Rendering Curves and Surfaces

Ed Angel

Professor Emeritus of Computer Science

University of New Mexico

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

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

Finite Differences

For equally spaced $\{u_k\}$ we define *finite differences*

$$\Delta^{(0)} p(u_k) = p(u_k)$$

$$\Delta^{(1)} p(u_k) = p(u_{k+1}) - p(u_k)$$

$$\Delta^{(m+1)} p(u_k) = \Delta^{(m)} p(u_{k+1}) - \Delta^{(m)} p(u_k)$$

For a polynomial of degree n,
the n^{th} finite difference is constant

Building a Finite Difference Table

The University of New Mexico

$p(u)=1+3u+2u^2+u^3$

t	0	1	2	3	4	5
p	1	7	23	55	109	191
$\Delta^{(1)}$**p**	6	16	32	54	82	
$\Delta^{(2)}$**p**	10	16	22	28		
$\Delta^{(3)}$**p**	6	6	6			

Finding the Next Values

Starting at the bottom, we can work up generating new values for the polynomial

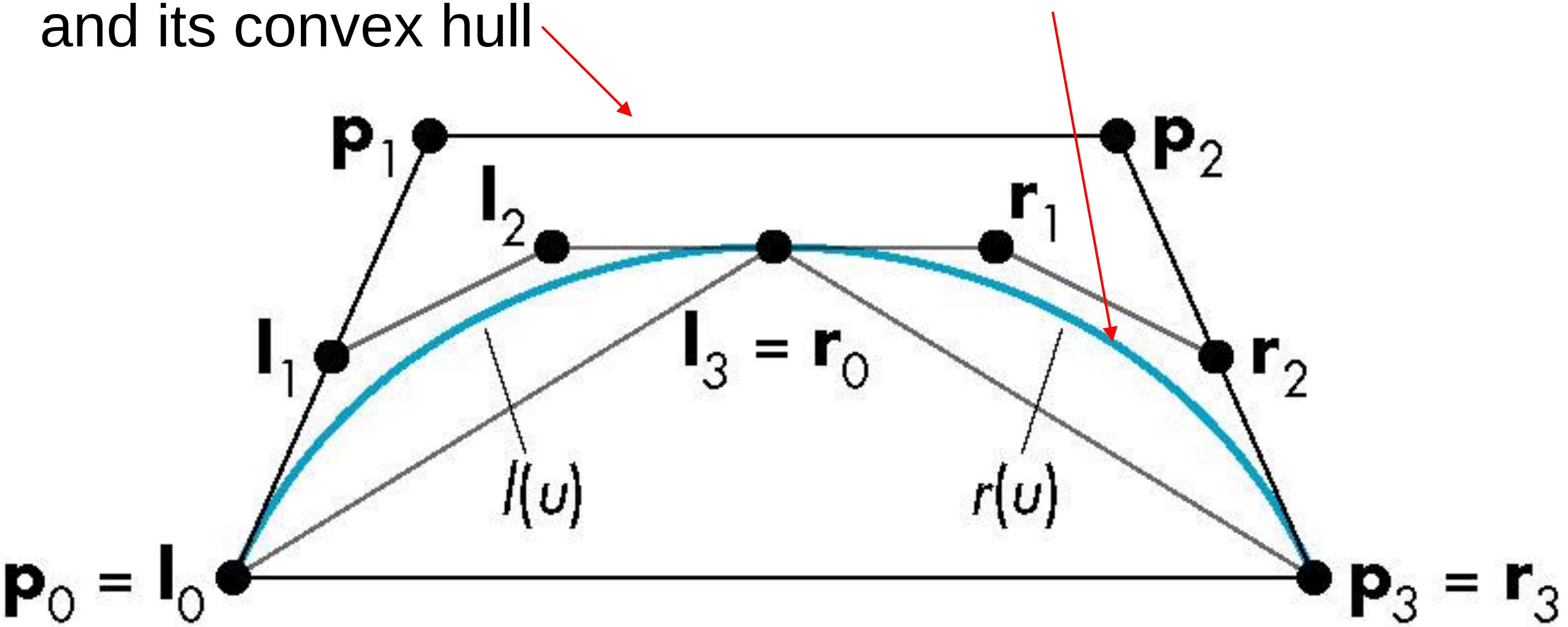
t	0	1	2	3	4	5
$\mathbf{p}$	1	7	23	55	109	191
$\Delta^{(1)}\mathbf{p}$	6	16	32	54	82	
$\Delta^{(2)}\mathbf{p}$	10	16	22	28		
$\Delta^{(3)}\mathbf{p}$	6	6	6			

