# Introduction to Computer Graphics with WebGL


Ed Angel

Professor Emeritus of Computer Science

Founding Director, Arts, Research, Technology and Science Laboratory

University of New Mexico

The University of New Mexico

# Curves and Surfaces


Ed Angel

Professor Emeritus of Computer Science

University of New Mexico

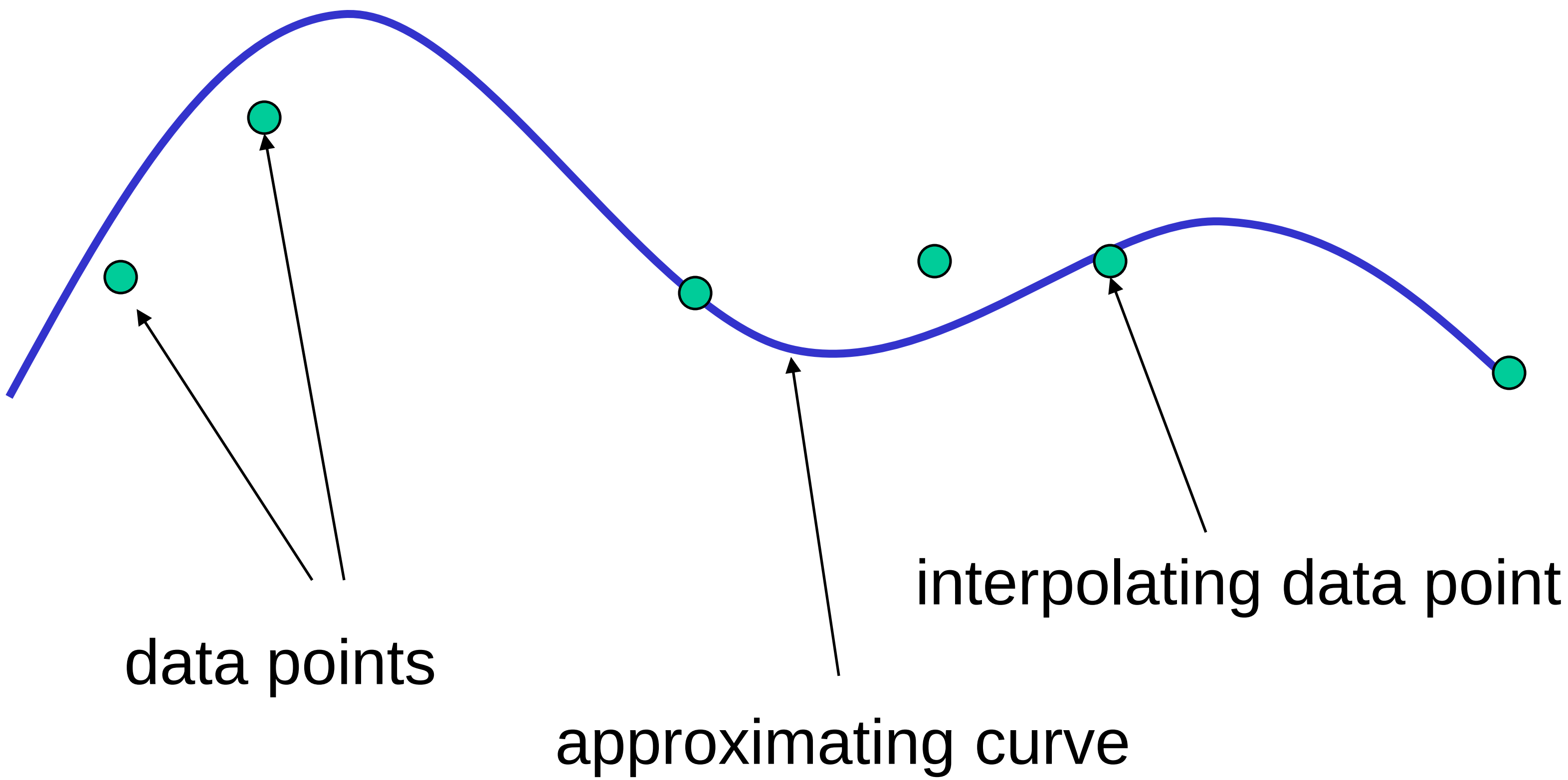
# Objectives

- 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



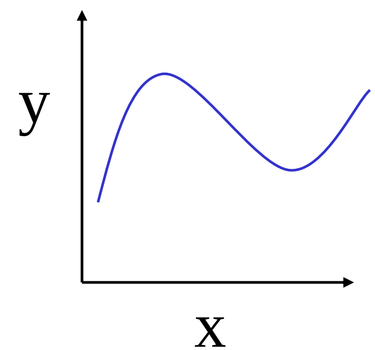
# What Makes a Good Representation?

- 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?

