

CSE 167:
Introduction to Computer Graphics
Lecture #13: Curves

Jürgen P. Schulze, Ph.D.
University of California, San Diego
Fall Quarter 2017

Announcements

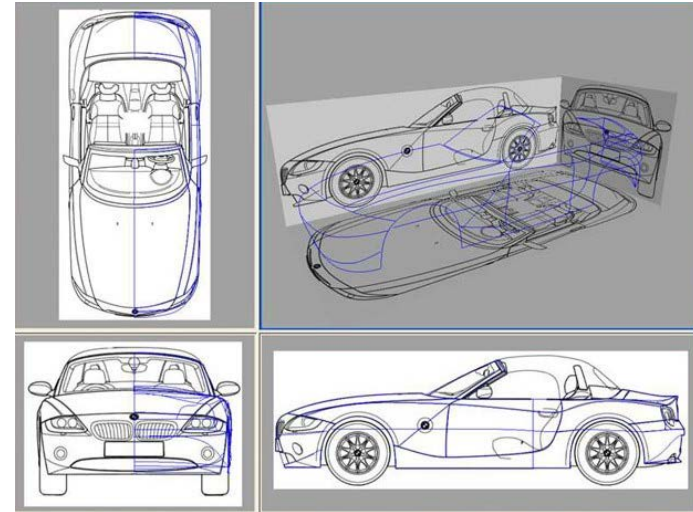
- ▶ Project 4 due Monday Nov 27 at 2pm
- ▶ Next Tuesday: midterm #2
- ▶ Topics: everything since first midterm including lecture this Thursday

Lecture Overview

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

Modeling

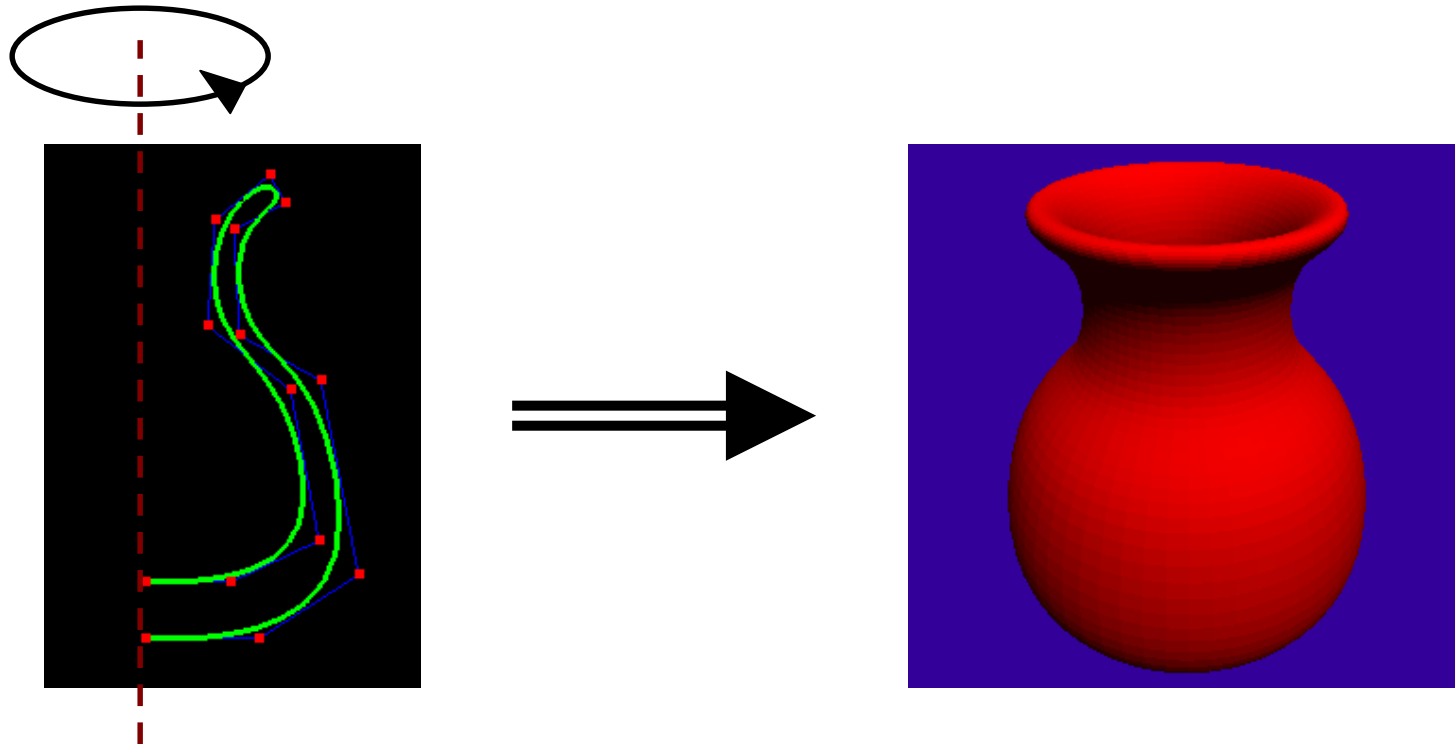
- ▶ Creating 3D objects
- ▶ How to construct complex surfaces?
- ▶ Goal
 - ▶ Specify objects with control points
 - ▶ Objects should be visually pleasing (smooth)
- ▶ Start with curves, then surfaces



What can curves be used for?

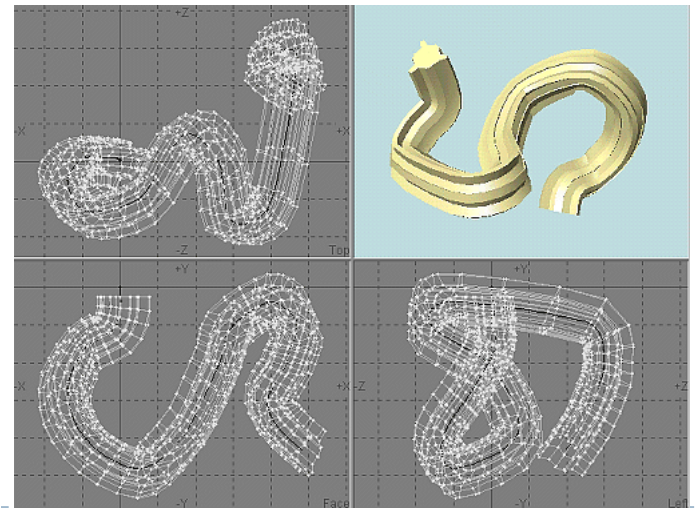
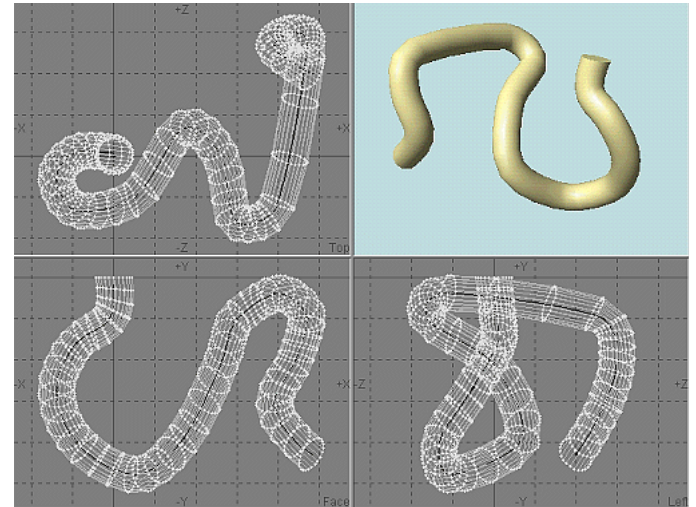
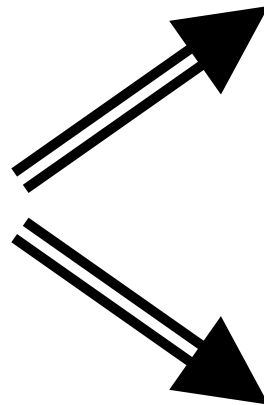
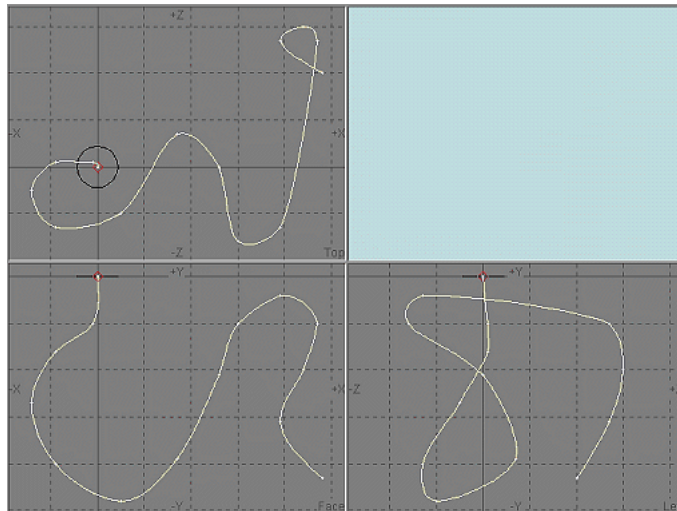
Curves

