CMSC427
Computer Graphics

Matthias Zwicker

Fall 2019

Today

Curves

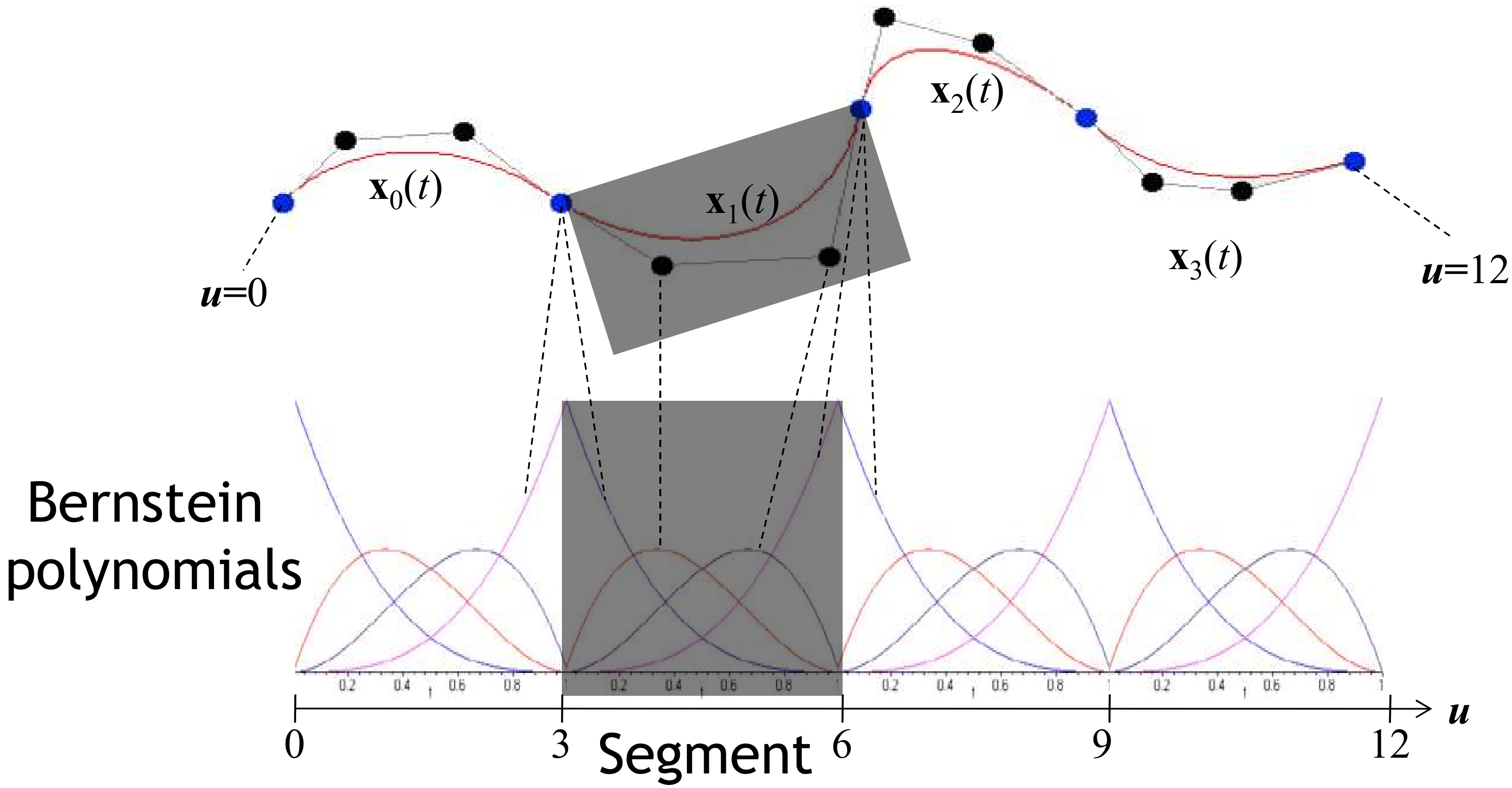
- NURBS

Surfaces

- Parametric surfaces
- Bilinear patch
- Bicubic Bézier patch
- Advanced surface modeling

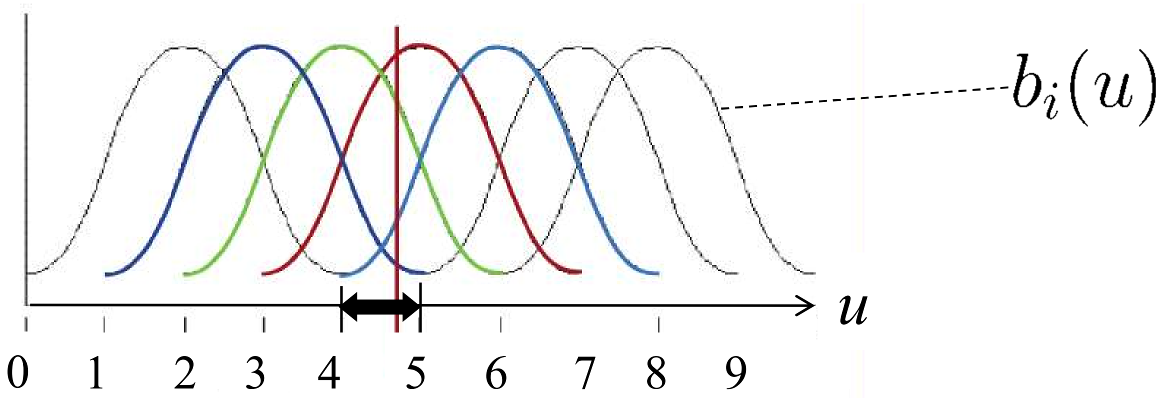
Piecewise Bézier curves

- Each segment contains four control points, four Bernstein polynomials
- Disadvantage: continuity only if control points are placed accordingly



B-splines

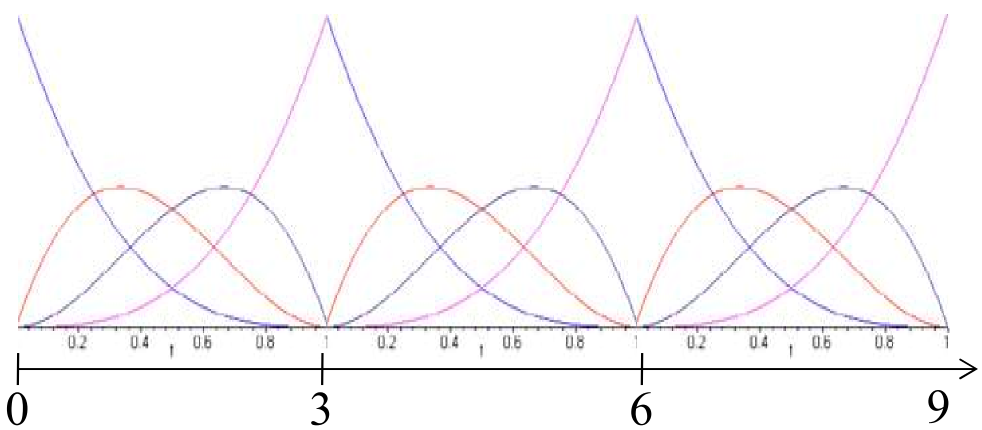
- Same idea, but generalized polynomial blending functions
- Uniform B-splines have only one type of blending function: B-spline (basis) function b_i
- B-spline function of degree n is C^{n-1} continuous
- Local support, at each point u exactly $n+1$ functions are non-zero



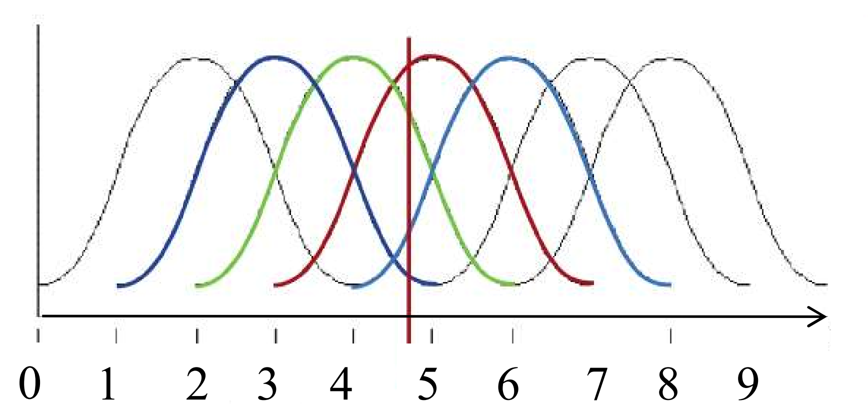
Uniform B-spline (basis) functions of degree 3

B-splines

- Widely used for curve and surface modeling
- Advantages over Bézier curves
  - Built-in continuity
  - Local support: curve only affected by nearby control points



Bernstein polynomials, deg. 3



B-spline basis functions, deg. 3

B-splines

- Weighted average of control points $\mathbf{p}_i$ using B-spline functions $b_i(u)$

$$\mathbf{x}(u) = \sum b_i(u)\mathbf{p}_i$$

- Positive, partition of unity => convex hull property
- Matrix form (note different basis matrix; caution: last lecture, matrices were transposed)

$$\underbrace{\begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}}_{\mathbf{B}_{Bez}}$$

$$\mathbf{x}(u) = \underbrace{\begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix}}_{\mathbf{T}} \underbrace{\frac{1}{6}\begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix}}_{\mathbf{B}_{B-spline}} \underbrace{\begin{bmatrix} \mathbf{p}_i \\ \mathbf{p}_{i+1} \\ \mathbf{p}_{i+2} \\ \mathbf{p}_{i+3} \end{bmatrix}}_{\mathbf{G}_{B-spline}}$$

where $t = u - i$ and $i = \lfloor u \rfloor$

Generalization: NURBS

http://en.wikipedia.org/wiki/Non-uniform_rational_B-spline

- Non-Uniform Rational B-splines
- Interactive explanation
  http://www.ibiblio.org/e-notes/Splines/nurbs.html

- NOTE: notation now uses t instead of u for curve parameter

Non-uniform B-splines

Knot vector

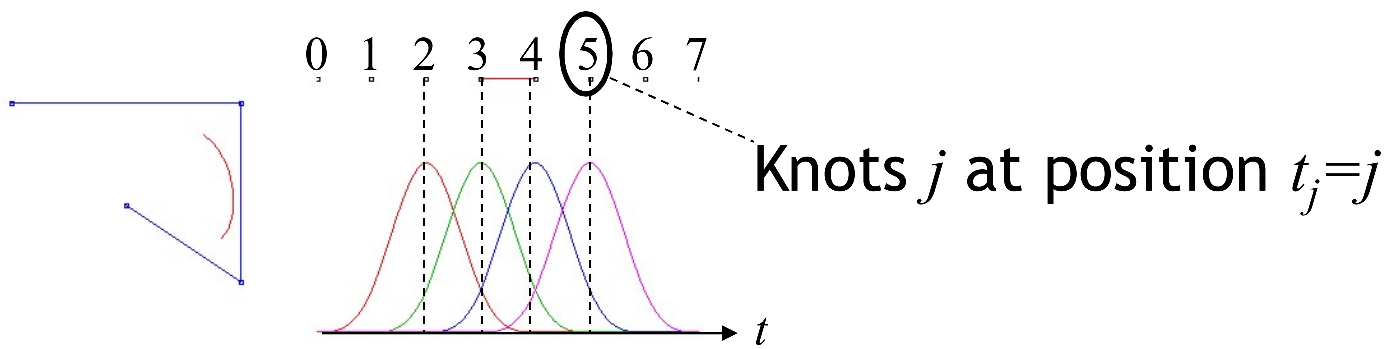
- Defines B-spline bases functions
- Uniform B-spline bases and Bernstein polynomials are special cases for specific knot vectors

Advantage

- Can make corners if desired (not possible with uniform b-splines)

Knot vector

- Knot vector is vector of locations $\{t_j\}$ on the t axis
  - B-spline function of degree n uses $n+2$ knots
- (Uniform) B-splines use a uniform knot vector $t_j=j$

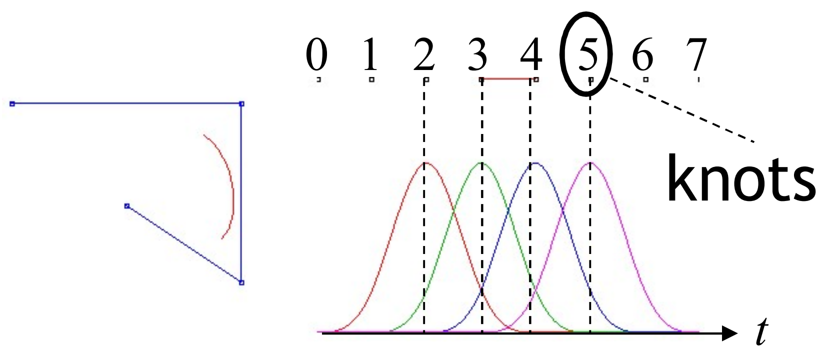


Uniform knot vector

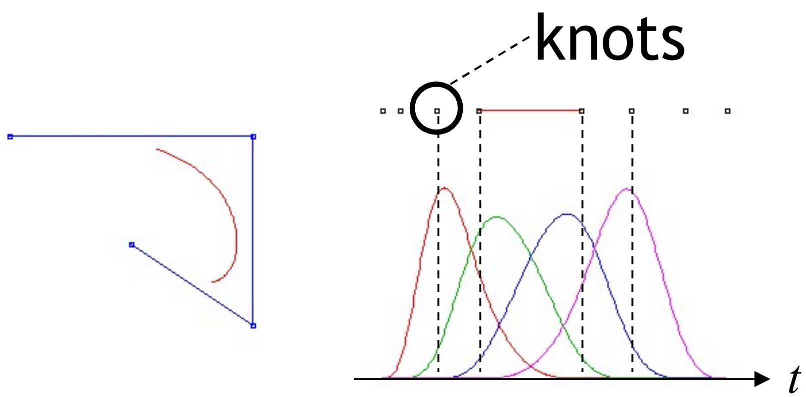
http://ibiblio.org/e-notes/Splines/basis.html

