CSE 167:
Introduction to Computer Graphics
Lecture #11: Surface Patches

Jürgen P. Schulze, Ph.D.
University of California, San Diego
Fall Quarter 2014

Announcements

- Project 4 due tomorrow at 3:30pm
  - Don't forget to upload source code to Ted by 3:30pm!

Siggraph Video Showing

- > I wanted to make sure you and your colleagues and students knew
  > about our upcoming special event, a showing of the SIGGRAPH 2014
  > Computer Animation Festival down at the Reuben Fleet,
  > **6:30 PM Saturday evening, November 15.**
  >
  > Full info, including the program, is on our web site at
  > http://san-diego.siggraph.org
  >
  > Tickets are priced "below cost" at $3 to make sure no one has to
  > miss this for financial reasons.
  >
  > We're hoping to have a good turnout; tell anyone you know who
  > might be interested. If you are able to forward this email
  > to others, please do. Hope you can come, put your feet up,
  > and enjoy the show with other local enthusiasts.
  >
  > Best, Mike Pique 858.354.4391

Bézier Curve Properties

Overview:

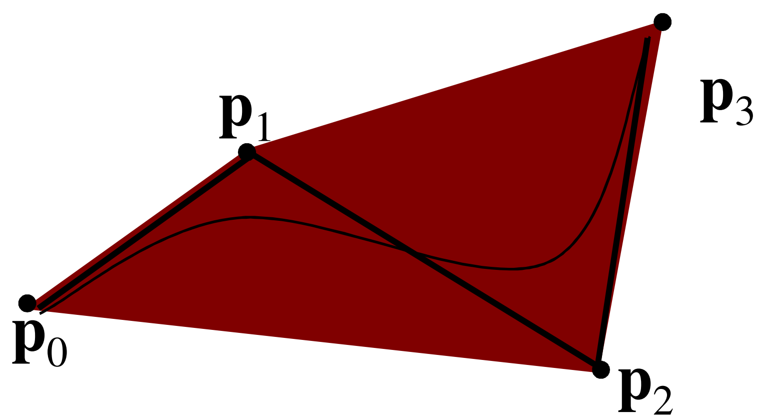
- Convex Hull property
- Affine Invariance

Definitions

- Convex hull of a set of points:
  - Polyhedral volume created such that all lines connecting any two points lie completely inside it (or on its boundary)
- Convex combination of a set of points:
  - Weighted average of the points, where all weights between 0 and 1, sum up to 1
- Any convex combination of a set of points lies within the convex hull

Convex Hull Property

- A Bézier curve is a convex combination of the control points (by definition, see Bernstein polynomials)
- A Bézier curve is always inside the convex hull
  - Makes curve predictable
  - Allows culling, intersection testing, adaptive tessellation
- Demo: http://www.cs.princeton.edu/~min/cs426/jar/bezier.html



Affine Invariance

Transforming Bézier curves

- Two ways to transform:
  - Transform the control points, then compute resulting spline points
  - Compute spline points, then transform them
- Either way, we get the same points
  - Curve is defined via affine combination of points
  - Invariant under affine transformations (i.e., translation, scale, rotation, shear)
  - Convex hull property remains true

Cubic Polynomial Form

Start with Bernstein form:

$$\mathbf{x}(t) = \left(-t^3 + 3t^2 - 3t + 1\right)\mathbf{p}_0 + \left(3t^3 - 6t^2 + 3t\right)\mathbf{p}_1 + \left(-3t^3 + 3t^2\right)\mathbf{p}_2 + \left(t^3\right)\mathbf{p}_3$$

Regroup into coefficients of t :

$$\mathbf{x}(t) = \left(-\mathbf{p}_0 + 3\mathbf{p}_1 - 3\mathbf{p}_2 + \mathbf{p}_3\right)t^3 + \left(3\mathbf{p}_0 - 6\mathbf{p}_1 + 3\mathbf{p}_2\right)t^2 + \left(-3\mathbf{p}_0 + 3\mathbf{p}_1\right)t + \left(\mathbf{p}_0\right)1$$

$$\mathbf{x}(t) = \mathbf{a}t^3 + \mathbf{b}t^2 + \mathbf{c}t + \mathbf{d}$$

$$\mathbf{a} = \left(-\mathbf{p}_0 + 3\mathbf{p}_1 - 3\mathbf{p}_2 + \mathbf{p}_3\right)$$
$$\mathbf{b} = \left(3\mathbf{p}_0 - 6\mathbf{p}_1 + 3\mathbf{p}_2\right)$$
$$\mathbf{c} = \left(-3\mathbf{p}_0 + 3\mathbf{p}_1\right)$$
$$\mathbf{d} = \left(\mathbf{p}_0\right)$$

- Good for fast evaluation
  - Precompute constant coefficients (**a**,**b**,**c**,**d**)
- Not much geometric intuition

Cubic Matrix Form

$$\mathbf{x}(t) = \begin{bmatrix} \vec{\mathbf{a}} & \vec{\mathbf{b}} & \vec{\mathbf{c}} & \mathbf{d} \end{bmatrix} \begin{bmatrix} t^3 \\ t^2 \\ t \\ 1 \end{bmatrix}$$

$$\vec{\mathbf{a}} = \left(-\mathbf{p}_0 + 3\mathbf{p}_1 - 3\mathbf{p}_2 + \mathbf{p}_3\right)$$
$$\vec{\mathbf{b}} = \left(3\mathbf{p}_0 - 6\mathbf{p}_1 + 3\mathbf{p}_2\right)$$
$$\vec{\mathbf{c}} = \left(-3\mathbf{p}_0 + 3\mathbf{p}_1\right)$$
$$\mathbf{d} = \left(\mathbf{p}_0\right)$$

$$\mathbf{x}(t) = \underbrace{\begin{bmatrix} \mathbf{p}_0 & \mathbf{p}_1 & \mathbf{p}_2 & \mathbf{p}_3 \end{bmatrix}}_{\mathbf{G}_{Bez}} \underbrace{\begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}}_{\mathbf{B}_{Bez}} \underbrace{\begin{bmatrix} t^3 \\ t^2 \\ t \\ 1 \end{bmatrix}}_{\mathbf{T}}$$

- Other types of cubic splines use different basis matrices $\mathbf{B}_{\mathrm{Bez}}$

Cubic Matrix Form

- In 3D: 3 equations for x, y and z:

$$\mathbf{x}_x(t) = \begin{bmatrix} p_{0x} & p_{1x} & p_{2x} & p_{3x} \end{bmatrix} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} t^3 \\ t^2 \\ t \\ 1 \end{bmatrix}$$

$$\mathbf{x}_y(t) = \begin{bmatrix} p_{0y} & p_{1y} & p_{2y} & p_{3y} \end{bmatrix} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} t^3 \\ t^2 \\ t \\ 1 \end{bmatrix}$$

$$\mathbf{x}_z(t) = \begin{bmatrix} p_{0z} & p_{1z} & p_{2z} & p_{3z} \end{bmatrix} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} t^3 \\ t^2 \\ t \\ 1 \end{bmatrix}$$

Matrix Form

- Bundle into a single matrix

$$\mathbf{x}(t) = \begin{bmatrix} p_{0x} & p_{1x} & p_{2x} & p_{3x} \\ p_{0y} & p_{1y} & p_{2y} & p_{3y} \\ p_{0z} & p_{1z} & p_{2z} & p_{3z} \end{bmatrix} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} t^3 \\ t^2 \\ t \\ 1 \end{bmatrix}$$

$$\mathbf{x}(t) = \mathbf{G}_{Bez}\mathbf{B}_{Bez}\mathbf{T}$$
$$\mathbf{x}(t) = \mathbf{C}\ \mathbf{T}$$

- Efficient evaluation
  - Pre-compute $\mathbf{C}$
  - Take advantage of existing 4x4 matrix hardware support

Lecture Overview

- Polynomial Curves
  - Introduction
  - Polynomial functions
- Bézier Curves
  - Introduction
  - Drawing Bézier curves
  - Piecewise Bézier curves

Drawing Bézier Curves

- Draw *line segments* or individual pixels
- Approximate the curve as a series of line segments (*tessellation*)
  - Uniform sampling
  - Adaptive sampling
  - Recursive subdivision