- ▶ Surface of revolution



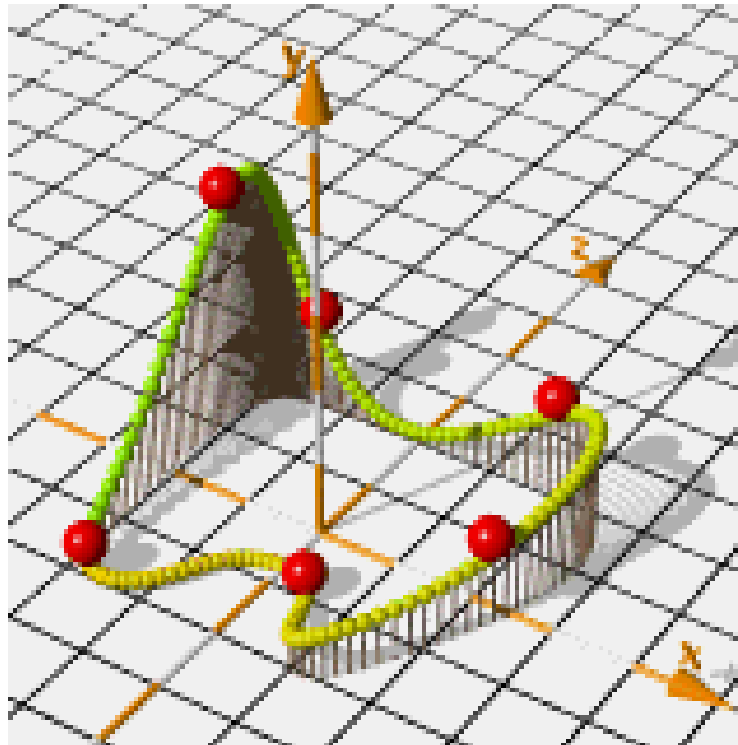
Curves

▶ Extruded/swept surfaces



Curves

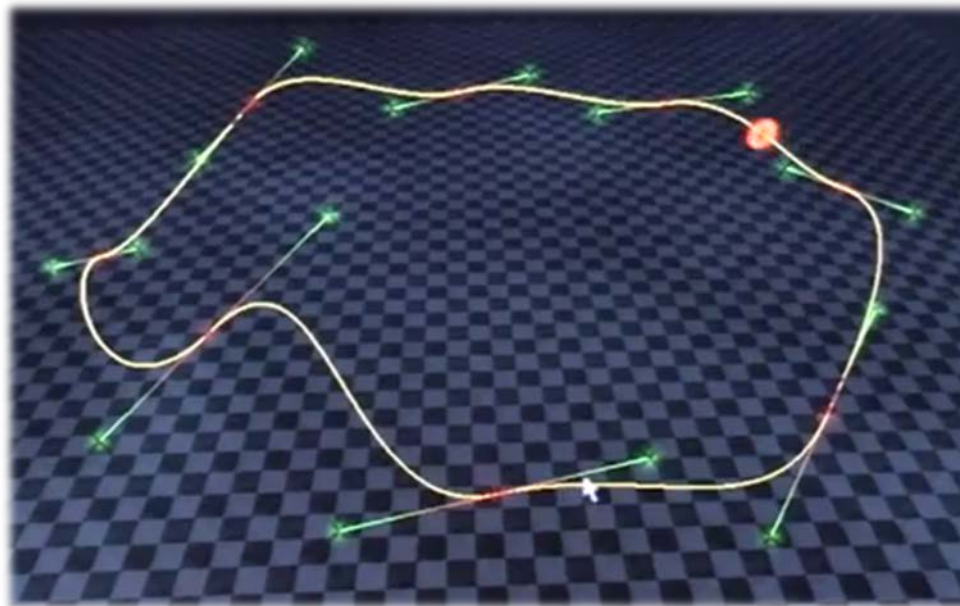
- ▶ **Animation**
 - ▶ Provide a “track” for objects
 - ▶ Use as camera path



Video

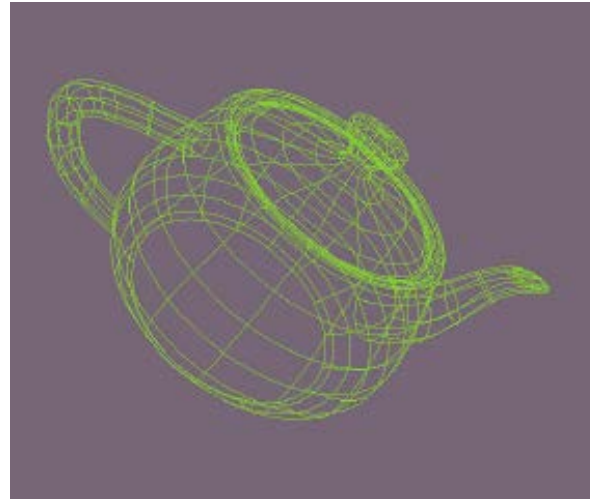
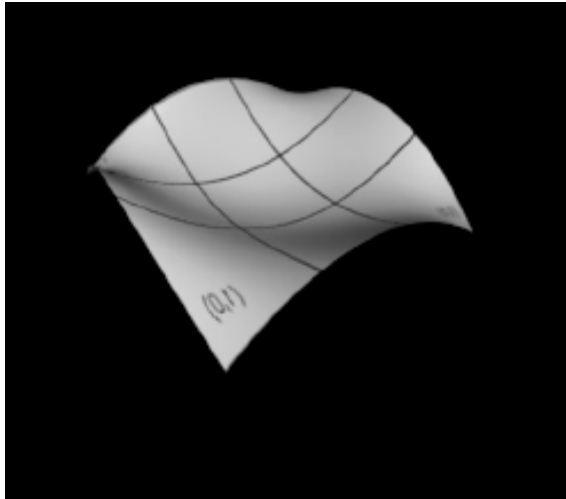
- ▶ **Bezier Curves**

- ▶ <http://www.youtube.com/watch?v=hIDYJNEiYvU>



Curves

- ▶ Can be generalized to surface patches



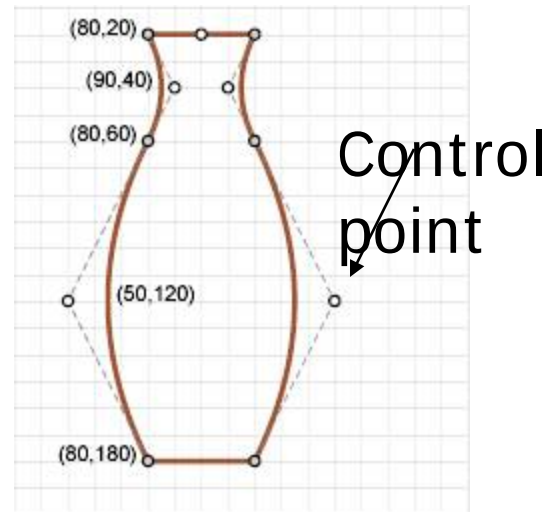
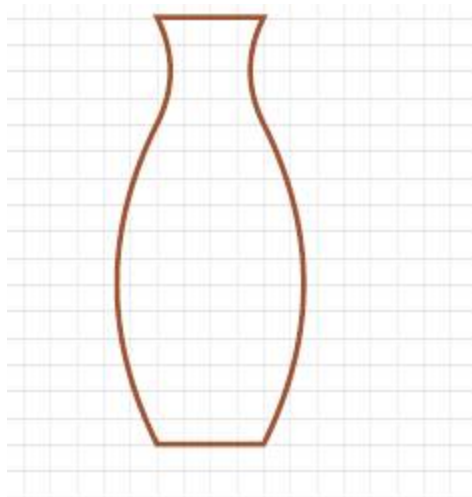
Curve Representation

Why not specify many points along a curve and connect with lines:

- ▶ Can't get smooth results when magnified – more points needed
- ▶ Large storage and CPU requirements

Instead: specify a curve with a small number of “control points”

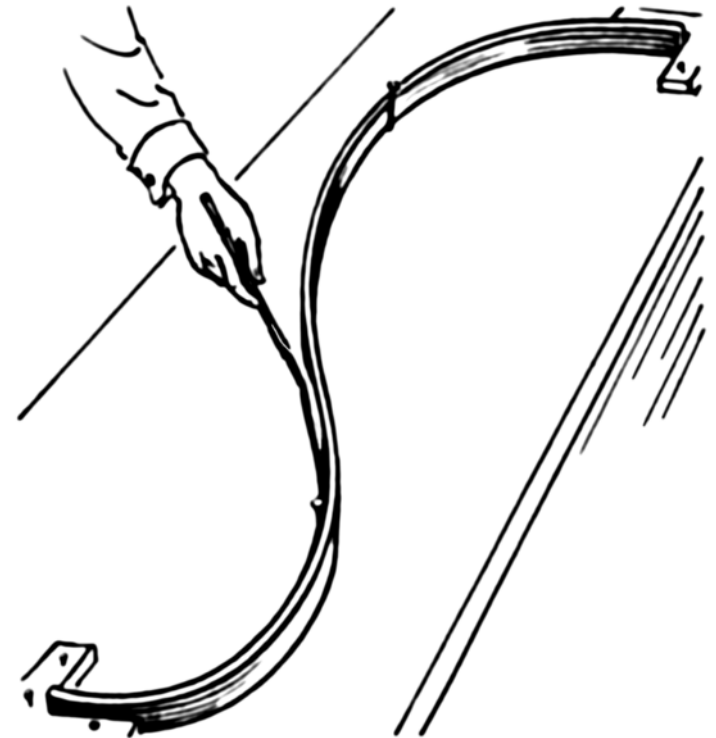
- ▶ Known as a *spline curve* or *spline*.