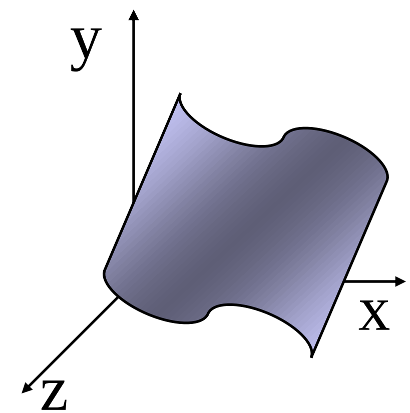
# Explicit Representation

- 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 solve for points that satisfy

# 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

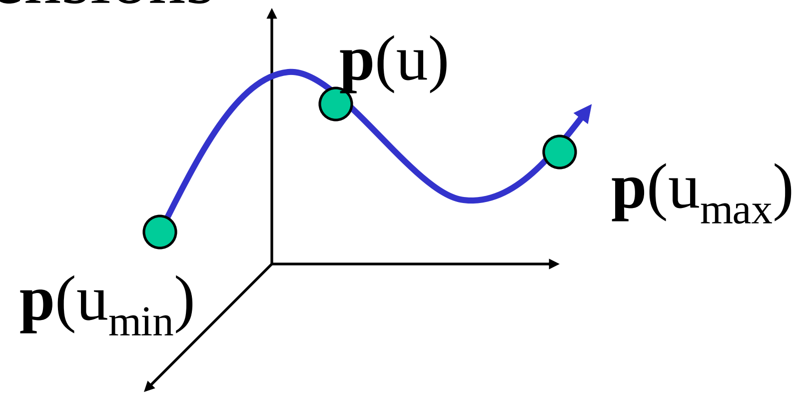
# Parametric Curves

- Separate equation for each spatial variable

$$x=x(u)$$
$$y=y(u)$$
$$z=z(u)$$

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

- 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

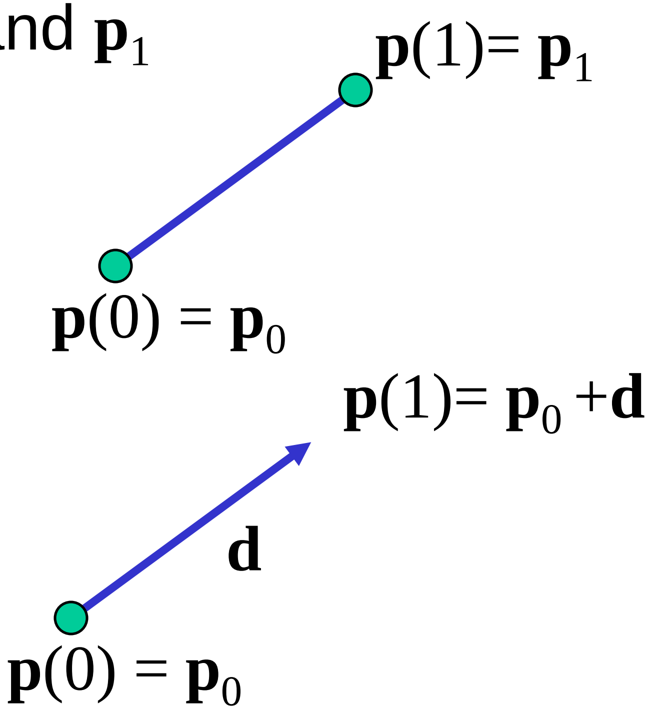
We can normalize u to be over the interval (0,1)

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 $\mathbf{d}$

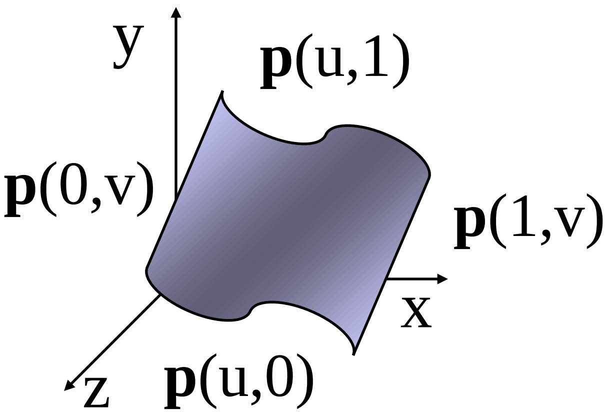
$$\mathbf{p}(u)=\mathbf{p}_0+u\mathbf{d}$$



# 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

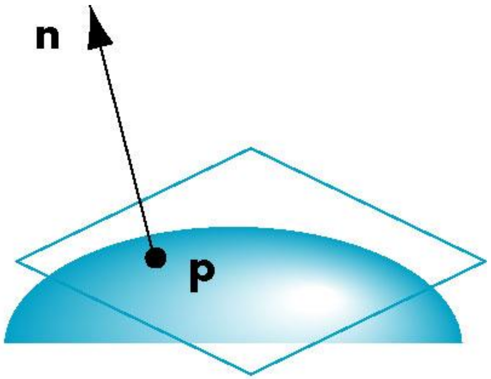
# Normals

We can differentiate with respect to $u$ and $v$ to obtain the normal at any point $\mathbf{p}$