Knot vector

- Nonuniform B-splines use an arbitrary knot vector



Uniform knot vector



Nonuniform knot vector

http://ibiblio.org/e-notes/Splines/basis.html

Nonuniform B-spline bases

Construction using knot vector

- Recursive
- Generate higher order bases step by step from lower order bases
- Can prove
  - Partition of unity (implies convex hull property, i.e., curve is always in convex hull of control points, https://en.wikipedia.org/wiki/Convex_hull)
  - Built-in continuity

Recursive construction

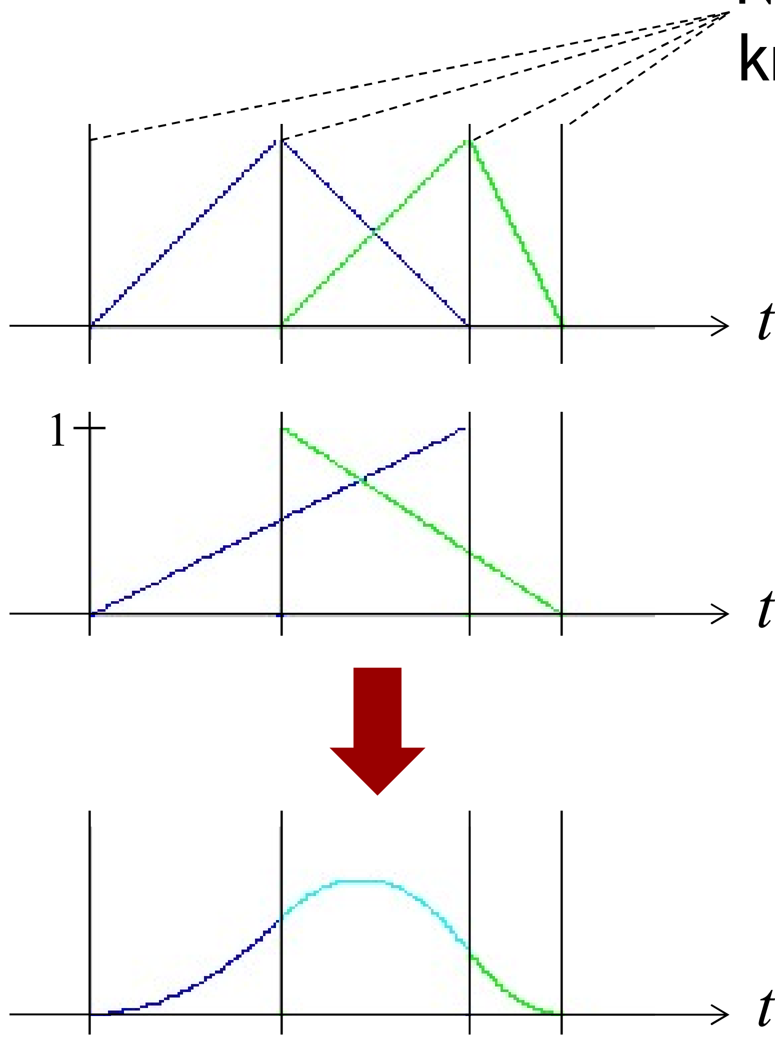
Nonuniform knot vector

Nonuniform, linear B-spline bases

Linear weighting function

Multiply & add

Quadratic B-spline basis



Recursive construction

Recipe

- Input: two neighboring basis functions of degree n
  - Multiply basis functions with linear weighting functions (one increasing, one decreasing)
  - Add
- Output: one basis function of degree $n+1$

For your reference...

- Recursive definition of non-uniform B-spline basis functions $b_{j,n}$
  - Function $b_{j,n}$ has degree n
  - Knot vector $\{t_j\}$

$$b_{j,0}(t) := \begin{cases} 1 & \text{if} \quad t_j \le t < t_{j+1} \\ 0 & \text{otherwise} \end{cases}$$

Basis functions of degree 0

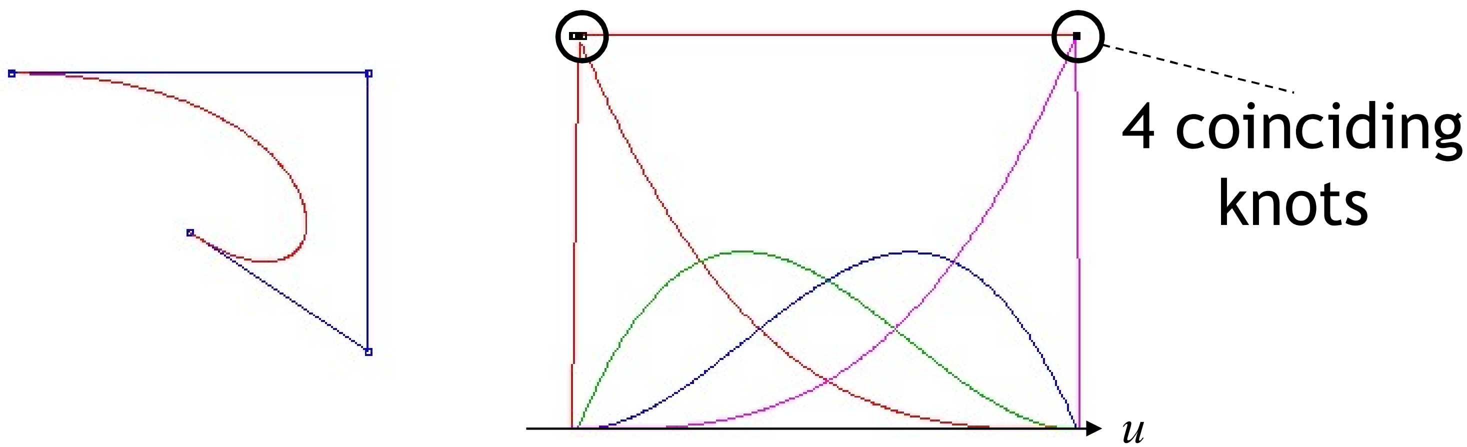
$$b_{j,n}(t) := \frac{t - t_j}{t_{j+n} - t_j} b_{j,n-1}(t) + \frac{t_{j+n+1} - t}{t_{j+n+1} - t_{j+1}} b_{j+1,n-1}(t).$$

Recursive definition of higher order functions

http://en.wikipedia.org/wiki/B-spline

Special cases

- Uniform B-splines have knot vector $t_j=j$
- Cubic Bézier curves $\{t_j\} = [0,0,0,0,1,1,1,1]$
  - Can make corners (C^1 discontinuity)
  - Non-uniform knot vectors allow mixing interpolating (e.g. at endpoints) and approximating



Bézier curve as B-spline with nonuniform knot vector

http://www.ibiblio.org/e-notes/Splines/basis.html

Generalization: NURBS

http://en.wikipedia.org/wiki/Non-uniform_rational_B-spline

- **N**on-**U**niform **R**ational **B**-splines
- Interactive explanation
  http://www.ibiblio.org/e-notes/Splines/nurbs.html

- NOTE: notation now uses t instead of u for curve parameter

Rational curves

- Big drawback of all polynomial curves
  - Can't make circles, ellipses, nor arcs, nor conic sections
- Rational B-spline
  - A type of rational function http://en.wikipedia.org/wiki/Rational_function
  - Add a weight to each control point i
  - Control points with homogeneous coordinates w_i

$$\mathbf{x}(u) = \sum_i b_i(u)\mathbf{p}_i$$

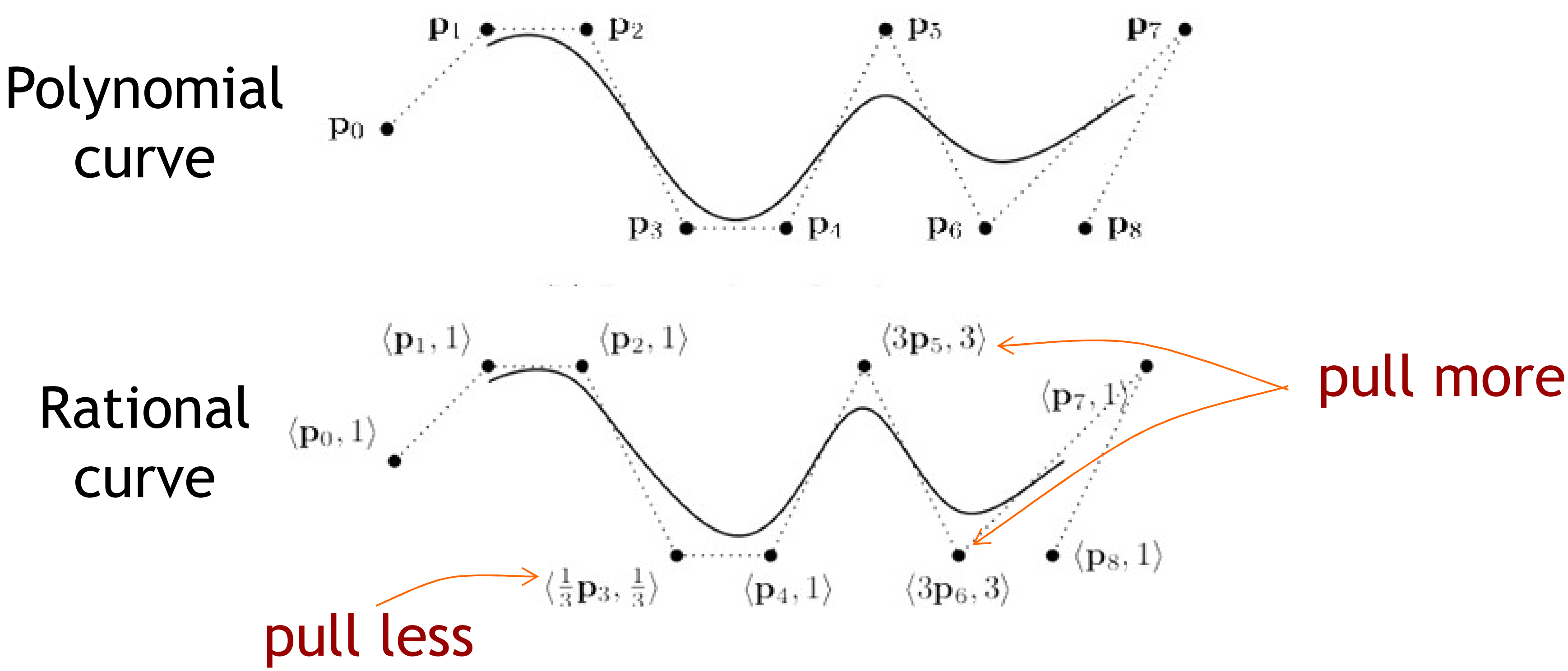
Polynomial curve
(b-spline, Bézier)

$$\mathbf{x}(u) = \frac{\sum_i b_i(u) w_i \mathbf{p}_i}{\sum_i b_i(u) w_i}$$

Rational curve
Not polynomial any more!