Spline: Definition

► Wikipedia:

- Term comes from flexible spline devices used by shipbuilders and draftsmen to draw smooth shapes.
- Spline consists of a long strip fixed in position at a number of points that relaxes to form a smooth curve passing through those points.

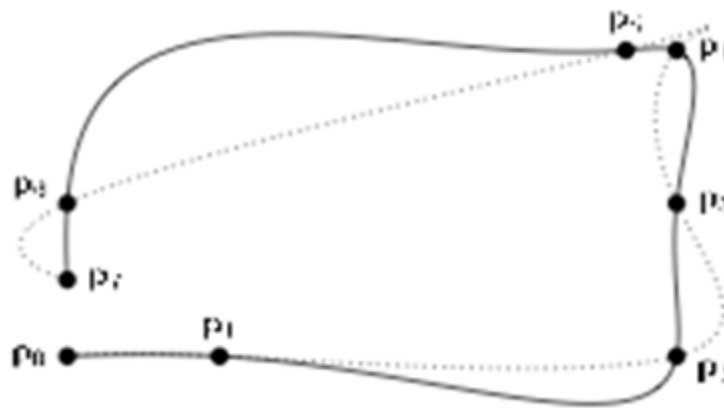


Lecture Overview

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

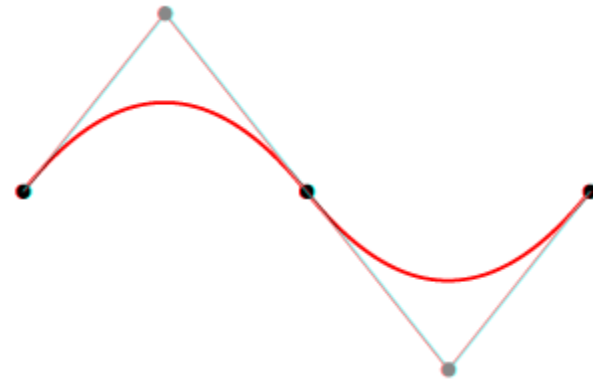
Interpolating Control Points

- ▶ “Interpolating” means that curve goes through all control points
- ▶ A.k.a. “Anchor Points”
- ▶ Seems most intuitive
- ▶ But hard to control exact behavior



Approximating Control Points

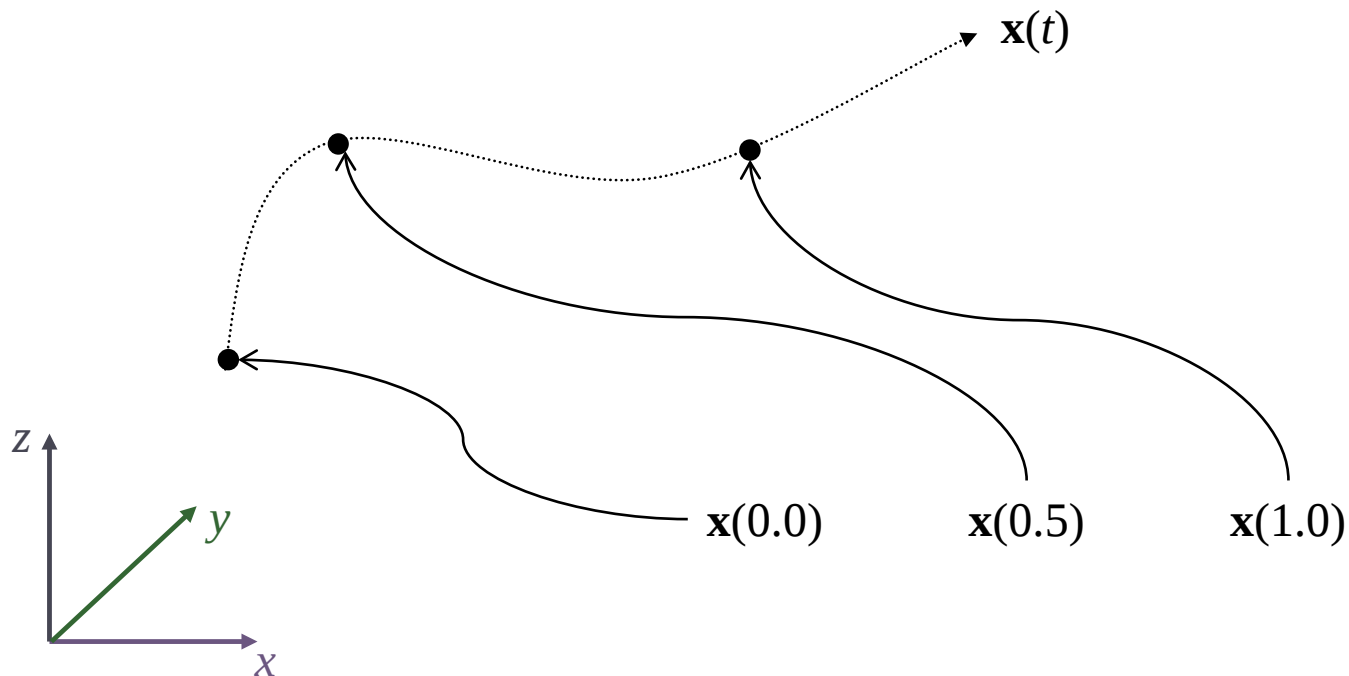
- ▶ Curve is “influenced” by control points



- ▶ Various types
- ▶ Most common: polynomial functions
 - ▶ Bézier spline (our focus)
 - ▶ B-spline (generalization of Bézier spline)
 - ▶ NURBS (Non Uniform Rational Basis Spline): used in CAD tools

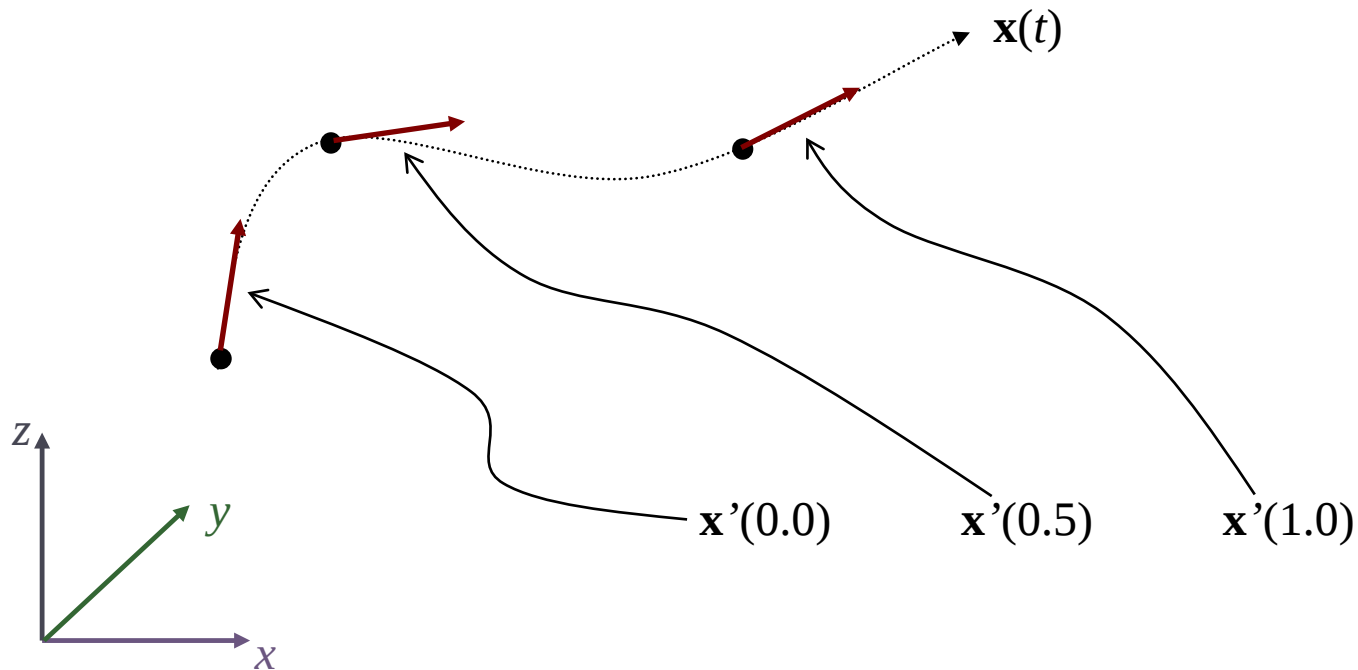
Mathematical Definition

- ▶ A vector valued function of one variable $\mathbf{x}(t)$
 - ▶ Given t , compute a 3D point $\mathbf{x}=(x,y,z)$
 - ▶ Could be interpreted as three functions: $x(t)$, $y(t)$, $z(t)$
 - ▶ Parameter t “moves a point along the curve”



Tangent Vector

- ▶ Derivative $\mathbf{x}'(t) = \frac{d\mathbf{x}}{dt} = (x'(t), y'(t), z'(t))$
- ▶ Vector $\mathbf{x}'$:
 - ▶ Points in direction of movement
 - ▶ Length corresponds to speed