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

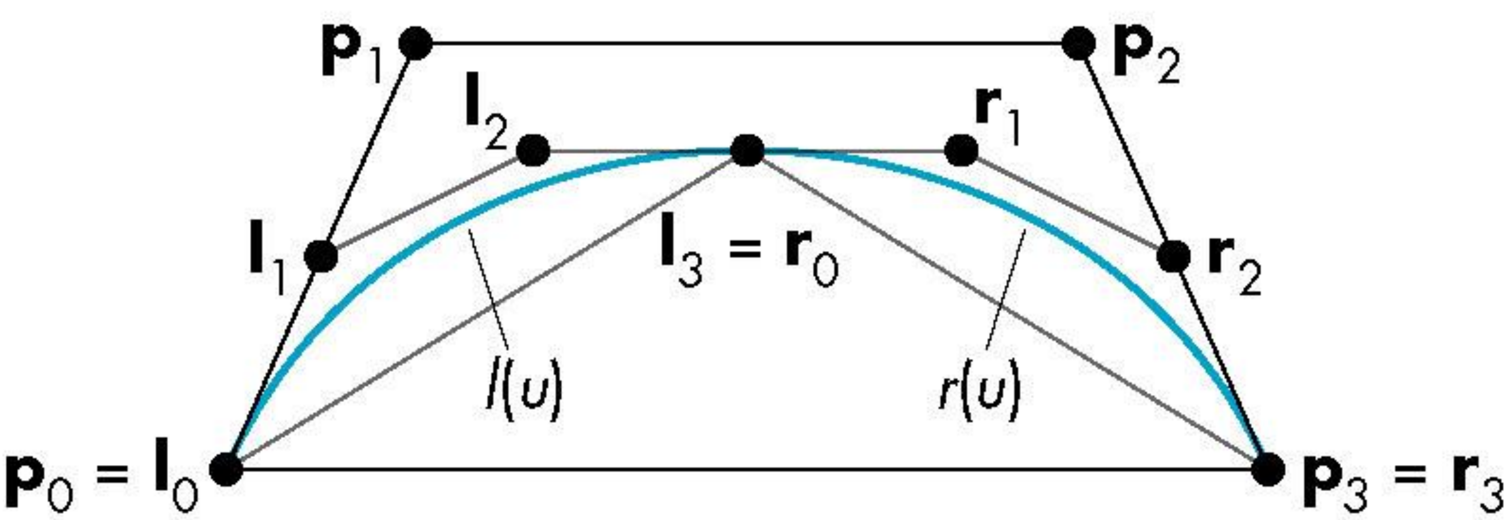
p_0, p_1 , p_2 , p_3 determine a cubic Bezier polynomial and its convex hull



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

l(u) and r(u)