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

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

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



# Parametric Planes

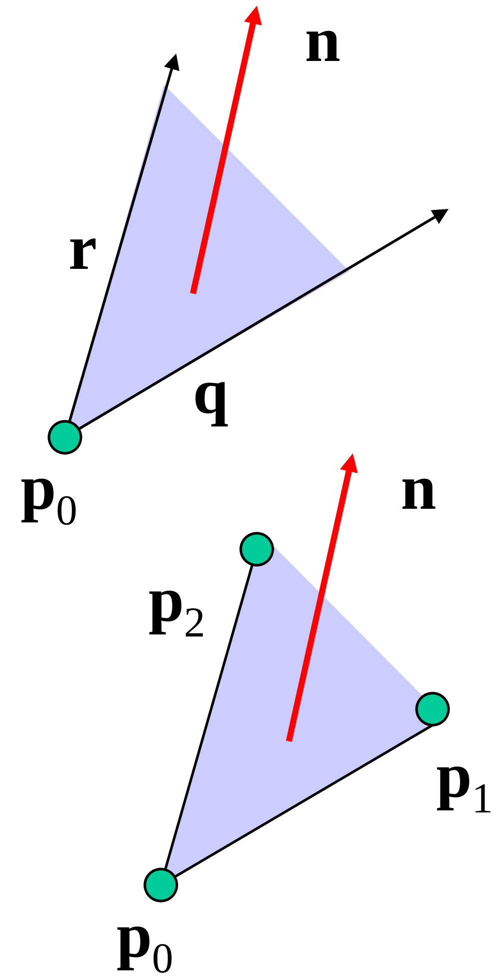
point-vector form

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

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

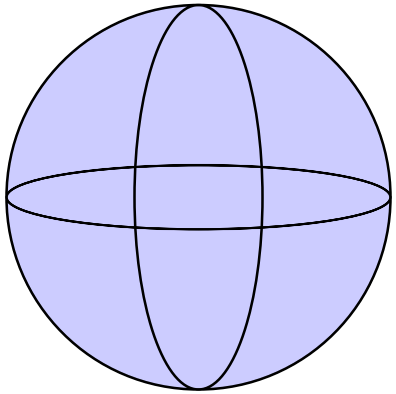
three-point form

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



# Parametric Sphere

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



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

$\theta$ constant: circles of constant longitude
$\phi$ constant: circles of constant latitude

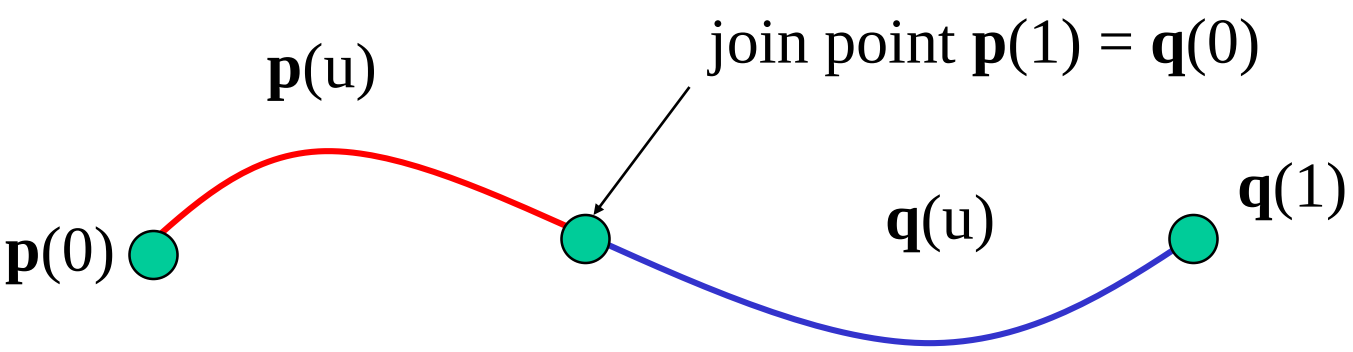
differentiate to show $\mathbf{n} = \mathbf{p}$

# 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

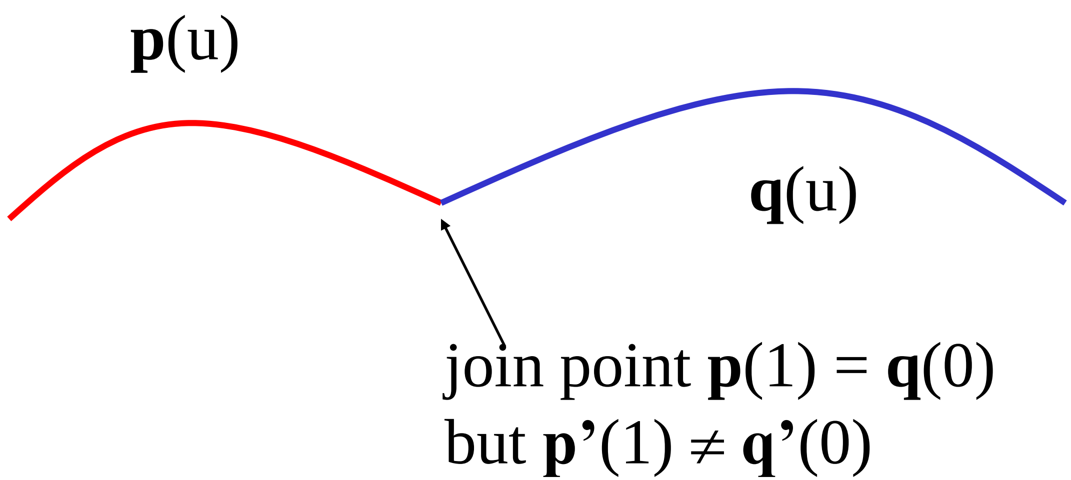
- Equivalently we need 3(N+1) independent conditions