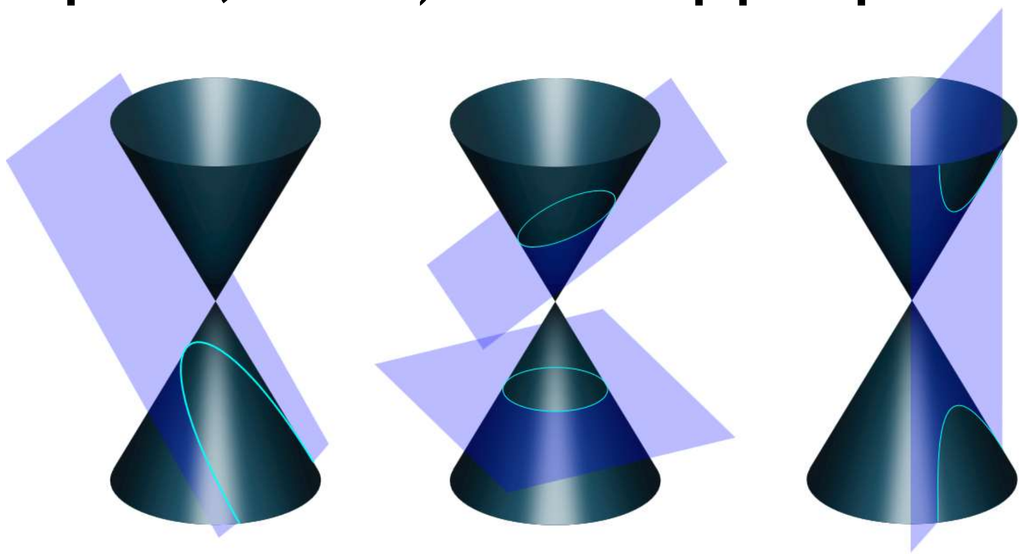
Rational curves

- Weight causes point to “pull” more (or less)
- With proper points & weights, can do circles



Rational curves

- Can generate curves for conic sections (circles, ellipses, etc.) with appropriate weights



http://en.wikipedia.org/wiki/Conic_section

- Need extra user interface to adjust the weights
- Often, hand-drawn curves are unweighted

Summary: NURBS

- Math is more complicated
  - (Non-uniform) knot vectors
  - Rational functions
- Very widely used for curve and surface modeling
  - Supported by virtually all 3D modeling tools
  - Open source modeling tool: http://www.blender.org
- Techniques for cutting, inserting, merging, revolving, etc…
- Applets
  - http://ibiblio.org/e-notes/Splines/Intro.htm

Today

Curves

- NURBS

Surfaces

- Parametric surfaces
- Bilinear patch
- Bicubic Bézier patch
- Advanced surface modeling

Curved surfaces

Curves

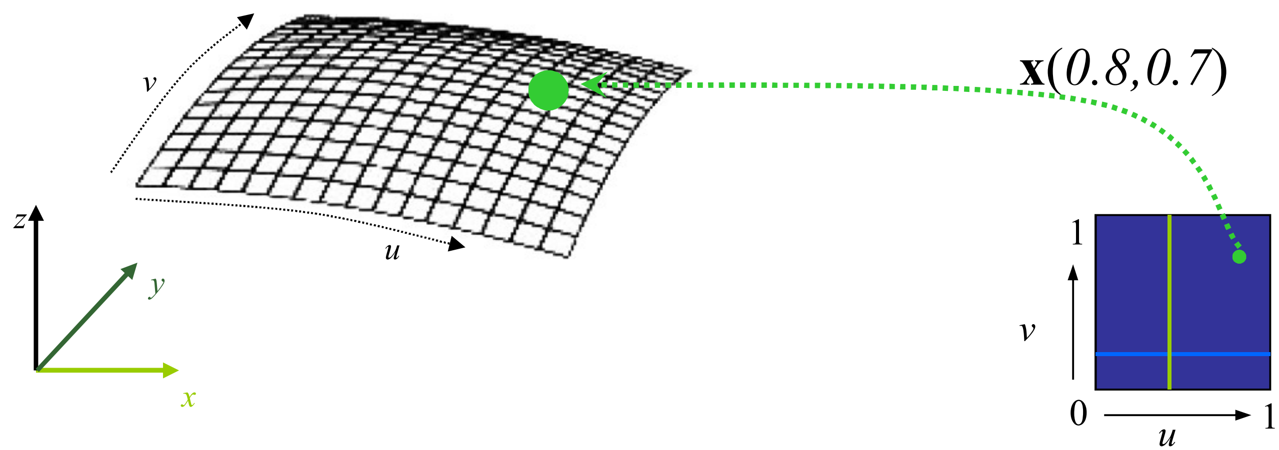
- Described by a 1D series of control points
- A function $\mathbf{x}(t)$
- Segments joined together to form a longer curve

Surfaces

- Described by a 2D mesh of control points
- Parameters have two dimensions (two dimensional parameter domain)
- A function $\mathbf{x}(u,v)$
- Patches joined together to form a bigger surface

Parametric surface patch

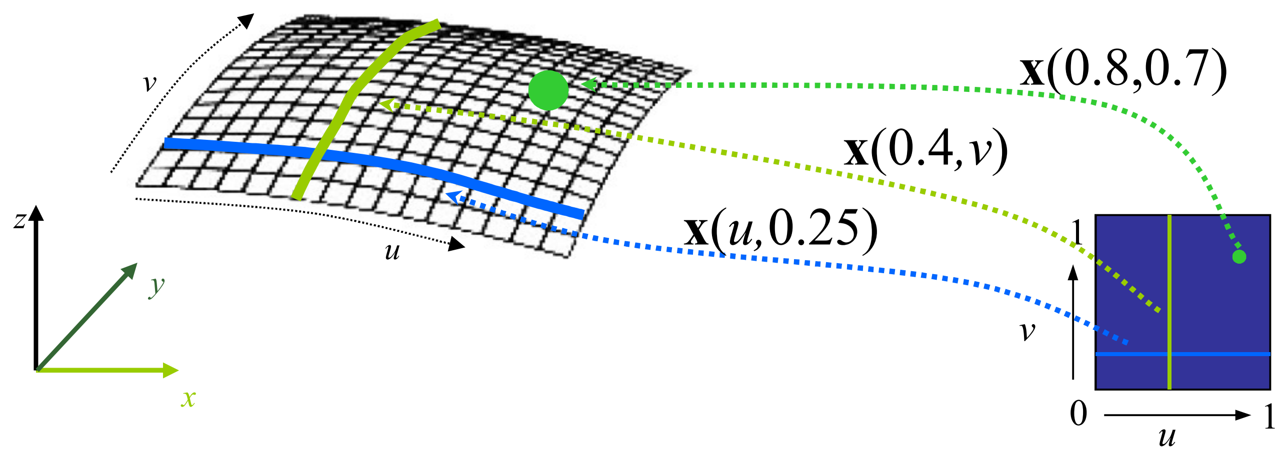
- $\mathbf{x}(u,v)$: $\mathbf{R}^2 \longrightarrow \mathbf{R}^3$ is a point in 3D space for any (u,v) pair
  - u,v each range from 0 to 1 (by convention)



2D parameter domain

Parametric surface patch

- $\mathbf{x}(u,v)$: $\mathbf{R}^2 \longrightarrow \mathbf{R}^3$ is a point in 3D space for any (u,v) pair
  - u,v each range from 0 to 1 (by convention)

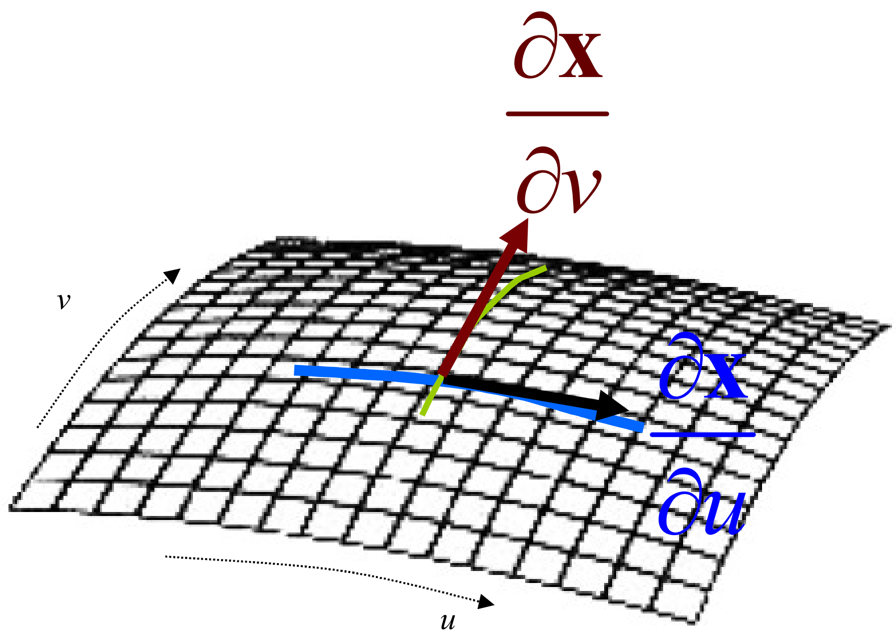


2D parameter domain

- Parametric curves
  - For fixed u_0, have a v curve $\mathbf{x}(u_0,v)$
  - For fixed v_0, have a u curve $\mathbf{x}(u,v_0)$
  - For any point on the surface, there is one pair of parametric curves that go through point

Tangents

- The tangent to a parametric curve is also tangent to the surface
- For any point on the surface, there are a pair of (parametric) tangent vectors given by the partial derivatives with respect to u and v
- Note: not necessarily perpendicular to each other



Tangents

Notation

- Tangent along u direction

$$\frac{\partial \mathbf{x}}{\partial u}(u, v) \quad \text{or} \quad \frac{\partial}{\partial u}\mathbf{x}(u, v) \quad \text{or} \quad \mathbf{x}_u(u, v)$$

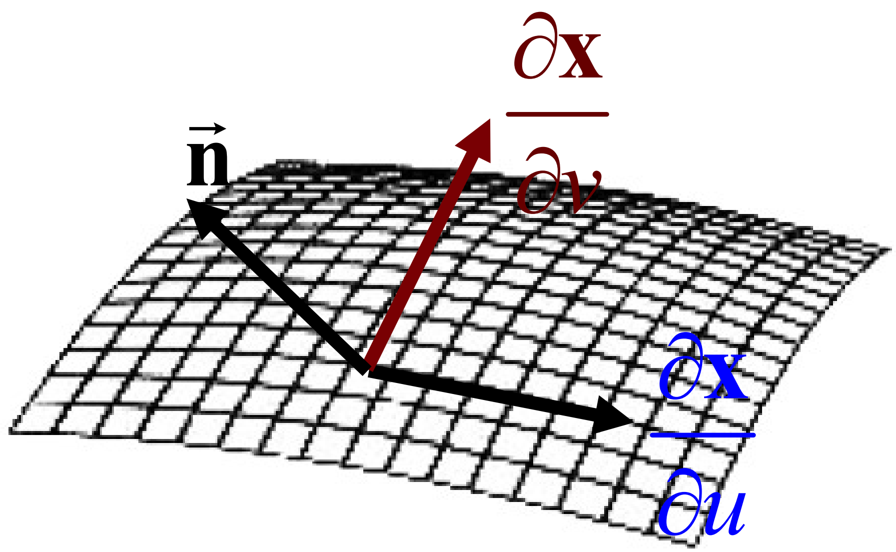
- Tangent along v direction

$$\frac{\partial \mathbf{x}}{\partial v}(u, v) \quad \text{or} \quad \frac{\partial}{\partial v}\mathbf{x}(u, v) \quad \text{or} \quad \mathbf{x}_v(u, v)$$

- Tangents are vector valued functions, i.e., vectors!

Surface normal

- Cross product of the two tangent vectors $\mathbf{x}_u(u,v) \times \mathbf{x}_v(u,v)$
- Order matters (determines normal orientation)
- Usually, want unit normal
  - Need to normalize by dividing through length



Today

Curves

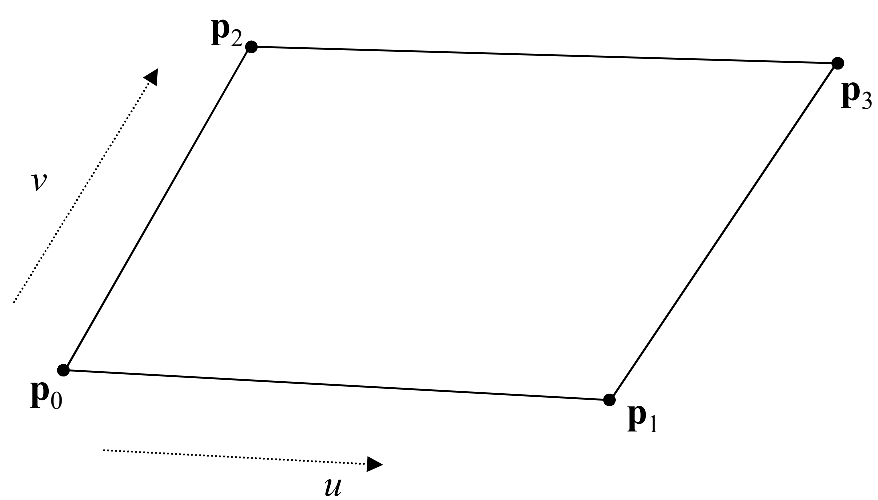
- NURBS

Surfaces

- Parametric surfaces
- Bilinear patch
- Bicubic Bézier patch
- Advanced surface modeling

Bilinear patch

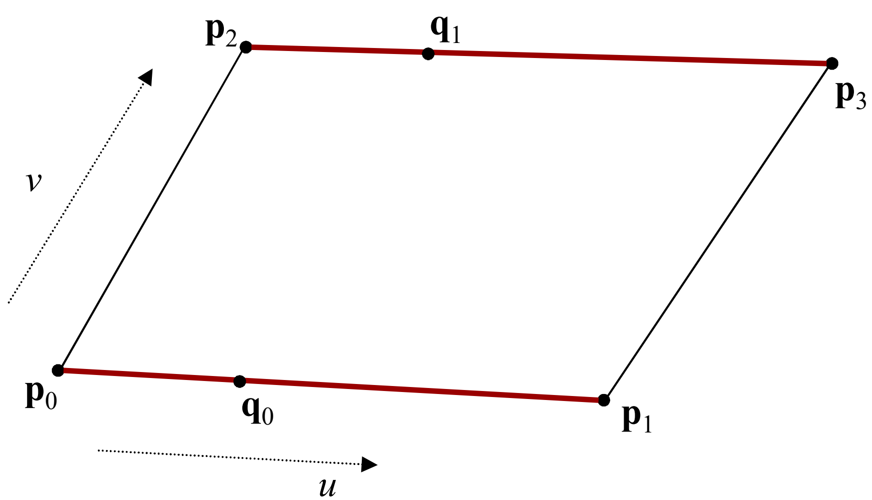
- Control mesh with four points $\mathbf{p_0}$, $\mathbf{p_1}$, $\mathbf{p_2}$, $\mathbf{p_3}$
- Compute $\mathbf{x}(u,v)$ using a two-step construction



Bilinear patch (step 1)