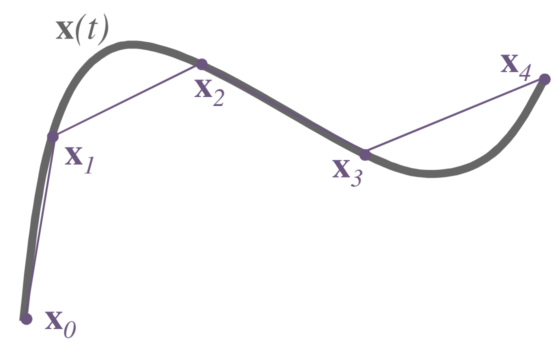
Uniform Sampling

- Approximate curve with N straight segments
  - N chosen in advance
  - Evaluate

$$\mathbf{x}_i = \mathbf{x}(t_i) \text{ where } t_i = \frac{i}{N} \text{ for } i = 0, 1, \ldots, N$$

$$\mathbf{x}_i = \vec{\mathbf{a}}\frac{i^3}{N^3} + \vec{\mathbf{b}}\frac{i^2}{N^2} + \vec{\mathbf{c}}\frac{i}{N} + \mathbf{d}$$

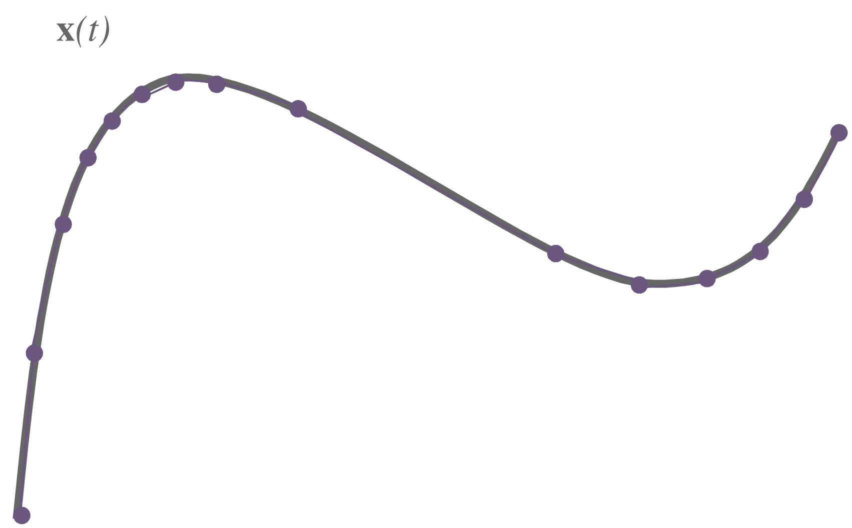
  - Connect the points with lines
- Too few points?
  - Poor approximation
  - "Curve" is faceted
- Too many points?
  - Slow to draw too many line segments
  - Segments may draw on top of each other



UCSD

Adaptive Sampling

- Use only as many line segments as you need
  - Fewer segments where curve is mostly flat
  - More segments where curve bends
  - Segments never smaller than a pixel

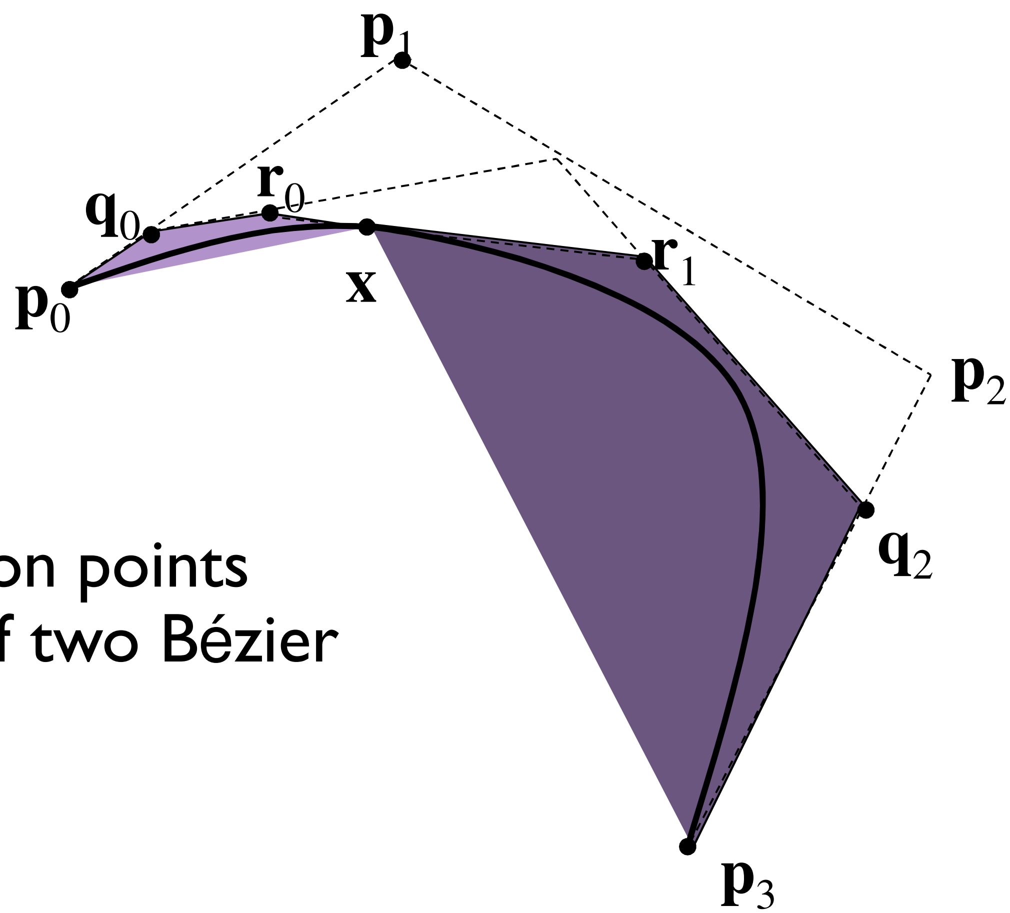


UCSD

Recursive Subdivision

- Any cubic curve segment can be expressed as a Bézier curve
- Any piece of a cubic curve is itself a cubic curve
- Therefore:
  - Any Bézier curve can be broken down into smaller Bézier curves

De Casteljau Subdivision



- De Casteljau construction points are the control points of two Bézier sub-segments

Adaptive Subdivision Algorithm

- Use De Casteljau construction to split Bézier segment in half
- For each half
  - If "flat enough": draw line segment
  - Else: recurse
- Curve is flat enough if hull is flat enough
  - Test how far the approximating control points are from a straight segment
    - If less than one pixel, the hull is flat enough

Drawing Bézier Curves With OpenGL

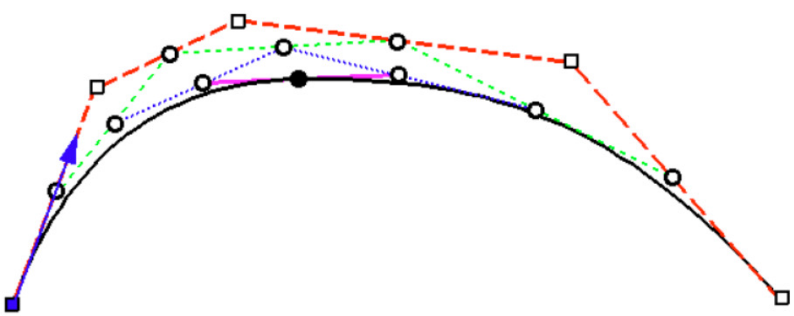
- Indirect OpenGL support for drawing curves:
  - Define evaluator map (`glMap`)
  - Draw line strip by evaluating map (`glEvalCoord`)
  - Optimize by pre-computing coordinate grid (`glMapGrid` and `glEvalMesh`)
- More details about OpenGL implementation:
  - http://www.cs.duke.edu/courses/fall09/cps124/notes/12_curves/opengl_nurbs.pdf

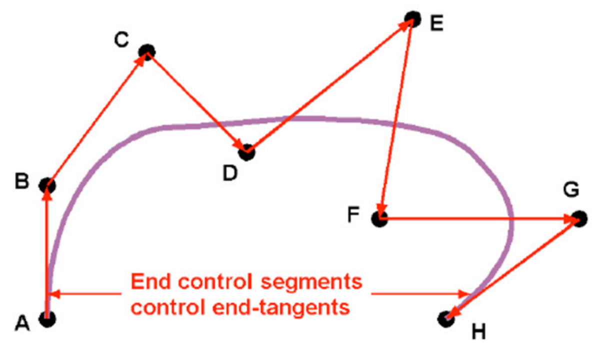
Lecture Overview

- Polynomial Curves
  - Introduction
  - Polynomial functions
- Bézier Curves
  - Introduction
  - Drawing Bézier curves
  - Piecewise Bézier curves