- 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



# Cubic Parametric Polynomials

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

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

- 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

The University of New Mexico

# Introduction to Computer Graphics with WebGL


Ed Angel

Professor Emeritus of Computer Science

Founding Director, Arts, Research, Technology and Science Laboratory

University of New Mexico

# Designing Parametric Cubic Curves

Ed Angel
Professor Emeritus of Computer Science
University of New Mexico

# Objectives

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

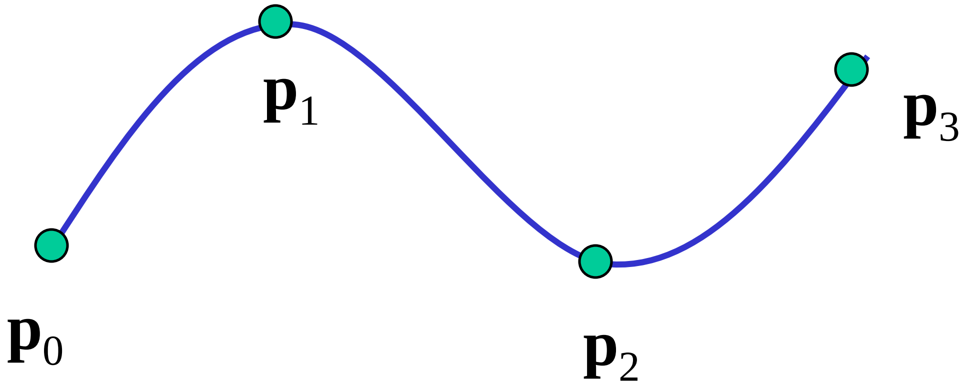
# Matrix-Vector Form

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

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

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

# 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 $u=0, 1/3, 2/3, 1$

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

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

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

# Interpolation Matrix

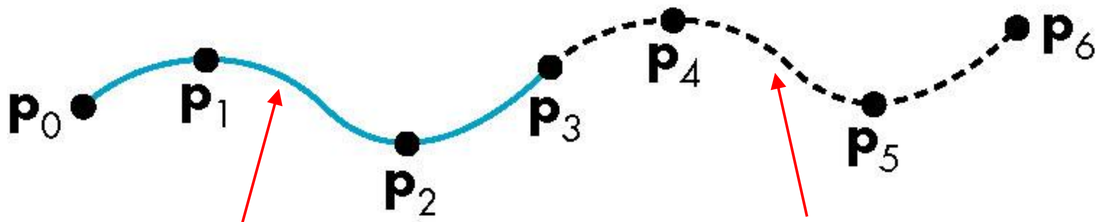
Solving for **c** we find the *interpolation matrix*

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

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

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)

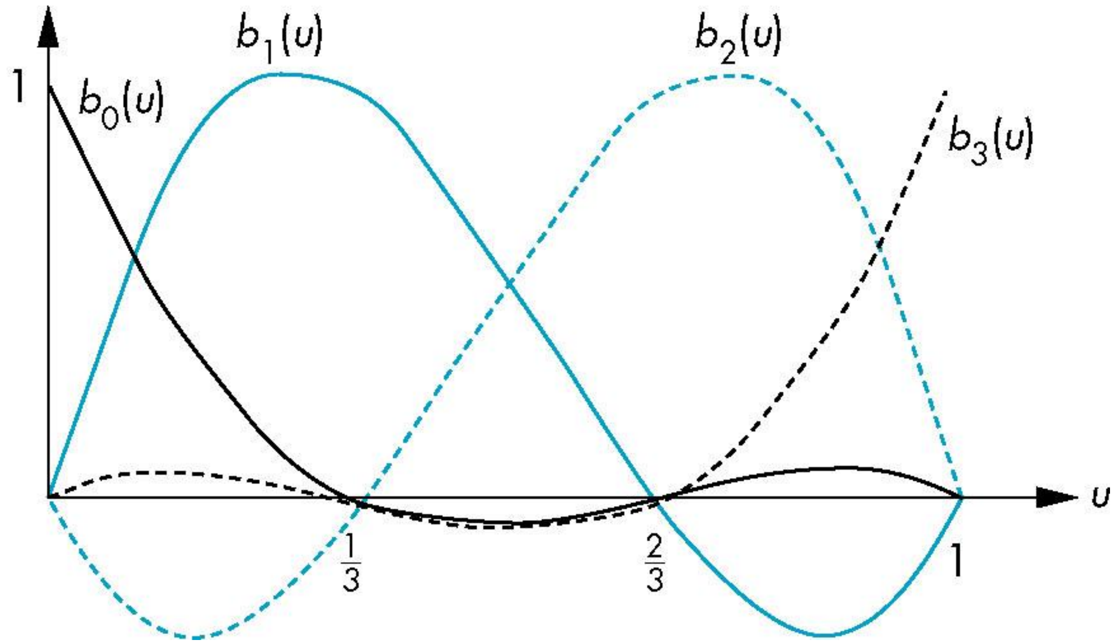
$$p(u)=\mathbf{u}^T\mathbf{c}=\mathbf{u}^T\mathbf{M}_I\mathbf{p} = \mathbf{b}(u)^T\mathbf{p}$$