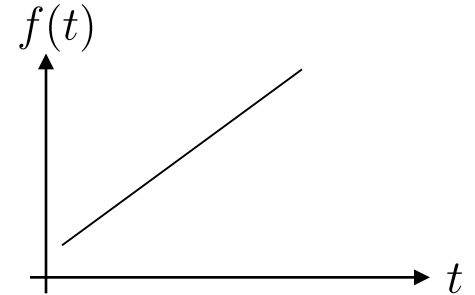


Lecture Overview

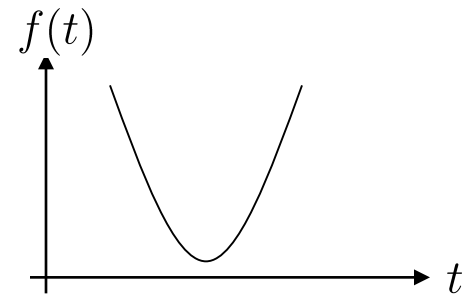
- ▶ Polynomial Curves
 - ▶ Introduction
 - ▶ Polynomial functions
- ▶ Bézier Curves
 - ▶ Introduction
 - ▶ Drawing Bézier curves
 - ▶ Piecewise Bézier curves

Polynomial Functions

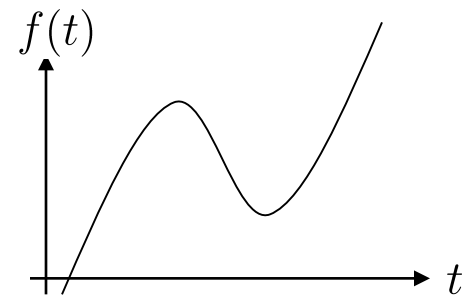
- ▶ **Linear:** $f(t) = at + b$
(1st order)



- ▶ **Quadratic:** $f(t) = at^2 + bt + c$
(2nd order)



- ▶ **Cubic:** $f(t) = at^3 + bt^2 + ct + d$
(3rd order)

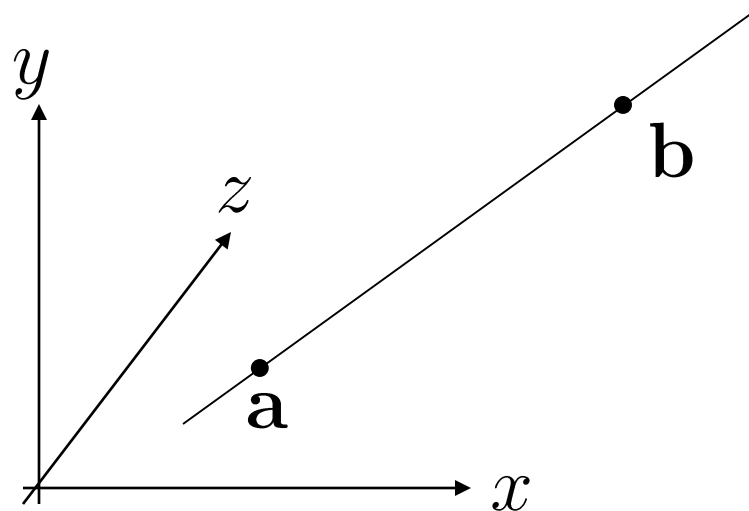


Polynomial Curves in 3D

- ▶ Linear $\mathbf{x}(t) = \mathbf{a}t + \mathbf{b}$

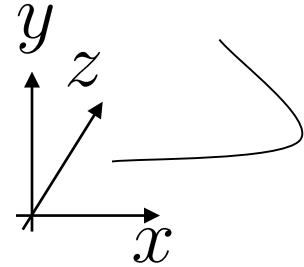
$$\mathbf{x} = (x, y, z), \mathbf{a} = (a_x, a_y, a_z), \mathbf{b} = (b_x, b_y, b_z)$$

- ▶ Evaluated as:
$$x(t) = a_x t + b_x$$
$$y(t) = a_y t + b_y$$
$$z(t) = a_z t + b_z$$

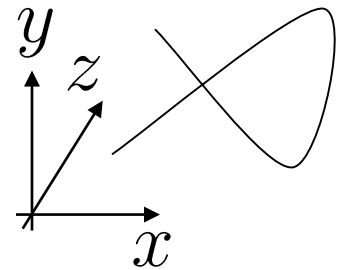


Polynomial Curves in 3D

► **Quadratic:** $\mathbf{x}(t) = \mathbf{a}t^2 + \mathbf{b}t + \mathbf{c}$
(2nd order)



► **Cubic:** $\mathbf{x}(t) = \mathbf{a}t^3 + \mathbf{b}t^2 + \mathbf{c}t + \mathbf{d}$
(3rd order)



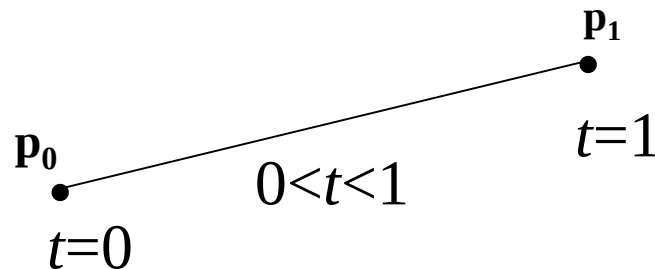
► We usually define the curve for $0 \leq t \leq 1$

Control Points

- ▶ Polynomial coefficients **a**, **b**, **c**, **d** can be interpreted as *control points*
 - ▶ Remember: **a**, **b**, **c**, **d** have x, y, z components each
- ▶ But: they do not intuitively describe the shape of the curve
- ▶ Goal: intuitive control points

Weighted Average

- ▶ Based on linear interpolation (LERP)
 - ▶ Weighted average between two values
 - ▶ “Value” could be a number, vector, color, ...
- ▶ Interpolate between points $\mathbf{p}_0$ and $\mathbf{p}_1$ with parameter t
 - ▶ Defines a “curve” that is straight (first-order spline)

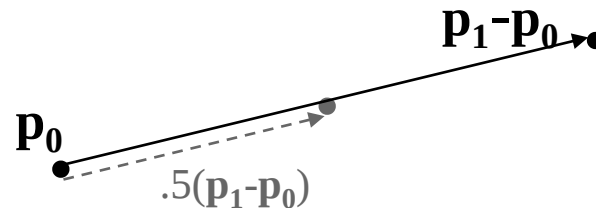


$$\mathbf{x}(t) = \text{Lerp}(t, \mathbf{p}_0, \mathbf{p}_1) = (1 - t)\mathbf{p}_0 + t \mathbf{p}_1$$

Linear Polynomial

$$\mathbf{x}(t) = \underbrace{(\mathbf{p}_1 - \mathbf{p}_0)}_{\text{vector } \mathbf{a}} t + \underbrace{\mathbf{p}_0}_{\text{point } \mathbf{b}}$$

- ▶ Curve is based at point $\mathbf{p}_0$
- ▶ Add the vector, scaled by t



Matrix Form

$$\mathbf{x}(t) = \begin{bmatrix} \mathbf{p}_0 & \mathbf{p}_1 \end{bmatrix} \begin{bmatrix} -1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} t \\ 1 \end{bmatrix} = \mathbf{GBT}$$

► Geometry matrix $\mathbf{G} = \begin{bmatrix} \mathbf{p}_0 & \mathbf{p}_1 \end{bmatrix}$

► Geometric basis $\mathbf{B} = \begin{bmatrix} -1 & 1 \\ 1 & 0 \end{bmatrix}$

► Polynomial basis $T = \begin{bmatrix} t \\ 1 \end{bmatrix}$

► In components
$$\mathbf{x}(t) = \begin{bmatrix} p_{0x} & p_{1x} \\ p_{0y} & p_{1y} \\ p_{0z} & p_{1z} \end{bmatrix} \begin{bmatrix} -1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} t \\ 1 \end{bmatrix}$$

Summary

1. Grouped by points $\mathbf{p}$: weighted average

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

2. Grouped by t : linear polynomial

$$\mathbf{x}(t) = (\mathbf{p}_1 - \mathbf{p}_0)t + \mathbf{p}_0$$

3. Matrix form:

$$\mathbf{x}(t) = \begin{bmatrix} \mathbf{p}_0 & \mathbf{p}_1 \end{bmatrix} \begin{bmatrix} -1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} t \\ 1 \end{bmatrix}$$

Tangent

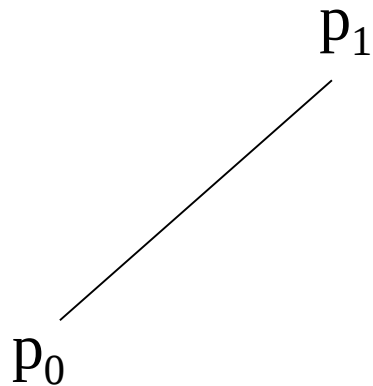
- ▶ Weighted average $\mathbf{x}'(t) = (-1)\mathbf{p}_0 + (+1)\mathbf{p}_1$
- ▶ Polynomial $\mathbf{x}'(t) = 0t + (\mathbf{p}_1 - \mathbf{p}_0)$
- ▶ Matrix form $\mathbf{x}'(t) = \begin{bmatrix} \mathbf{p}_0 & \mathbf{p}_1 \end{bmatrix} \begin{bmatrix} -1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix}$

