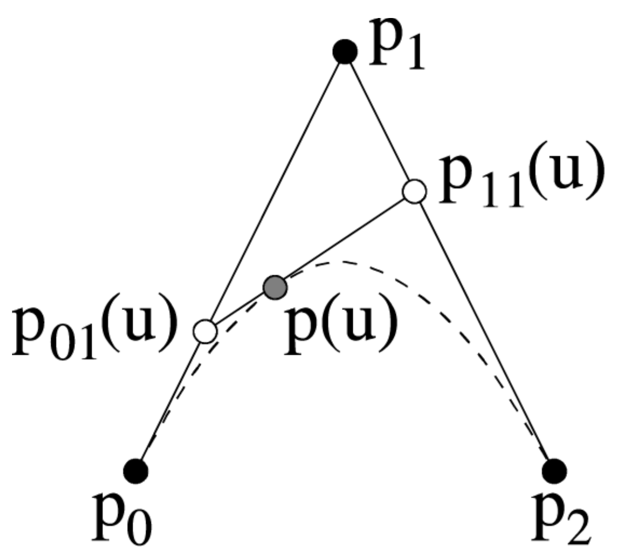
- The complete solution from the algorithm for three iterations:

$$\begin{aligned}\mathbf{p}_{01}(u) &= (1-u)\mathbf{p}_0 + u\mathbf{p}_1 \\ \mathbf{p}_{11}(u) &= (1-u)\mathbf{p}_1 + u\mathbf{p}_2.\end{aligned}$$

$$\mathbf{p}(u) = (1-u)\mathbf{p}_{01}(u) + u\mathbf{p}_{11}(u)$$

$$\begin{array}{lll} \mathbf{p_0} & & \\ \mathbf{p_1} & \mathbf{p_{01}} & \\ \mathbf{p_2} & \mathbf{p_{11}} & \mathbf{p(u)} \\ & & \uparrow \end{array}$$

Final Value

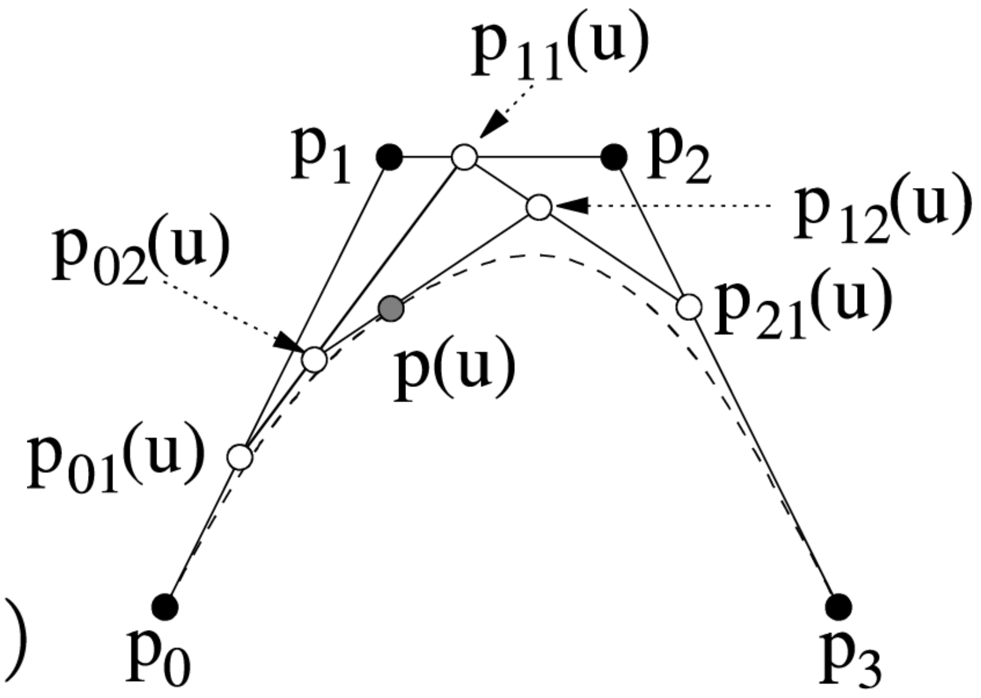


Pics/Math courtesy of Dave Mount @ UMD-CP

The de Casteljau Algorithm

- The solution after four iterations:

$\mathbf{p}_0$			
$\mathbf{p}_1$	$\mathbf{p}_{01}$		
$\mathbf{p}_2$	$\mathbf{p}_{11}$	$\mathbf{p}_{02}$	
$\mathbf{p}_3$	$\mathbf{p}_{21}$	$\mathbf{p}_{12}$	$\mathbf{p(u)}$



Pics/Math courtesy of Dave Mount @ UMD-CP

The de Casteljau Algorithm

- Input: $\mathrm{p}_0, \mathrm{p}_1, \mathrm{p}_2 \ldots \mathrm{p}_n \in R^3 \, , \, t \in R$
- Iteratively set:

$$\mathrm{p}_{ir}(t) = (1-t)\mathrm{p}_{i(r-1)}(t) + t\,\mathrm{p}_{(i+1)(r-1)}(t) \quad \begin{cases} r = 1,\ldots,n \\ i = 0,\ldots,n-r \end{cases}$$

and $\mathrm{p}_{i0}(t) = \mathrm{p}_i$

Then $\mathrm{p}_{0n}(t)$ is the point with parameter value *t* on the Bézier curve defined by the p_i' s

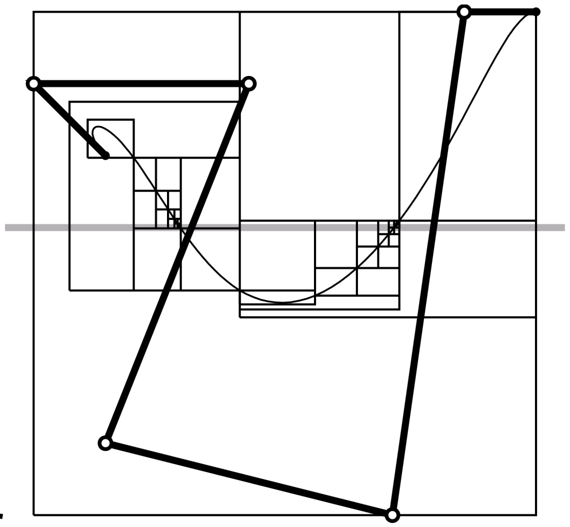
De Casteljau: Arc Segment Animation

De Casteljau: Cubic Curve Animation

Animated by Max Peysakhov @ Drexel University

Subdivision

- Common in many areas of graphics, CAD, CAGD, vision
- Basic idea
  - primitives def'd by control polygons
  - set of control points is not unique
    - more than one way to compute a curve
  - subdivision refines representation of an object by introducing more control points
- Allows for local modification
- Subdivide to pixel resolution



Pics/Math courtesy of G. Farin @ ASU

Bézier Curve Subdivision

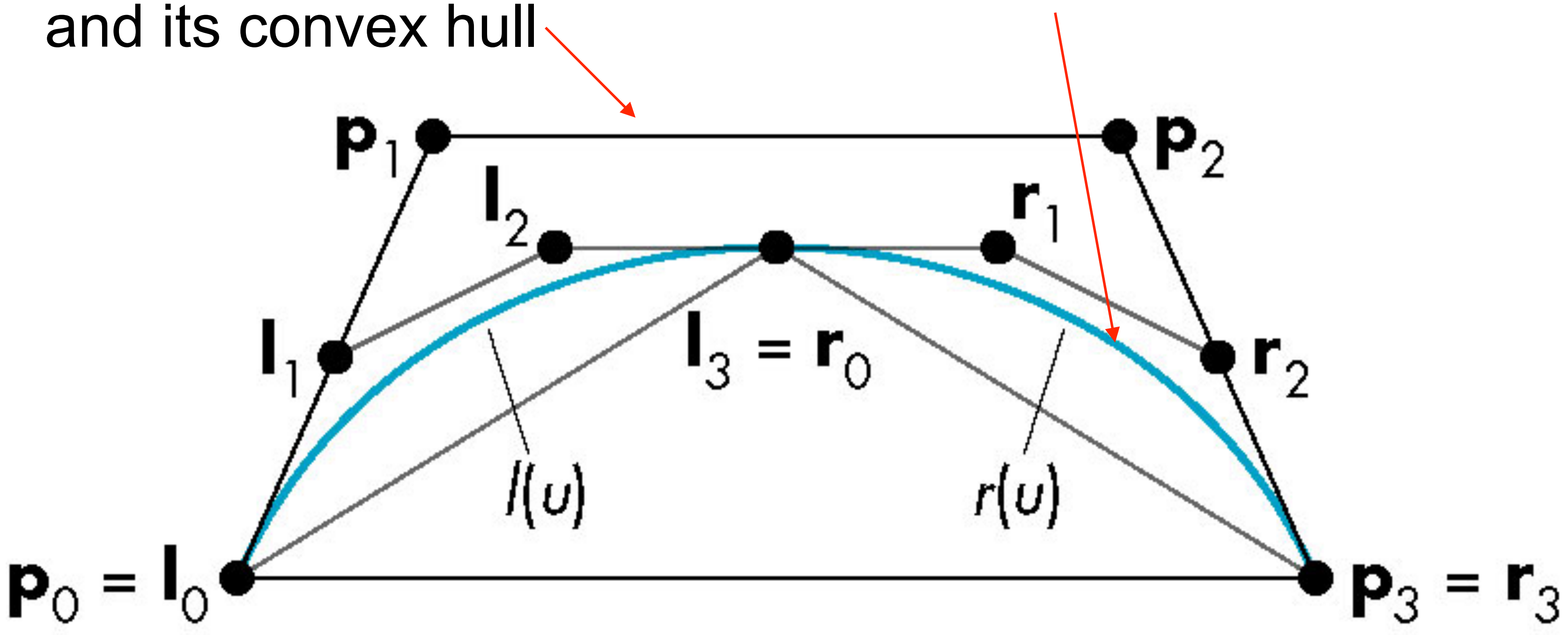
- Subdivision allows display of curves at different/adaptive levels of resolution
- Rendering systems (OpenGL, ActiveX, etc) only display polygons or lines
- Subdivision generates the lines/facets that approximate the curve/surface
  - output of subdivision sent to renderer