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

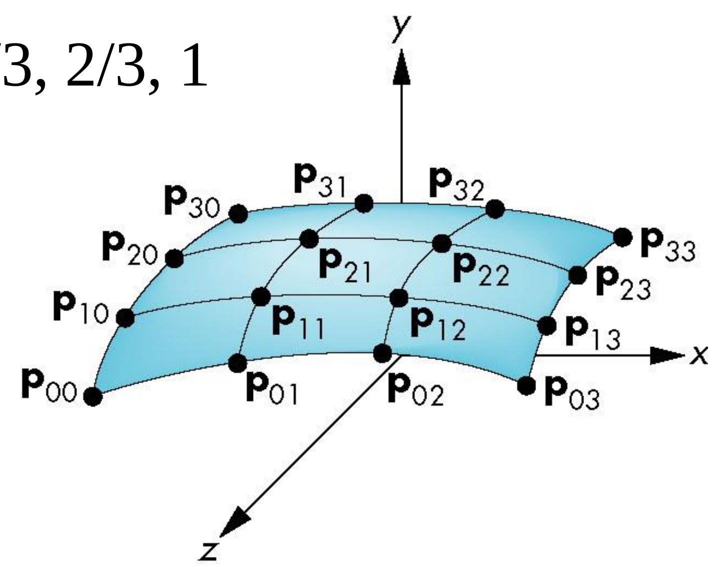


# Interpolating Patch

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

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

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



# Matrix Form

The University of New Mexico

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

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

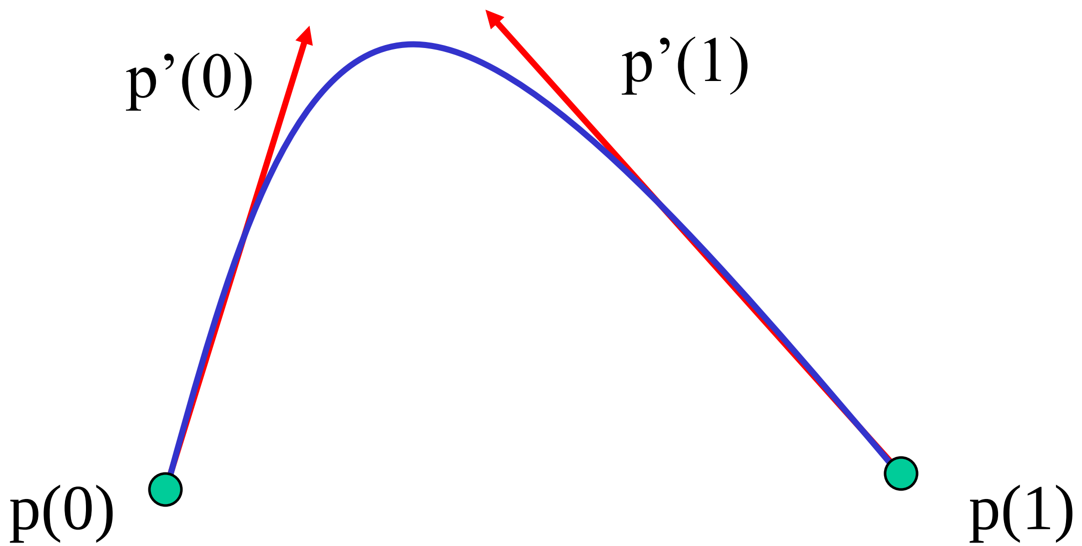
Each $b_i(u)b_j(v)$ is a blending patch

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

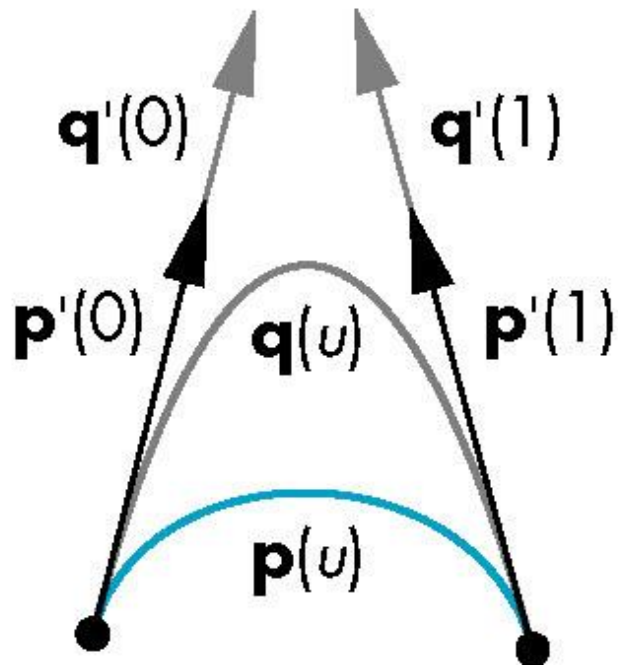
However, the Hermite form is the basis of the Bezier form

