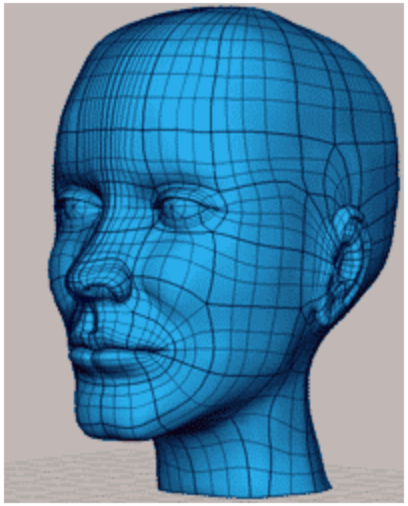
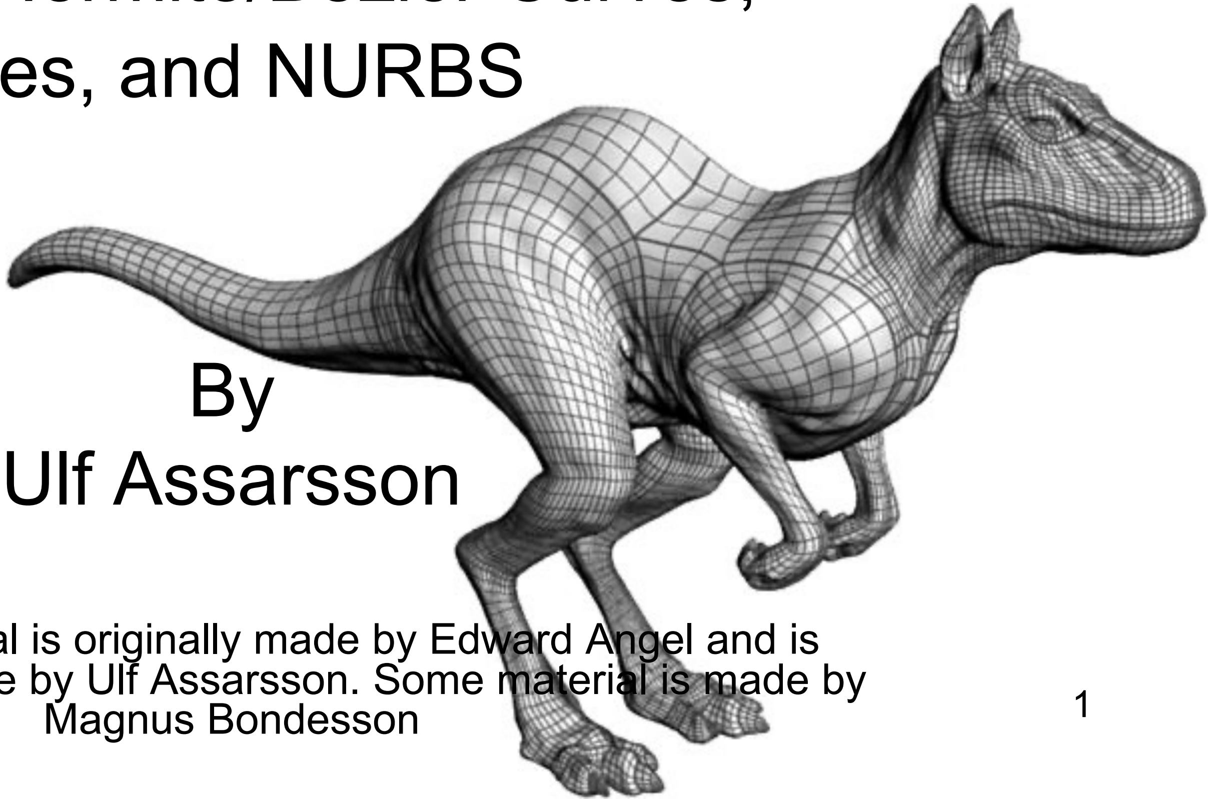


Computer Graphics

Curves and Surfaces

Interpolating/Hermite/Bezier Curves, B-Splines, and NURBS

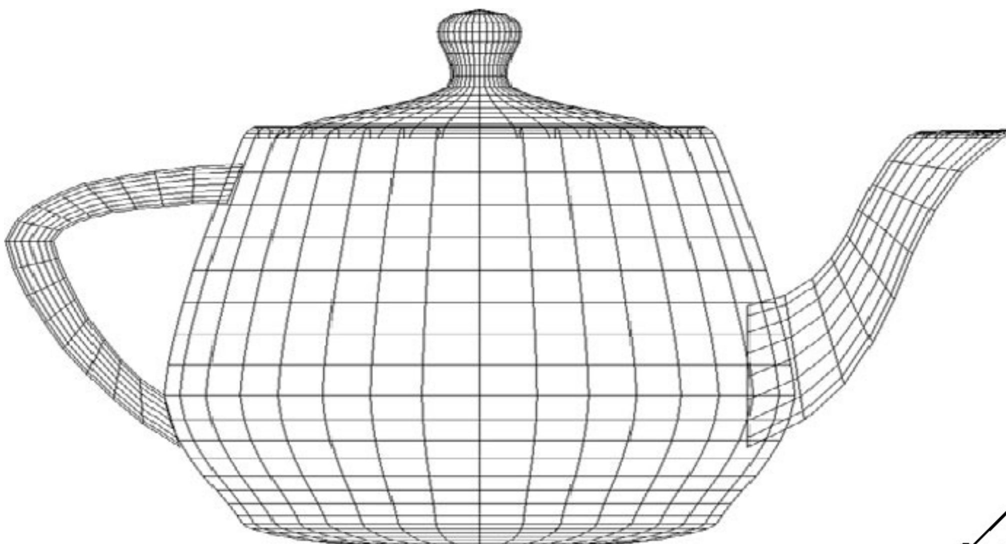


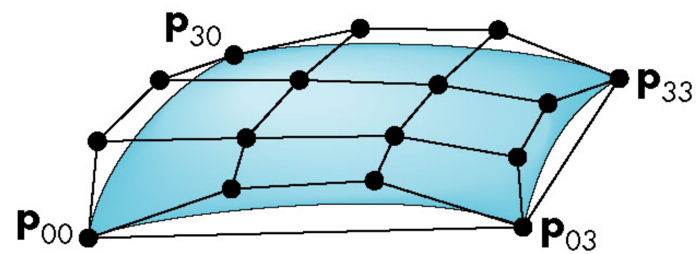
By
Ulf Assarsson

Most of the material is originally made by Edward Angel and is adapted to this course by Ulf Assarsson. Some material is made by Magnus Bondesson

Utah Teapot

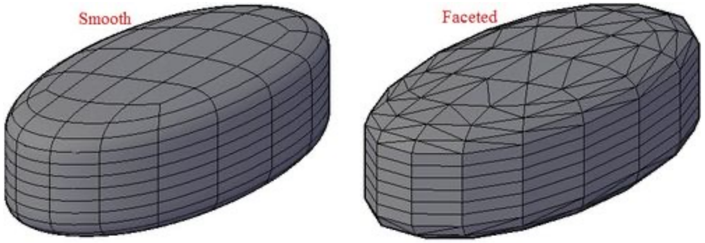
- Most famous data set in computer graphics
- Widely available as a list of 306 3D vertices and the indices that define 32 Bezier patches



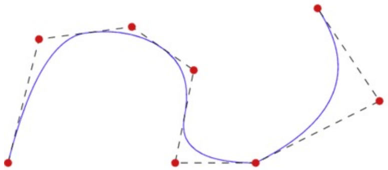


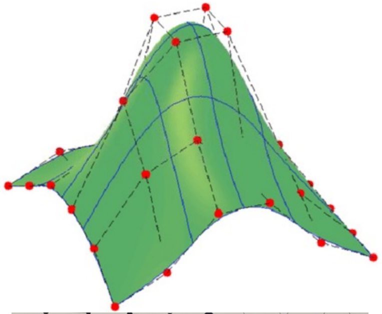
A Bezier patch

Curves and Curved Surfaces



- Reason: may want
  - smooth shapes from few control points.
  - Infinite resolutions (e.g., in movie rendering). No discretization.
- Vast topic, e.g.,
  - Bezier patches:
    - can describe all *polynomial* surfaces
      - (quadratic, cubic, quartic, quintic,…).
  - NURBS
    - standard for CAD, more flexibility.
    - Not in course book (Real-Time Rendering)
  - Subdivision surfaces:
    - Good for smoothing arbitrary triangle meshes
    - Popular in rendering
    - E.g., Loop subdivision, Catmull-Clark subdivision, …
    - Often easier to grasp on your own, compared to NURBS.

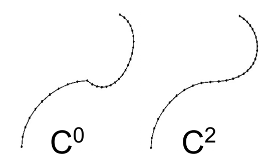




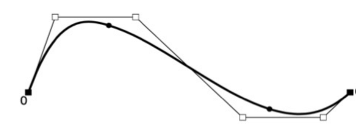
Outline

Goal is to explain NURBS curves/surfaces…

- Introduce types of curves and surfaces
  - *Explicit* – not general, easy to compute.
  - *Implicit* – general, non-easy to compute.
  - *Parametric* - general **and** easy to compute. We choose this.
- A complete curve is split into curve segments, each defined by a polynomial (per x,y,z coordinate), e.g., *cubic polynomials*.
  - Introducing *Interpolating/Hermite/Bezie*r curves.

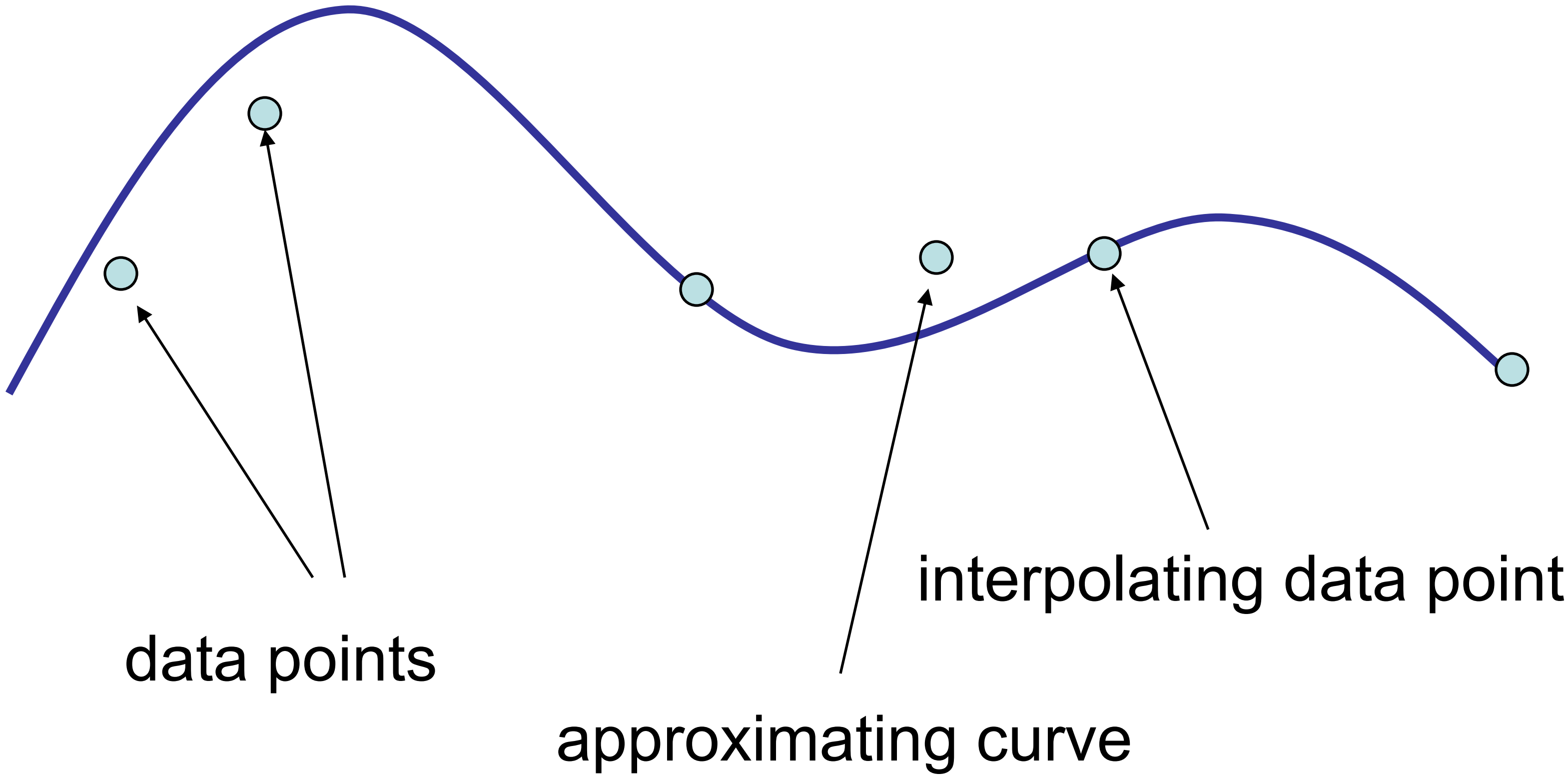


- Adjacent segments should preferably have C^2 continuity:
  - Leads to *B-Splines* with a blending function (a spline) per control point
    - Each spline consists of 4 cubical polynomials, forming a bell shape translated along *u*.
    - (Also, four bells will overlap at each point on the complete curve.)



- NURBS – a generalization of B-Splines:
  - Control points at non-uniform locations along parameter *u*.
  - Individual weights (i.e., importance) per control point
  - popular in CAD systems

Modeling with Curves



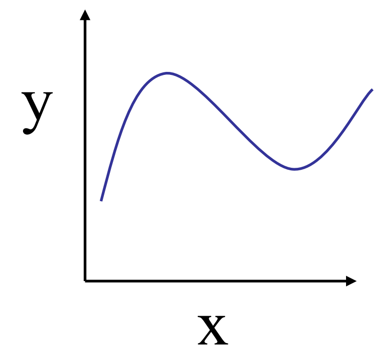
What Makes a Good Representation?