deCasteljau Recursion

- We can use the convex hull property of Bezier curves to obtain an efficient recursive method that does not require any function evaluations
  - Uses only the values at the control points
- Based on the idea that "any polynomial and any part of a polynomial is a Bezier polynomial for properly chosen control data"

Splitting a Cubic Bezier

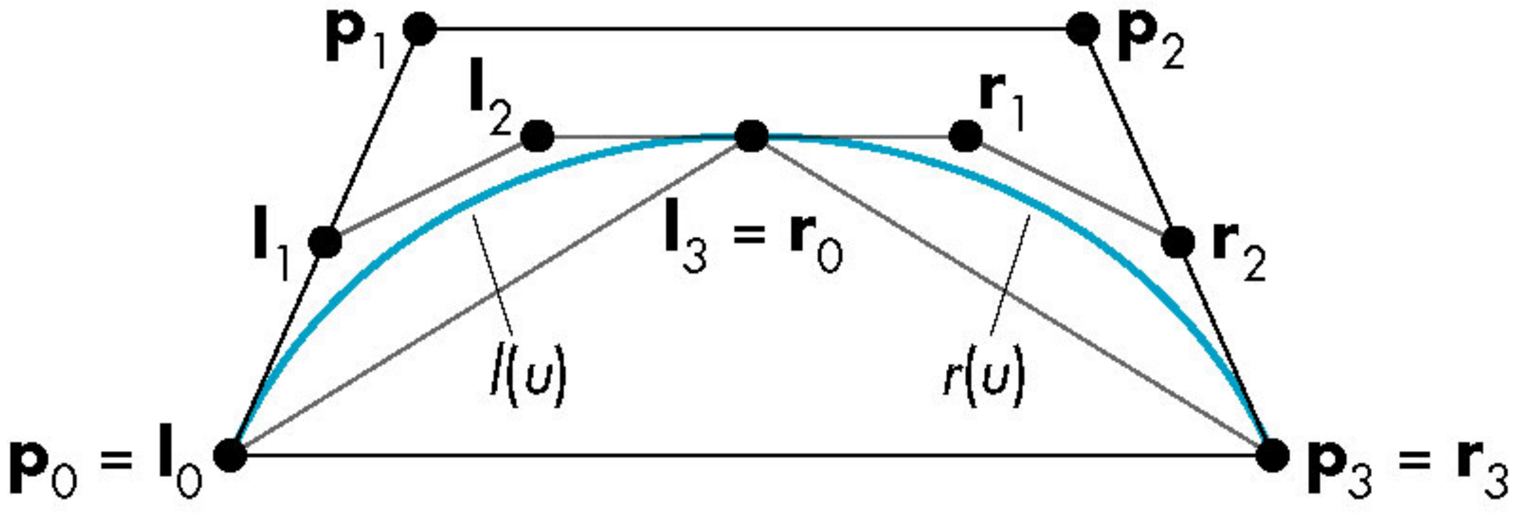
p_0, p_1, p_2, p_3 determine a cubic Bezier polynomial and its convex hull