- For a given value of u, evaluate the linear curves on the two u-direction edges
- Use the same value u for both:

$$\mathbf{q_1} = Lerp(u, \mathbf{p_2}, \mathbf{p_3})$$

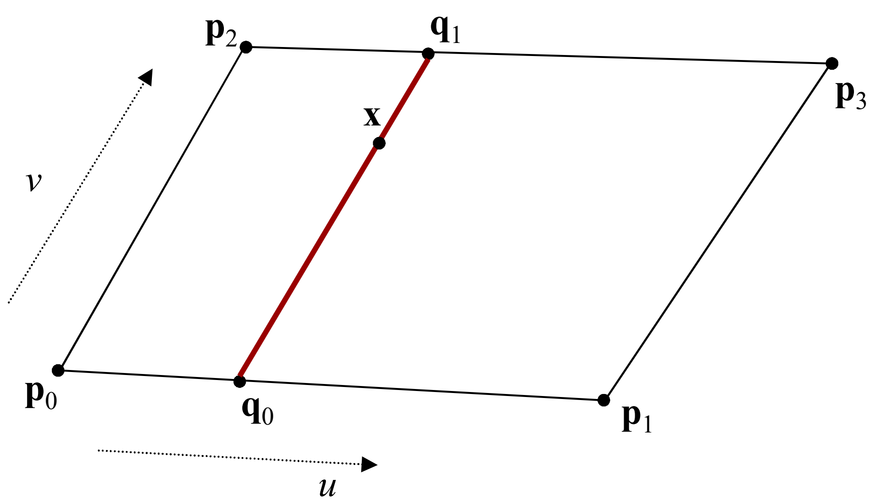


$$\mathbf{q_0} = Lerp(u, \mathbf{p_0}, \mathbf{p_1})$$

Bilinear patch (step 2)

- Consider that $\mathbf{q}_0$, $\mathbf{q}_1$ define a line segment
- Evaluate it using v to get $\mathbf{x}$

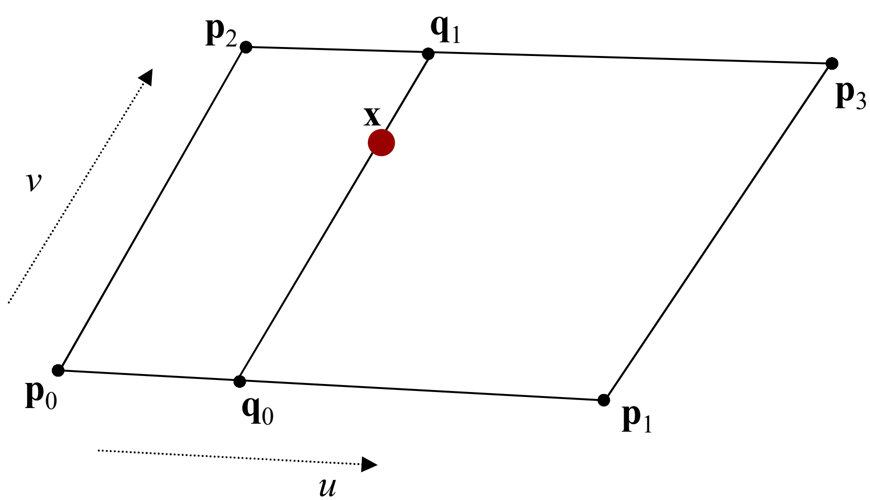
$$\mathbf{x} = Lerp(v, \mathbf{q}_0, \mathbf{q}_1)$$



Bilinear patch

- Combining the steps, we get the full formula

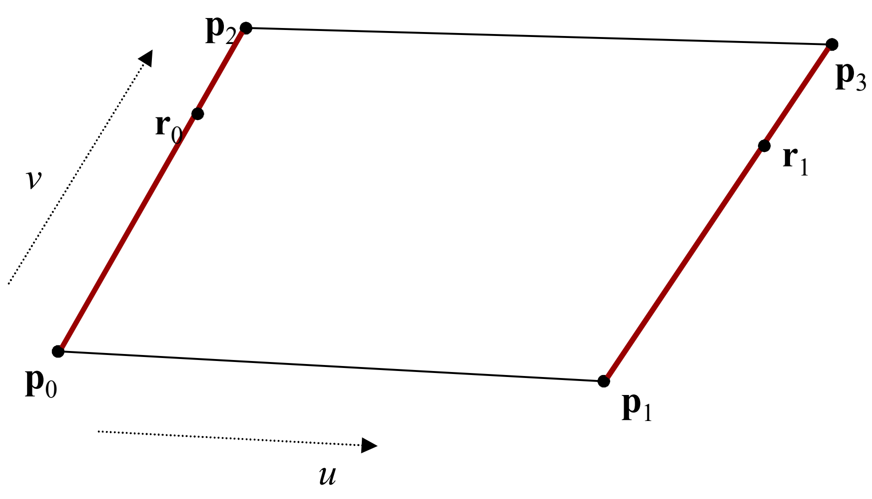
$$\mathbf{x}(u,v) = Lerp(v, Lerp(u, \mathbf{p}_0, \mathbf{p}_1), Lerp(u, \mathbf{p}_2, \mathbf{p}_3))$$



Bilinear patch

- Try the other order
- Evaluate first in the v direction

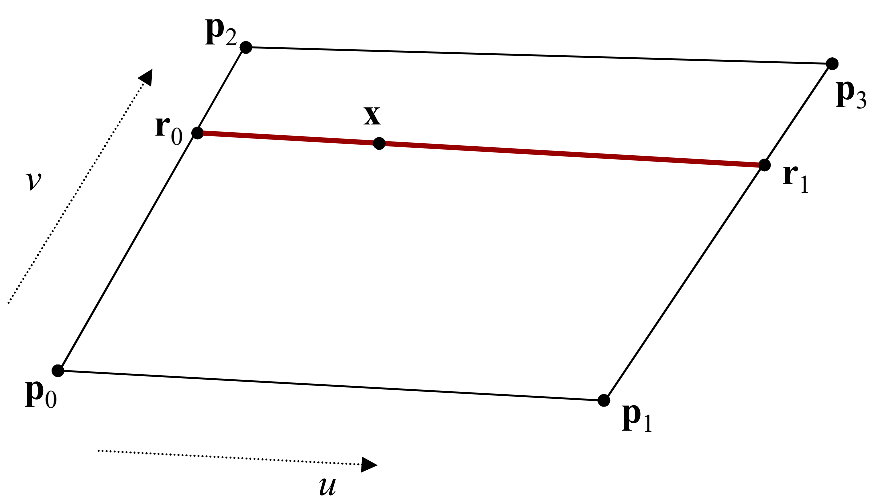
$$\mathbf{r}_0 = Lerp(v, \mathbf{p}_0, \mathbf{p}_2) \qquad \mathbf{r}_1 = Lerp(v, \mathbf{p}_1, \mathbf{p}_3)$$



Bilinear patch

- Consider that $\mathbf{r_0}$, $\mathbf{r_1}$ define a line segment
- Evaluate it using u to get $\mathbf{x}$

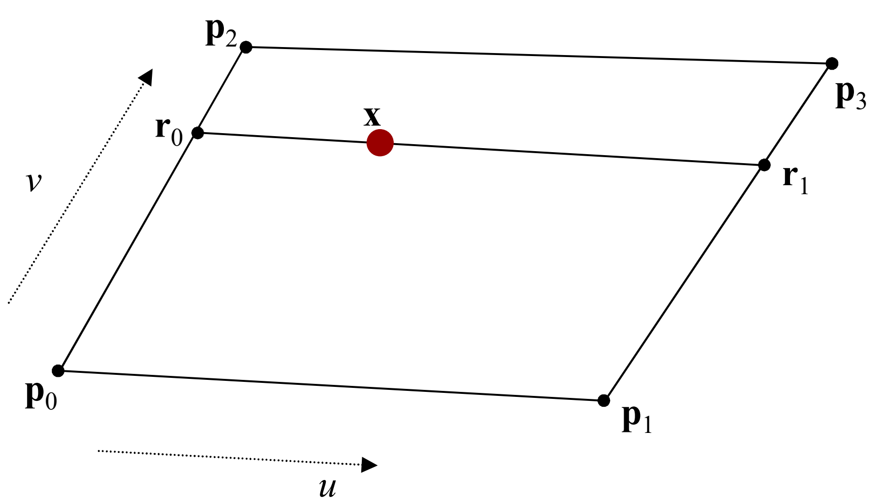
$$\mathbf{x} = Lerp(u, \mathbf{r}_0, \mathbf{r}_1)$$



Bilinear patch

- The full formula for the v direction first:

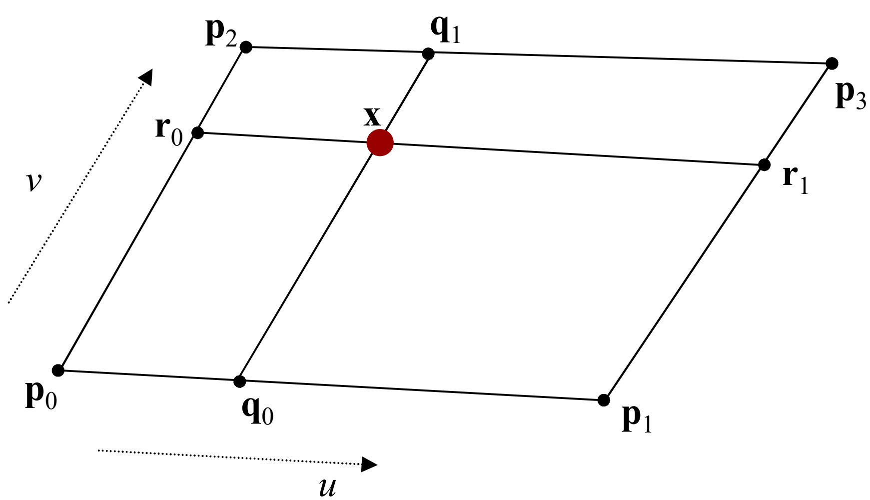
$$\mathbf{x}(u,v) = Lerp(u, Lerp(v, \mathbf{p}_0, \mathbf{p}_2), Lerp(v, \mathbf{p}_1, \mathbf{p}_3))$$



Bilinear patch

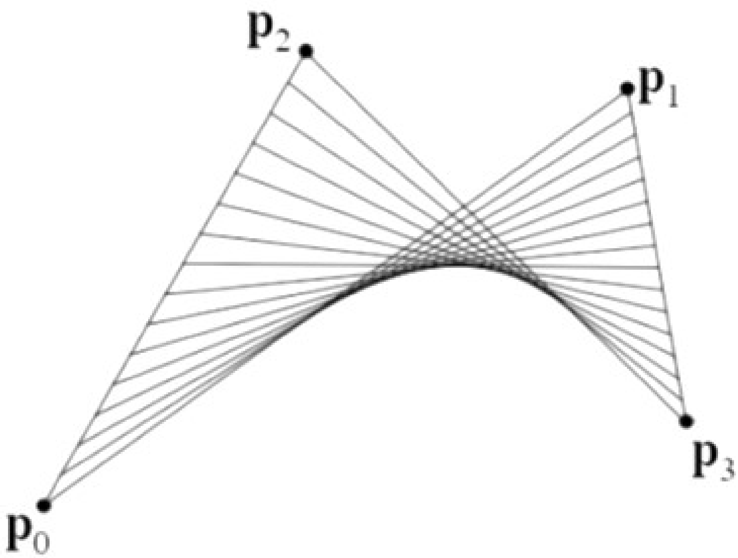
- It works out the same either way!

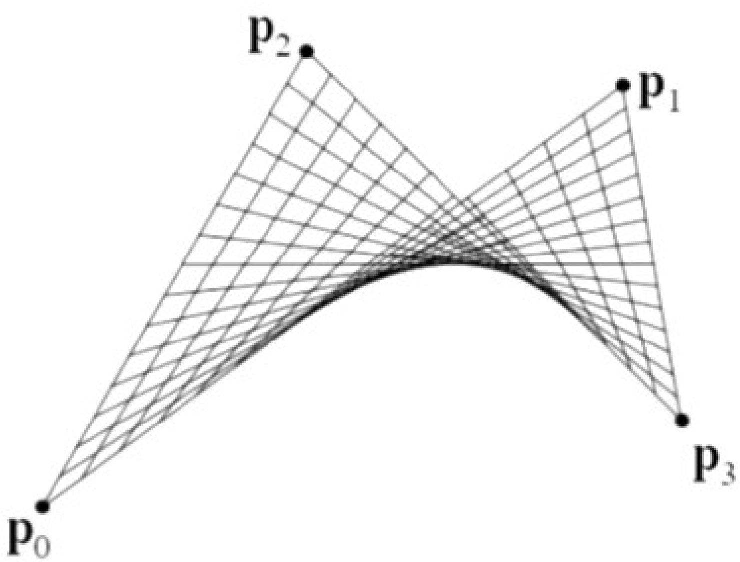
$$\boxed{\begin{aligned}\mathbf{x}(u,v) &= Lerp(v, Lerp(u,\mathbf{p}_0,\mathbf{p}_1), Lerp(u,\mathbf{p}_2,\mathbf{p}_3))\\ \mathbf{x}(u,v) &= Lerp(u, Lerp(v,\mathbf{p}_0,\mathbf{p}_2), Lerp(v,\mathbf{p}_1,\mathbf{p}_3))\end{aligned}}$$



Bilinear patch

- Visualization





Bilinear patches

- Weighted sum of control points

$$\mathbf{x}(u,v) = (1-u)(1-v)\mathbf{p}_0 + u(1-v)\mathbf{p}_1 + (1-u)v\mathbf{p}_2 + uv\mathbf{p}_3$$