- 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?

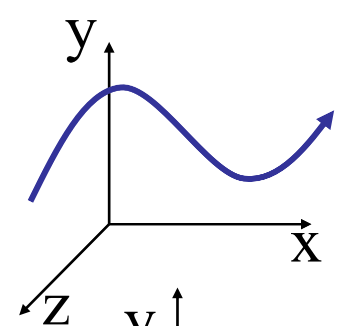
Explicit Representation

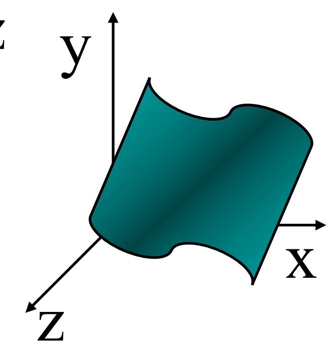
- Most familiar form of curve in 2D

$$y=f(x)$$



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



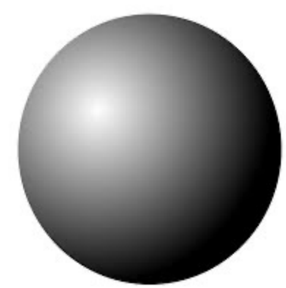


Implicit Representation

- Two dimensional curve(s)

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

- 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



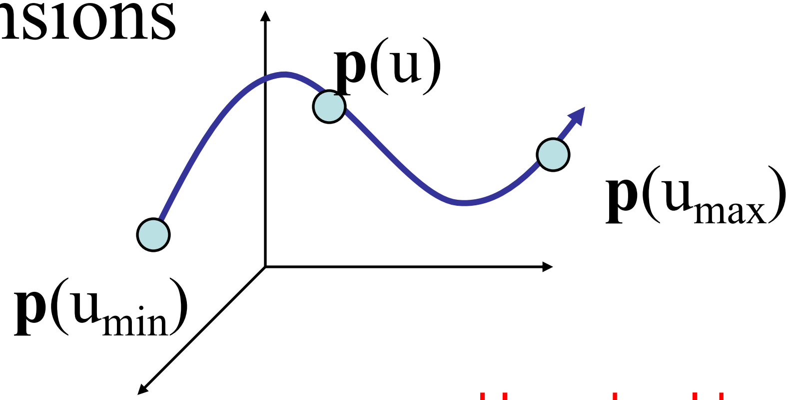
Parametric Curves

- Separate equation for each spatial variable

$$x = \mathrm{x}(\mathrm{u})$$
$$y = \mathrm{y}(\mathrm{u})$$
$$z = \mathrm{z}(\mathrm{u})$$

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

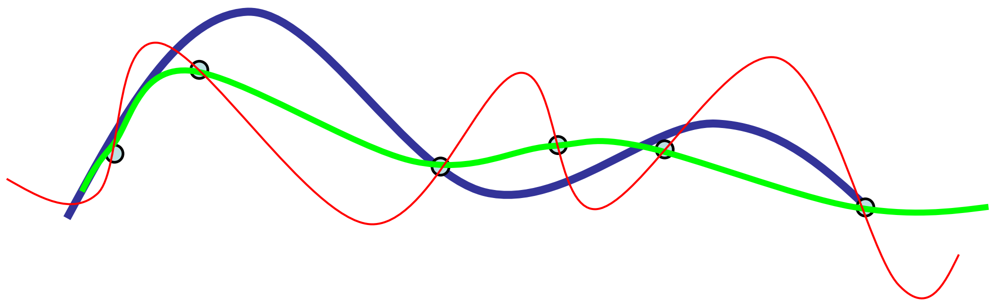
- For $u_{min} \le u \le u_{max}$ we trace out a curve in two or three dimensions



How should we create the parametric functions x(u), y(u), z(u)?

We create the curves from user-given control points. But how choose a curve for these points?:

Selecting Functions



- Usually we can select many "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

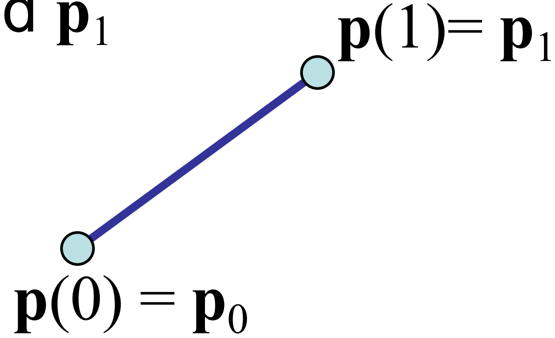
Lines are easy…

Parametric Lines

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

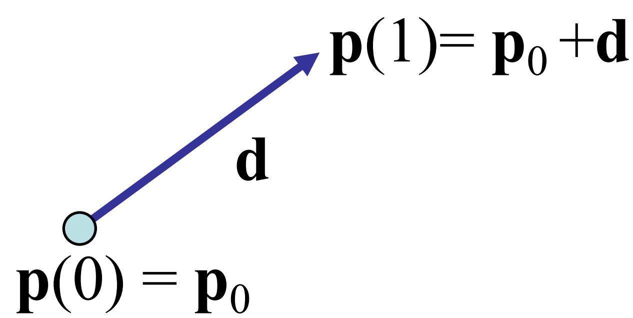
Line connecting two points $\mathbf{p}_0$ and $\mathbf{p}_1$

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



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

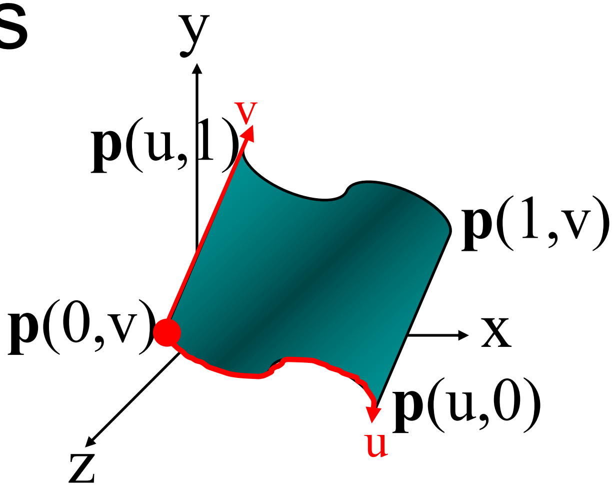
$$\mathbf{p}(u)=\mathbf{p}_0+u\mathbf{d}$$



Parametric Surfaces

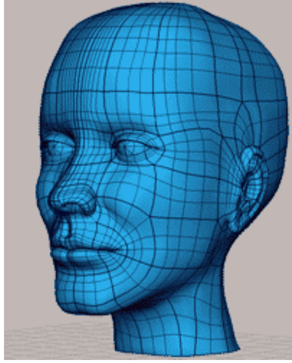
- Surfaces require 2 parameters

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



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

If we have the x(u,v), y(u,v), z(u,v) functions, we can compute the normal at any point *u,v*.

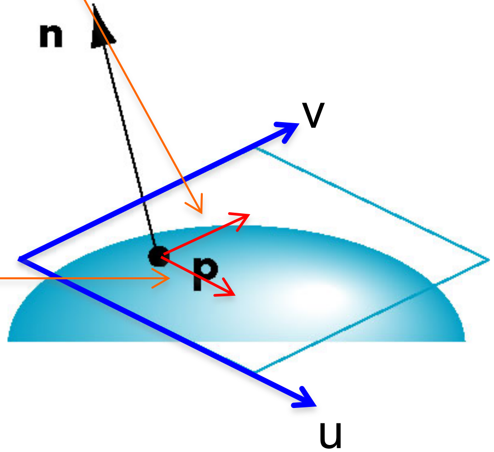


Normals

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

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

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



Parametric Planes

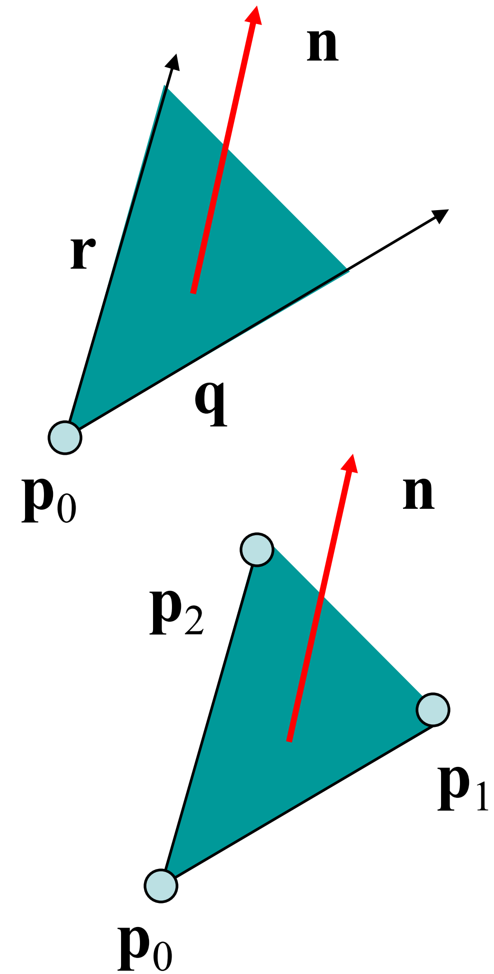
point-vector form

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

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

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

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



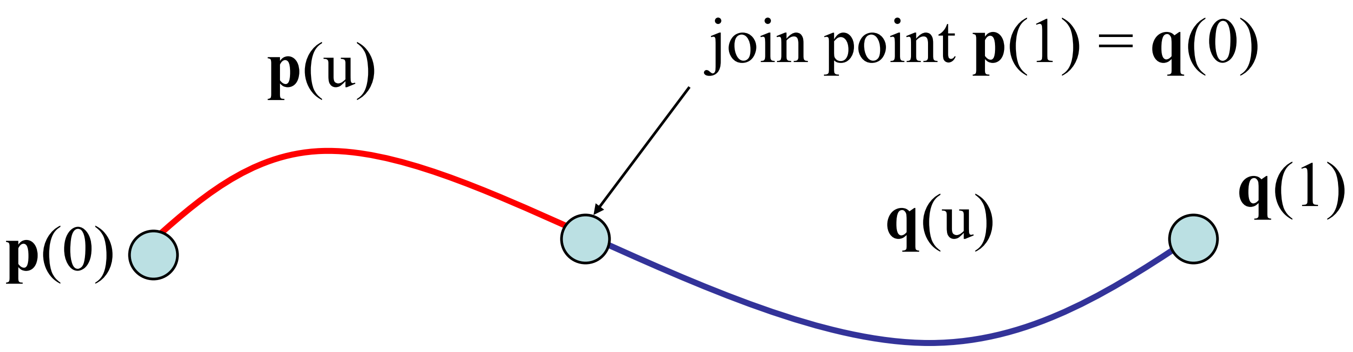
We create the curves from user-given control points by using curve segments…

Curve Segments

- We can normalize u, so each curve is written

  $\mathbf{p}(u)=[x(u), y(u), z(u)]^T, \quad 0 \le u \le 1$

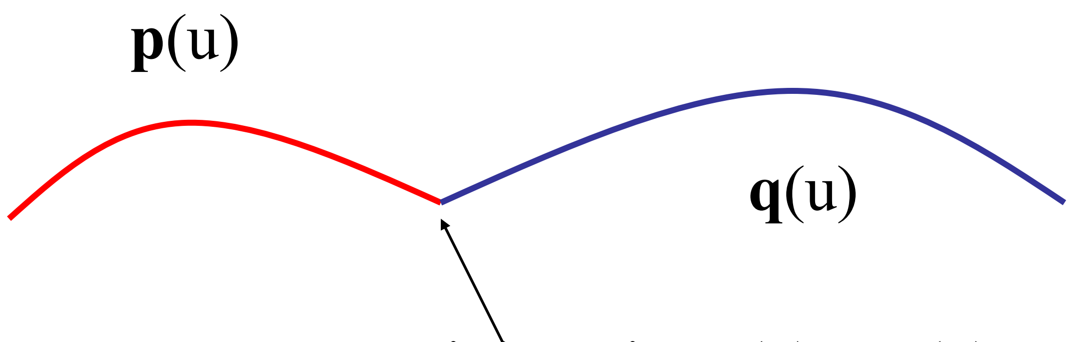
- In classical numerical methods, we design a single global curve.
- In computer graphics and CAD, it is better to design small connected curve *segments*



How should we describe curve segments?

We choose Polynomials

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



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

Let's worry about that later. First let's scrutinize the polynomials!

Parametric Polynomial Curves

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

- Cubic polynomials gives N=M=L=3

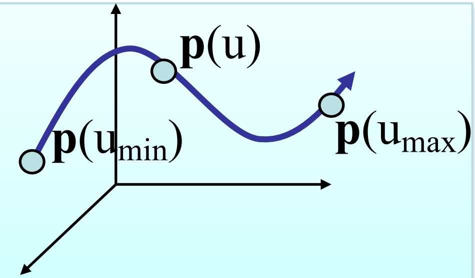
$p(u) = c_0 + c_1 u + c_2 u^2 + c_3 u^3$

Remember:

$x=x(u)$

$y=y(u)$

$z=z(u)$



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

where p is any of x, y, z. It is just the numerical c_k values that differ.

Let's assume cubic polynomials!

Cubic Parametric Polynomials

Linear. Quadratic. Cubic. Quartic.