Consider left half l(u) and right half r(u)

l(u) and r(u)

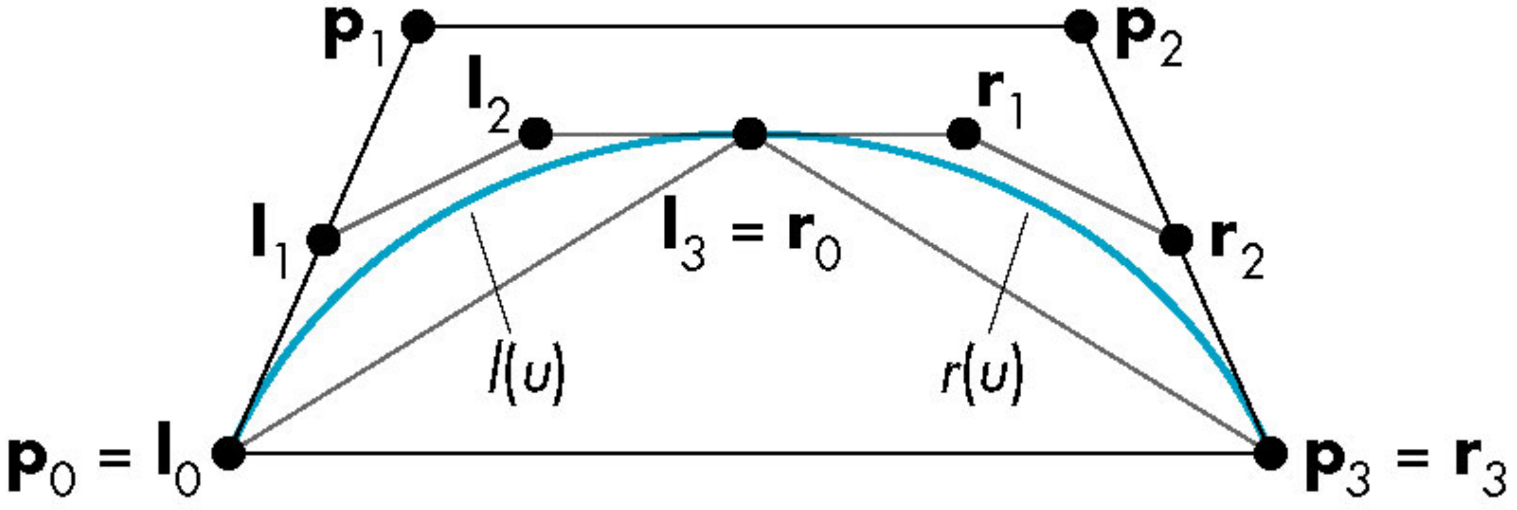
Since l(u) and r(u) are Bezier curves, we should be able to find two sets of control points $\{l_0, l_1, l_2, l_3\}$ and $\{r_0, r_1, r_2, r_3\}$ that determine them



Convex Hulls

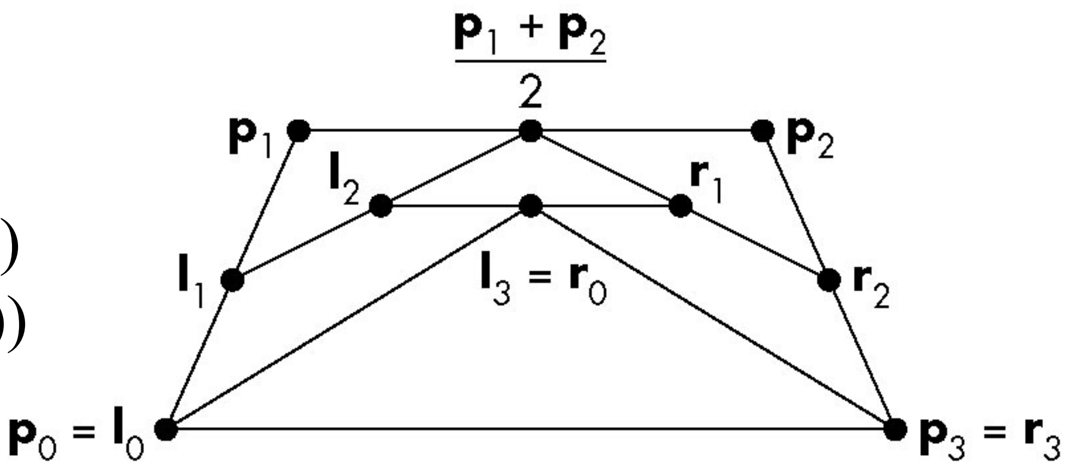
$\{l_0, l_1, l_2, l_3\}$ and $\{r_0, r_1, r_2, r_3\}$ each have a convex hull that that is closer to p(u) than the convex hull of $\{p_0, p_1, p_2, p_3\}$ This is known as the *variation diminishing property*.