More Control Points

- Cubic Bézier curve limited to 4 control points
  - Cubic curve can only have one inflection (point where curve changes direction of bending)
  - Need more control points for more complex curves
- k-1 order Bézier curve with k control points

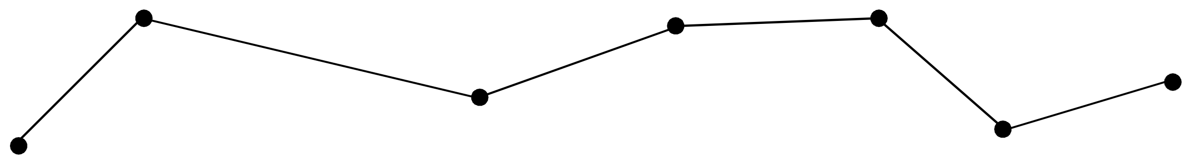




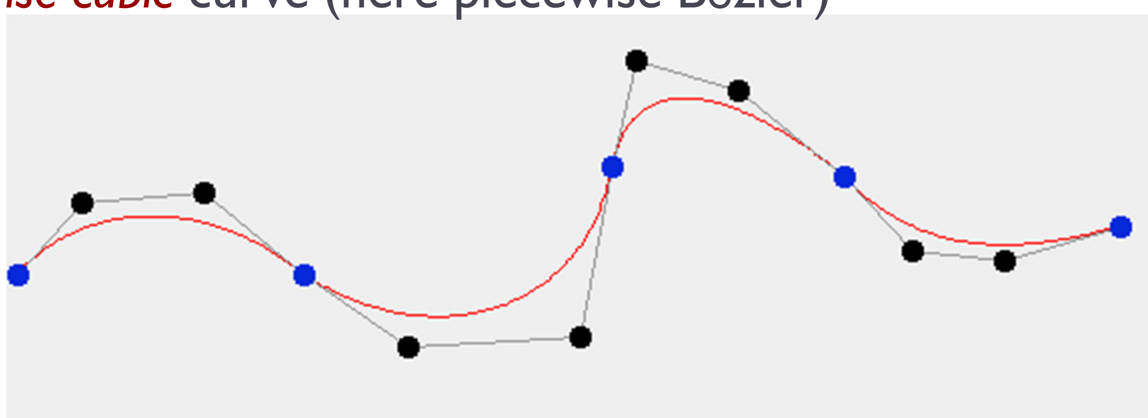
- Hard to control and hard to work with
  - Intermediate points don't have obvious effect on shape
  - Changing any control point changes the whole curve
  - Want *local support*: each control point only influences nearby portion of curve

Piecewise Curves

- Sequence of line segments
  - *Piecewise linear* curve

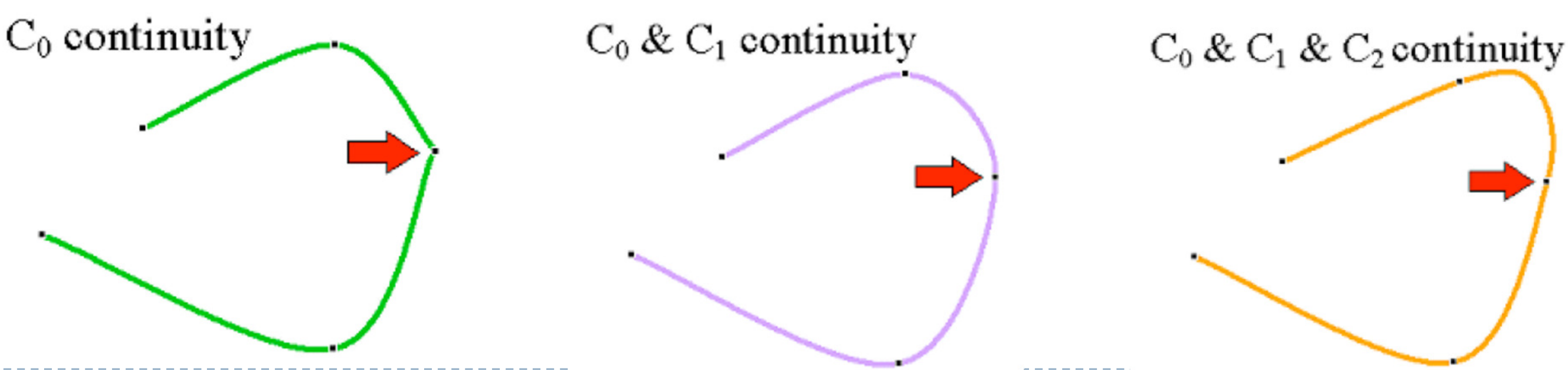


- Sequence of simple (low-order) curves, end-to-end
  - Known as a *piecewise polynomial curve*
- Sequence of cubic curve segments
  - *Piecewise cubic* curve (here piecewise Bézier)



Parametric Continuity

- C^0 continuity:
  - Curve segments are connected
- C^1 continuity:
  - C^0 & 1st-order derivatives agree
  - Curves have same tangents
  - Relevant for smooth shading
- C^2 continuity:
  - C^1 & 2nd-order derivatives agree
  - Curves have same tangents and curvature
  - Relevant for high quality reflections



Overview

- Piecewise Bezier curves
- Bezier surfaces

Global Parameterization

- Given N curve segments $\mathbf{x}_0(t)$, $\mathbf{x}_1(t)$, …, $\mathbf{x}_{N-1}(t)$
- Each is parameterized for t from 0 to 1
- Define a piecewise curve
  - Global parameter u from 0 to N

$$\mathbf{x}(u) = \begin{cases} \mathbf{x}_0(u), & 0 \le u \le 1 \\ \mathbf{x}_1(u-1), & 1 \le u \le 2 \\ \vdots & \vdots \\ \mathbf{x}_{N-1}(u-(N-1)), & N-1 \le u \le N \end{cases}$$

$$\mathbf{x}(u) = \mathbf{x}_i(u-i), \text{ where } i = \lfloor u \rfloor \quad (\text{and } \mathbf{x}(N) = \mathbf{x}_{N-1}(1))$$

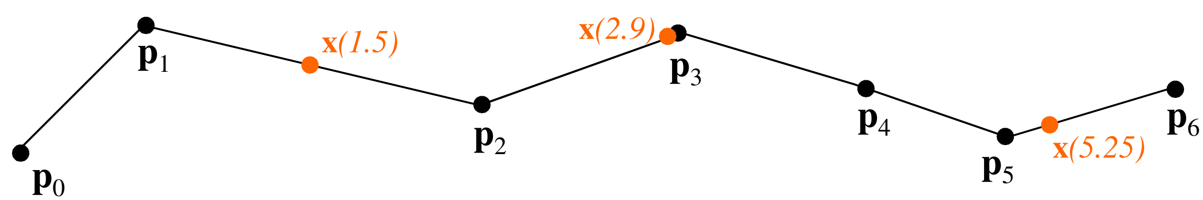
- Alternate: solution u also goes from 0 to 1

$$\mathbf{x}(u) = \mathbf{x}_i(Nu-i), \text{ where } i = \lfloor Nu \rfloor$$

Piecewise-Linear Curve

- Given N+1 points $\mathbf{p}_0, \mathbf{p}_1, \ldots, \mathbf{p}_N$
- Define curve

$$\mathbf{x}(u) = Lerp(u - i, \mathbf{p}_i, \mathbf{p}_{i+1}), \qquad i \le u \le i+1$$
$$= (1 - u + i)\mathbf{p}_i + (u - i)\mathbf{p}_{i+1}, \quad i = \lfloor u \rfloor$$



- N+1 points define N linear segments
- $\mathbf{x}(i)=\mathbf{p}_i$
- C^0 continuous by construction
- C^1 at $\mathbf{p}_i$ when $\mathbf{p}_i - \mathbf{p}_{i-1} = \mathbf{p}_{i+1} - \mathbf{p}_i$