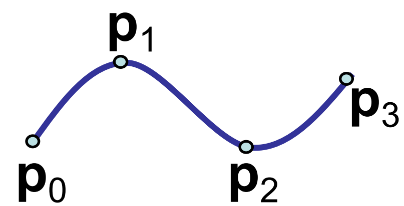
- Cubic polynomials give balance between ease of evaluation and flexibility in design

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

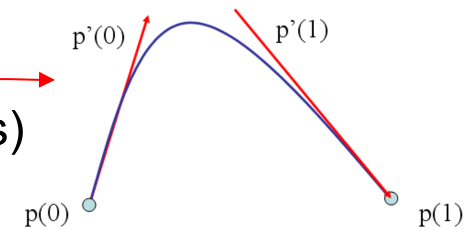
$p(u) = c_0 + c_1u + c_2u^2 + c_3u^3$

- 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

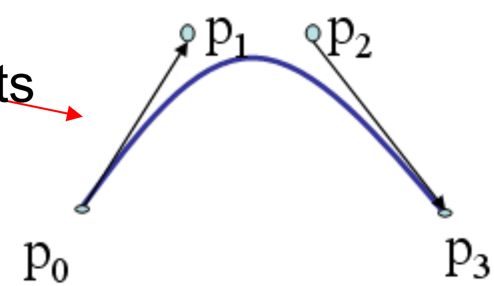
Some Types of Curves



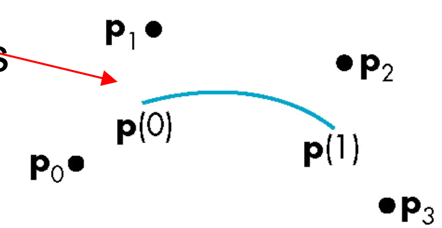
- Introduce the types of curves
  - Interpolating
    - Blending polynomials for interpolation of 4 control points (fit curve to 4 control points)
  - Hermite



    - fit curve to 2 control points + 2 derivatives (tangents)
  - Bezier
    - 2 interpolating control points + 2 intermediate points to define the tangents



  - B-spline – use points of adjacent curve segments
    - To get C^1 and C^2 continuity
  - NURBS
    - Different weights of the control points
    - The control points can be at non-uniform *u,v* intervalls



- Analyze them

Matrix-Vector Form

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

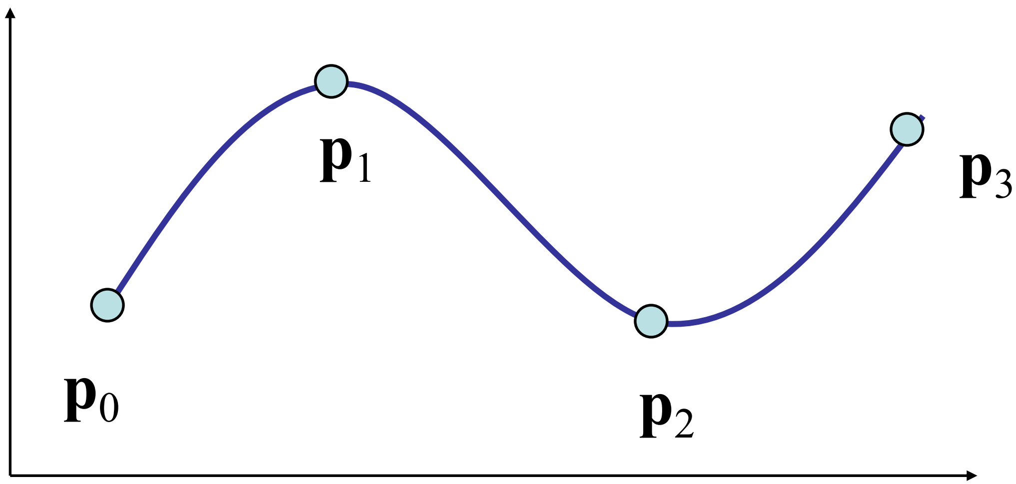
p(u) = $c_0 + c_1u + c_2u^2 + c_3u^3$

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

$$[1 \; u \; u^2 u^3] \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} = [c_0 \; c_1 \; c_2 \; c_3] \begin{bmatrix} 1 \\ u \\ u^2 \\ u^3 \end{bmatrix}$$

Interpolating Curve

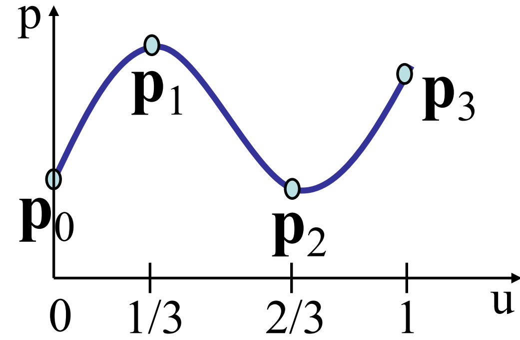


Given four data (control) points $\mathbf{p}_0$, $\mathbf{p}_1$,$\mathbf{p}_2$, $\mathbf{p}_3$ determine cubic $\mathbf{p}(u)$ which passes through them

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

Let's create an equation system!

Interpolation Equations



$p(u) = c_0 + c_1u + c_2u^2 + c_3u^3$

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

$$\begin{cases} p(0) = p_0 = c_0 \\ p(1/3) = p_1 = c_0+(1/3)c_1+(1/3)^2c_2+(1/3)^3c_3 \\ p(2/3) = p_2 = c_0+(2/3)c_1+(2/3)^2c_2+(2/3)^3c_3 \\ p(1) = p_3 = c_0+c_1+c_2+c_3 \end{cases}$$

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

$$\mathbf{p} = \begin{bmatrix} p_0 \\ p_1 \\ p_2 \\ p_3 \end{bmatrix} = \mathbf{Ac} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & \left(\frac{1}{3}\right) & \left(\frac{1}{3}\right)^2 & \left(\frac{1}{3}\right)^3 \\ 1 & \left(\frac{2}{3}\right) & \left(\frac{2}{3}\right)^2 & \left(\frac{2}{3}\right)^3 \\ 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

$\mathbf{p=Ac}$

I.e., $\mathbf{c=A^{-1}p}$

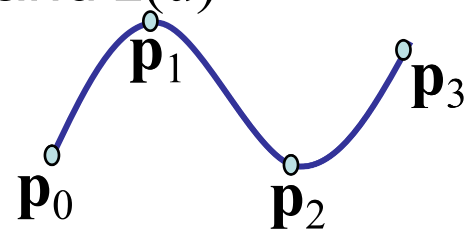
Interpolation Matrix

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

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

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

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



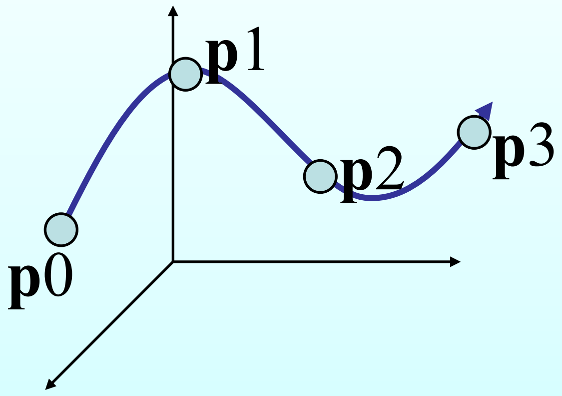
$p(u) = c_0 + c_1 u + c_2 u^2 + c_3 u^3$

Interpolation Matrix

$p(u) = c_0 + c_1 u + c_2 u^2 + c_3 u^3$ means:

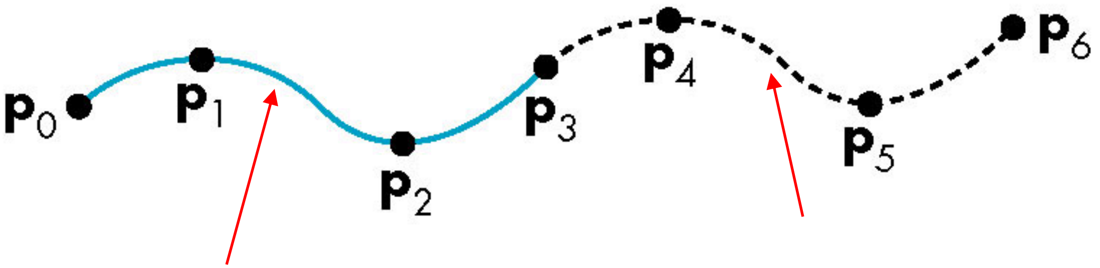
$x = x(u) = c_{x0} + c_{x1} u + c_{x2} u^2 + c_{x3} u^3$
$y = y(u) = c_{y0} + c_{y1} u + c_{y2} u^2 + c_{y3} u^3$
$z = z(u) = c_{z0} + c_{z1} u + c_{z2} u^2 + c_{z3} u^3$

where
$\mathbf{c}_x = M_I \, \mathbf{p}_x$
$\mathbf{c}_y = M_I \, \mathbf{p}_y$
$\mathbf{c}_z = M_I \, \mathbf{p}_z$



$\mathbf{p}_x$ are the x coordinates of $p_0 \ldots p_3$
$\mathbf{p}_y$ are the y coordinates of $p_0 \ldots p_3$
$\mathbf{p}_z$ are the z coordinates of $p_0 \ldots p_3$

Interpolating Multiple Segments



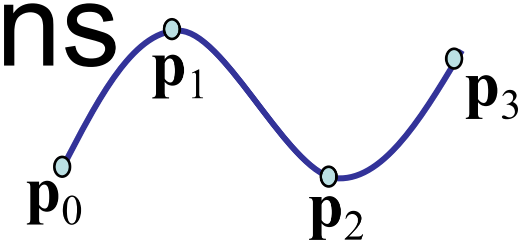
use $\mathbf{p} = [p_0\ p_1\ p_2\ p_3]^T$

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

We have continuity of the curve at the join points but not continuity of the curve's derivatives. I.e., curve is not smooth.

Let's ignore that a few more slides...

Blending Functions



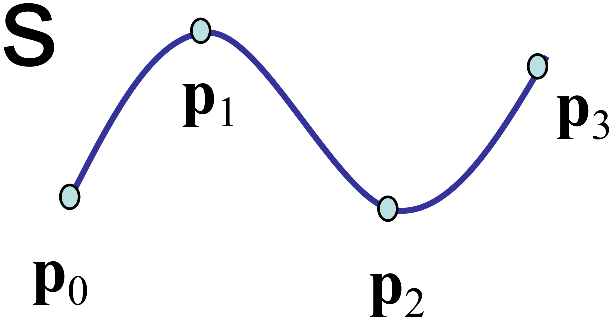
Rewriting the equation for p(u)

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

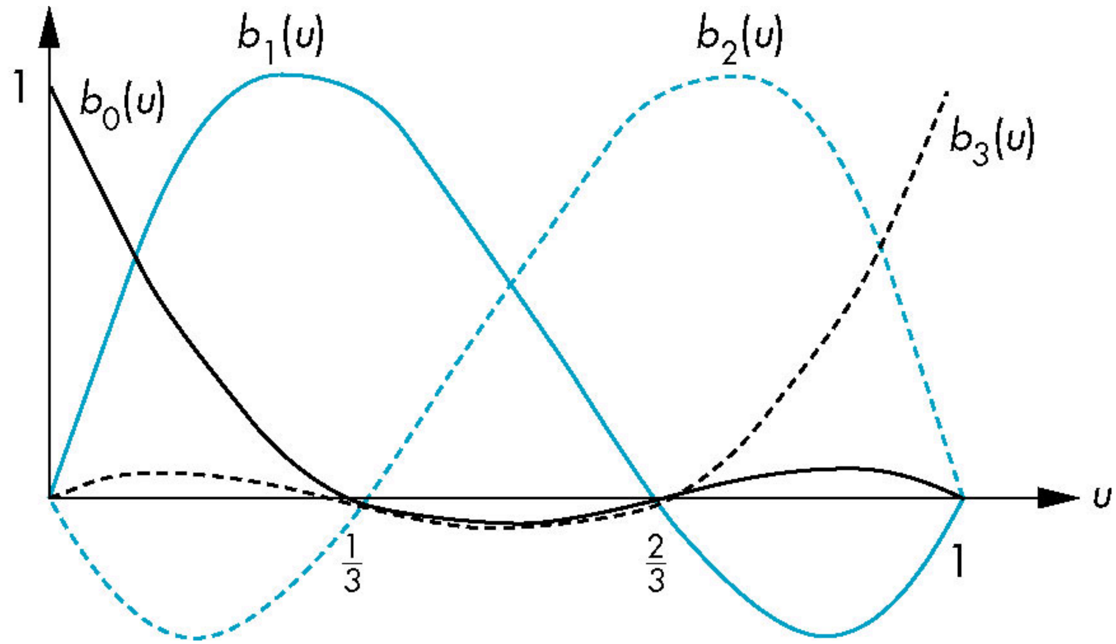
where $b(u) = [b_0(u)\ b_1(u)\ b_2(u)\ b_3(u)]^T$ is an array of *blending polynomials* such that $p(u) = b_0(u)p_0 + b_1(u)p_1 + b_2(u)p_2 + b_3(u)p_3$

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

Blending Functions

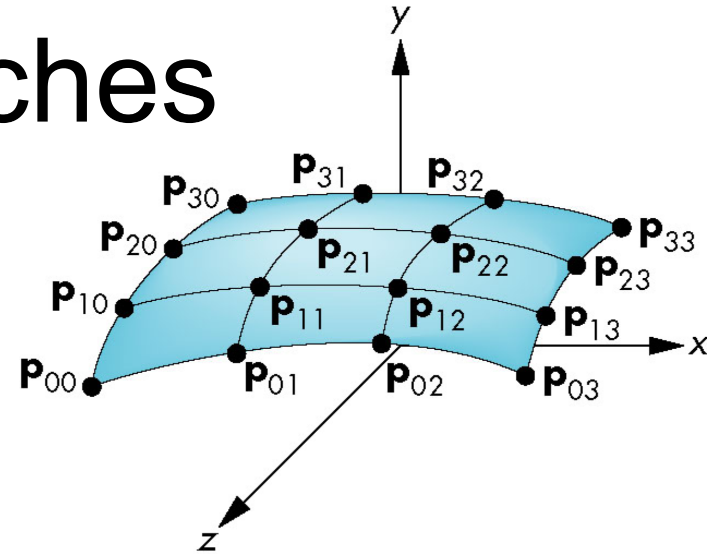


"Weight curves for each control point **p** at a certain u"



$$p(u) = b_0(u)p_0 + b_1(u)p_1 + b_2(u)p_2 + b_3(u)p_3$$

Blending Patches



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

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

$$\mathbf{P} = \begin{bmatrix} p_{00} & p_{01} & p_{02} & p_{03} \\ p_{10} & p_{11} & p_{12} & p_{13} \\ p_{20} & p_{21} & p_{22} & p_{23} \\ p_{30} & p_{31} & p_{32} & p_{33} \end{bmatrix}$$

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

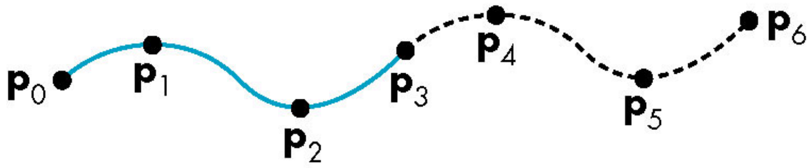
Shows that we can build and analyze surfaces from our knowledge of curves.