The polyline from l_0 to l_3 $(=r_0)$ to r_3 is an approximation to p(u). Repeating recursively we get better approximations.



Efficient Form

$l_0 = p_0$
$r_3 = p_3$
$l_1 = ½(p_0 + p_1)$
$r_1 = ½(p_2 + p_3)$
$l_2 = ½(l_1 + ½(p_1 + p_2))$
$r_1 = ½(r_2 + ½(p_1 + p_2))$
$l_3 = r_0 = ½(l_2 + r_1)$



Requires only shifts and adds!

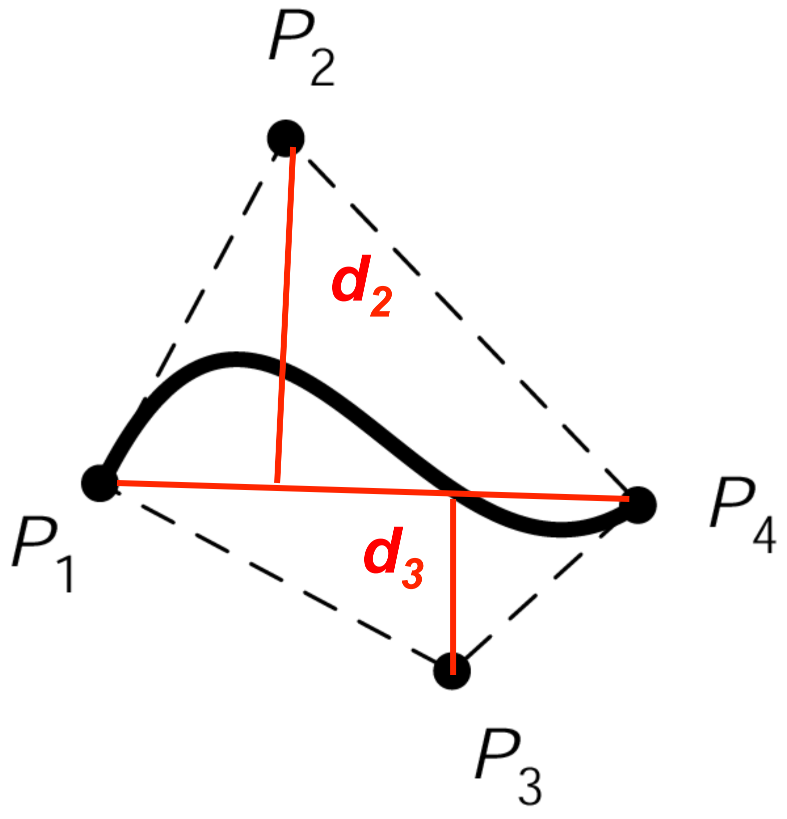
Drawing Parametric Curves

Two basic ways:

- *Iterative evaluation* of $x(t)$, $y(t)$, $z(t)$ for incrementally spaced values of t
  - can't easily control segment lengths and error
- ***Recursive Subdivision***
  via de Casteljau, that stops when control points get sufficiently close to the curve
  - i.e. when the curve is nearly a straight line

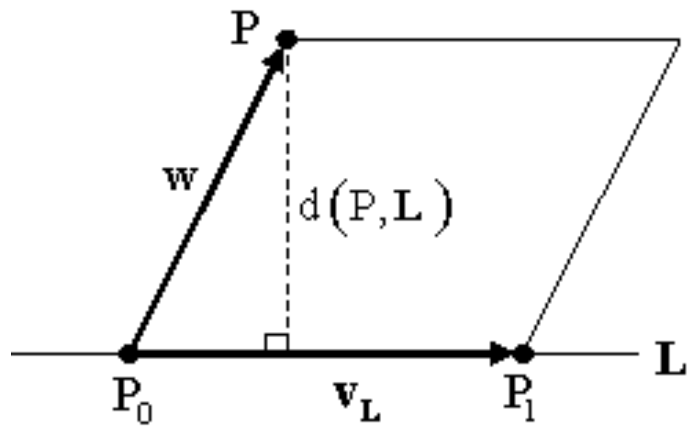
Drawing Parametric Curves via Recursive Subdivision