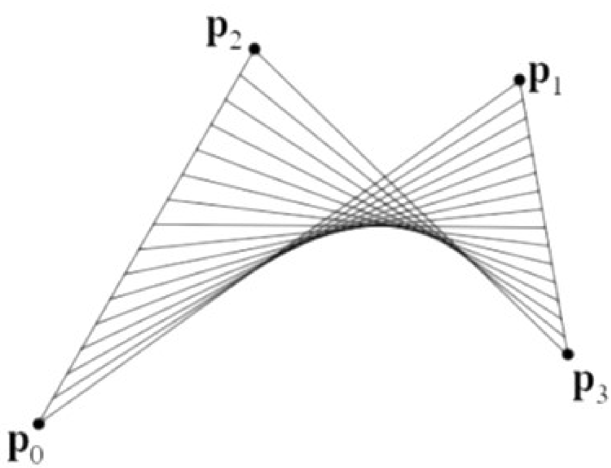
- Bilinear polynomial

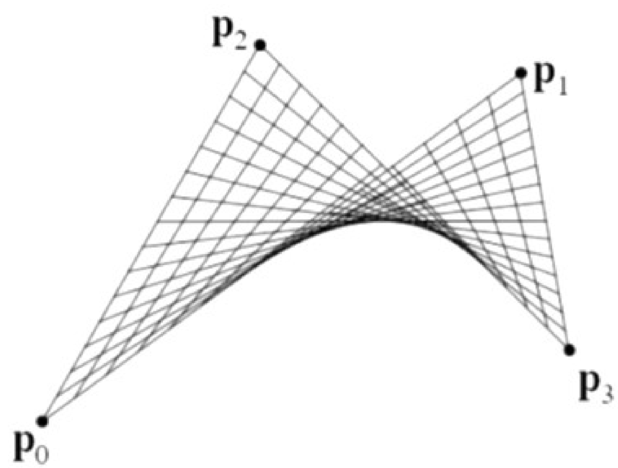
$$\mathbf{x}(u,v) = (\mathbf{p}_0 - \mathbf{p}_1 - \mathbf{p}_2 + \mathbf{p}_3)uv + (\mathbf{p}_1 - \mathbf{p}_0)u + (\mathbf{p}_2 - \mathbf{p}_0)v + \mathbf{p}_0$$

- Matrix form exists, too

Properties

- Interpolates the control points
- The boundaries are straight line segments
- If all 4 points of the control mesh are co-planar, the patch is flat
- If the points are not coplanar, get a curved surface
  - saddle shape, AKA hyperbolic paraboloid
- The parametric curves are all straight line segments!
  - a (doubly) *ruled surface:* has (two) straight lines through every point





- Not terribly useful as a modeling primitive

Today

Curves

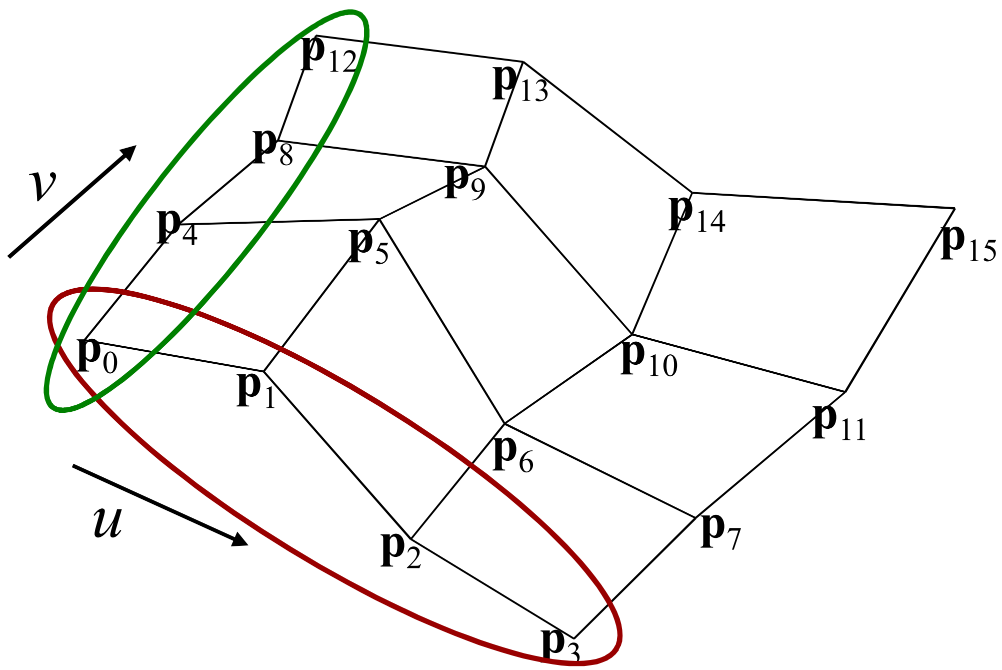
- NURBS

Surfaces

- Parametric surfaces
- Bilinear patch
- Bicubic Bézier patch
- Advanced surface modeling

Bicubic Bézier patch

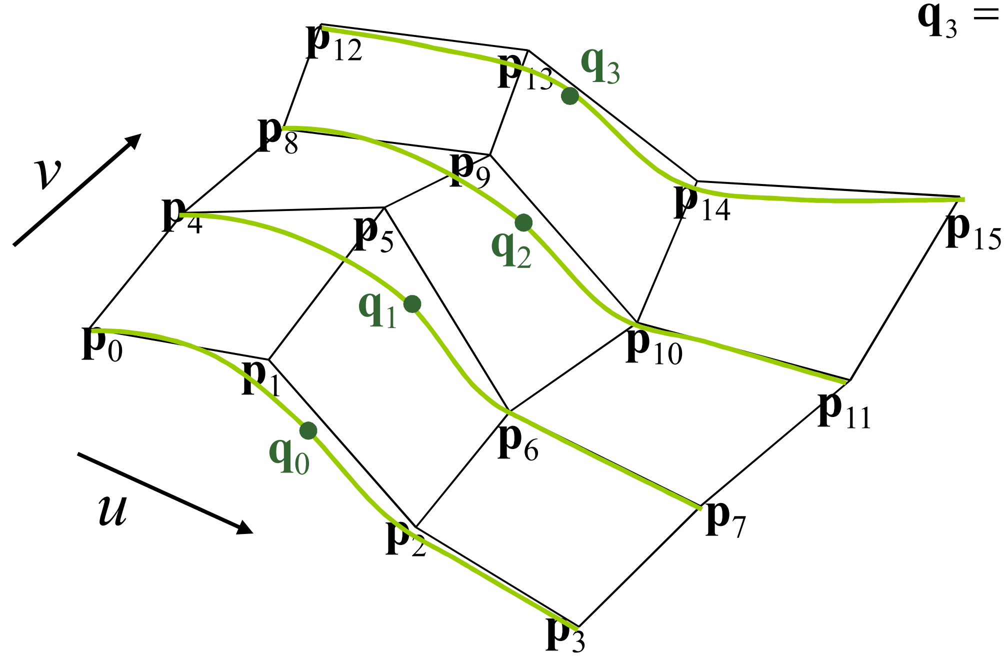
- Grid of 4x4 control points, $\mathbf{p}_0$ through $\mathbf{p}_{15}$
- Four rows of control points define Bézier curves along u
  $\mathbf{p}_0,\mathbf{p}_1,\mathbf{p}_2,\mathbf{p}_3$; $\mathbf{p}_4,\mathbf{p}_5,\mathbf{p}_6,\mathbf{p}_7$; $\mathbf{p}_8,\mathbf{p}_9,\mathbf{p}_{10},\mathbf{p}_{11}$; $\mathbf{p}_{12},\mathbf{p}_{13},\mathbf{p}_{14},\mathbf{p}_{15}$
- Four columns define Bézier curves along v
  $\mathbf{p}_0,\mathbf{p}_4,\mathbf{p}_8,\mathbf{p}_{12}$; $\mathbf{p}_1,\mathbf{p}_6,\mathbf{p}_9,\mathbf{p}_{13}$; $\mathbf{p}_2,\mathbf{p}_6,\mathbf{p}_{10},\mathbf{p}_{14}$; $\mathbf{p}_3,\mathbf{p}_7,\mathbf{p}_{11},\mathbf{p}_{15}$



Bicubic Bézier patch (step 1)

- Evaluate four u-direction Bézier curves at u
- Get intermediate points $\mathbf{q}_0 \ldots \mathbf{q}_3$

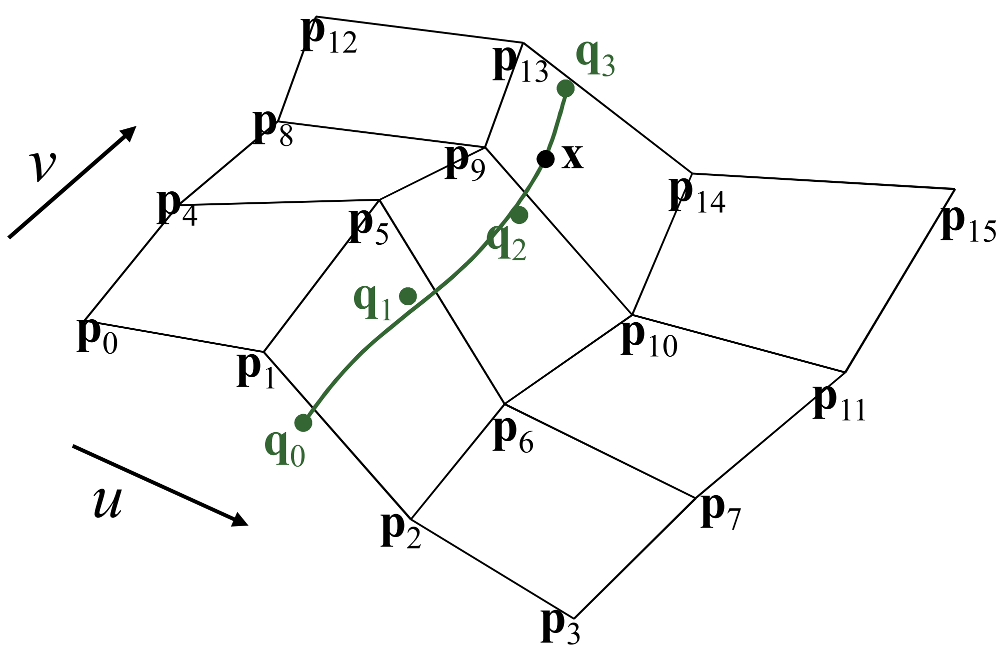
$$\mathbf{q}_0 = Bez(u, \mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3)$$
$$\mathbf{q}_1 = Bez(u, \mathbf{p}_4, \mathbf{p}_5, \mathbf{p}_6, \mathbf{p}_7)$$
$$\mathbf{q}_2 = Bez(u, \mathbf{p}_8, \mathbf{p}_9, \mathbf{p}_{10}, \mathbf{p}_{11})$$
$$\mathbf{q}_3 = Bez(u, \mathbf{p}_{12}, \mathbf{p}_{13}, \mathbf{p}_{14}, \mathbf{p}_{15})$$



Bicubic Bézier patch (step 2)

- Points $\mathbf{q}_0 \dots \mathbf{q}_3$ define a Bézier curve
- Evaluate it at v

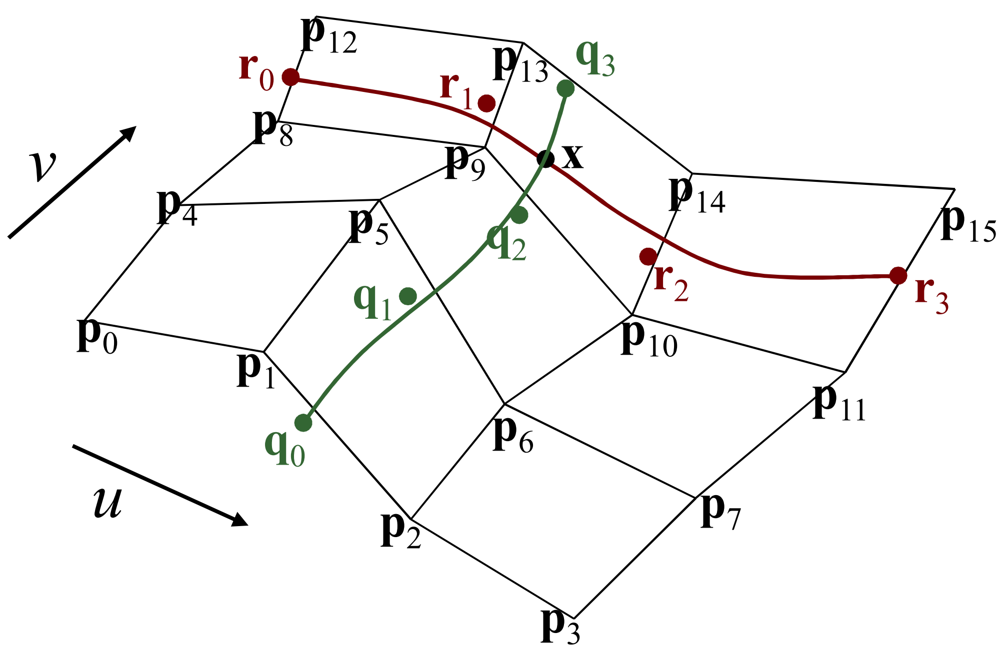
$$\mathbf{x}(u,v) = Bez(v, \mathbf{q}_0, \mathbf{q}_1, \mathbf{q}_2, \mathbf{q}_3)$$