Lecture Overview

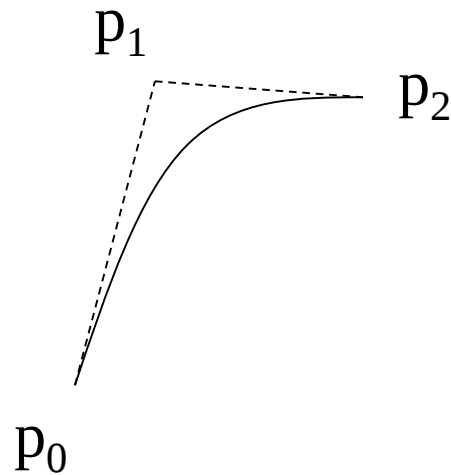
- ▶ Polynomial Curves
 - ▶ Introduction
 - ▶ Polynomial functions
- ▶ Bézier Curves
 - ▶ **Introduction**
 - ▶ Drawing Bézier curves
 - ▶ Piecewise Bézier curves

Bézier Curves

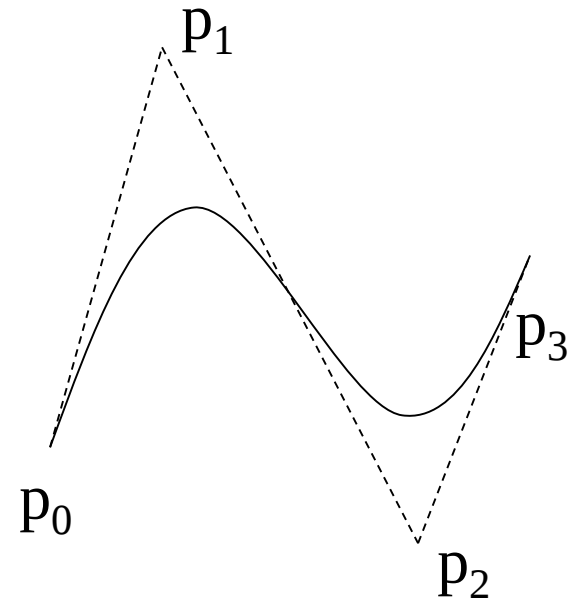
- ▶ Are a higher order extension of linear interpolation



Linear



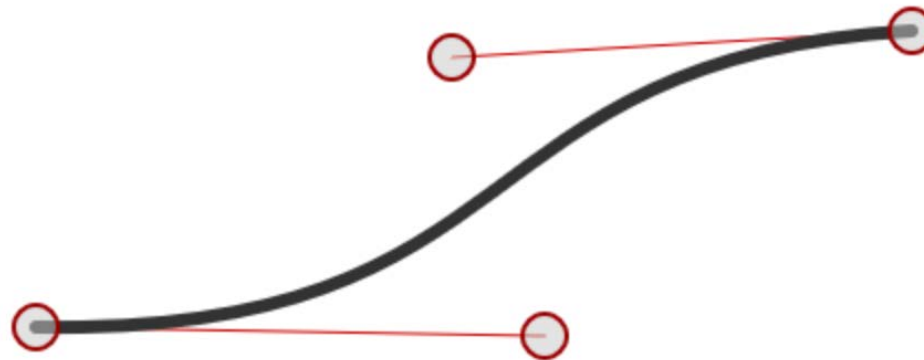
Quadratic



Cubic

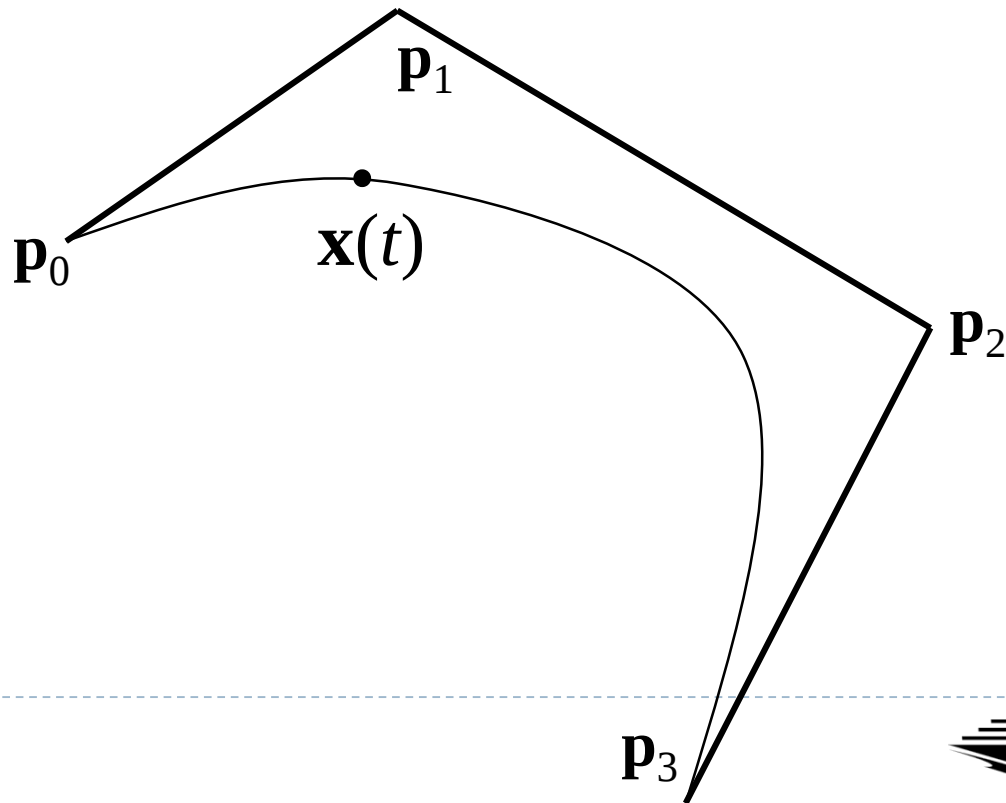
Bézier Curves

- ▶ Give intuitive control over curve with control points
 - ▶ Endpoints are interpolated, intermediate points are approximated
- ▶ Demo:
 - ▶ <http://blogs.sitepointstatic.com/examples/tech/canvas-curves/bezier-curve.html>



Cubic Bézier Curve

- ▶ Most commonly used case
- ▶ Defined by four control points:
 - ▶ Two interpolated endpoints (points are on the curve)
 - ▶ Two points control the tangents at the endpoints
- ▶ Points $\mathbf{x}$ on curve defined as function of parameter t



Algorithmic Construction

- ▶ **Algorithmic construction**
 - ▶ *De Casteljau* algorithm, developed at Citroen in 1959, named after its inventor Paul de Casteljau (pronounced “Cast-all-’Joe”)
 - ▶ Developed independently from Bézier’s work: Bézier created the formulation using blending functions, Casteljau devised the recursive interpolation algorithm

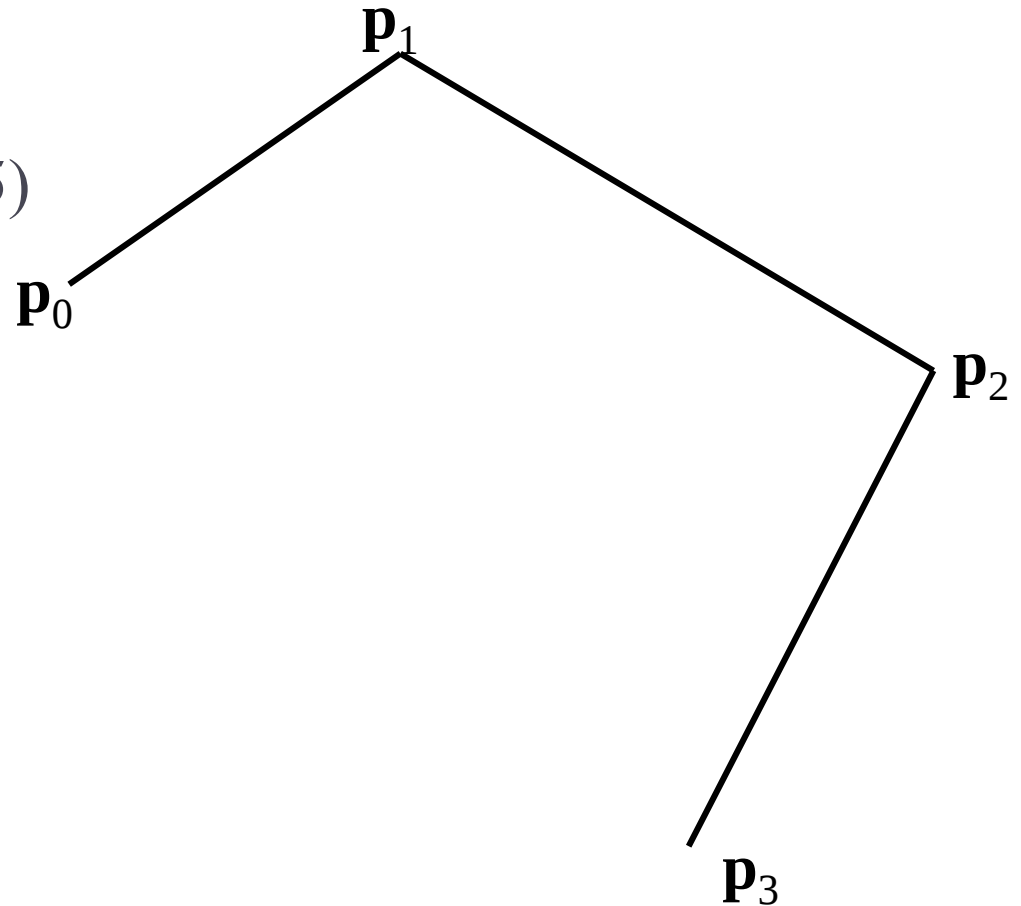
De Casteljau Algorithm

- ▶ A recursive series of linear interpolations
 - ▶ Works for any order Bezier function, not only cubic
- ▶ Not very efficient to evaluate
 - ▶ Other forms more commonly used
- ▶ But:
 - ▶ Gives intuition about the geometry
 - ▶ Useful for subdivision