# 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

# 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

# Introduction to Computer Graphics with WebGL

Ed Angel

Professor Emeritus of Computer Science

Founding Director, Arts, Research, Technology and Science Laboratory

University of New Mexico

# Bezier and Spline Curves and Surfaces

Ed Angel

Professor Emeritus of Computer Science

University of New Mexico

# Objectives

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

The University of New Mexico

# 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

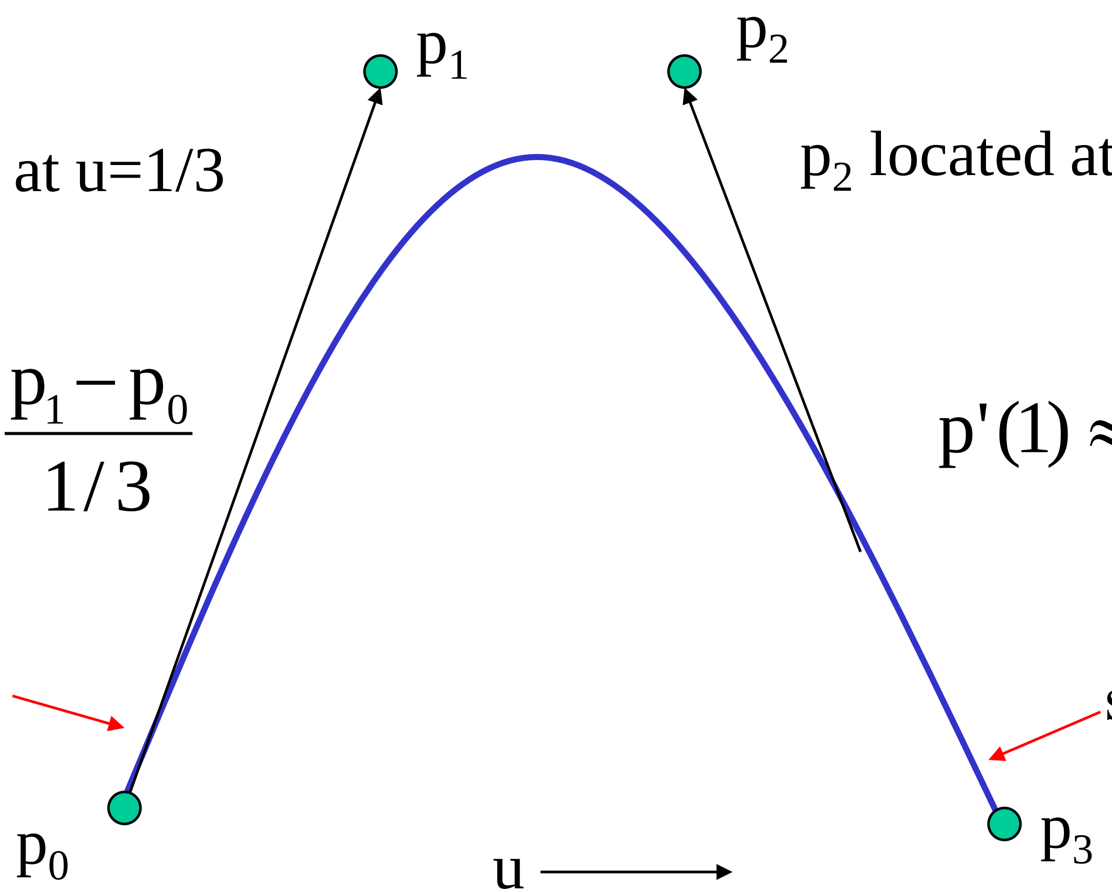
# Approximating Derivatives

$p_1$ located at u=1/3

$p_2$ located at u=2/3

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

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




slope p’(0)

slope p’(1)

# 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

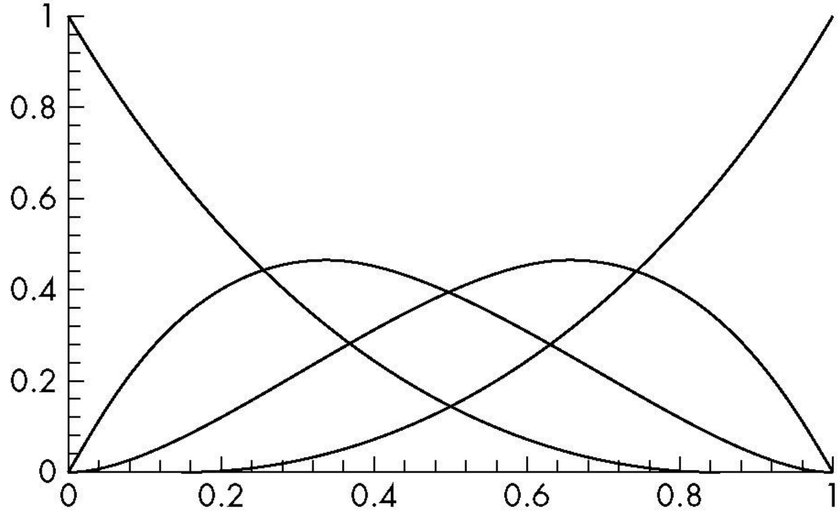
$$\mathbf{M}_B = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -3 & 3 & 0 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{bmatrix}$$