UCSD

Piecewise Bézier curve

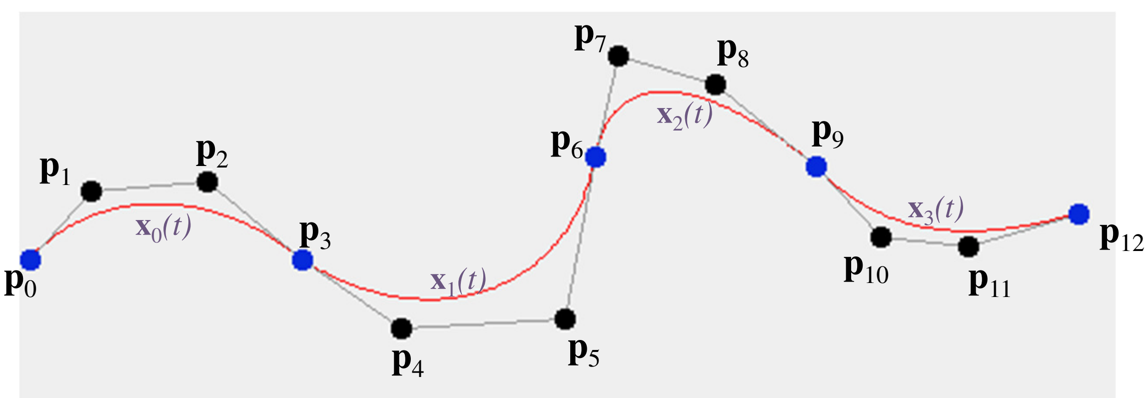
- Given $3N+1$ points $\mathbf{p}_0, \mathbf{p}_1, \ldots, \mathbf{p}_{3N}$
- Define N Bézier segments:

$$\mathbf{x}_0(t) = B_0(t)\mathbf{p}_0 + B_1(t)\mathbf{p}_1 + B_2(t)\mathbf{p}_2 + B_3(t)\mathbf{p}_3$$

$$\mathbf{x}_1(t) = B_0(t)\mathbf{p}_3 + B_1(t)\mathbf{p}_4 + B_2(t)\mathbf{p}_5 + B_3(t)\mathbf{p}_6$$

$$\vdots$$

$$\mathbf{x}_{N-1}(t) = B_0(t)\mathbf{p}_{3N-3} + B_1(t)\mathbf{p}_{3N-2} + B_2(t)\mathbf{p}_{3N-1} + B_3(t)\mathbf{p}_{3N}$$



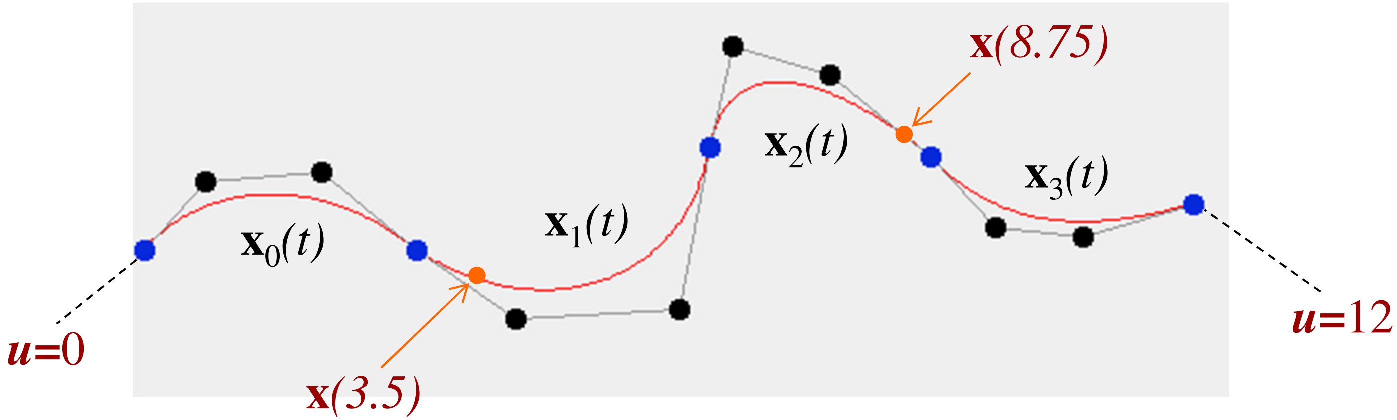
UCSD

Piecewise Bézier Curve

- Parameter in 0<=u<=3N

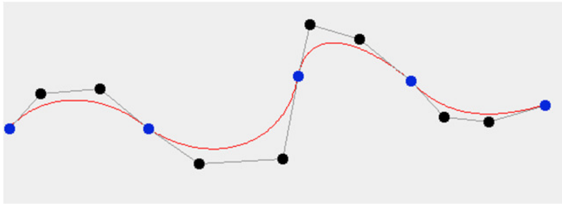
$$\mathbf{x}(u) = \begin{cases} \mathbf{x}_0(\frac{1}{3}u), & 0 \le u \le 3 \\ \mathbf{x}_1(\frac{1}{3}u - 1), & 3 \le u \le 6 \\ \vdots & \vdots \\ \mathbf{x}_{N-1}(\frac{1}{3}u - (N-1)), & 3N-3 \le u \le 3N \end{cases}$$

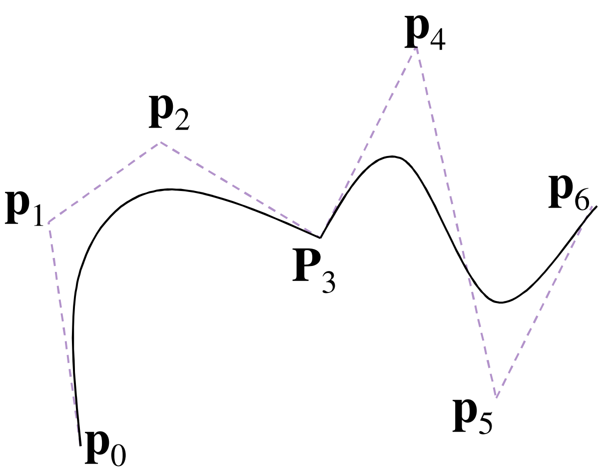
$\mathbf{x}(u) = \mathbf{x}_i\left(\frac{1}{3}u - i\right)$, where $i = \left\lfloor \frac{1}{3}u \right\rfloor$



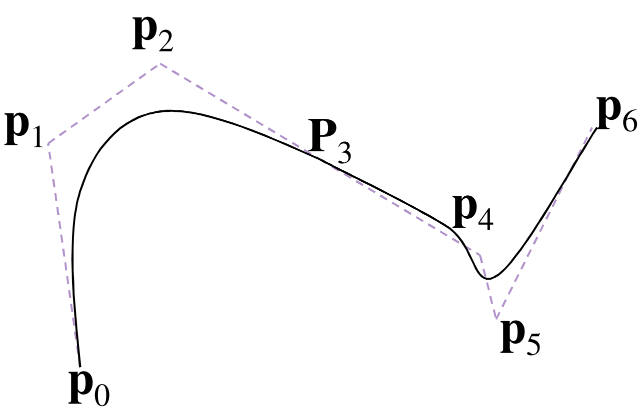
Piecewise Bézier Curve

- $3N+1$ points define N Bézier segments
- $\mathbf{x}(3i)=\mathbf{p}_{3i}$
- C_0 continuous by construction
- C_1 continuous at $\mathbf{p}_{3i}$ when $\mathbf{p}_{3i} - \mathbf{p}_{3i-1} = \mathbf{p}_{3i+1} - \mathbf{p}_{3i}$
- C_2 is harder to achieve