De Casteljau Algorithm

► Given:

- Four control points
- A value of t (here $t \approx 0.25$)

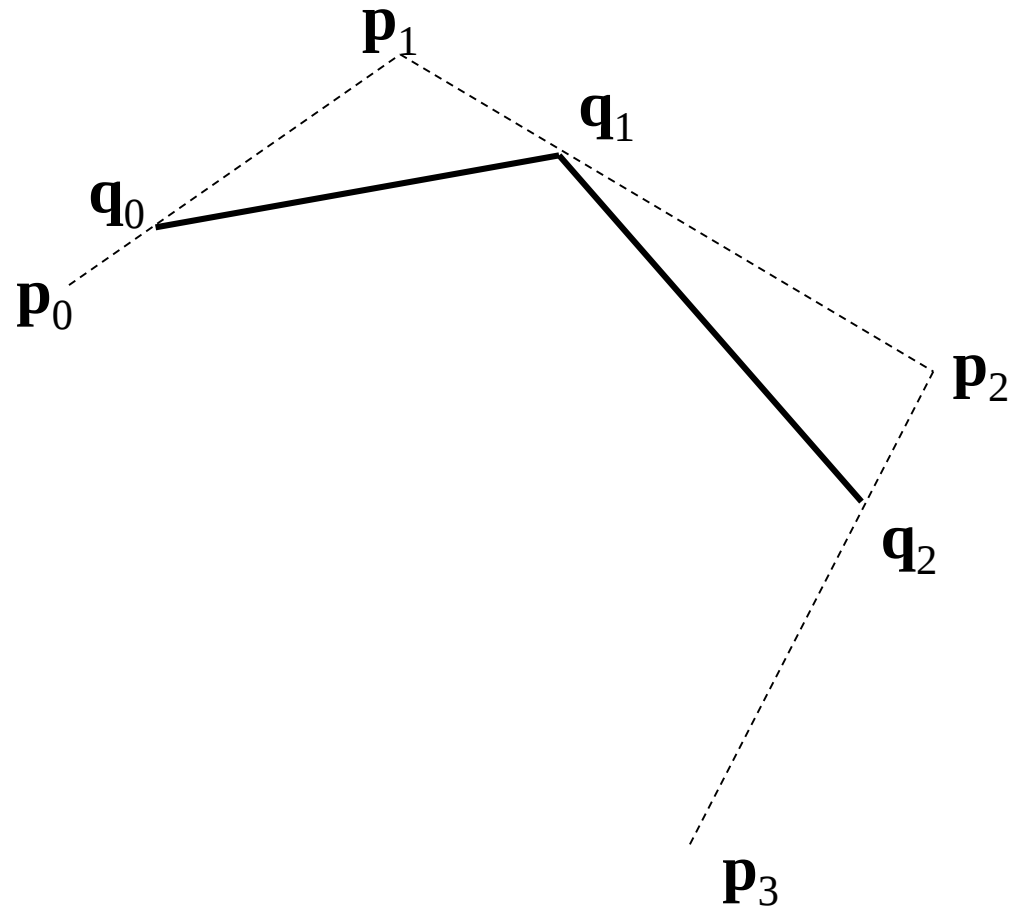


De Casteljau Algorithm

$$\mathbf{q}_0(t) = \text{Lerp}(t, \mathbf{p}_0, \mathbf{p}_1)$$

$$\mathbf{q}_1(t) = \text{Lerp}(t, \mathbf{p}_1, \mathbf{p}_2)$$

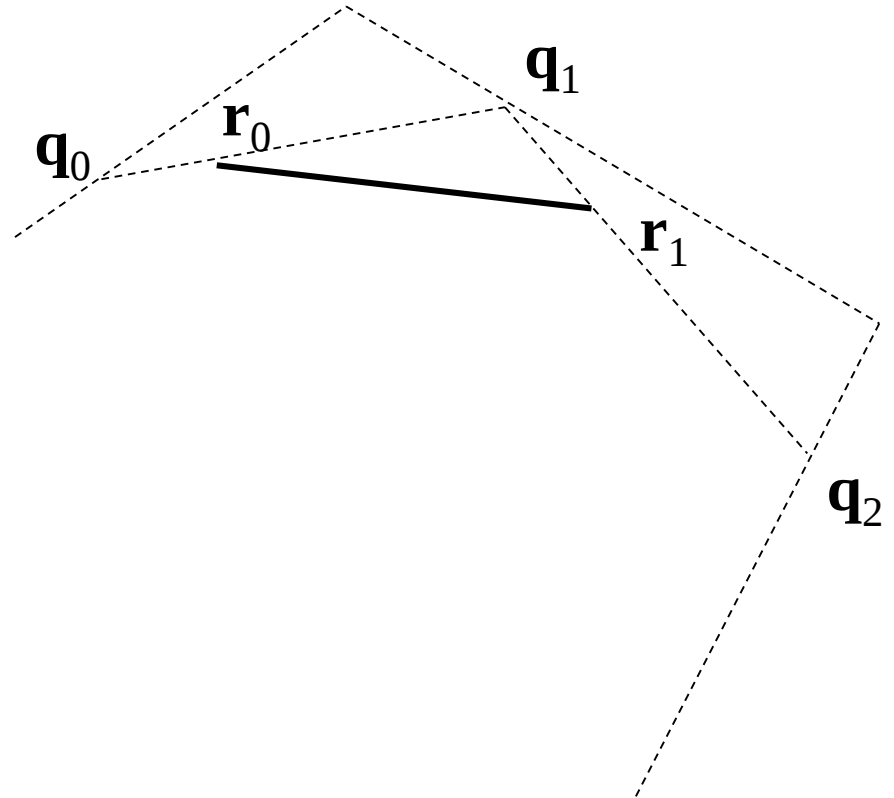
$$\mathbf{q}_2(t) = \text{Lerp}(t, \mathbf{p}_2, \mathbf{p}_3)$$



De Casteljau Algorithm

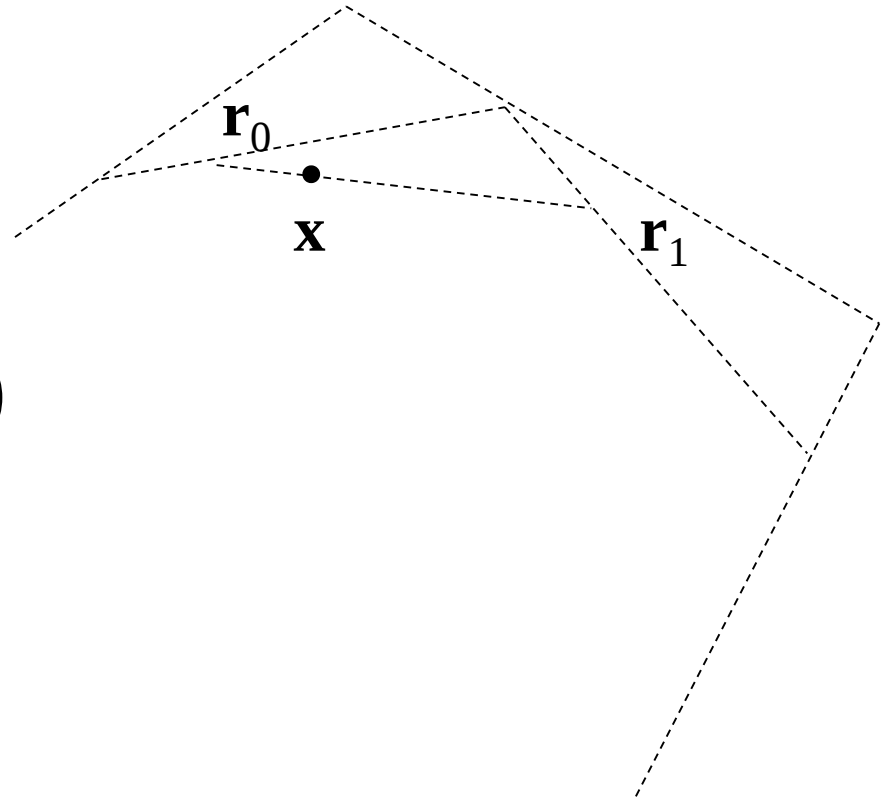
$$\mathbf{r}_0(t) = \text{Lerp}(t, \mathbf{q}_0(t), \mathbf{q}_1(t))$$