- Idea: stop subdivision when segment is flat enough to be drawn w/ straight line
- *Curve Flatness Test*:
  - based on the convex hull
  - if $\boldsymbol{d_2}$ and $\boldsymbol{d_3}$ are both less than some ε, then the curve is declared flat



FYI: Computing the Distance from a Point to a Line

- Line is defined with two points
- Basic idea:
  - Project point P onto the line
  - Find the location of the projection



$$\mathrm{d}(P,L) = \frac{(y_0 - y_1)x + (x_1 - x_0)y + (x_0 y_1 - x_1 y_0)}{\sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2}}$$

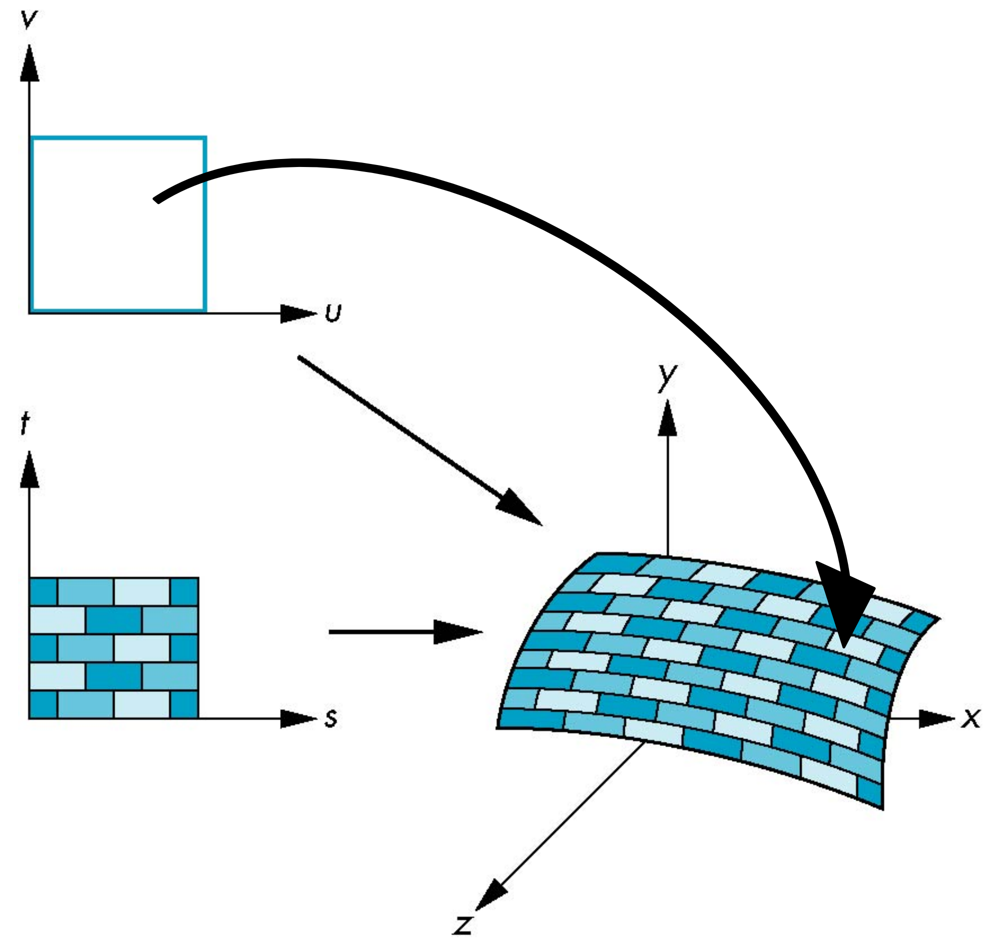
Drawing Parametric Curves via Recursive Subdivision

The Algorithm:

```
• DrawCurveRecSub(curve,e)
    - If straight(curve,e) then DrawLine(curve)
    - Else
        • SubdivideCurve(curve,LeftCurve,RightCurve)
        • DrawCurveRecSub(LeftCurve,e)
        • DrawCurveRecSub(RightCurve,e)
```