C_1 discontinuous



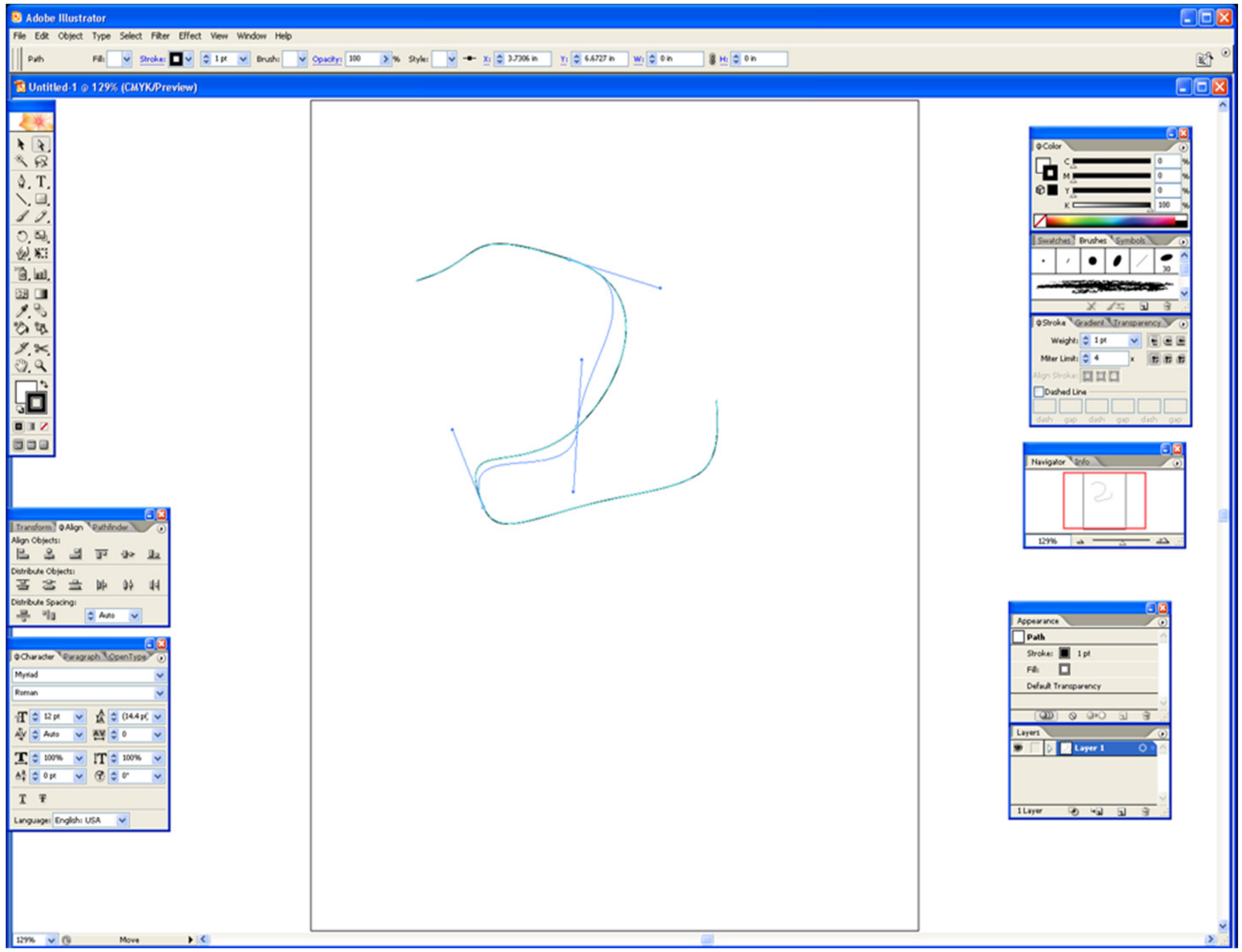
C_1 continuous

Piecewise Bézier Curves

- Used often in 2D drawing programs
- Inconveniences
  - Must have 4 or 7 or 10 or 13 or … (1 plus a multiple of 3) control points
  - Some points interpolate, others approximate
  - Need to impose constraints on control points to obtain C^1 continuity
  - C_2 continuity more difficult
- Solutions
  - User interface using "Bézier handles"
  - Generalization to B-splines or NURBS

Bézier Handles

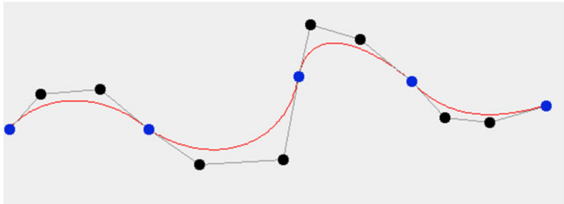
- Segment end points (interpolating) presented as curve control points
- Midpoints (approximating points) presented as "handles"
- Can have option to enforce C_1 continuity

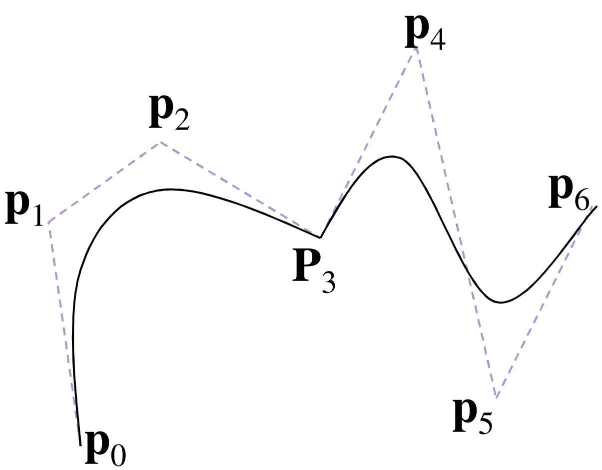


Adobe Illustrator

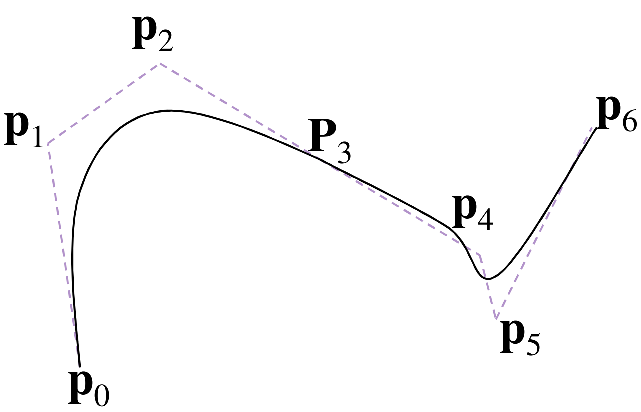
Piecewise Bézier Curve

- $3N+1$ points define N Bézier segments
- $\mathbf{x}(3i)=\mathbf{p}_{3i}$
- C_0 continuous by construction
- C_1 continuous at $\mathbf{p}_{3i}$ when $\mathbf{p}_{3i} - \mathbf{p}_{3i-1} = \mathbf{p}_{3i+1} - \mathbf{p}_{3i}$
- C_2 is harder to achieve





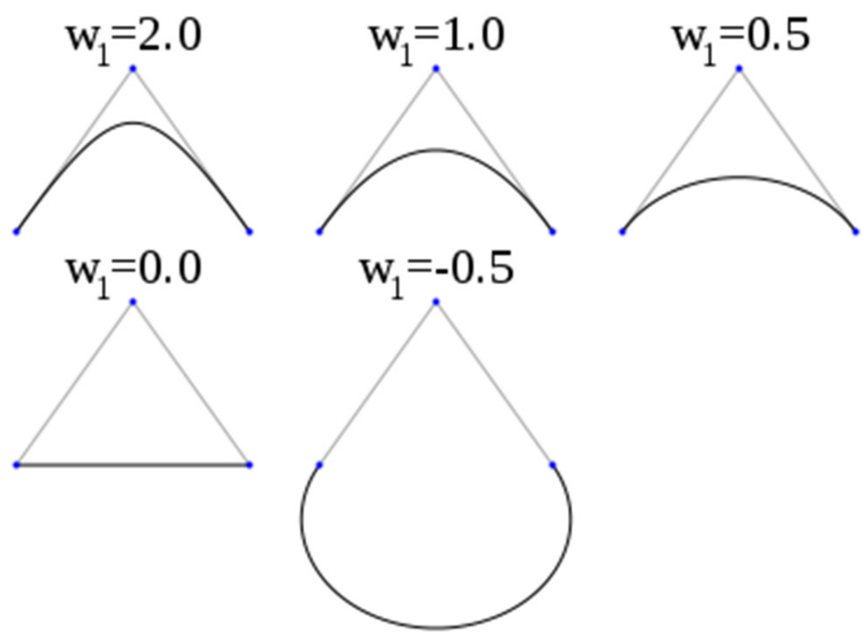
C_1 discontinuous



C_1 continuous

Rational Curves

- Weight causes point to "pull" more (or less)
- Can model circles with proper points and weights,
- Below: rational quadratic Bézier curve (three control points)



UCSD

B-Splines

- B as in **B**asis-Splines
- Basis is blending function
- Difference to Bézier blending function:
  - B-spline blending function can be zero outside a particular range (limits scope over which a control point has influence)
- B-Spline is defined by control points and range in which each control point is active.