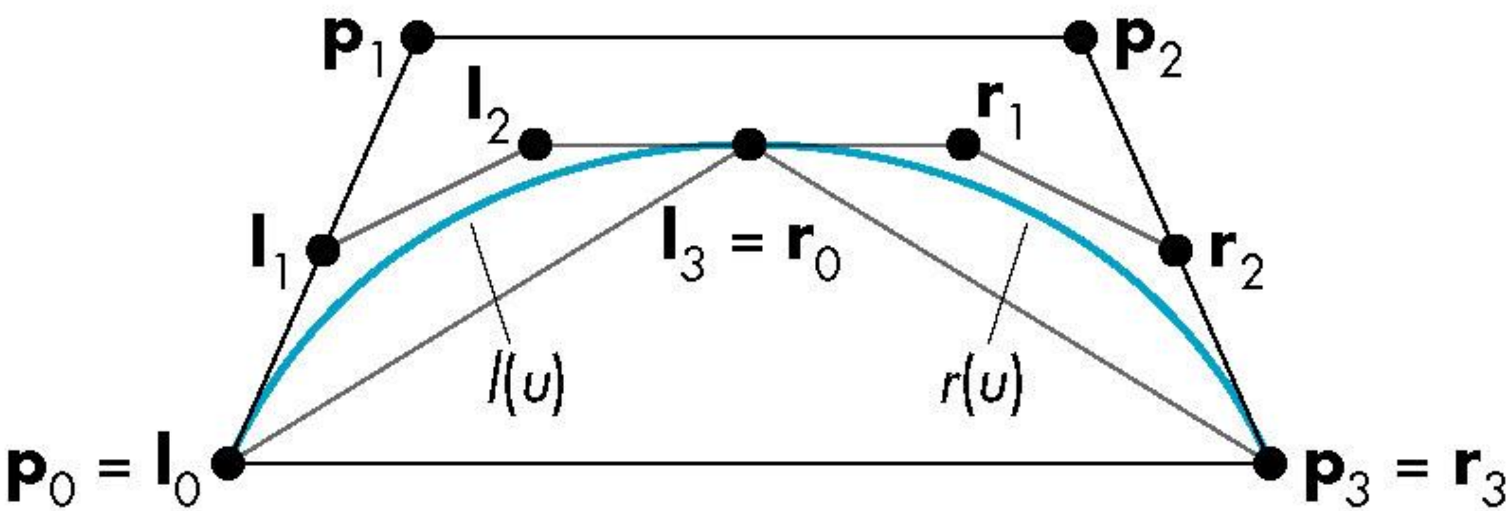
Since l(u) and r(u) are Bezier curves, we should be able to find two sets of control points $\{l_0, l_1, l_2, l_3\}$ and $\{r_0, r_1, r_2, r_3\}$ that determine them



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.



Equations

Start with Bezier equations $p(u)=\mathbf{u}^T M_B \mathbf{p}$

$l(u)$ must interpolate $p(0)$ and $p(1/2)$

$$l(0) = l_0 = p_0$$
$$l(1) = l_3 = p(1/2) = 1/8(p_0 +3 p_1 +3 p_2 + p_3)$$

Matching slopes, taking into account that $l(u)$ and $r(u)$ only go over half the distance as $p(u)$

$$l'(0) = 3(l_1 - l_0) = p'(0) = 3/2(p_1 - p_0)$$
$$l'(1) = 3(l_3 – l_2) = p'(1/2) = 3/8(- p_0 - p_1+ p_2 + p_3)$$

Symmetric equations hold for $r(u)$

Efficient Form

$l_0 = p_0$

$r_3 = p_3$

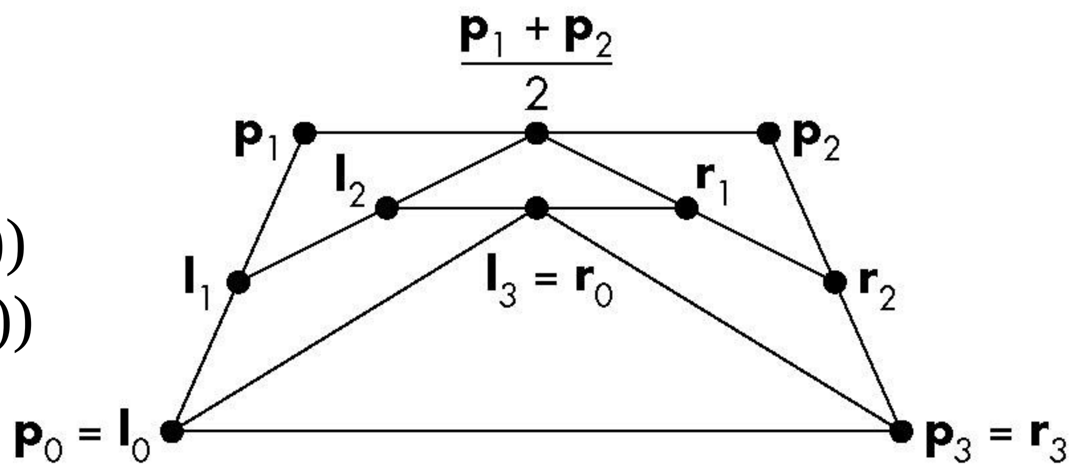
$l_1 = ½(p_0 + p_1)$

$r_1 = ½(p_2 + p_3)$

$l_2 = ½(l_1 + ½(p_1 + p_2))$

$r_1 = ½(r_2 + ½(p_1 + p_2))$

$l_3 = r_0 = ½(l_2 + r_1)$



Requires only shifts and adds!