Biparametric Patch

- (u,v) pair maps to a 3D point on patch

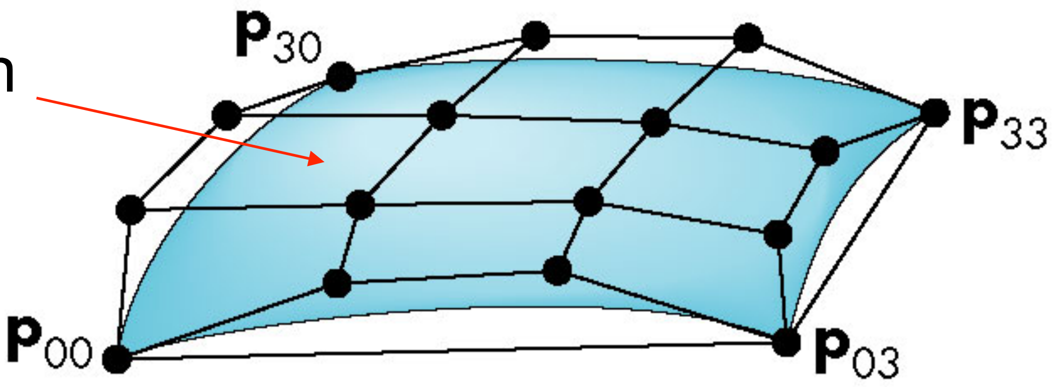


Bezier Patches

Using same data array $\mathbf{P}=[\bar{\mathrm{p}}_{\mathrm{ij}}]$ as with interpolating form

$$\vec{p}(u,v) = \sum_{i=0}^{3}\sum_{j=0}^{3} b_i(u) b_j(v) \vec{p}_{ij} = u^T \mathbf{M}_B \mathbf{P} \mathbf{M}_B^T v$$

Patch lies in convex hull



Bézier Patches

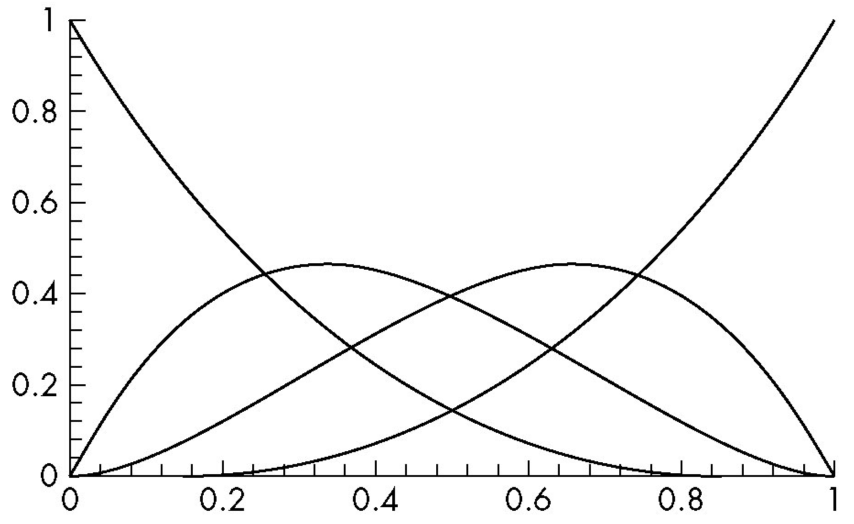
- Expanding the summation

$$\vec{p}(u,v) = \sum_{i=0}^{3}\sum_{j=0}^{3} b_i(u) b_j(v) \vec{p}_{ij} =$$

$$b_0(u)b_0(v)\vec{p}_{00} +$$
$$b_0(u)b_1(v)\vec{p}_{01} +$$
$$b_0(u)b_2(v)\vec{p}_{02} +$$
$$b_0(u)b_3(v)\vec{p}_{03} +$$
$$b_1(u)b_0(v)\vec{p}_{10} +$$
$$etc.$$

Bezier Blending Functions

$$\mathbf{b}(u) = \begin{bmatrix} (1-u)^3 \\ 3u(1-u)^2 \\ 3u^2(1-u) \\ u^3 \end{bmatrix}$$



Note that all zeros are at 0 and 1 which forces the functions to be smooth over (0,1)

Faceting

Faceting