$$\mathbf{r}_1(t) = \text{Lerp}(t, \mathbf{q}_1(t), \mathbf{q}_2(t))$$

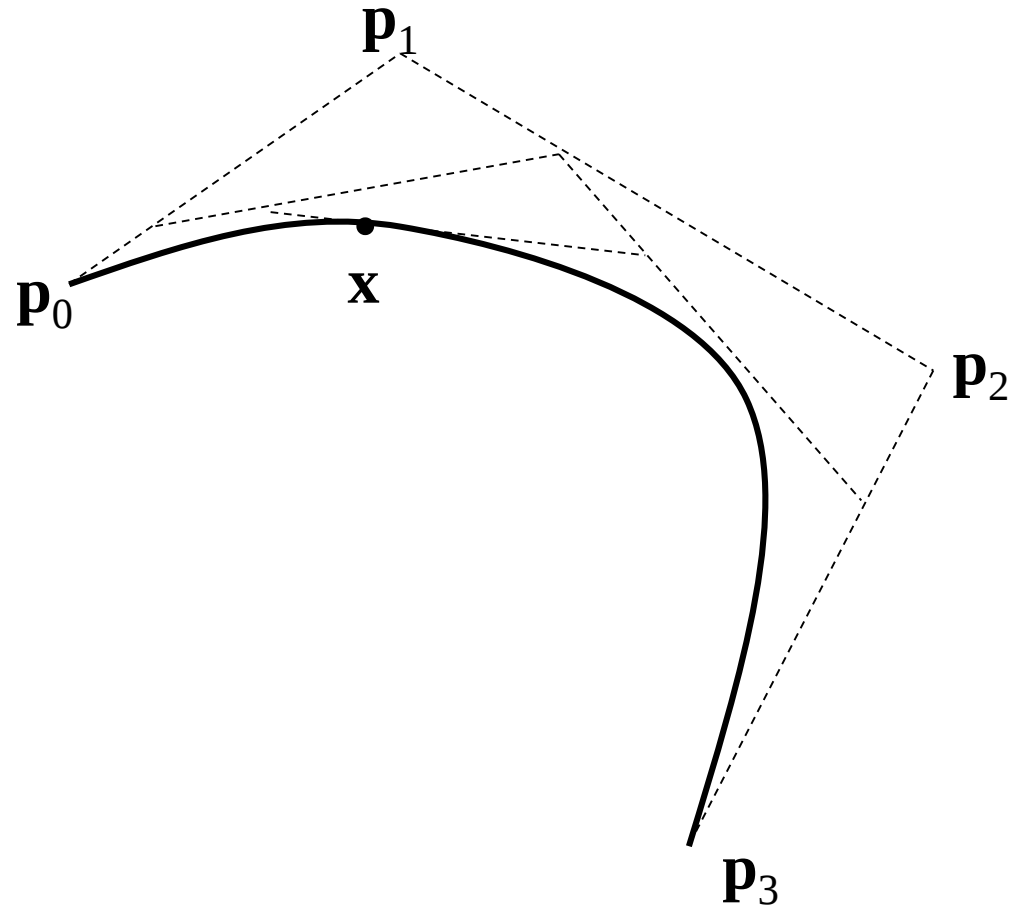


De Casteljau Algorithm

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



De Casteljau Algorithm

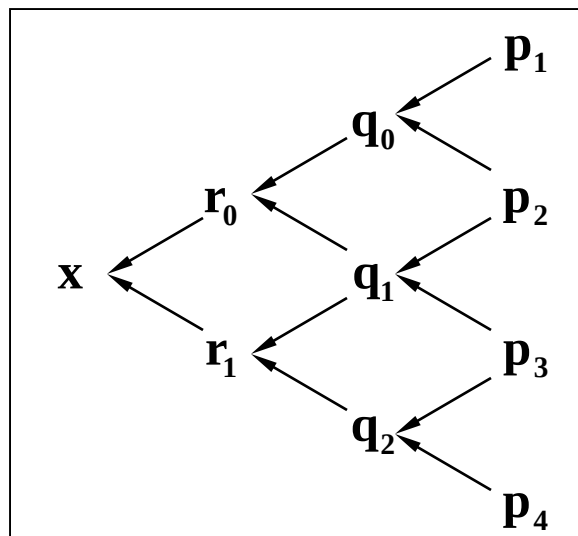


► Demo

- <https://www.jasondavies.com/animated-bezier/>

Recursive Linear Interpolation

$$\begin{aligned}
 \mathbf{x} = \text{Lerp}(t, \mathbf{r}_0, \mathbf{r}_1) \quad & \mathbf{r}_0 = \text{Lerp}(t, \mathbf{q}_0, \mathbf{q}_1) \quad \mathbf{q}_0 = \text{Lerp}(t, \mathbf{p}_0, \mathbf{p}_1) \quad \mathbf{p}_0 \\
 & \mathbf{r}_1 = \text{Lerp}(t, \mathbf{q}_1, \mathbf{q}_2) \quad \mathbf{q}_1 = \text{Lerp}(t, \mathbf{p}_1, \mathbf{p}_2) \quad \mathbf{p}_1 \\
 & \quad \quad \quad \mathbf{q}_2 = \text{Lerp}(t, \mathbf{p}_2, \mathbf{p}_3) \quad \mathbf{p}_2 \\
 & \quad \quad \quad \quad \quad \mathbf{p}_3
 \end{aligned}$$



Expand the LERPs

$$\mathbf{q}_0(t) = \text{Lerp}(t, \mathbf{p}_0, \mathbf{p}_1) = (1-t)\mathbf{p}_0 + t\mathbf{p}_1$$

$$\mathbf{q}_1(t) = \text{Lerp}(t, \mathbf{p}_1, \mathbf{p}_2) = (1-t)\mathbf{p}_1 + t\mathbf{p}_2$$

$$\mathbf{q}_2(t) = \text{Lerp}(t, \mathbf{p}_2, \mathbf{p}_3) = (1-t)\mathbf{p}_2 + t\mathbf{p}_3$$

$$\mathbf{r}_0(t) = \text{Lerp}(t, \mathbf{q}_0(t), \mathbf{q}_1(t)) = (1-t)((1-t)\mathbf{p}_0 + t\mathbf{p}_1) + t((1-t)\mathbf{p}_1 + t\mathbf{p}_2)$$

$$\mathbf{r}_1(t) = \text{Lerp}(t, \mathbf{q}_1(t), \mathbf{q}_2(t)) = (1-t)((1-t)\mathbf{p}_1 + t\mathbf{p}_2) + t((1-t)\mathbf{p}_2 + t\mathbf{p}_3)$$

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

$$\begin{aligned} &= (1-t)((1-t)((1-t)\mathbf{p}_0 + t\mathbf{p}_1) + t((1-t)\mathbf{p}_1 + t\mathbf{p}_2)) \\ &\quad + t((1-t)((1-t)\mathbf{p}_1 + t\mathbf{p}_2) + t((1-t)\mathbf{p}_2 + t\mathbf{p}_3)) \end{aligned}$$

Weighted Average of Control Points

- ▶ Regroup for p :

$$\mathbf{x}(t) = (1-t)((1-t)((1-t)\mathbf{p}_0 + t\mathbf{p}_1) + t((1-t)\mathbf{p}_1 + t\mathbf{p}_2)) \\ + t((1-t)((1-t)\mathbf{p}_1 + t\mathbf{p}_2) + t((1-t)\mathbf{p}_2 + t\mathbf{p}_3))$$

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

$$\mathbf{x}(t) = \overbrace{(-t^3 + 3t^2 - 3t + 1)}^{B_0(t)} \mathbf{p}_0 + \overbrace{(3t^3 - 6t^2 + 3t)}^{B_1(t)} \mathbf{p}_1 \\ + \underbrace{(-3t^3 + 3t^2)}_{B_2(t)} \mathbf{p}_2 + \underbrace{(t^3)}_{B_3(t)} \mathbf{p}_3$$

Cubic Bernstein Polynomials

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

The cubic *Bernstein polynomials* :

$$B_0(t) = -t^3 + 3t^2 - 3t + 1$$

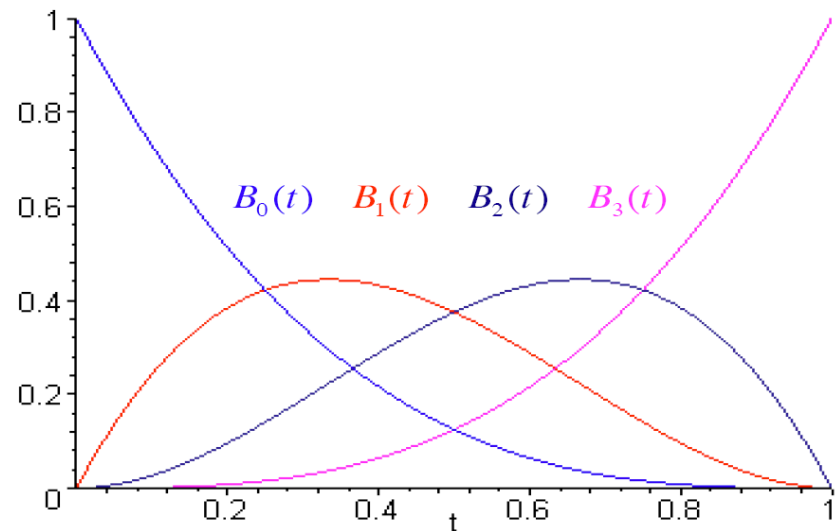
$$B_1(t) = 3t^3 - 6t^2 + 3t$$

$$B_2(t) = -3t^3 + 3t^2$$

$$B_3(t) = t^3$$

$$\sum B_i(t) = 1$$

Bernstein Cubic Polynomials



- Weights $B_i(t)$ add up to 1 for any value of t

General Bernstein Polynomials

$$B_0^1(t) = -t + 1$$

$$B_1^1(t) = t$$

$$B_0^2(t) = t^2 - 2t + 1$$

$$B_1^2(t) = -2t^2 + 2t$$

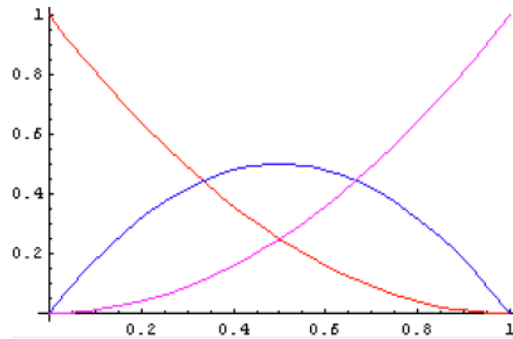
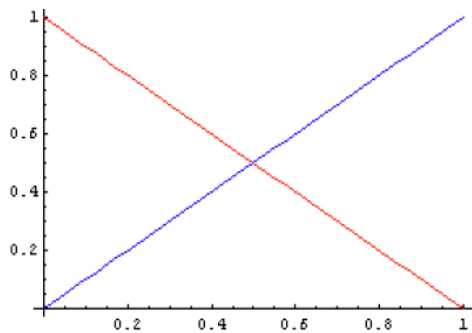
$$B_2^2(t) = t^2$$