NURBS

- **N**on **U**niform **R**ational **B-S**plines
- Generalization of Bézier curves
- Non uniform:
- Combine B-Splines (limited scope of control points) and Rational Curves (weighted control points)
- Can exactly model conic sections (circles, ellipses)
- OpenGL support: see `gluNurbsCurve`
- Demo: http://bentonian.com/teaching/AdvGraph0809/demos/Nurbs2d/index.html
- http://mathworld.wolfram.com/NURBSCurve.html

Overview

- Bi-linear patch
- Bi-cubic Bézier patch
- Advanced parametric surfaces

Curved Surfaces

Curves

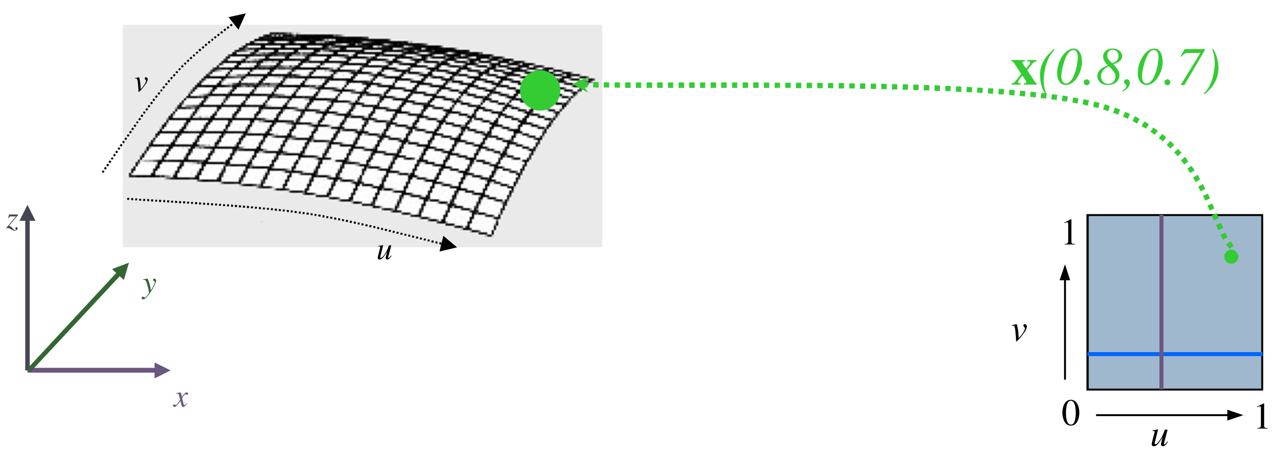
- Described by a 1D series of control points
- A function $\mathbf{x}(t)$
- Segments joined together to form a longer curve

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

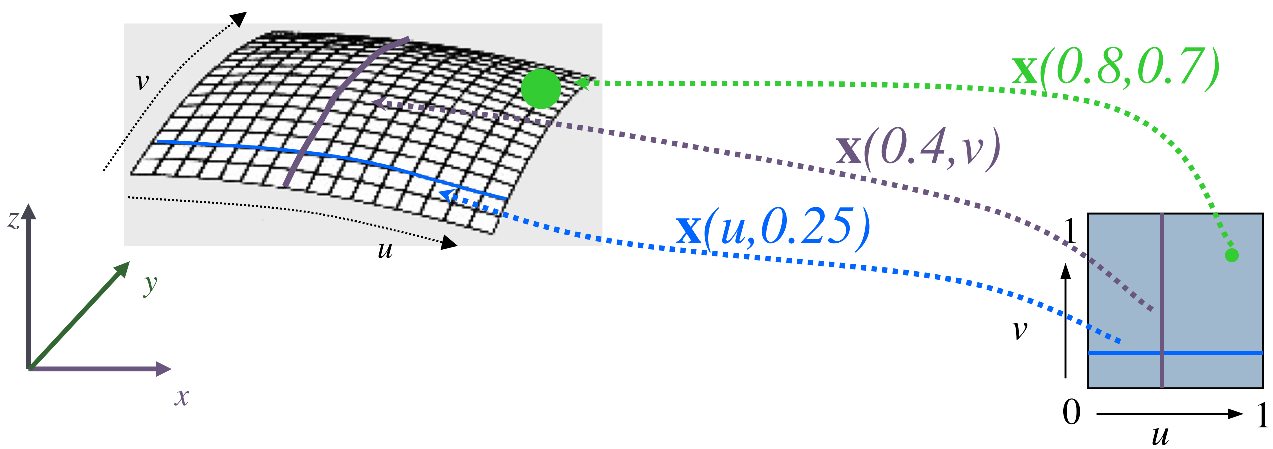
- $\mathbf{x}(u,v)$ describes a point in space for any given (u,v) pair
  - u,v each range from 0 to 1



2D parameter domain

Parametric Surface Patch

- $\mathbf{x}(u,v)$ describes a point in space for any given (u,v) pair
  - u,v each range from 0 to 1

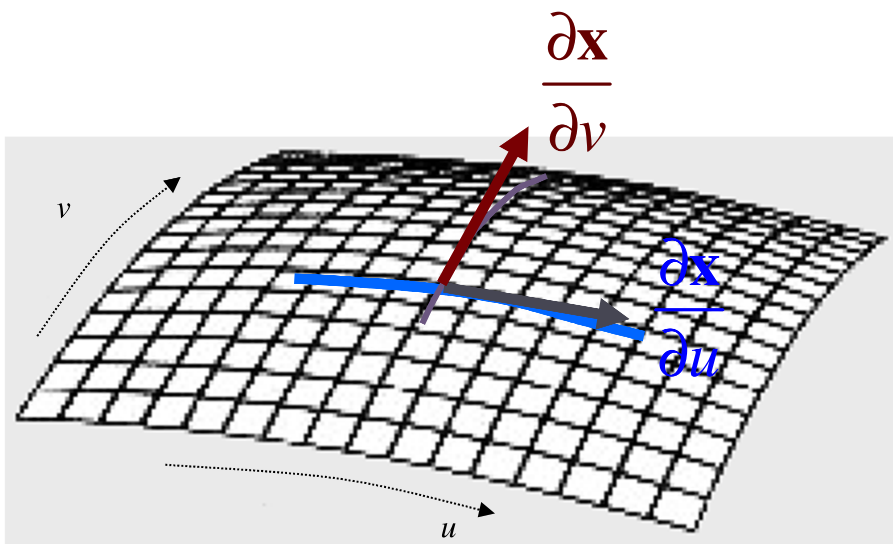


2D parameter domain

- 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 are a pair of parametric curves through that 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
- Note: these vectors are not necessarily perpendicular to each other



UCSD

Tangents

- Notation:
  - The tangent along a u curve, AKA the tangent in the u direction, is written as:

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

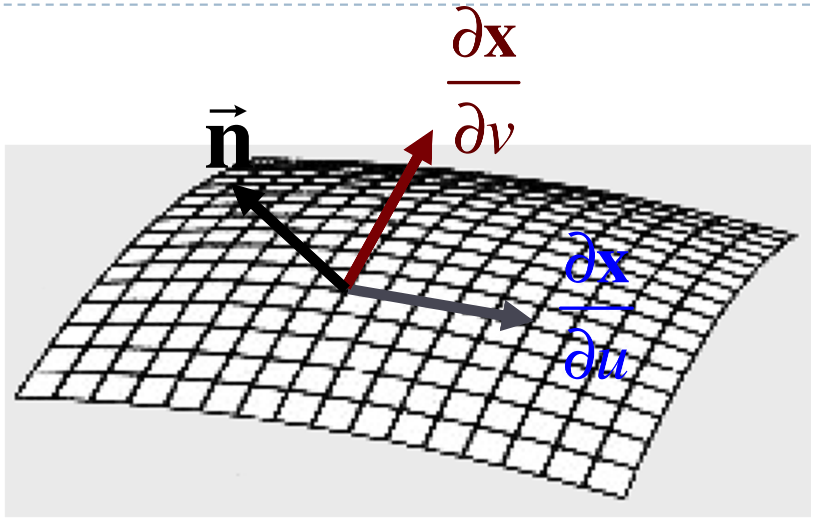
  - The tangent along a v curve, AKA the tangent in the v direction, is written as:

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

- Note that each of these is a vector-valued function:
  - At each point $\mathbf{x}(u,v)$ on the surface, we have tangent vectors $\tfrac{\partial}{\partial u}\mathbf{x}(u,v)$ and $\tfrac{\partial}{\partial v}\mathbf{x}(u,v)$

Surface Normal

- Normal is cross product of the two tangent vectors
- Order matters!