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 $\mathbf{q}$

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

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

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

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

Matrices

Interpolating to Bezier

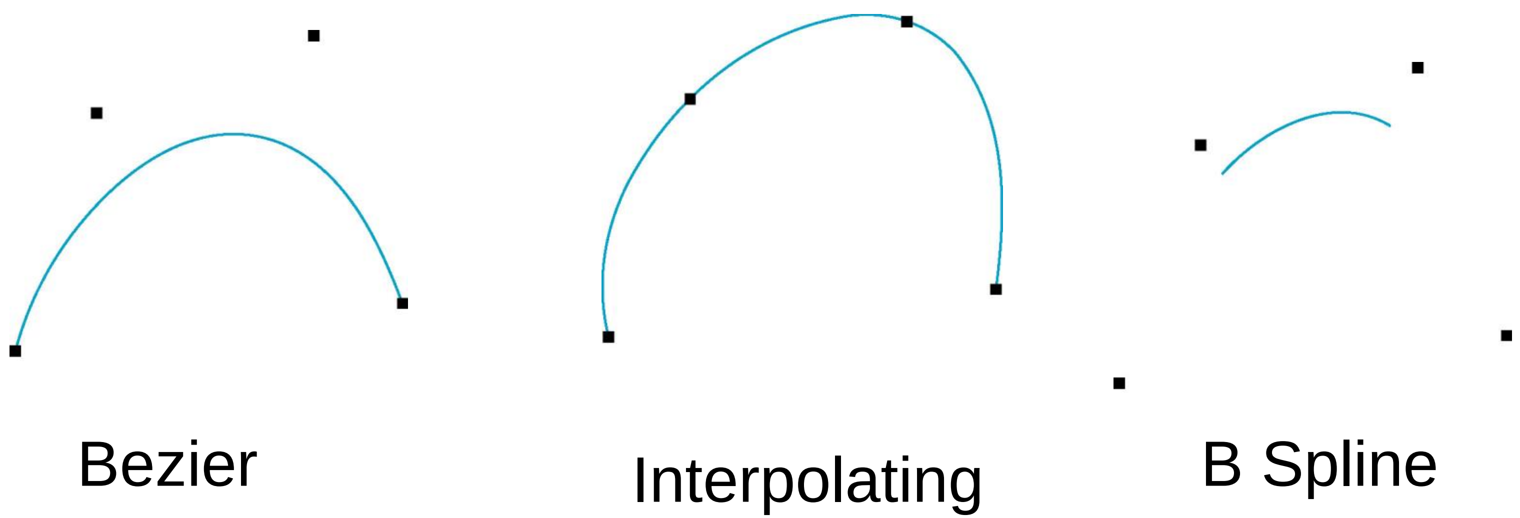
$$\mathbf{M}_B^{-1}\mathbf{M}_I = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -\frac{5}{6} & 3 & -\frac{3}{2} & \frac{1}{3} \\ \frac{1}{3} & -\frac{3}{2} & 3 & -\frac{5}{6} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