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

Patch: $p(u,v) = \mathbf{u}^T\mathbf{C}\,\mathbf{v} = \mathbf{u}^T\mathbf{M}_I\,\mathbf{P}\,\mathbf{M}_I^T\,\mathbf{v} = \mathbf{b}(u)^T\,\mathbf{P}\,\mathbf{b}(v)^T$

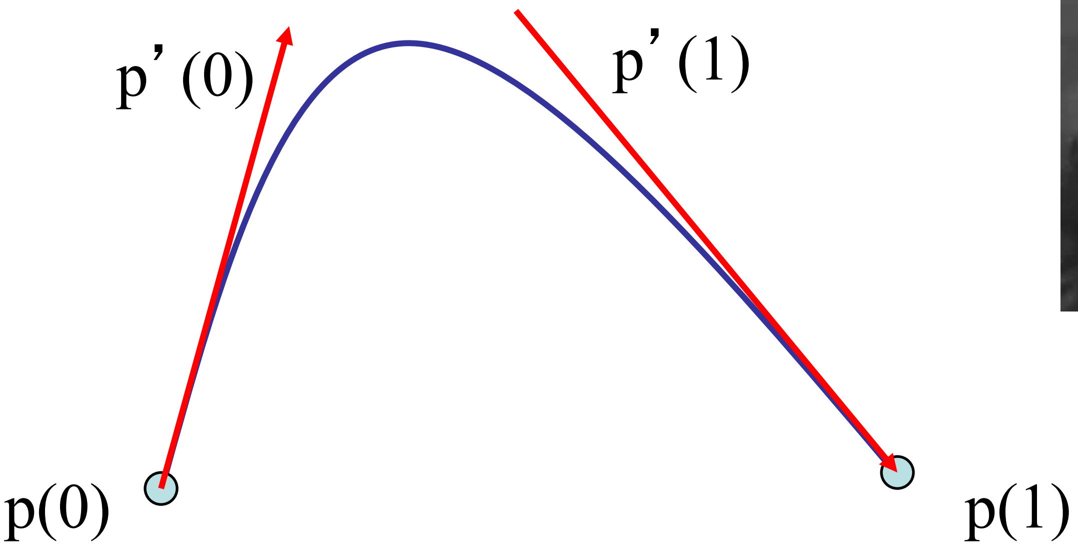
Hermite Curves and Surfaces

- Our interpolating curves have discontinuities between curve segments
  - –Discontinuous derivatives at join points:



- Hermite curves solves this…

Hermite Form

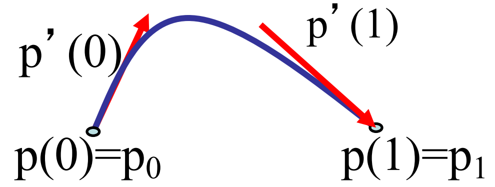


Charles Hermite, 1822-1901

Use two interpolating conditions and two derivative conditions per segment

Ensures continuity and first derivative continuity between segments

Equations



$p(u) = c_0+uc_1+u^2c_2+u^3c_3$

Interpolating conditions are the same at ends

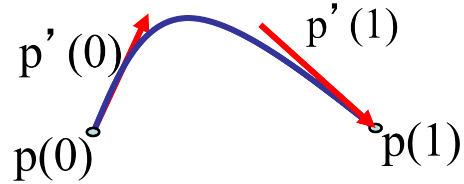
$$p(0) = p_0 = c_0$$
$$p(1) = p_1 = c_0+c_1+c_2+c_3$$

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

Evaluating at end points

$$p'(0) = p'_0 = c_1$$
$$p'(1) = p'_1 = c_1+2c_2+3c_3$$

$$\mathbf{q} = \begin{bmatrix} p_0 \\ p_1 \\ p'_0 \\ p'_1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 2 & 3 \end{bmatrix}\mathbf{c}$$



Matrix Form

$$\mathbf{q} = \begin{bmatrix} p_0 \\ p_1 \\ p'_0 \\ p'_1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 2 & 3 \end{bmatrix} \mathbf{c}$$

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

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

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

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

Blending Polynomials

$\mathrm{p(u)} = \mathrm{u}^T\mathbf{M}_H\mathbf{q}$ => $\mathrm{p(u)} = \mathbf{b(u)}^T\mathbf{q}$

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

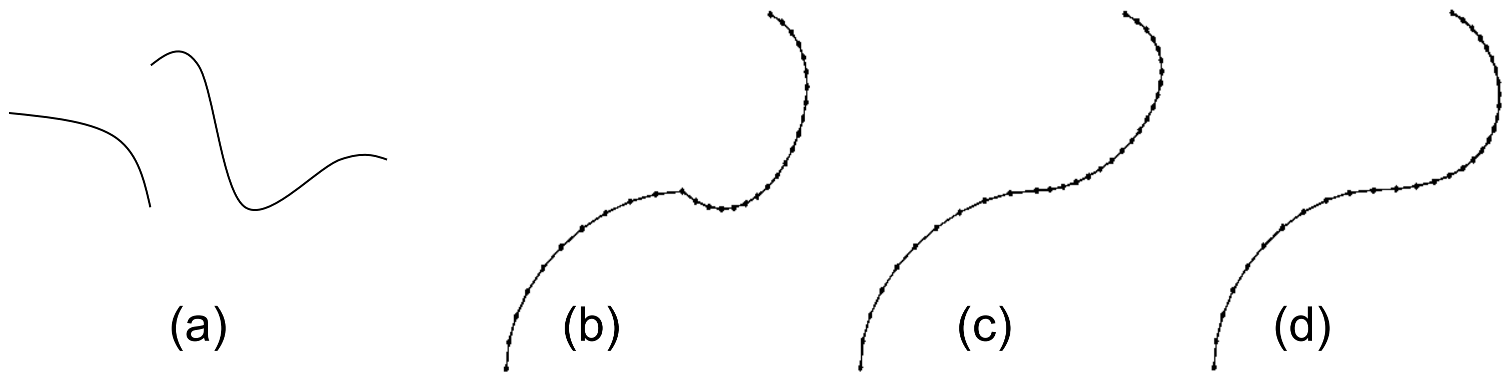
$\mathbf{u} = [1\ u\ u^2 u^3]$

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

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

Continuity



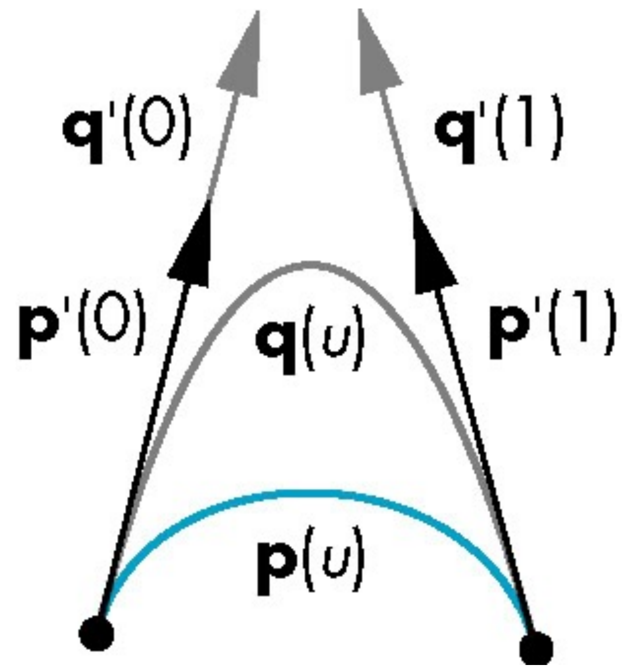
- A) Non-continuous
- B) C^0-continuous
- C) G^1-continuous
- D) C^1-continuous
- (C^2-continuous)

See page 726-727 in Real-time Rendering, 4th ed.

G^1-continuity Example

- Here the p and q have the same tangents at the ends of the segment but different derivatives (lengths)
  - This generates different Hermite curves

- This techniques is used in drawing applications



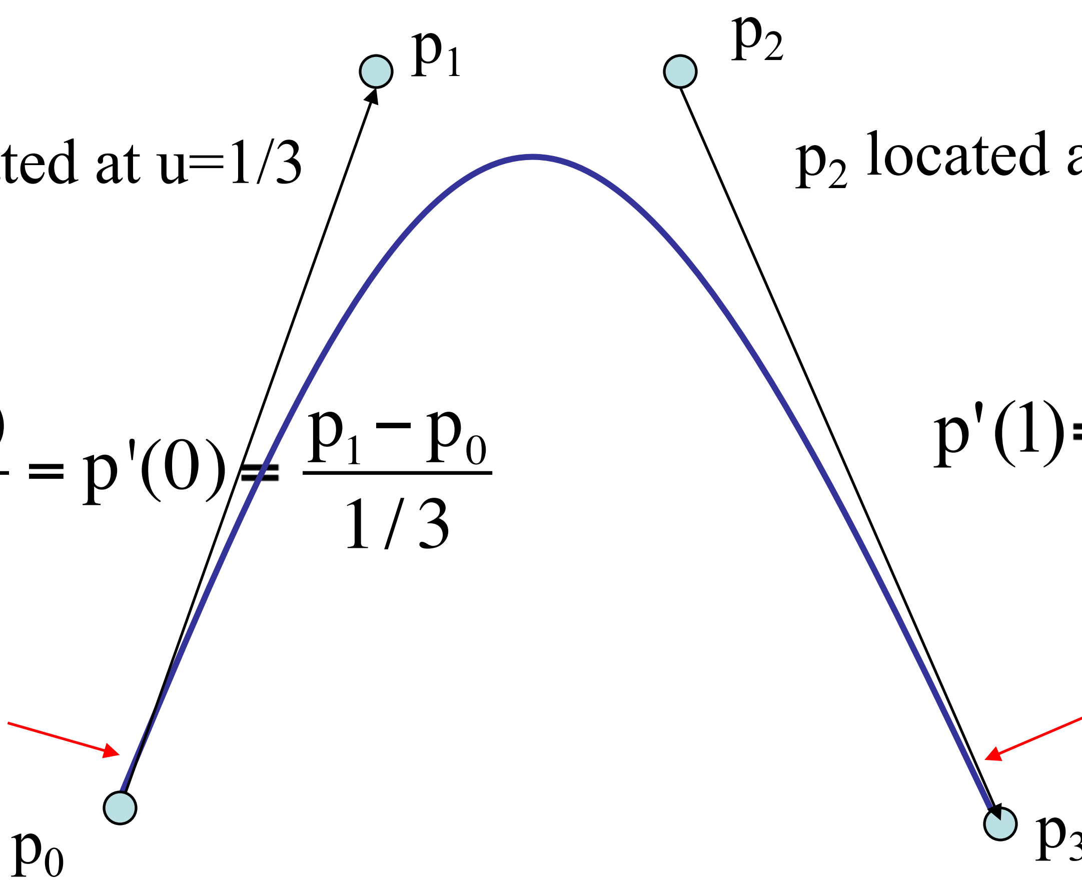
Reflections should be at least C^1

Example

Bezier Curves

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

Computing Derivatives



p_1 located at u=1/3

p_2 located at u=2/3

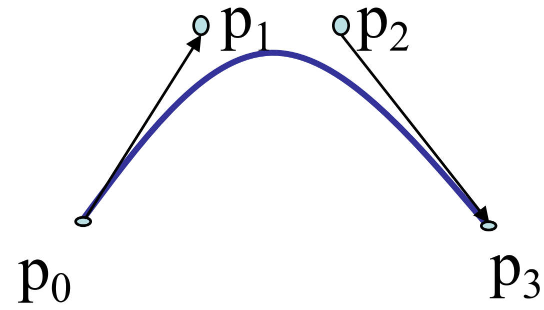
$$\frac{dp(u=0)}{du} = p'(0) = \frac{p_1 - p_0}{1/3}$$

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

slope p' (0)

slope p' (1)

Equations



$p(u) = c_0+uc_1+u^2c_2+u^3c_3$

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

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

$$\left.\begin{aligned} p'(0) &\approx \frac{p_1-p_0}{1/3} \\ p'(1) &\approx \frac{p_3-p_2}{1/3} \end{aligned}\right\}$$