Bicubic Bézier patch

- Same result in either order (evaluate u before v or vice versa)

$$
\begin{aligned}
\mathbf{q}_0 &= Bez(u, \mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3) \\
\mathbf{q}_1 &= Bez(u, \mathbf{p}_4, \mathbf{p}_5, \mathbf{p}_6, \mathbf{p}_7) \\
\mathbf{q}_2 &= Bez(u, \mathbf{p}_8, \mathbf{p}_9, \mathbf{p}_{10}, \mathbf{p}_{11}) \\
\mathbf{q}_3 &= Bez(u, \mathbf{p}_{12}, \mathbf{p}_{13}, \mathbf{p}_{14}, \mathbf{p}_{15}) \\
\mathbf{x}(u,v) &= Bez(v, \mathbf{q}_0, \mathbf{q}_1, \mathbf{q}_2, \mathbf{q}_3)
\end{aligned}
\quad \Leftrightarrow \quad
\begin{aligned}
\mathbf{r}_0 &= Bez(v, \mathbf{p}_0, \mathbf{p}_4, \mathbf{p}_8, \mathbf{p}_{12}) \\
\mathbf{r}_1 &= Bez(v, \mathbf{p}_1, \mathbf{p}_5, \mathbf{p}_9, \mathbf{p}_{13}) \\
\mathbf{r}_2 &= Bez(v, \mathbf{p}_2, \mathbf{p}_6, \mathbf{p}_{10}, \mathbf{p}_{14}) \\
\mathbf{r}_3 &= Bez(v, \mathbf{p}_3, \mathbf{p}_7, \mathbf{p}_{11}, \mathbf{p}_{15}) \\
\mathbf{x}(u,v) &= Bez(u, \mathbf{r}_0, \mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3)
\end{aligned}
$$



Tensor product formulation

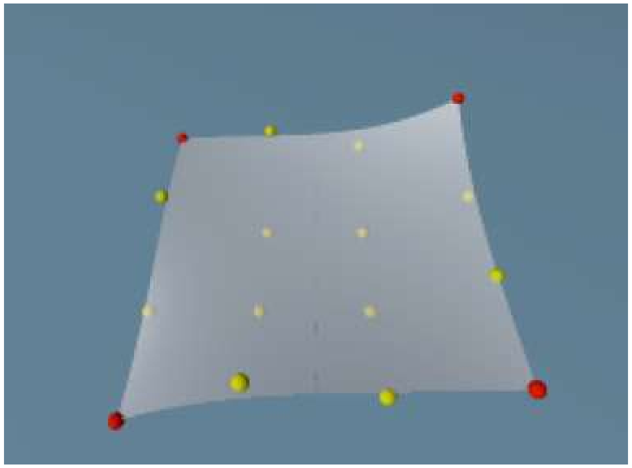
- Corresponds to weighted average formulation
- Construct two-dimensional weighting function as product of two one-dimensional functions
  - Bernstein polynomials B_i, B_j as for curves

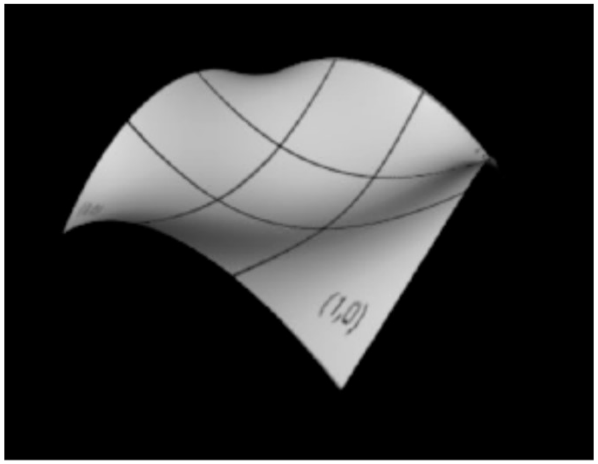
$$\mathbf{x}(u,v) = \sum_i \sum_j \mathbf{p}_{i,j} B_i(u) B_j(v)$$

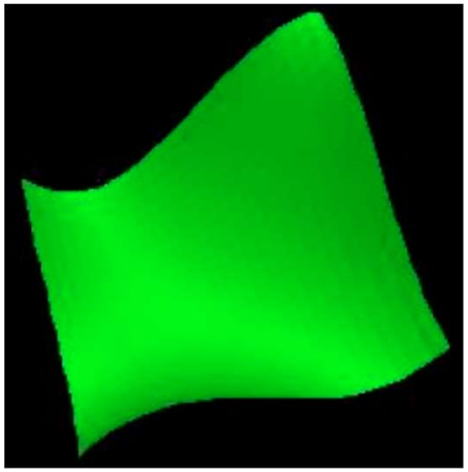
- Same tensor product construction applies to higher order Bézier and NURBS surfaces

Bicubic Bézier patch: properties

- Convex hull: any point on the surface will fall within the convex hull of the control points
- Interpolates 4 corner points
- Approximates other 12 points, which act as “handles”
- The boundaries of the patch are the Bézier curves defined by the points on the mesh edges
- The parametric curves are all Bézier curves

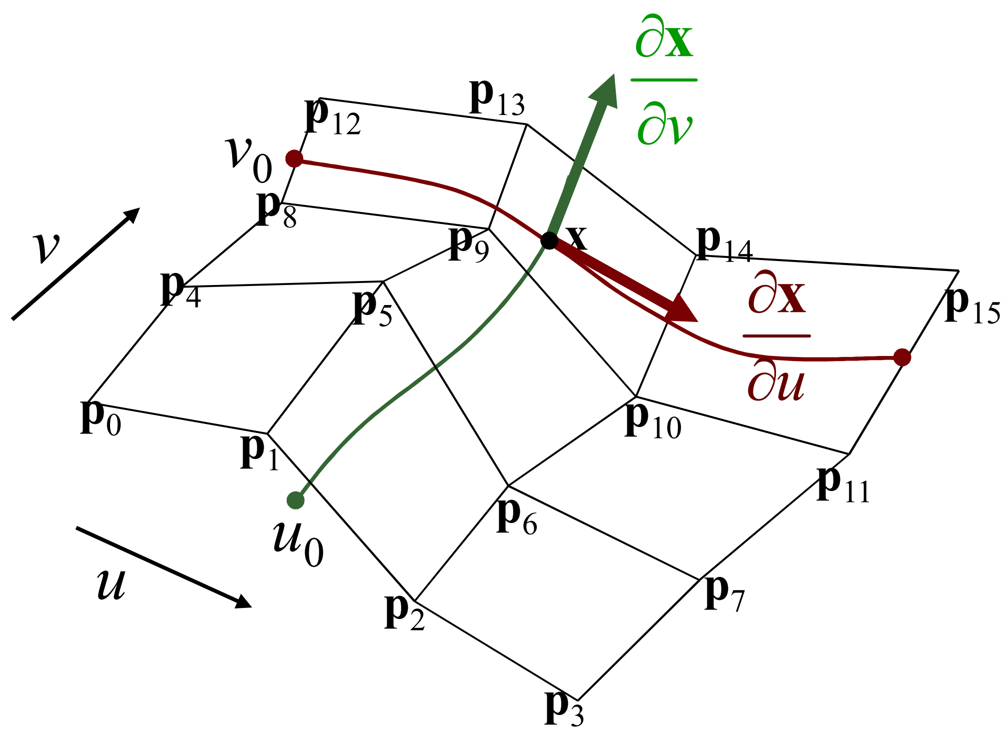






Tangents of Bézier patch

- Remember parametric curves $\mathbf{x}(u,v_0)$, $\mathbf{x}(u_0,v)$ where v_0, u_0 is fixed
- Tangents to surface = tangents to parametric curves
- Tangents are partial derivatives of $\mathbf{x}(u,v)$
- Normal is cross product of the tangents



Tangents of Bézier patch

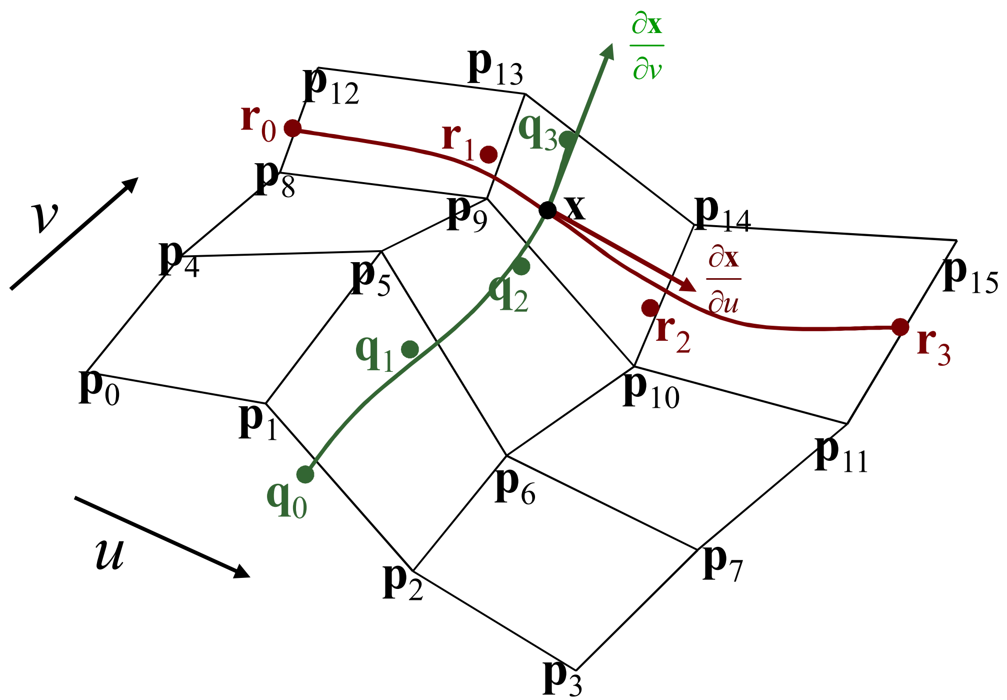
$$\mathbf{q}_0 = Bez(u,\mathbf{p}_0,\mathbf{p}_1,\mathbf{p}_2,\mathbf{p}_3) \qquad \mathbf{r}_0 = Bez(v,\mathbf{p}_0,\mathbf{p}_4,\mathbf{p}_8,\mathbf{p}_{12})$$

$$\mathbf{q}_1 = Bez(u,\mathbf{p}_4,\mathbf{p}_5,\mathbf{p}_6,\mathbf{p}_7) \qquad \mathbf{r}_1 = Bez(v,\mathbf{p}_1,\mathbf{p}_5,\mathbf{p}_9,\mathbf{p}_{13})$$

$$\mathbf{q}_2 = Bez(u,\mathbf{p}_8,\mathbf{p}_9,\mathbf{p}_{10},\mathbf{p}_{11}) \qquad \mathbf{r}_2 = Bez(v,\mathbf{p}_2,\mathbf{p}_6,\mathbf{p}_{10},\mathbf{p}_{14})$$

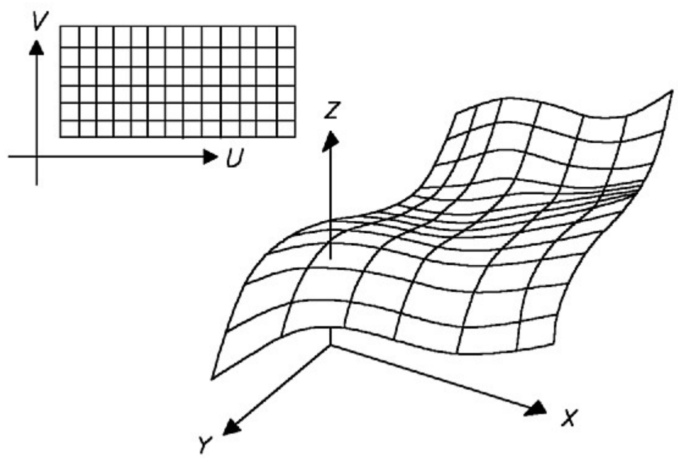
$$\mathbf{q}_3 = Bez(u,\mathbf{p}_{12},\mathbf{p}_{13},\mathbf{p}_{14},\mathbf{p}_{15}) \qquad \mathbf{r}_3 = Bez(v,\mathbf{p}_3,\mathbf{p}_7,\mathbf{p}_{11},\mathbf{p}_{15})$$

$$\frac{\partial \mathbf{x}}{\partial v}(u,v) = Bez'(v,\mathbf{q}_0,\mathbf{q}_1,\mathbf{q}_2,\mathbf{q}_3) \qquad \frac{\partial \mathbf{x}}{\partial u}(u,v) = Bez'(u,\mathbf{r}_0,\mathbf{r}_1,\mathbf{r}_2,\mathbf{r}_3)$$