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

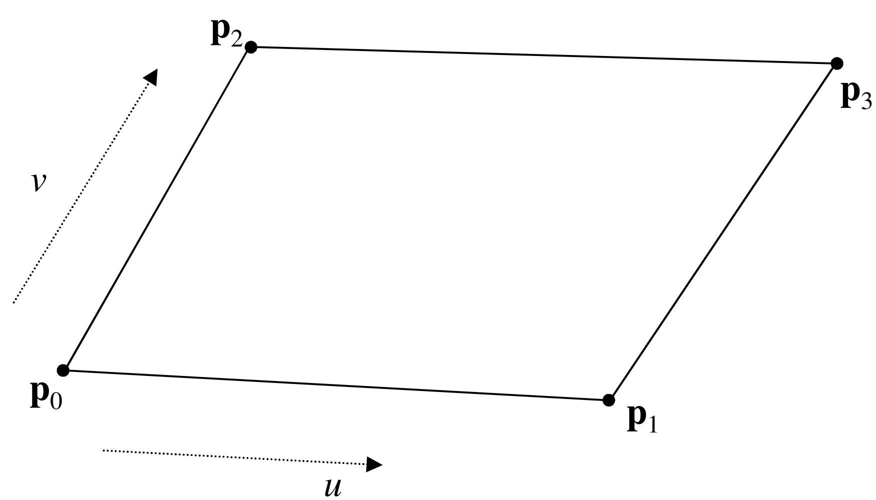
Typically we are interested in the unit normal, so we need to normalize

$$\vec{\mathbf{n}}^*(u,v) = \frac{\partial \mathbf{x}}{\partial u}(u,v) \times \frac{\partial \mathbf{x}}{\partial v}(u,v)$$

$$\vec{\mathbf{n}}(u,v) = \frac{\vec{\mathbf{n}}^*(u,v)}{\left|\vec{\mathbf{n}}^*(u,v)\right|}$$

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 scheme

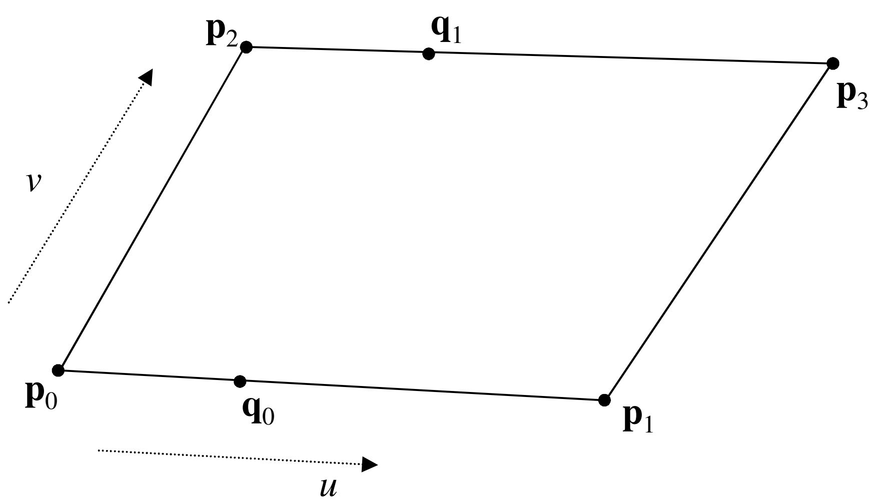


UCSD

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:

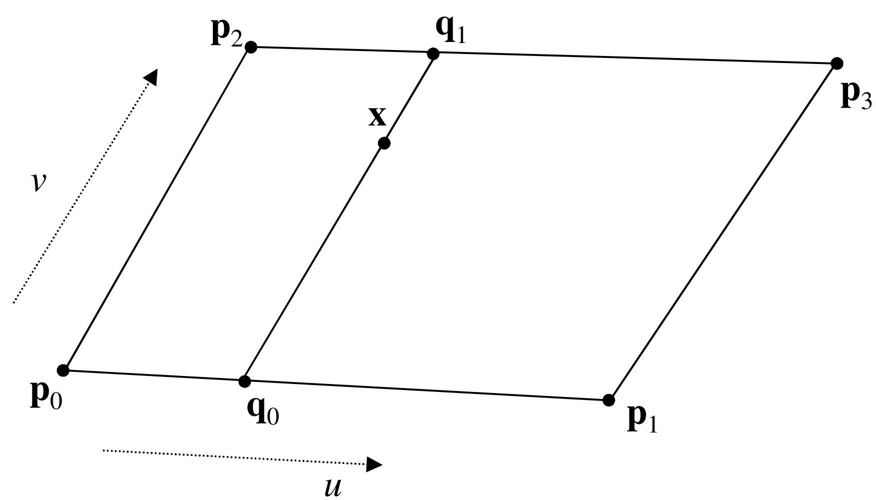
$$\mathbf{q_0} = Lerp(u, \mathbf{p_0}, \mathbf{p_1}) \qquad \mathbf{q_1} = Lerp(u, \mathbf{p_2}, \mathbf{p_3})$$



Bilinear Patch (Step 2)

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

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

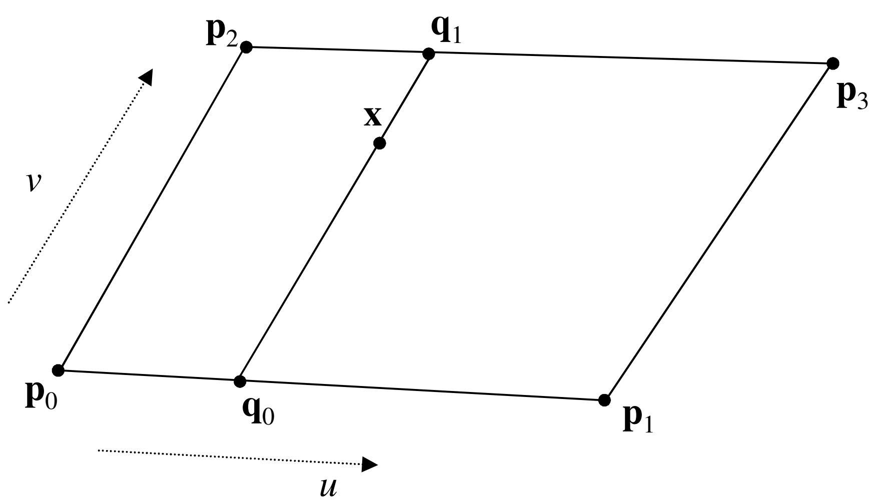


UCSD

Bilinear Patch

- Combining the steps, we get the full formula

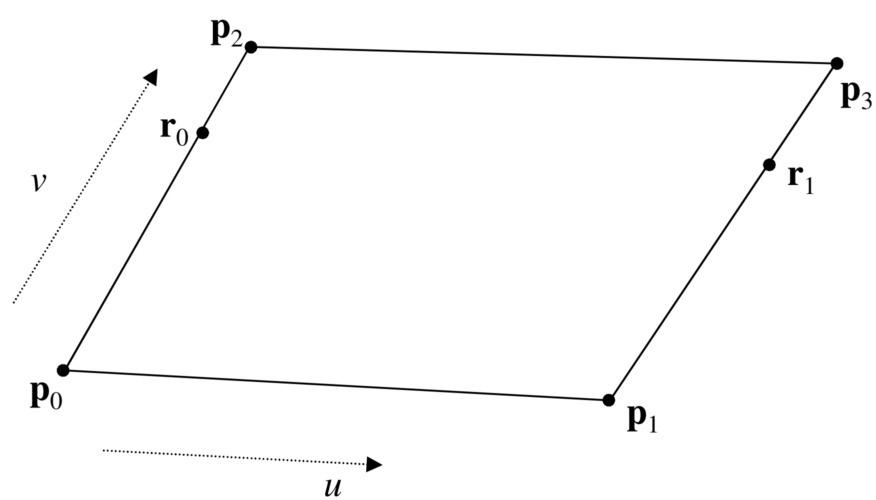
$$\mathbf{x}(u,v) = Lerp(v, Lerp(u, \mathbf{p}_0, \mathbf{p}_1), Lerp(u, \mathbf{p}_2, \mathbf{p}_3))$$