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

$\Rightarrow \mathbf{Bp=Ac}$

$\Rightarrow \mathbf{c=A^{-1}Bp}$

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

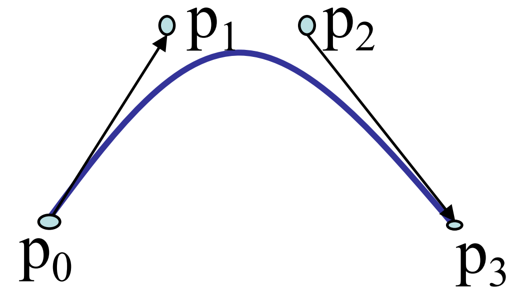
Bezier Matrix

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

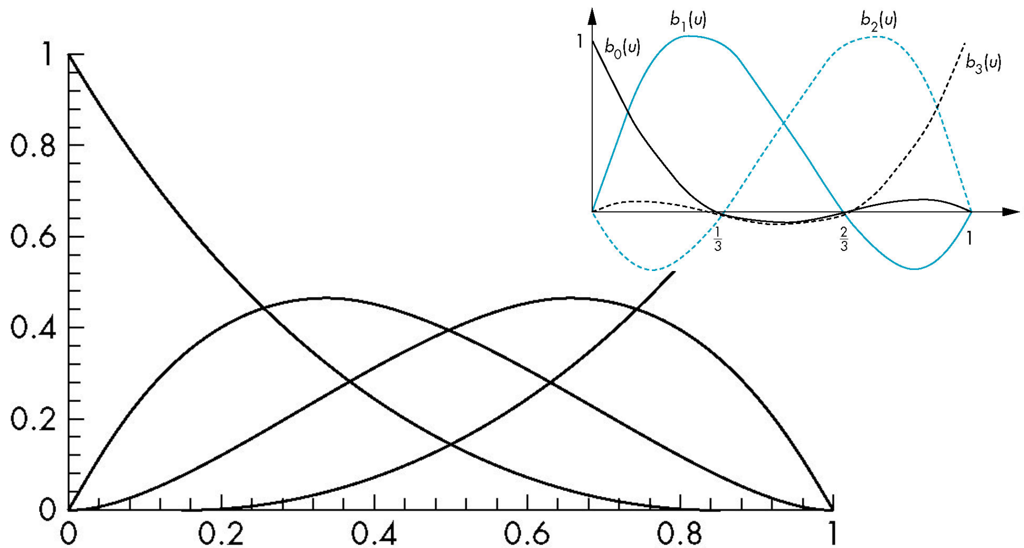
$$\mathrm{p}(\mathrm{u}) = \mathbf{u}^{\mathrm{T}}\mathbf{M}_B\mathbf{p} = \mathbf{b}(\mathrm{u})^{\mathrm{T}}\mathbf{p}$$

blending functions

Blending Functions



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

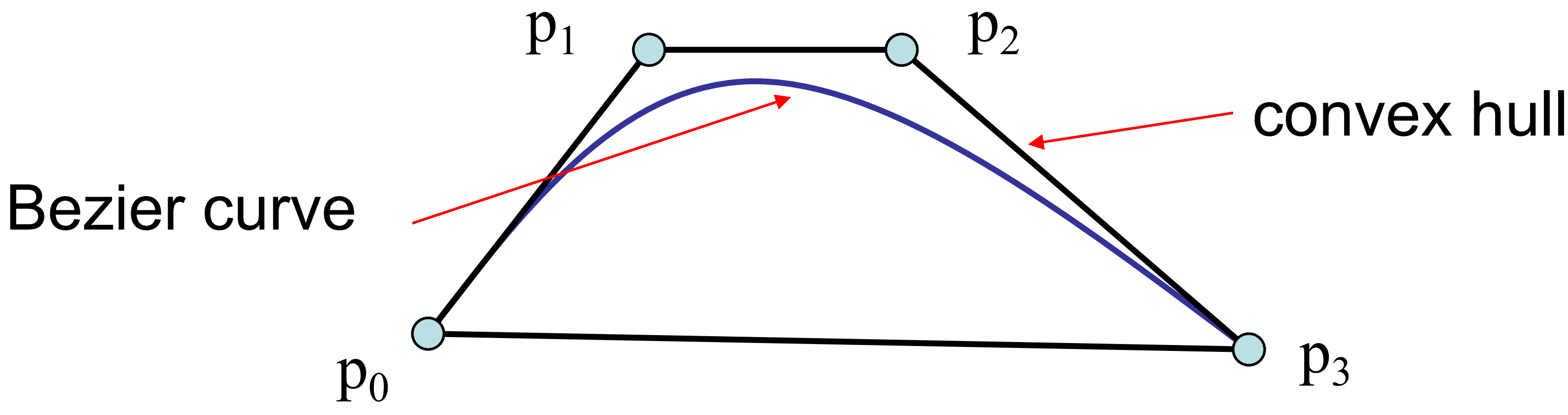


Note that all zeros are at 0 and 1 which forces the functions to be smoother over (0,1)

Smoother because the curve stays inside the convex hull, and therefore does not have room to fluctuate so much.

Convex Hull Property

- At given u, all weights being within [0,1] and sum of all weights = 1 ensures 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



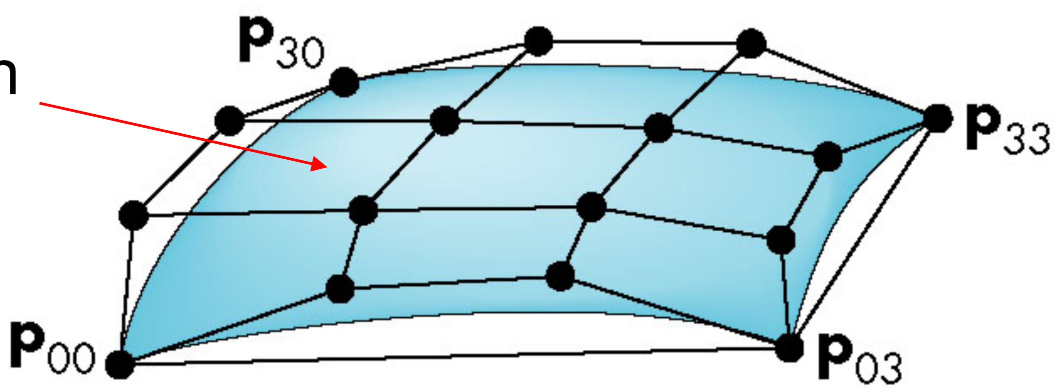
Bezier Patches

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

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

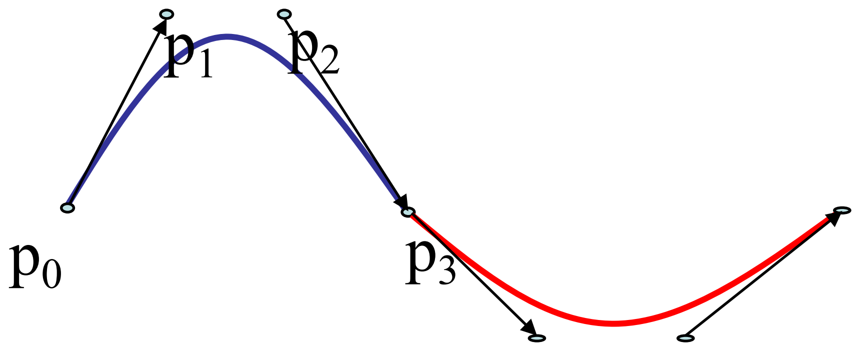
$$p(u,v)=\sum_{i=0}^{3}\sum_{j=0}^{3}b_i(u)\,b_j(v)\,p_{ij}=u^T\mathbf{M}_B\mathbf{P}\mathbf{M}_B^T v$$

Patch lies in convex hull



Analysis

- Although the Bezier form is much better than the interpolating form, the derivatives are not continuous at join points

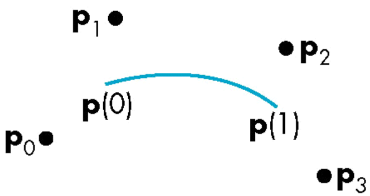


- What shall we do to solve this?

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



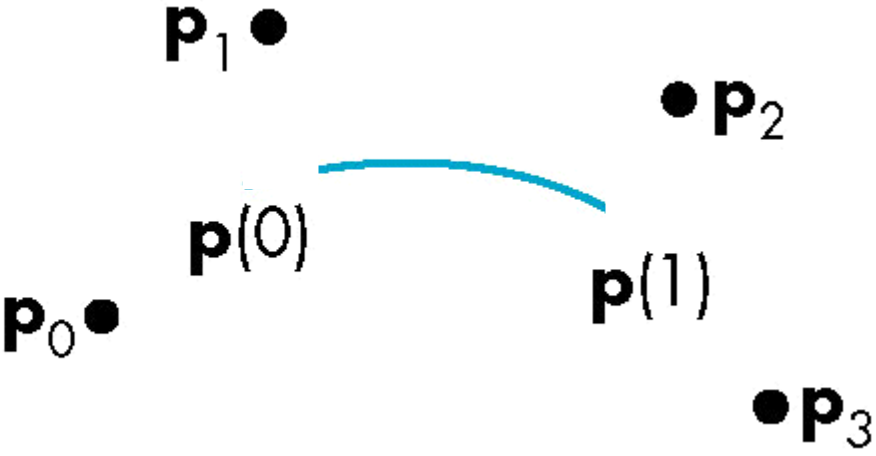
- Allows us to apply more continuity conditions to each segment
- For cubics, we can have continuity of the function and first and second derivatives at the join points

So what does the cubic B-spline matrix look like? …

Cubic B-spline Matrix

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

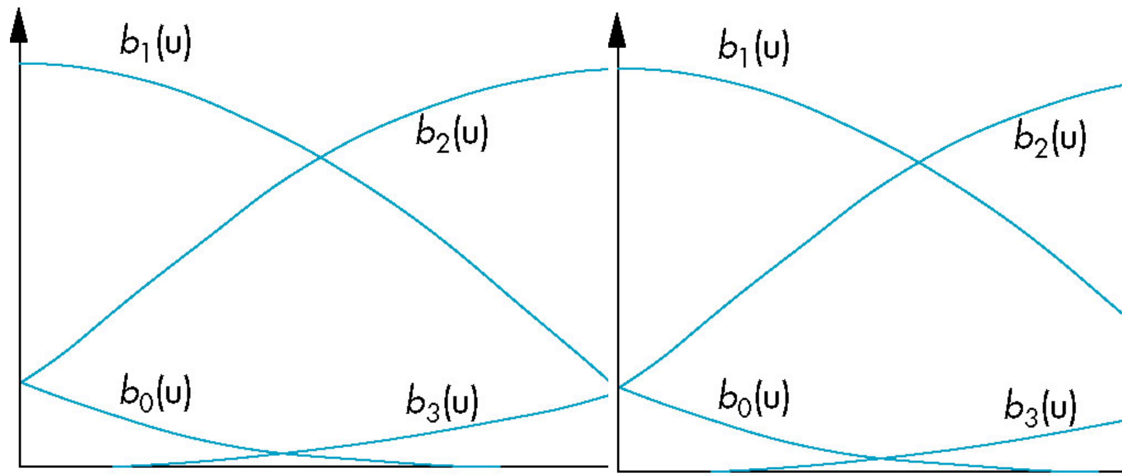
$$\mathbf{M}_S = \frac{1}{6}\begin{bmatrix} 1 & 4 & 1 & 0 \\ -3 & 0 & 3 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{bmatrix}$$



1st and 2nd derivatives are now continuous
as we can see on the blend functions…

Blending Functions

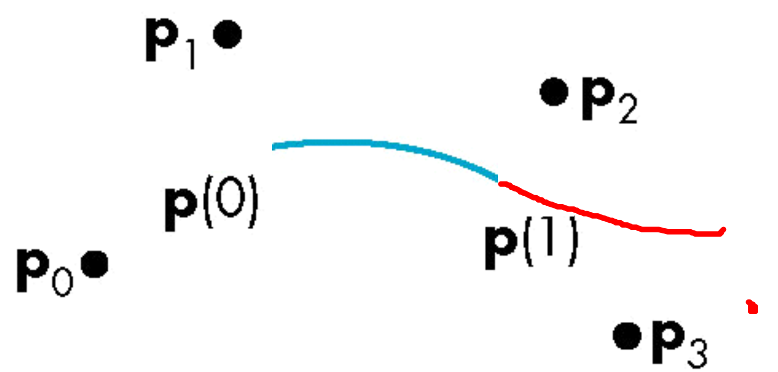
$$\mathbf{b}(u) = \frac{1}{6}\begin{array}{l} \color{red}{b_0:} \\ \color{red}{b_1:} \\ \\ \color{red}{b_3:} \end{array}\begin{bmatrix} (1-u)^3 \\ 4-6u^2+3u^3 \\ 1+3u+3u^2-3u^3 \\ u^3 \end{bmatrix}$$



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

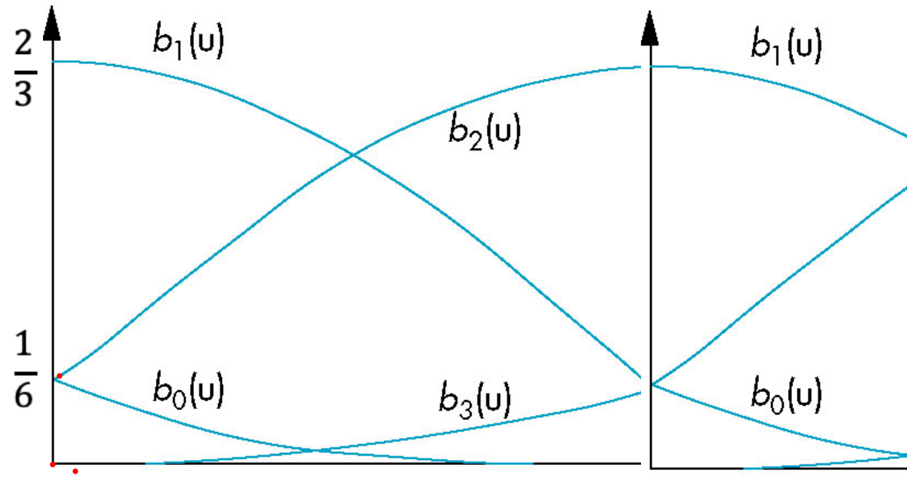
$p(u) = b_0(u)p_0 + b_1(u)p_1 + b_2(u)p_2 + b_3(u)p_3$

$$\mathbf{u}^T\mathbf{M}_S = \begin{bmatrix} 1 & u & u^2 & u^3 \end{bmatrix}\frac{1}{6}\begin{bmatrix} 1 & 4 & 1 & 0 \\ -3 & 0 & 3 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{bmatrix}$$



How compute the cubic B-spline matrix $\mathbf{M}_S$?