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

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)

# Bernstein Polynomials

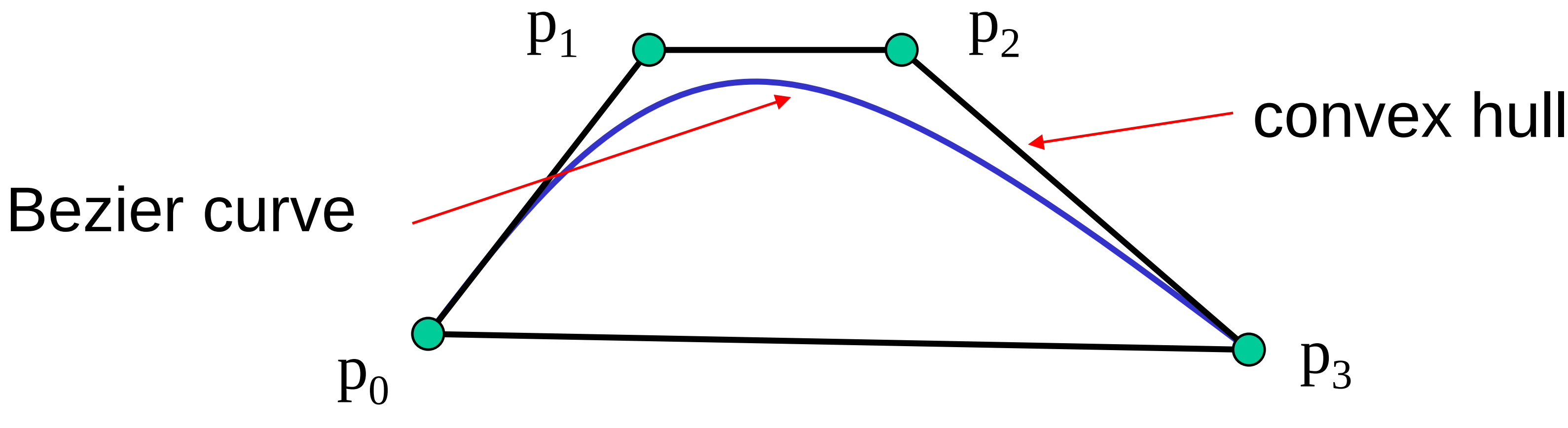
- The blending functions are a special case of the Bernstein polynomials

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

- 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

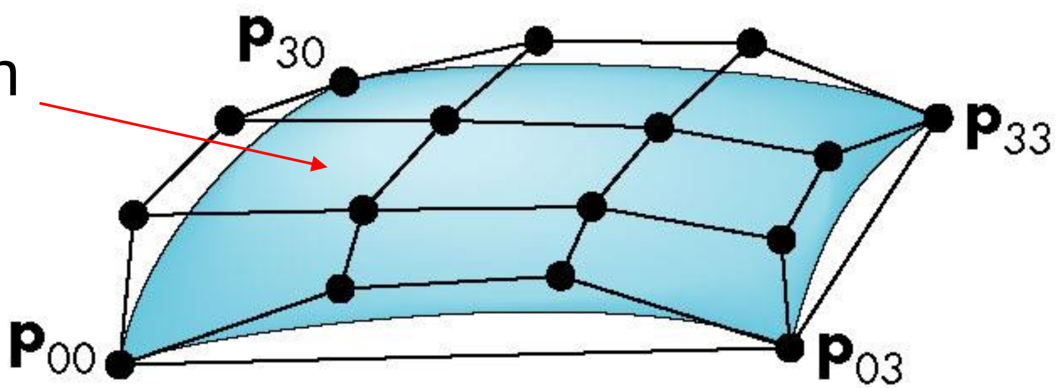


# Bezier Patches

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



# Analysis

- Although the Bezier form is much better than the interpolating form, we have the 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 fixed function OpenGL
  - Apply different conditions
    - Tricky without letting order increase