Tessellating a Bézier patch

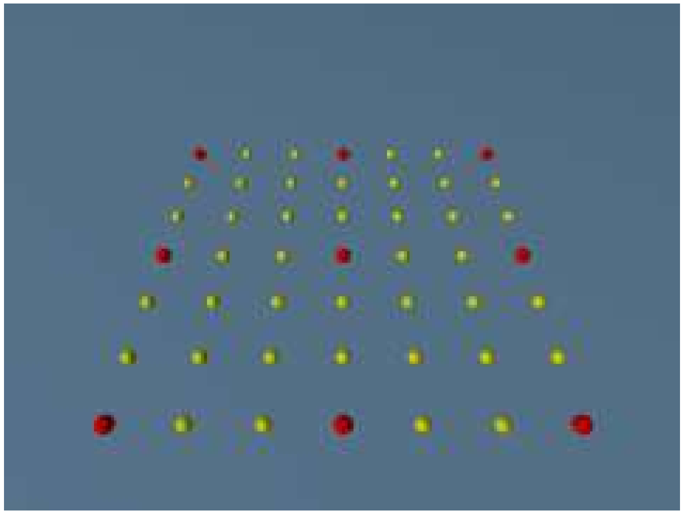
- Uniform tessellation is most straightforward
  - Evaluate points on uniform grid of u, v coordinates
  - Compute tangents at each point, take cross product to get per-vertex normal
  - Draw triangle strips (several choices of direction)



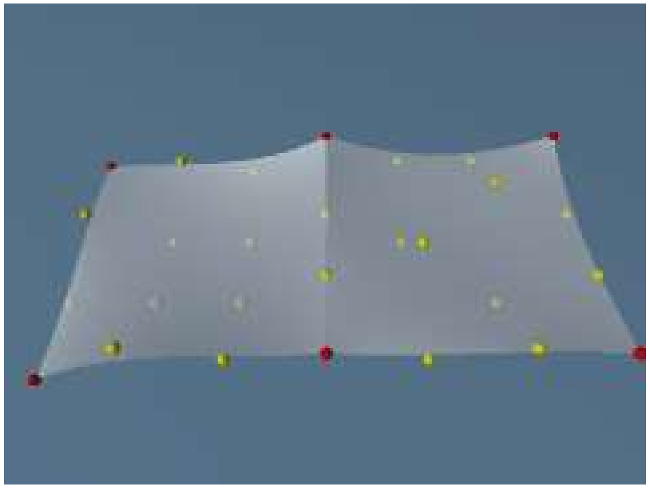
- Adaptive tessellation/recursive subdivision
  - Potential for “cracks” if patches on opposite sides of an edge divide differently
  - Tricky to get right, but can be done

Piecewise Bézier surface

- Lay out grid of adjacent meshes of control points
- For C^0 continuity, must share points on the edge
  - Each edge of a Bézier patch is a Bézier curve based only on the edge mesh points
  - So if adjacent meshes share edge points, the patches will line up exactly
- But we have a crease…



Grid of control points

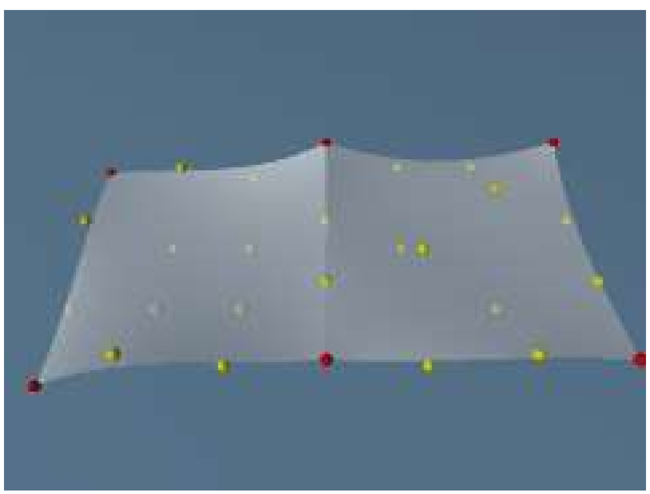


Piecewise Bézier surface

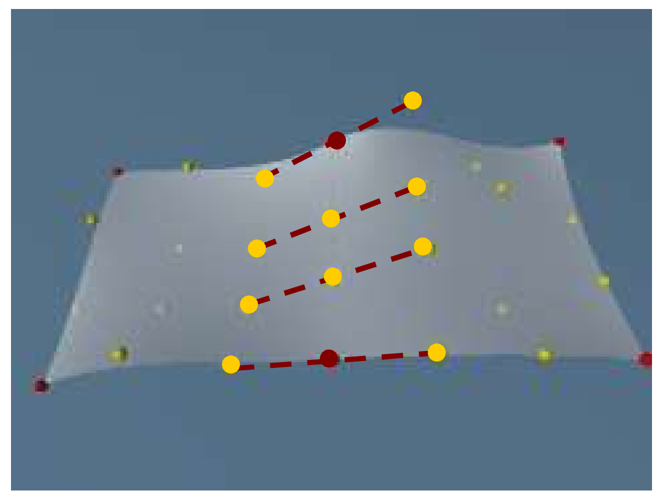
C^1 continuity

- Want parametric curves that cross each edge to have C^1 continuity
  - Handles must be equal-and-opposite across edge

C^0 continuous



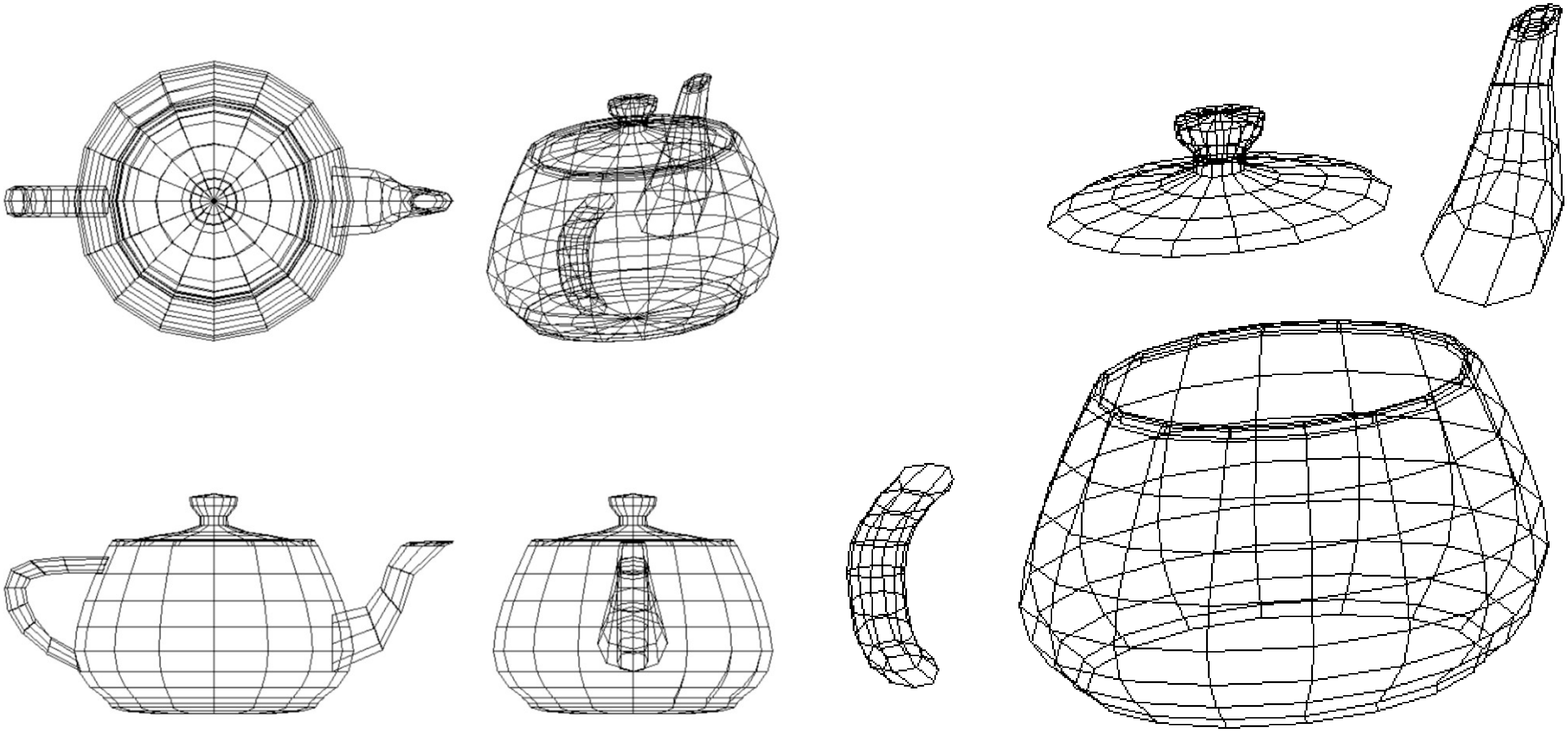
C^1 continuous



[http://www.spiritone.com/~english/cyclopedia/patches.html]

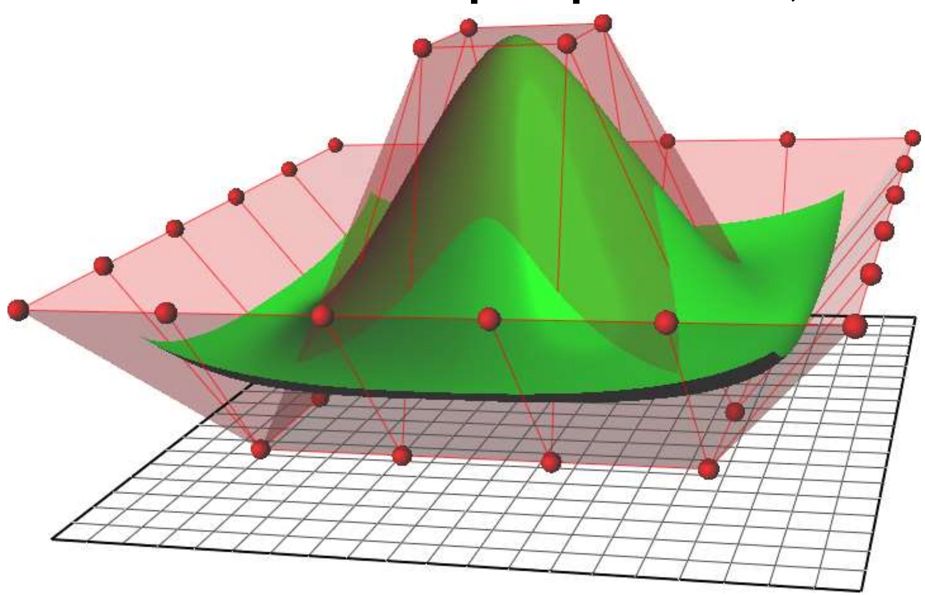
Modeling with Bézier patches

- Original Utah teapot specified as Bézier Patches
  http://en.wikipedia.org/wiki/Utah_teapot



B-spline/NURBS surfaces

- B-spline/NURBS patches instead of Bézier
- For the same reason as using B-spline/NURBS curves
  - More flexible (can model spheres)
  - Better mathematical properties, continuity



4th order NURBS patch

http://de.wikipedia.org/wiki/Non-Uniform_Rational_B-Spline

Today

Curves

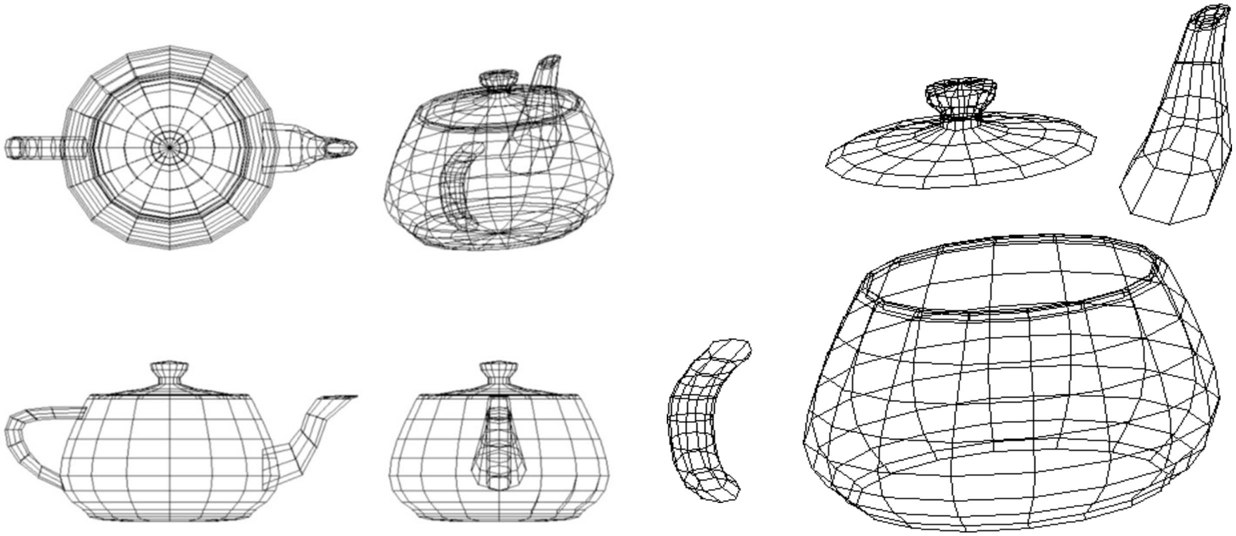
- NURBS

Surfaces

- Parametric surfaces
- Bilinear patch
- Bicubic Bézier patch
- Advanced surface modeling

Modeling headaches

- Original Teapot is not "watertight"
  http://en.wikipedia.org/wiki/Utah_teapot
  - Spout & handle intersect with body
  - No bottom
  - Hole in spout
  - Gap between lid and body