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



16 unknowns in $\mathbf{M}_S$. We need 16 equations:

- 5 for endpoint values:

$b_0(0)=b_1(1)$. $b_0(1)=0$. $b_1(0)=b_2(1)$. $b_2(0)=b_3(1)$. $b_3(0)=0$.

- Same 5 for endpoint 1st derivatives:

$b_0'(0)=b_1'(1)$. $b_0'(1)=0$. $b_1'(0)=b_2'(1)$. $b_2'(0)=b_3'(1)$. $b_3'(0)=0$.

- Same 5 for endpoint 2nd derivatives:

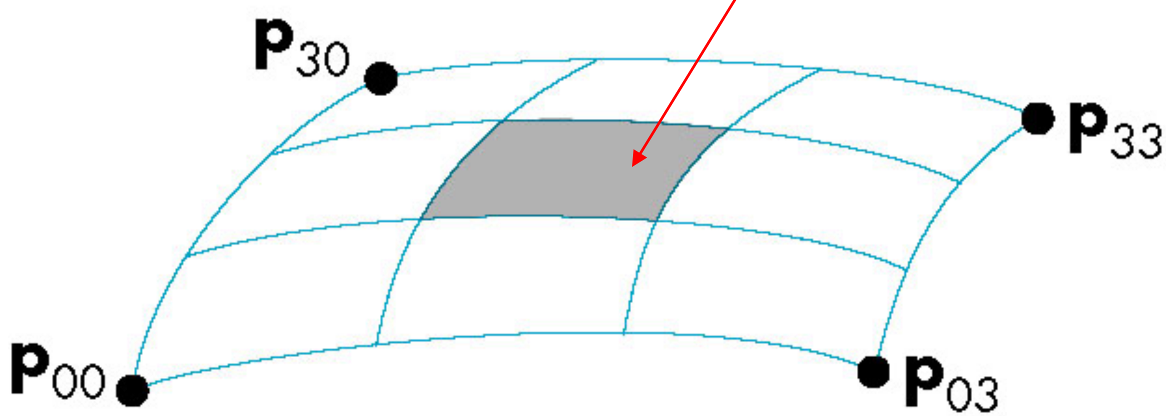
$b_0''(0)=b_1''(1)$. $b_0''(1)=0$. $b_1''(0)=b_2''(1)$. $b_2''(0)=b_3''(1)$. $b_3''(0)=0$.

- Sum = 1, everywhere: $b_0(u)+b_1(u)+b_2(u)+b_3(u)=1$, for $u \in [0,1]$. E.g., for $u=0$.

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



Let the 4 splines $b_{0..3}(u)$ create a basis function 

The blend function of each control point is just a translation along *u* of this basis function.

Basis Splines

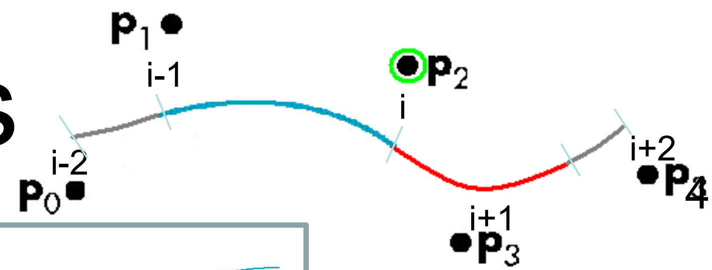
- If we examine the cubic B-spline from the perspective of each control (data) point, each interior point contributes (through the blending functions) to four segments
- We can rewrite p(u) in terms of *all* the data points along the curve as

The whole curve can be written as:

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

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

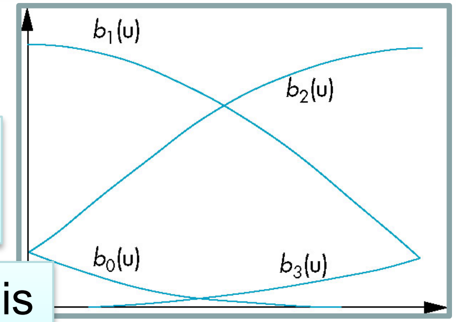
Basis Functions



Over this blue segment…

$$p(u) = \sum B_i(u) p_i = B_0(u) p_0 + \ldots B_{n-1}(u) p_{n-1}$$

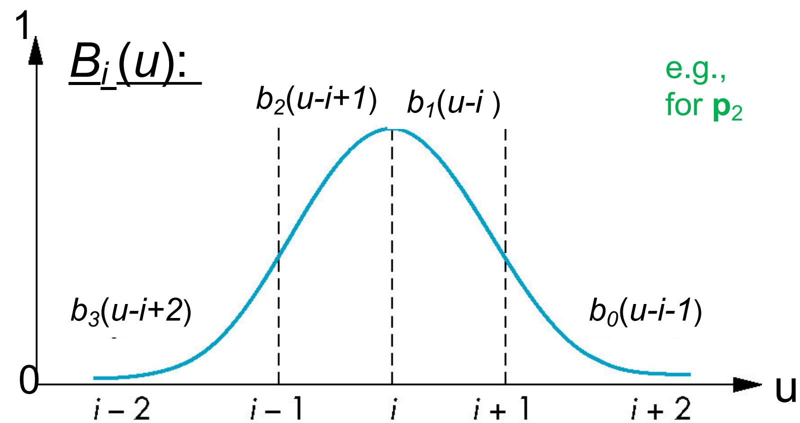
…these are the blending functions for control points $\mathbf{p}_0$ … $\mathbf{p}_3$



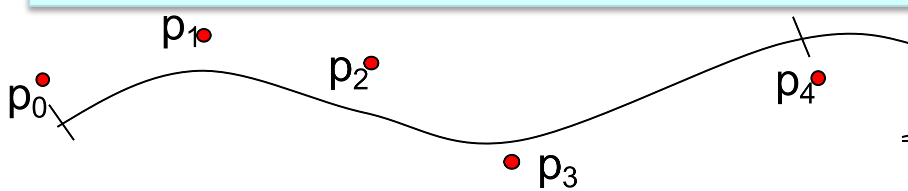
$$\begin{matrix} b_0(u) \\ b_1(u) \\ b_2(u) \\ b_3(u) \end{matrix} = \mathbf{b}(u) = \frac{1}{6}\begin{bmatrix} (1-u)^3 \\ 4-6u^2+3u^3 \\ 1+3u+3u^2-3u^3 \\ u^3 \end{bmatrix}$$

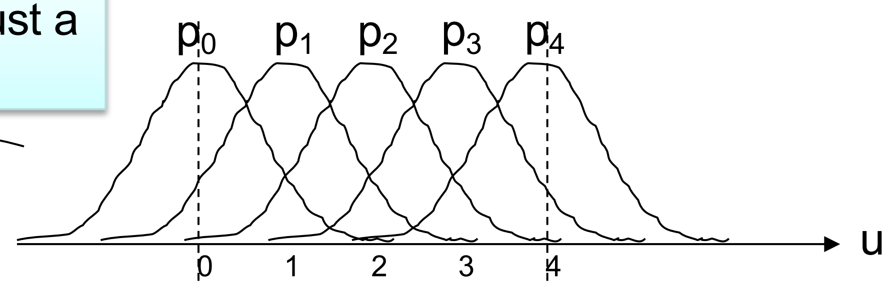
From the perspective of any control point $\mathbf{p}_i$ this is its weight, $B_i(u)$, over the complete curve $u=0\ldots n$:

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



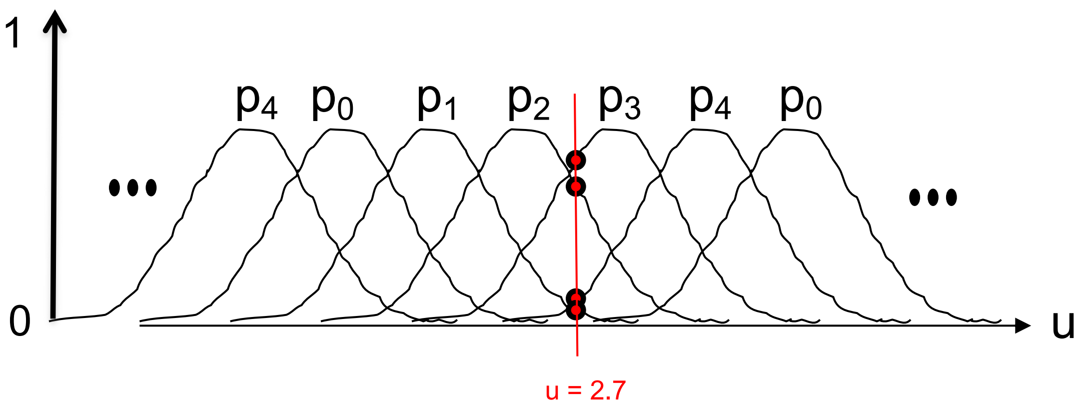
Each individual blending function $\mathbf{B}_i(u)$ is just a translation of the bell shape:

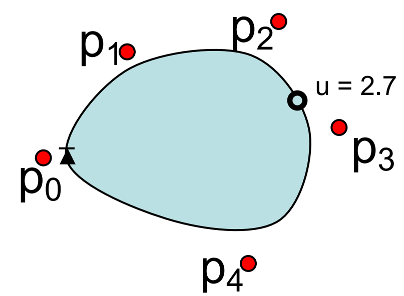




Weights for each point along the curve

One more example





$p(u) = B_0(u)p_0 + B_1(u)p_1 + B_2(u)p_2 + B_3(u)p_3 + B_4(u)p_4$

I.e.,: $p(u) = \sum B_i(u)\, p_i$

SUMMARY

B-Splines

These are our control points, p_0-p_8, to which we want to approximate a curve

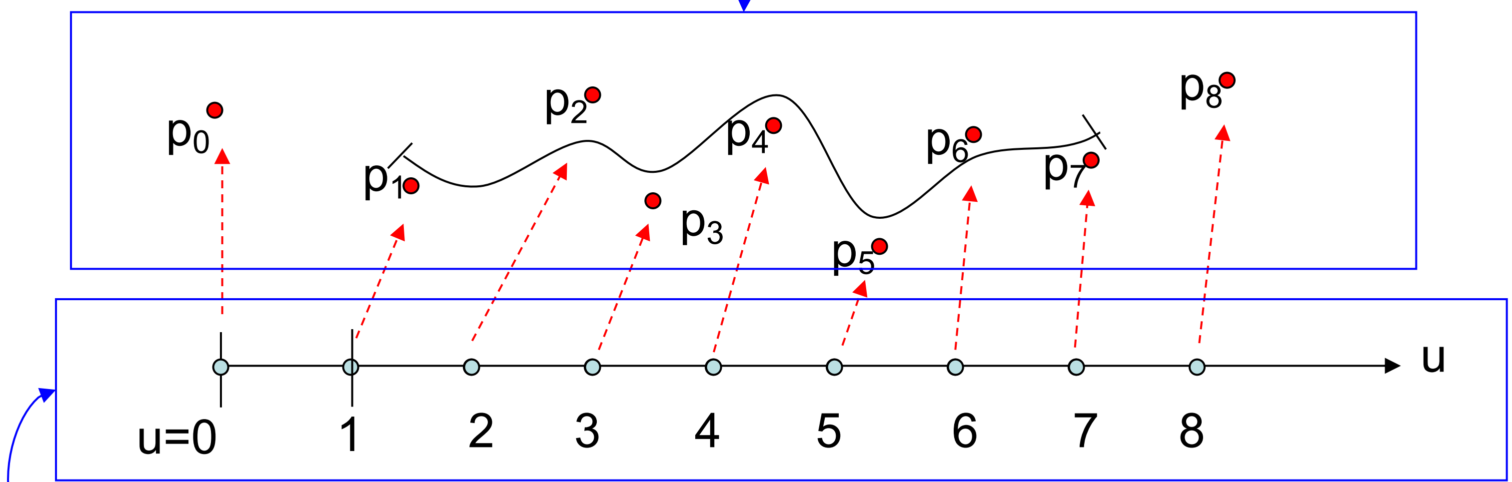
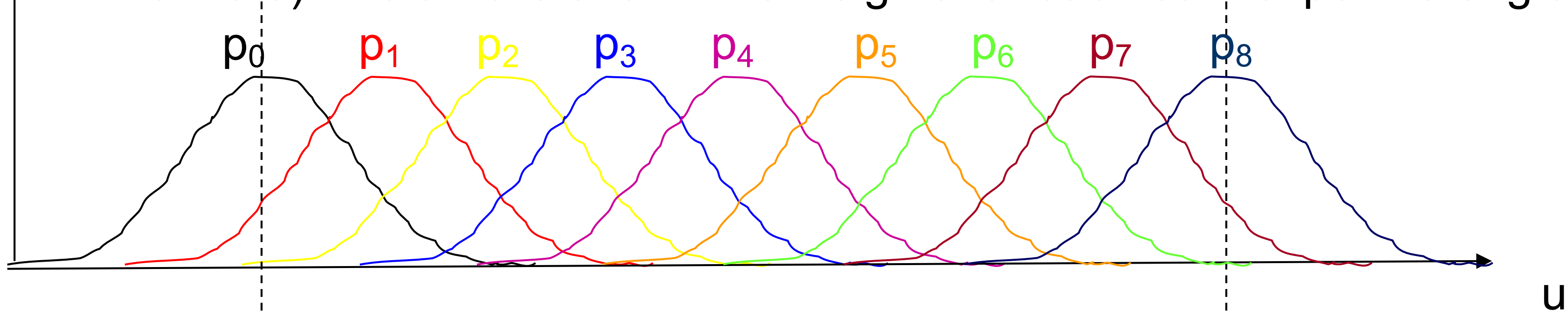


Illustration of how the control points are evenly (uniformly) distributed along the parameterisation u of the curve p(u).