B-Spline to Bezier

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

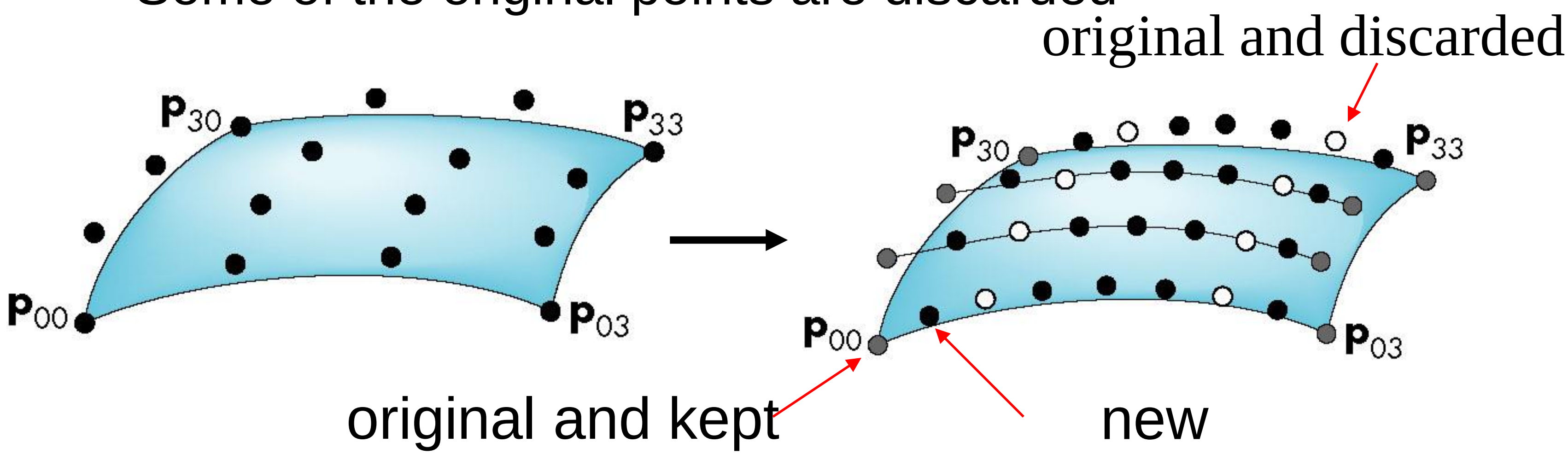
Example

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

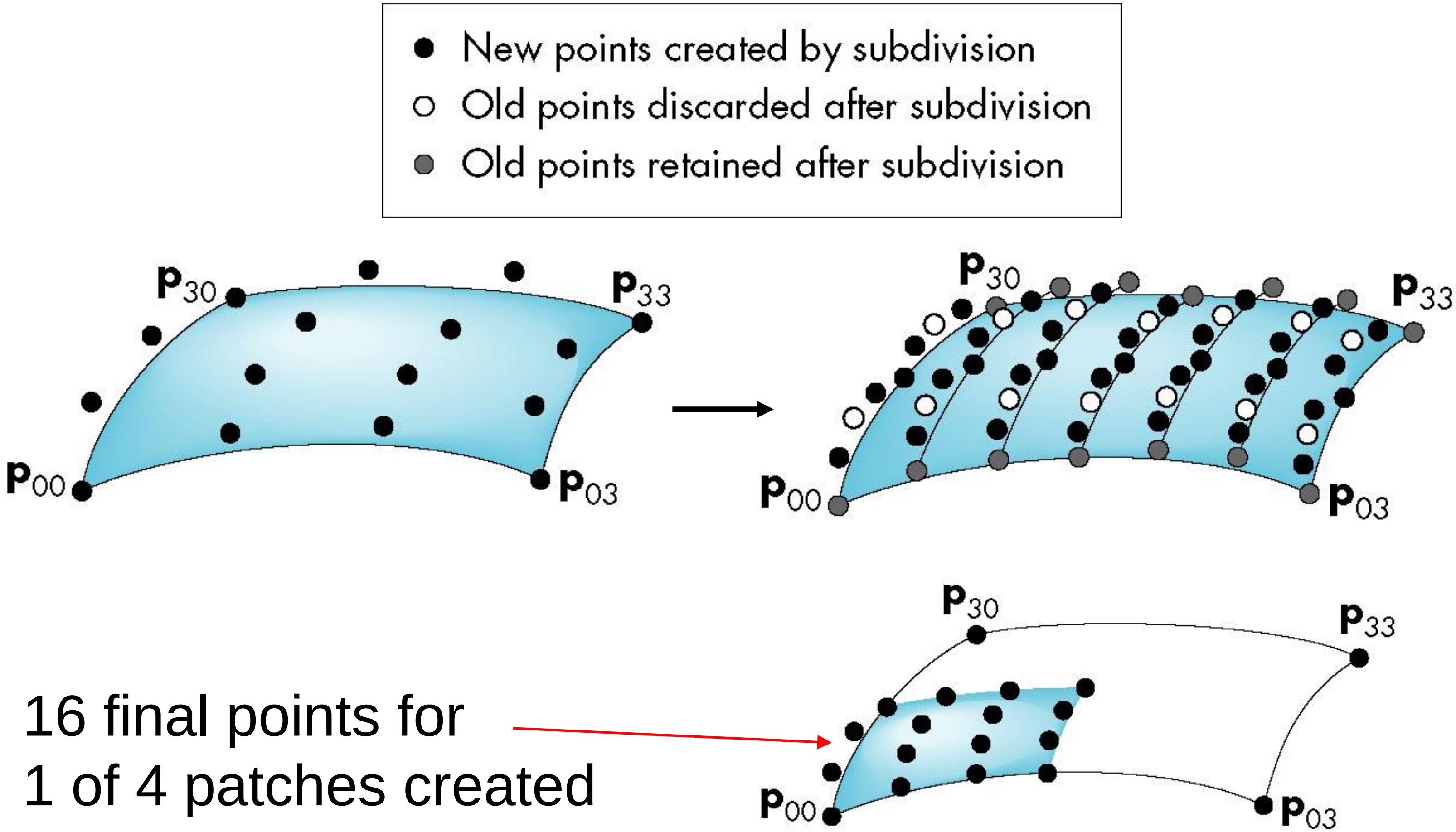


Surfaces

- Can apply the recursive method to surfaces if we recall that for a Bezier patch curves of constant u (or v) are Bezier curves in u (or v)
- First subdivide in u
  - Process creates new points
  - Some of the original points are discarded



Second Subdivision



16 final points for
1 of 4 patches created

Normals

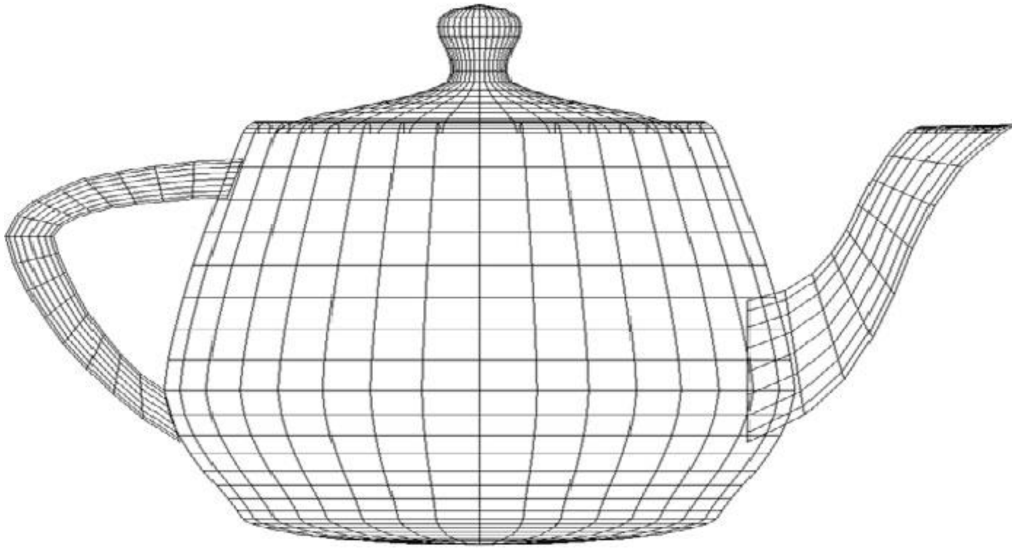
- For rendering we need the normals if we want to shade
  - Can compute from parametric equations

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

  - Can use vertices of corner points to determine
  - OpenGL can compute automatically

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



Quadrics

- Any quadric can be written as the quadratic form

  $\mathbf{p}^T\mathbf{A}\mathbf{p}+\mathbf{b}^T\mathbf{p}+c=0$ where $\mathbf{p}=[x, y, z]^T$

  with $\mathbf{A}$, $\mathbf{b}$ and c giving the coefficients
- Render by ray casting
  - Intersect with parametric ray $\mathbf{p}(\alpha)=\mathbf{p}_0+\alpha\mathbf{d}$ that passes through a pixel
  - Yields a scalar quadratic equation
    - No solution: ray misses quadric
    - One solution: ray tangent to quadric
    - Two solutions: entry and exit points