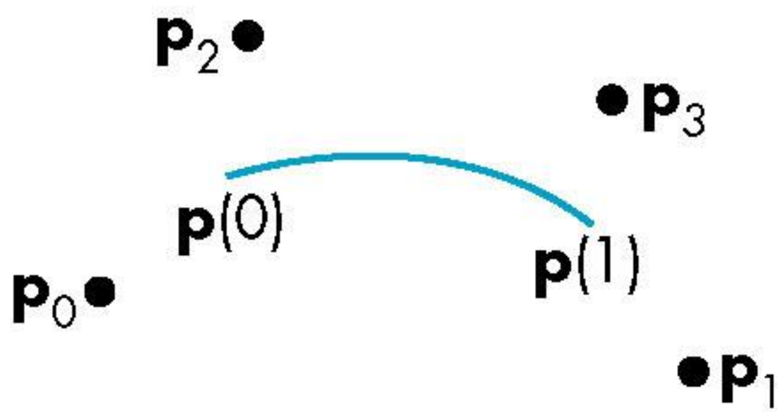
# B-Splines

- 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

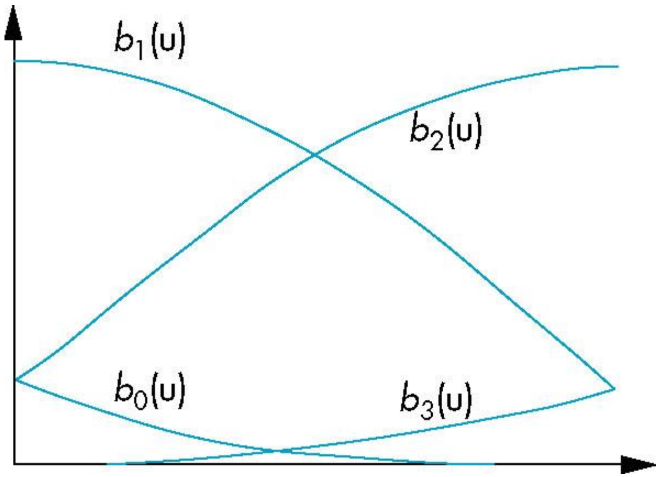
$$p(u) = \mathbf{u}^T\mathbf{M}_S\mathbf{p} = \mathbf{b}(u)^T\mathbf{p}$$

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

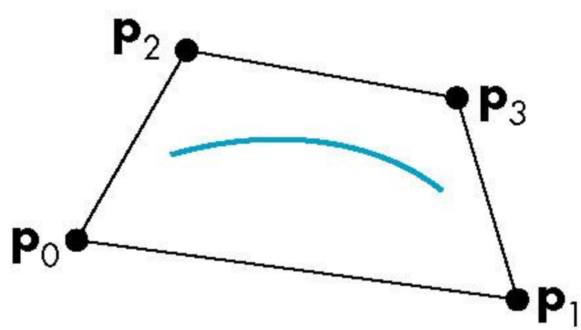


# Blending Functions

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



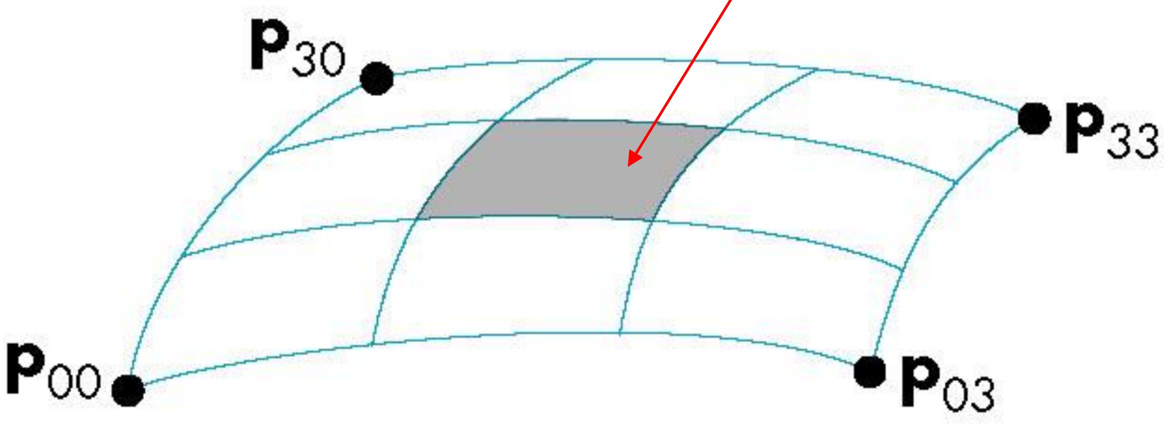

convex hull property



# B-Spline Patches

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

defined over only 1/9 of region



# 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

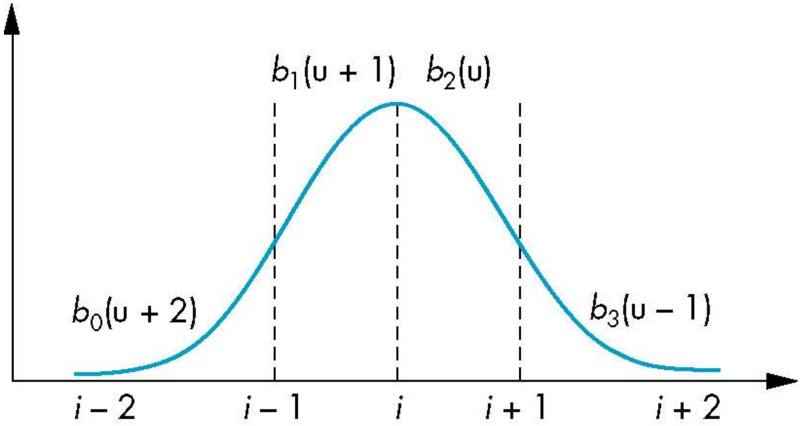
$$p(u) = \sum B_i(u)\, p_i$$

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

- Nonuniform Rational B-Spline 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