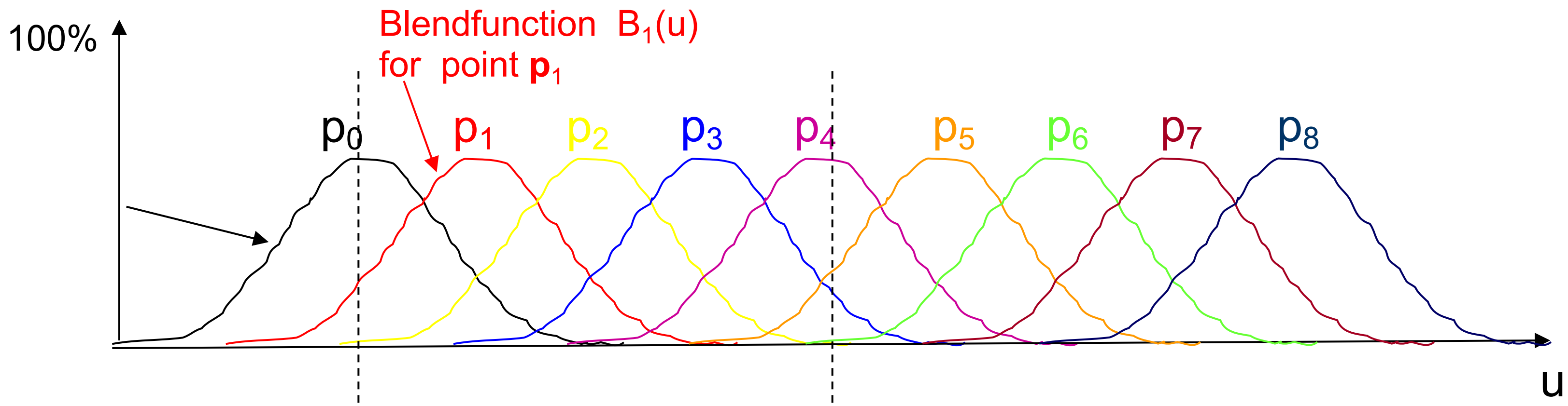
In each point p(u) of the curve (i.e., for a given u), the point is defined as a weighted sum of all control points (only the closest 4 surrounding will be nonzero). Below are shown the weights for each control point along $u=0 \rightarrow 8$

100 %



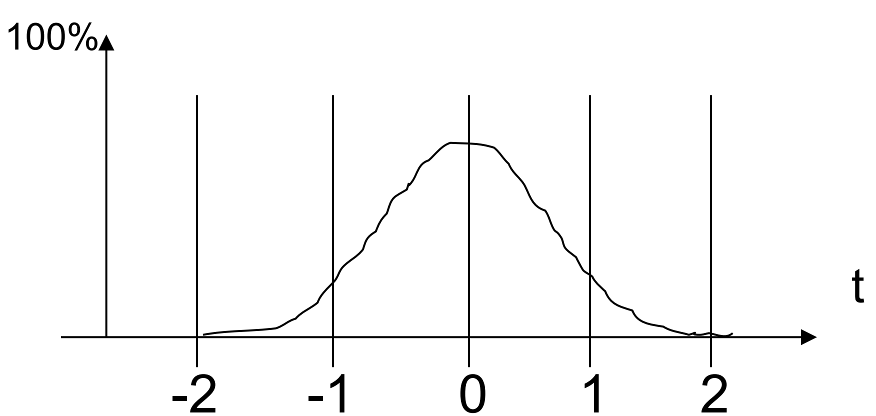
B-Splines

In each point p(u) of the curve, for a given u, the point is defined as a weighted sum of all control points (only the closest 4 surrounding will be nonzero). Below are shown the weights for each control point along u=0→8



The weight function (blend function) $B_i(u)$ for a point $\mathbf{p}_i$ can thus be written as a translation of a basis function B(t). $B_i(u) = B(u-i)$

B(t):



Our complete B-spline curve p(u) can thus be written as:

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

Generalizing Splines

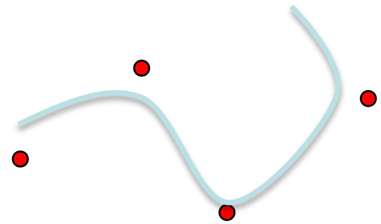
- Common to use *knot* vector:
  - array of the control-point indices: 0,1,2,3,4,5,6…
  - Can have repeated knots: 0,0,0,1,2,3,4,5,5,6,
    - Repeating a ctrl point 3x forces cubic spline to interpolate the point
    - If you want the curve to start at the first point and end at the last point, just repeat those 3 times: e.g., 0,0,0,1,2,3,4,5,6,6,6.

DEMO of B-Spline curve: (make duplicate knots)

(Cox-deBoor recursion gives method of evaluation - also known as de Casteljau-recursion, see page 721, RTR 4:th edition for details)

- We can extend to splines of any degree
  - Not just cubic polynomials (quartic, quintic…)
- Data and conditions do not have to be given at equally spaced *u* values:
  - Nonuniform (vs uniform splines)
  - Leads us to NURBS…

NURBS



NURBS = Non-Uniform Rational B-Splines

NURBS is similar to B-Splines except that:

1. The control points can have different weights, w_i, (heigher weight makes the curve go closer to that control point)
2. The control points do not have to be at uniform distances (u=0,1,2,3...) along the parameterisation u.
   E.g.: u=0, 0.5, 0.9, 4, 14,…

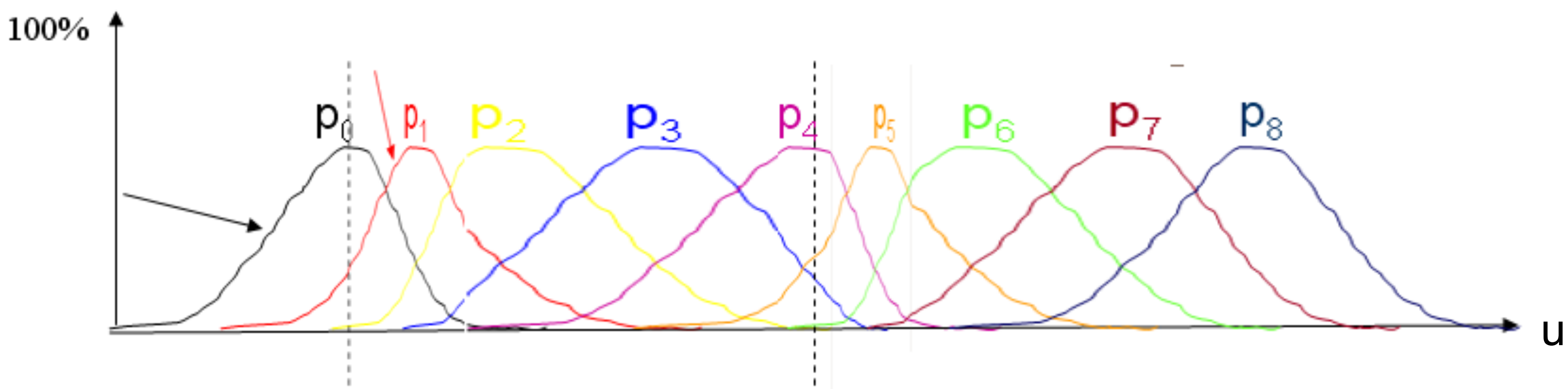
The NURBS-curve is thus defined as:

$$\mathbf{p}(u) = \frac{\sum_{i=0}^{n-1} B_i(u) w_i \mathbf{p}_i}{\sum_{i=0}^{n-1} B_i(u) w_i}$$

Division with the sum of the weights, to make the combined weights sum up to 1, at each position along the curve. (Otherwise, some scaling/translation of the curve is introduced, which is not desirable)

NURBS

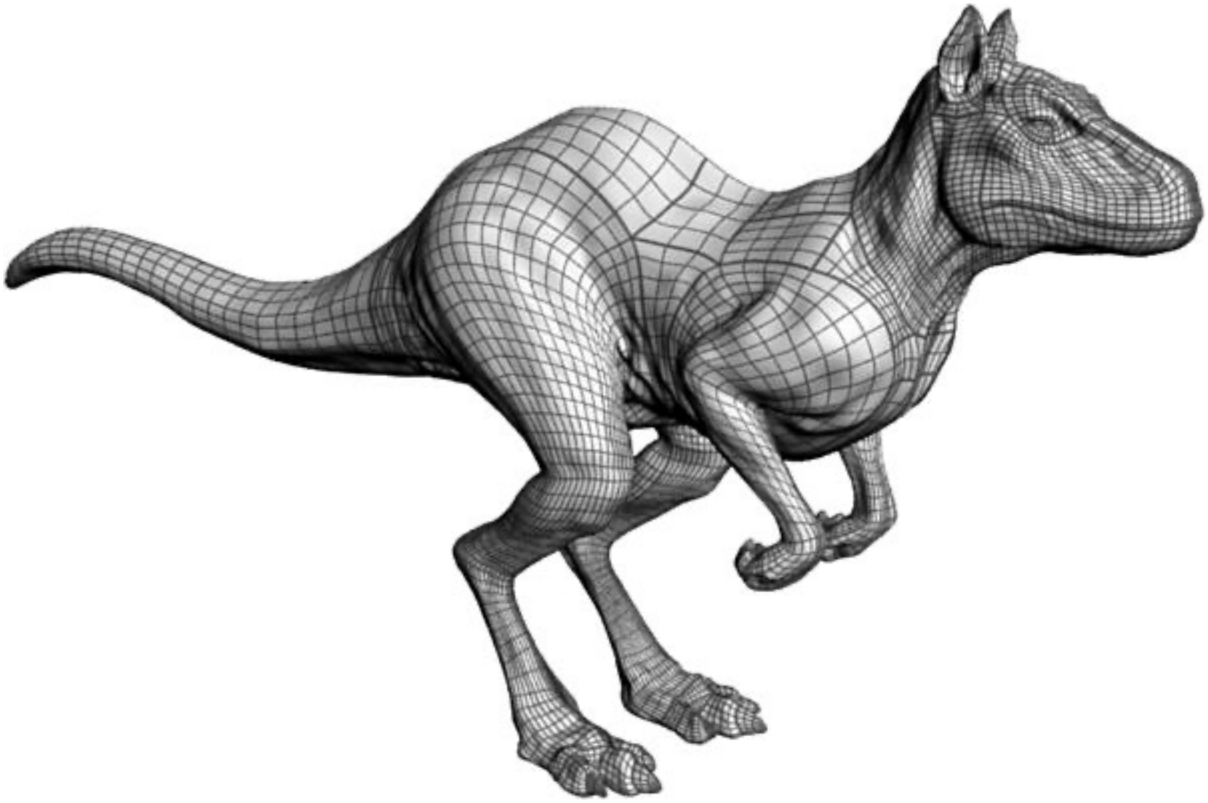
- Allowing control points at non-uniform distances means that the basis functions $B_{pi}()$ are being streched and non-uniformly located.
- E.g.:

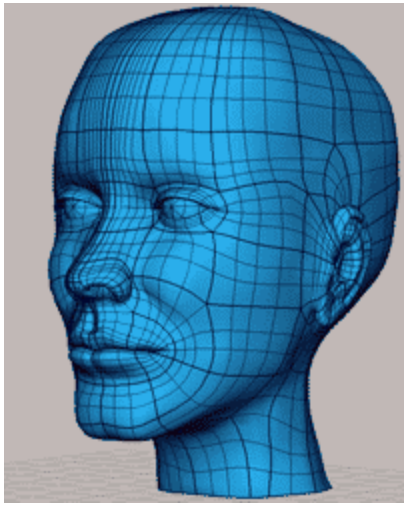


Each curve $B_{pi}()$ should of course look smooth and C^2 –continuous.
But it is not so easy to draw smoothly by hand...

(The sum of the weights are still =1 due to the division in previous slide.)

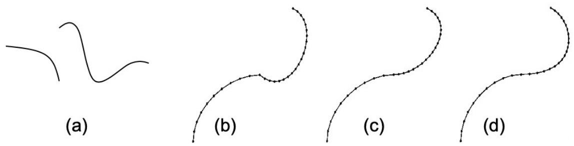
NURBS Surfaces - examples





What you need to know:

Continuity

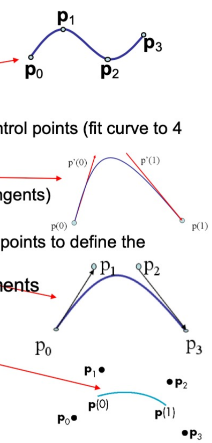


- A) Non-continuous
- B) C^0-continuous
- C) G^1-continuous
- D) C^1-continuous
- (C^2-continuous)

See page 726-727 in Real-time Rendering, 4th ed.