Quadrilateral topology

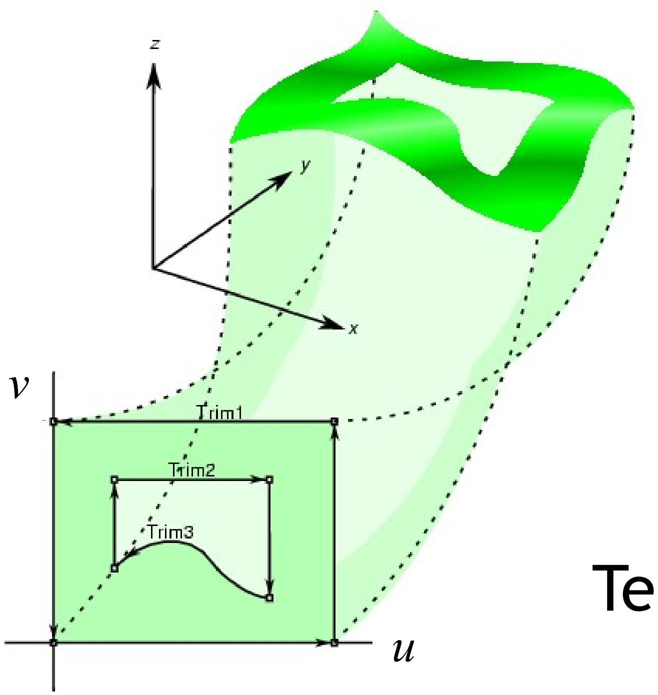
- Surfaces made up of quadrilateral patches
  - 4 corners
  - 4 (curved) boundaries

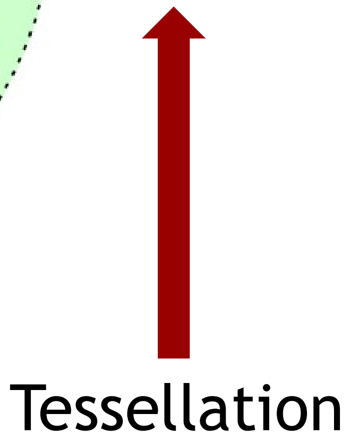
Makes it hard to

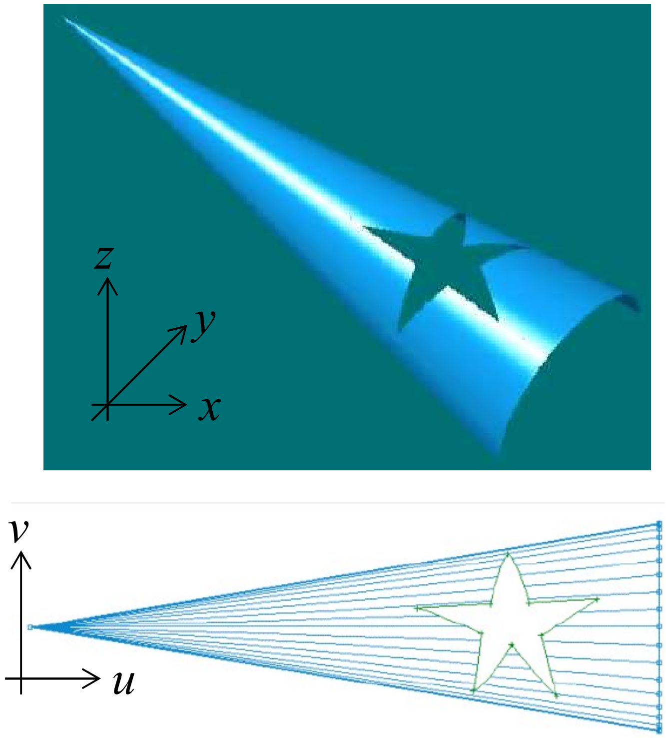
- join or abut curved pieces
- build surfaces with awkward topology or structure

Trim curves

- Cut away part of surface
- Define “holes” with trim curves in *u/v* domain
- Tessellation uses trim curve to define surface
- Still hard to fit different parts together





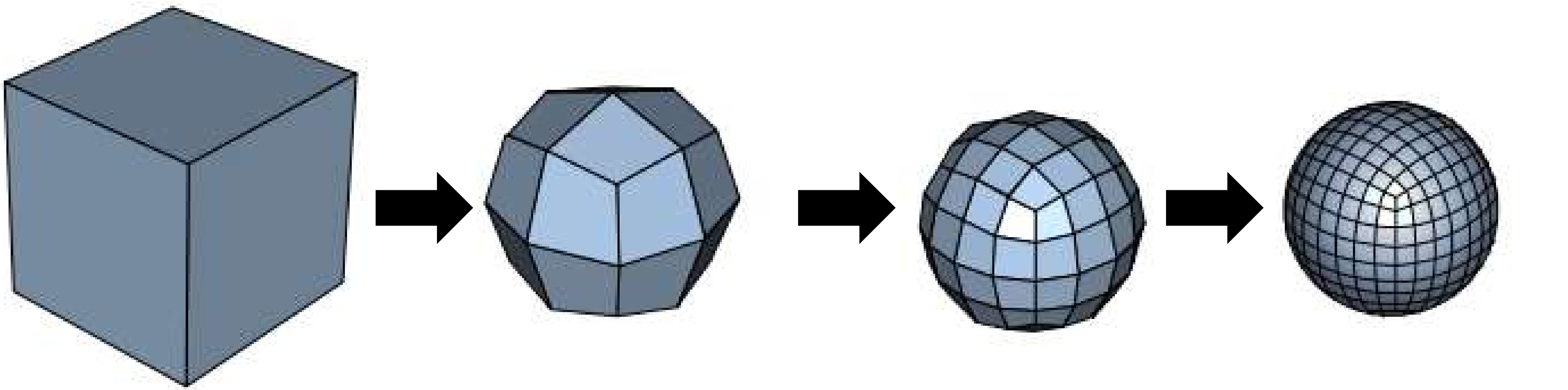


Subdivision surfaces

- Goal
  - Create smooth surfaces from small number of control points, like splines
  - More flexibility for the topology of the control points (not restricted to quadrilateral grid)
- Idea
  - Start with initial coarse polygon mesh
  - Create smooth surface recursively by
    1. Splitting (subdividing) mesh into finer polygons
    2. Smoothing the vertices of the polygons
    3. Repeat from 1.

Subdivision surfaces

http://en.wikipedia.org/wiki/Catmull%E2%80%93Clark_subdivision_surface

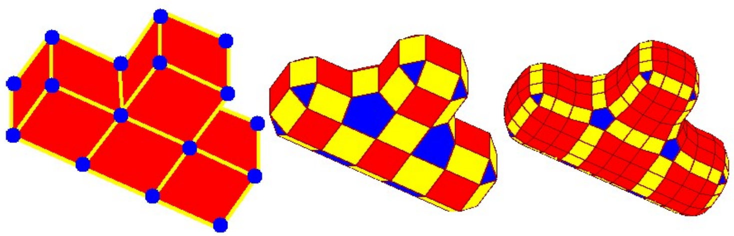


Input mesh | Subdivision & smoothing | Subdivision & smoothing | Subdivision & smoothing

Limit surface

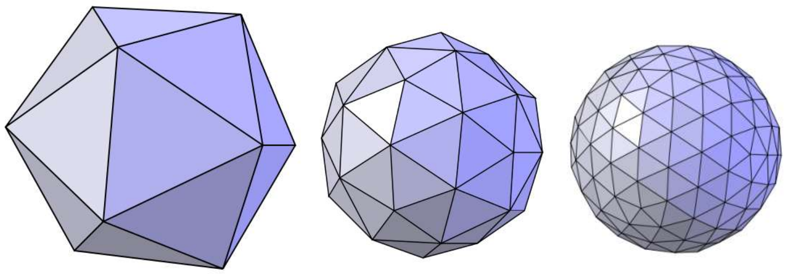
Subdivision schemes

- Various schemes available to subdivide and smooth



Doo-Sabin

http://en.wikipedia.org/wiki/Doo%E2%80%93Sabin_subdivision_surface



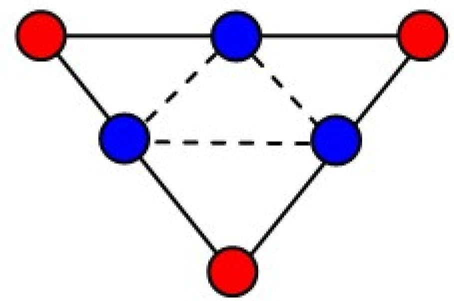
Loop

http://en.wikipedia.org/wiki/Loop_subdivision_surface

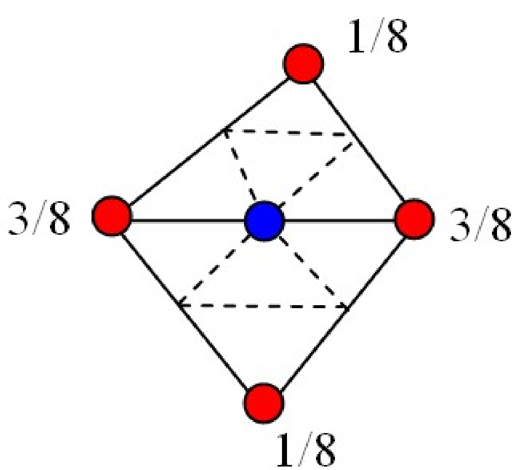
- All provide certain guarantees for smoothness of limit surface

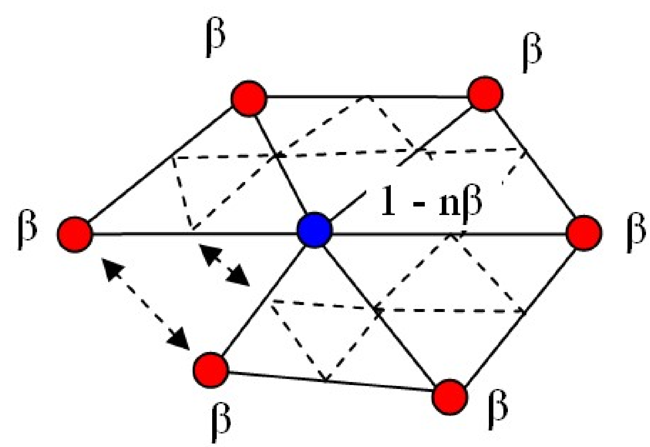
Loop subdivision

- Subdivision
  - Split each triangle into four



- Smoothing
  - New vertex positions as weighted average of neighbors
  - Different cases





Cases for β:

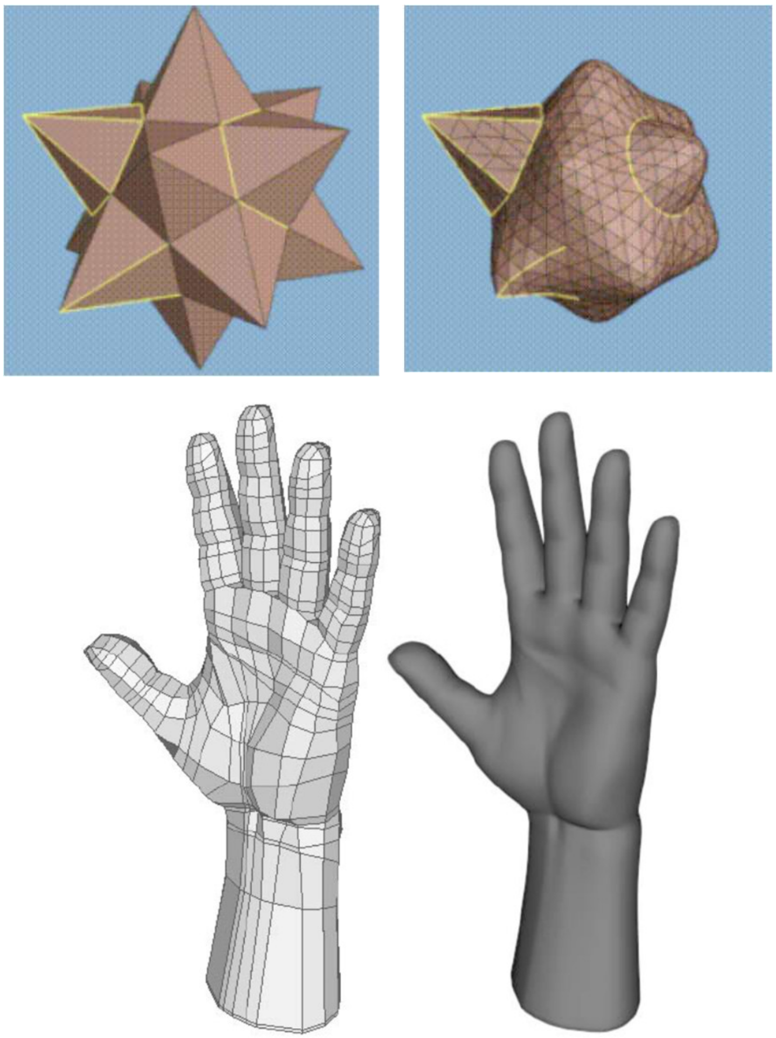
$$\beta = \begin{cases} \dfrac{3}{8n} & n > 3 \\[2ex] \dfrac{3}{16} & n = 3 \end{cases}$$

Number of neighbors n

http://graphics.stanford.edu/~mdfisher/subdivision.html

Subdivision surfaces

- Arbitrary mesh of control points
- Arbitrary topology or connectivity
  - Not restricted to quadrilateral topology
  - No global u,v parameters
- Work by recursively subdividing mesh faces
- Used in particular for character animation
  - One surface rather than collection of patches
  - Can deform geometry without creating cracks



Subdivision surfaces

Next time

- Implementing subdivision surfaces