Bilinear Patch

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

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

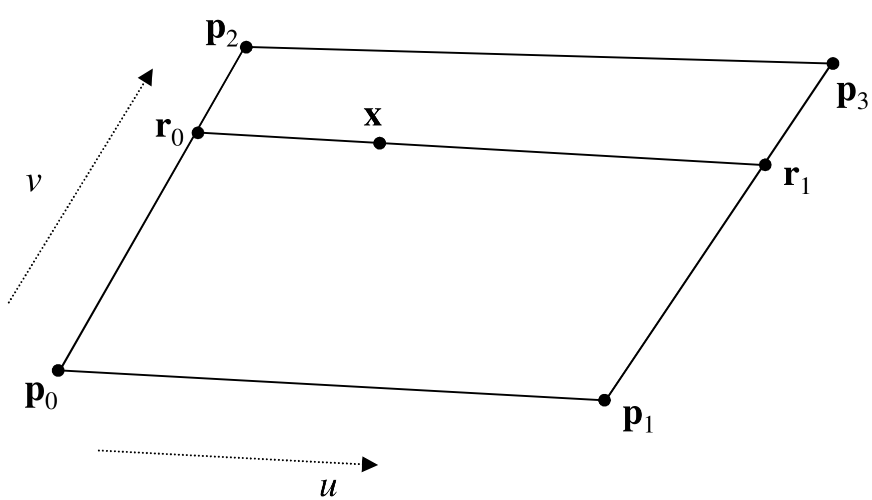


UCSD

Bilinear Patch

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

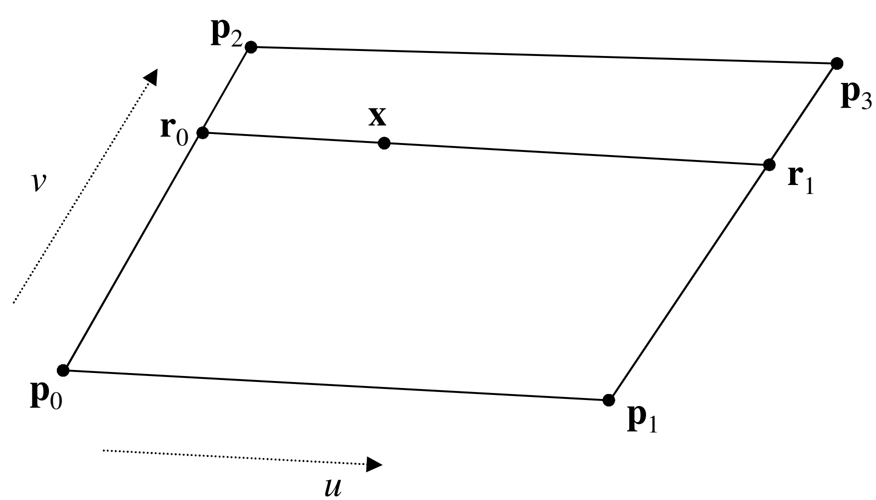
$$\mathbf{x} = Lerp(u, \mathbf{r}_0, \mathbf{r}_1)$$



Bilinear Patch

- The full formula for the v direction first:

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

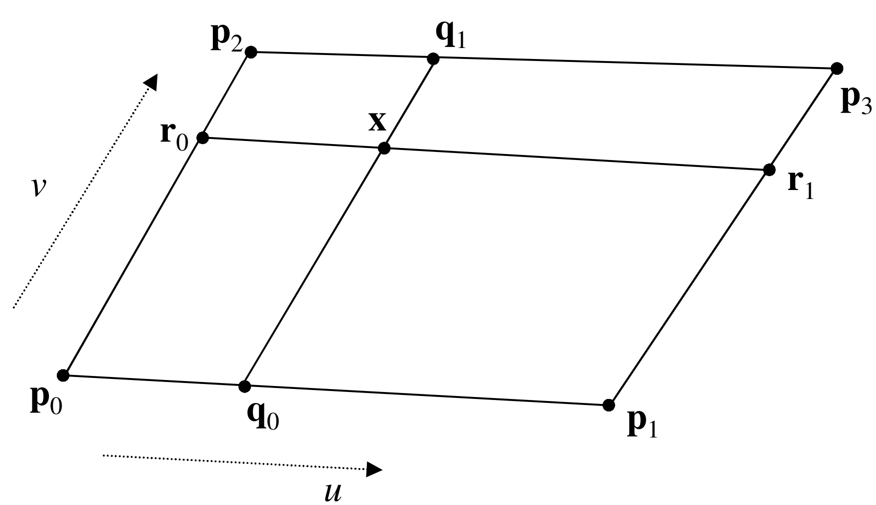


UCSD

Bilinear Patch

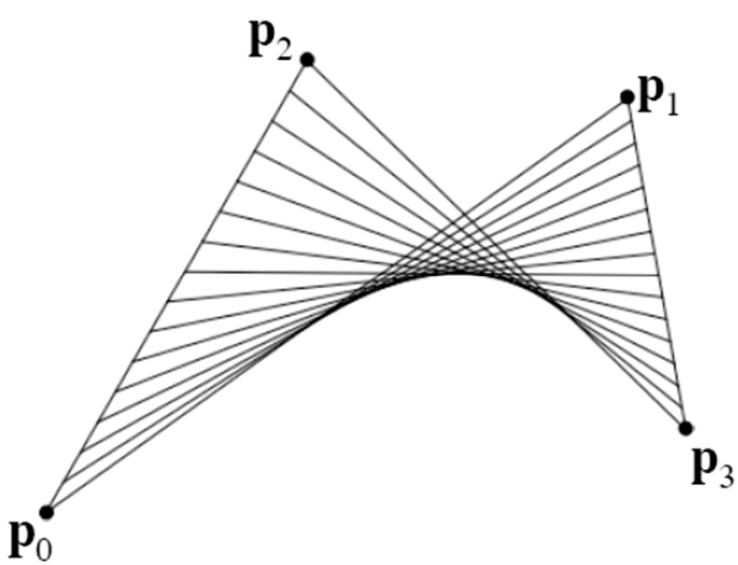
- Patch geometry is independent of the order of *u* and *v*

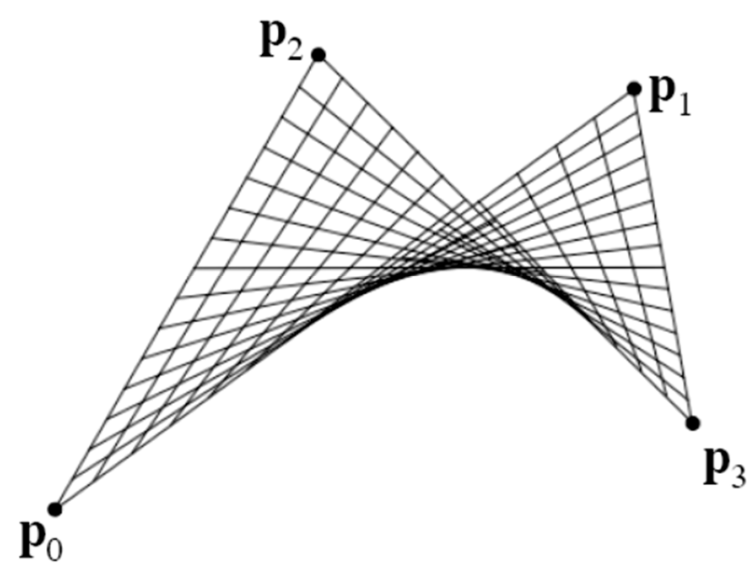
$$\mathbf{x}(u,v) = Lerp(v, Lerp(u, \mathbf{p}_0, \mathbf{p}_1), Lerp(u, \mathbf{p}_2, \mathbf{p}_3))$$
$$\mathbf{x}(u,v) = Lerp(u, Lerp(v, \mathbf{p}_0, \mathbf{p}_2), Lerp(v, \mathbf{p}_1, \mathbf{p}_3))$$



Bilinear Patch

- Visualization





UCSD