Objectives

- Introduce the types of curves
  - Interpolating
    - Blending polynomials for interpolation of 4 control points (fit curve to 4 control points)
  - Hermite
    - fit curve to 2 control points + 2 derivatives (tangents)
  - Bezier
    - 2 interpolating control points + 2 intermediate points to define the tangents
  - B-spline – use points of adjacent curve segments
    - To get C^1 and C^2 continuity
  - NURBS
    - Different weights of the control points
- Analyze them



SUMMARY

B-Splines

These are our control points, p_0-p_8, to which we want to approximate a curve

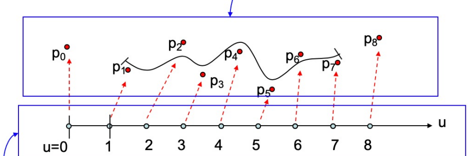
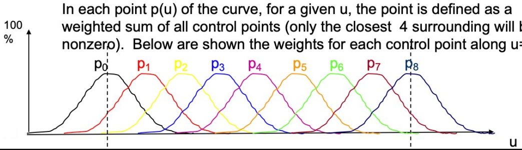


Illustration of how the control points are evenly (uniformly) distributed along the parameterisation u of the curve p(u).

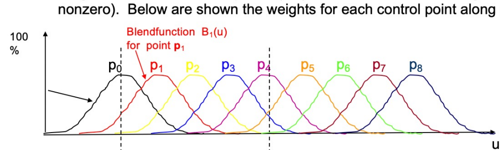
In each point p(u) of the curve, for a given u, the point is defined as a weighted sum of all control points (only the closest 4 surrounding will be nonzero). Below are shown the weights for each control point along u=0→8



SUMMARY

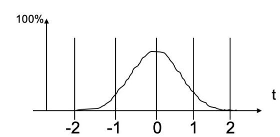
B-Splines

In each point p(u) of the curve, for a given u, the point is defined as a weighted sum of all control points (only the closest 4 surrounding will be nonzero). Below are shown the weights for each control point along u=0→8



The weight function (blend function) $B_i(u)$ for a point $\mathbf{p}_i$ can thus be written as a translation of a basis function B(t). $B_i(u) = B_t(u-i)$

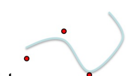
B(t):



Our complete B-spline curve p(u) can thus be written as:

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

NURBS



NURBS is similar to B-Splines except that:

1. The control points can have different weights, w_i, (heigher weight makes the curve go closer to that control point)
2. The control points do not have to be at uniform distances (u=0,1,2,3...) along the parameterisation u. E.g.: u=0, 0.5, 0.9, 4, 14,…

NURBS = Non-Uniform Rational B-Splines

The NURBS-curve is thus defined as:

$$\mathbf{p}(u) = \frac{\sum_{i=0}^{n} B_i(u) w_i \mathbf{p}_i}{\sum_{i=0}^{n} B_i(u) w_i}$$

Division with the sum of the weights, to make the combined weights sum up to 1, at each position along the curve. Otherwise, a scaling of (with the effect of also translating) the curve is introduced (which is not desirable)

Bonus slides

- Every polynomial curve can be exactly described by a bezier curve (by properly adjusting the control points).
- Rasterization of Bezier curves can be implemented highly efficiently using *de Casteljau recursion*.
- Thus, NURBS curves are often first converted to Bezier curves, to be efficiently rasterized.
- See following bonus slides for explanations…

Every Polynomial 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 $\mathbf{q}$

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

- There exist Bezier control points $\mathbf{p}$ such that

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

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

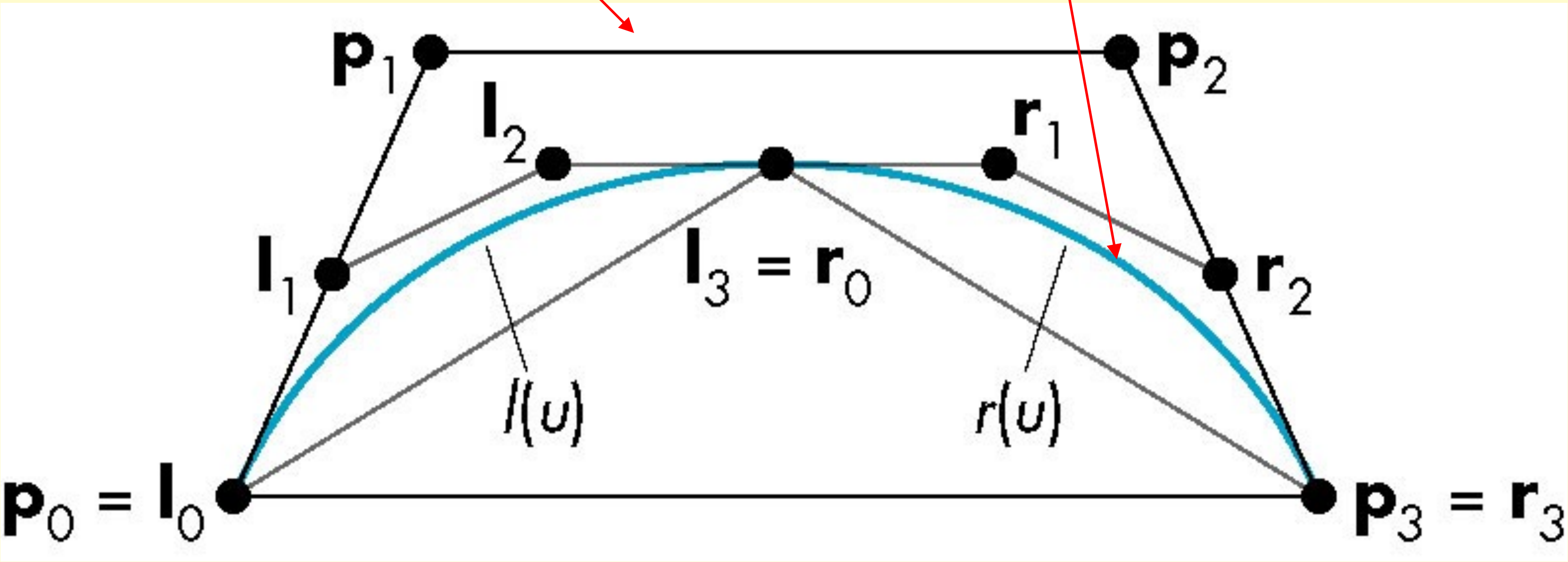
deCasteljau[1] Recursion

- 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"

[1] Paul de Casteljau and Pierre Bezier where engineers in the car industry. De Casteljau at Peugot at Bezier at Renault. Both developed Bezier-surfaces, unaware of each other.

Splitting a Cubic Bezier

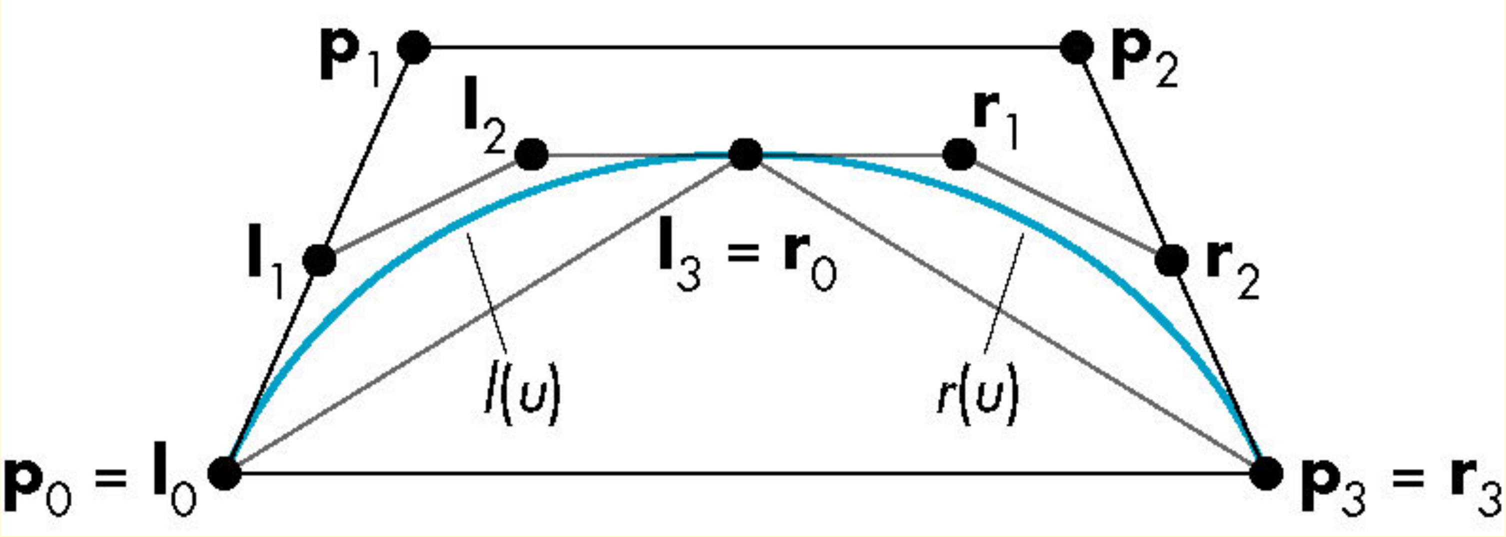
p_0, p_1 , p_2 , p_3 determine a cubic Bezier polynomial and its convex hull



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

l(u) and r(u)

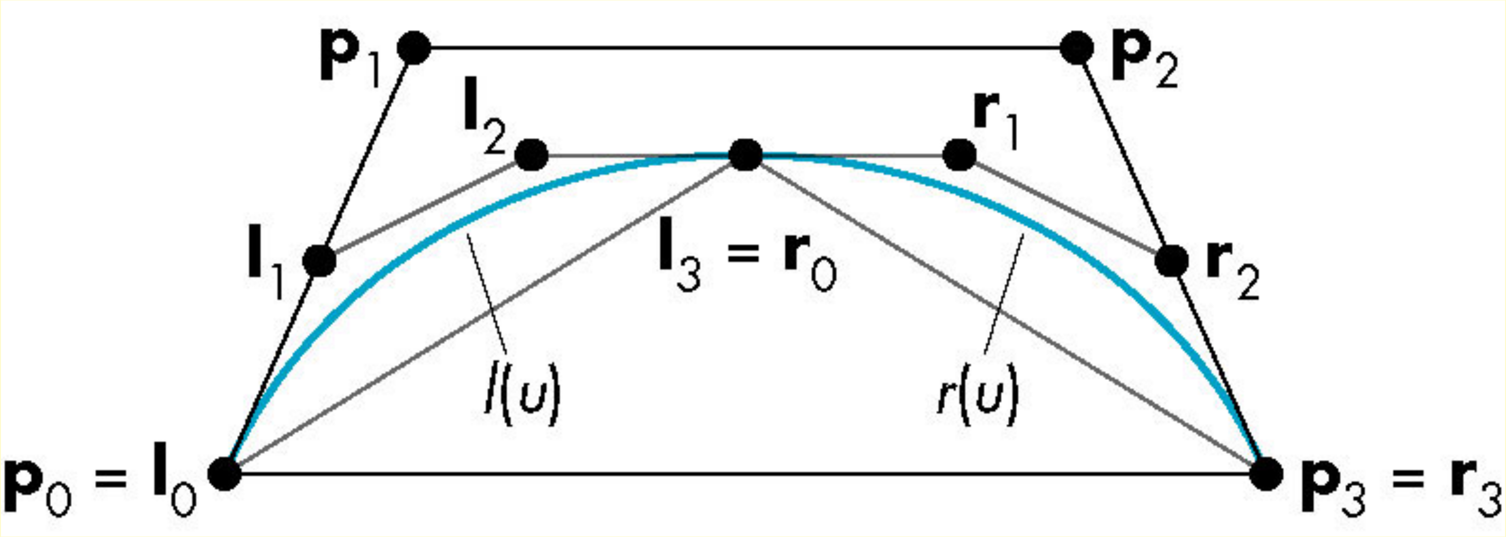
Since l(u) and r(u) are Bezier curves, we should be able to find two sets of control points $\{l_0, l_1, l_2, l_3\}$ and $\{r_0, r_1, r_2, r_3\}$ that determine them



Convex Hulls

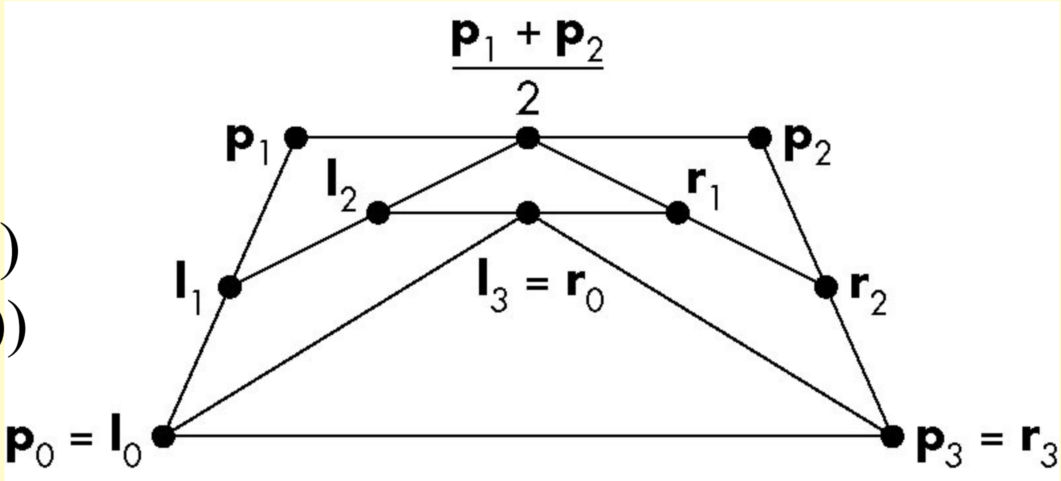
$\{l_0, l_1, l_2, l_3\}$ and $\{r_0, r_1, r_2, r_3\}$each have a convex hull that that is closer to p(u) than the convex hull of $\{p_0, p_1, p_2, p_3\}$ This is known as the *variation diminishing property*.

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

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



Requires only shifts and adds!

Then, recursively continue for the two new bezier curves $\{l_0, l_1, l_2, l_3\}$ and $\{r_0, r_1, r_2, r_3\}$ until desired precission is reached.