$$B_0^3(t) = -t^3 + 3t^2 - 3t + 1$$

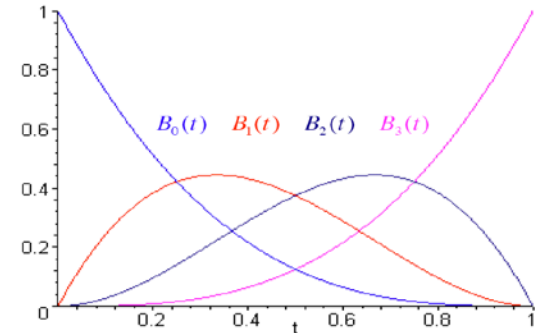
$$B_1^3(t) = 3t^3 - 6t^2 + 3t$$

$$B_2^3(t) = -3t^3 + 3t^2$$

$$B_3^3(t) = t^3$$



Bernstein Cubic Polynomials



$$B_i^n(t) = \binom{n}{i} (1-t)^{n-i} (t)^i$$

$$\binom{n}{i} = \frac{n!}{i!(n-i)!}$$

$$\sum B_i^n(t) = 1$$

$n!$ = factorial of n

$(n+1)! = n! \times (n+1)$

Any order Bézier Curves

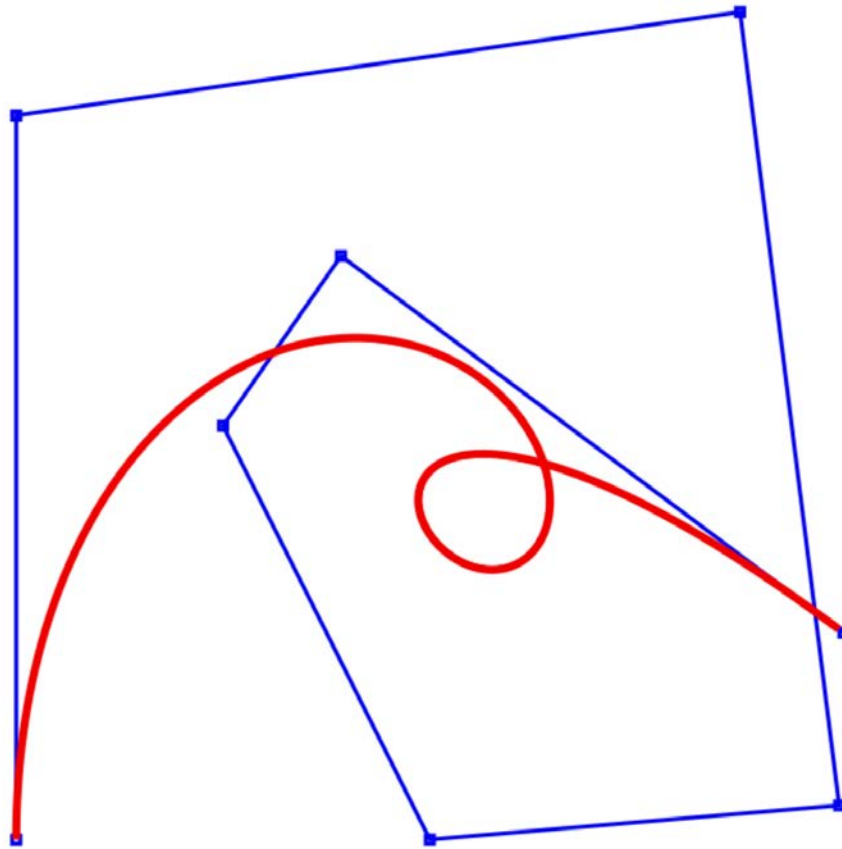
- ▶ n th-order Bernstein polynomials form n th-order Bézier curves

$$B_i^n(t) = \binom{n}{i} (1-t)^{n-i} (t)^i$$

$$\mathbf{x}(t) = \sum_{i=0}^n B_i^n(t) \mathbf{p}_i$$

Demo: Bezier curves of multiple orders

- ▶ <http://www.ibiblio.org/e-notes/Splines/bezier.html>

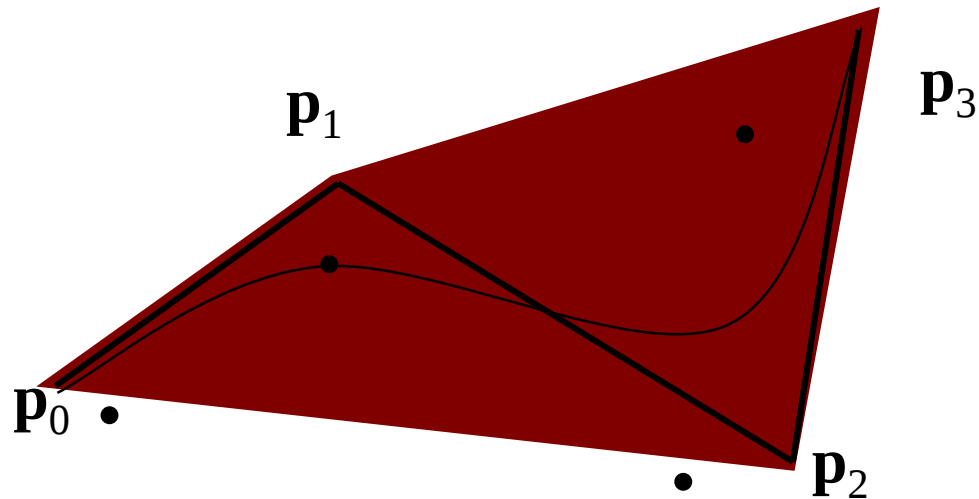


Useful Bézier Curve Properties

- ▶ Convex Hull property
- ▶ Affine Invariance

Convex Hull Property

- ▶ A Bézier curve is always inside the convex hull
 - ▶ Makes curve predictable
 - ▶ Allows culling, intersection testing, adaptive tessellation



Affine Invariance

Transforming Bézier curves

- ▶ Two ways to transform:
 - ▶ First transform control points, then compute spline points
 - ▶ First compute spline points, then transform them
- ▶ Results are identical
 - ▶ Invariant under affine transformations

Cubic Polynomial Form

Start with Bernstein form:

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

Regroup into coefficients of t :

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

$\mathbf{x}(t) = \mathbf{a}t^3 + \mathbf{b}t^2 + \mathbf{c}t + \mathbf{d}$	$\mathbf{a} = (-\mathbf{p}_0 + 3\mathbf{p}_1 - 3\mathbf{p}_2 + \mathbf{p}_3)$
	$\mathbf{b} = (3\mathbf{p}_0 - 6\mathbf{p}_1 + 3\mathbf{p}_2)$
	$\mathbf{c} = (-3\mathbf{p}_0 + 3\mathbf{p}_1)$
	$\mathbf{d} = (\mathbf{p}_0)$

- ▶ Good for fast evaluation
 - ▶ Precompute constant coefficients ($\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}$)
- ▶ Not much geometric intuition

Cubic Matrix Form

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

$$\begin{aligned} \vec{\mathbf{a}} &= (-\mathbf{p}_0 + 3\mathbf{p}_1 - 3\mathbf{p}_2 + \mathbf{p}_3) \\ \vec{\mathbf{b}} &= (3\mathbf{p}_0 - 6\mathbf{p}_1 + 3\mathbf{p}_2) \\ \vec{\mathbf{c}} &= (-3\mathbf{p}_0 + 3\mathbf{p}_1) \\ \mathbf{d} &= (\mathbf{p}_0) \end{aligned}$$

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

$$x(t) = G_{Bez} B_{Bez} T = C T$$

Matrix Form

- ▶ Other types of cubic splines use different basis matrices
- ▶ Efficient evaluation
 - ▶ Pre-compute $\mathbf{C}$
 - ▶ Use 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 $\mathbf{x}_i = \mathbf{x}(t_i)$ where $t_i = \frac{i}{N}$ for $i = 0, 1, \dots, N$

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

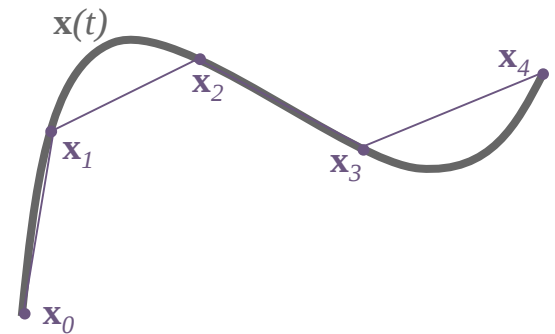
- ▶ Connect points with lines

- ▶ Too few points?

- ▶ Poor approximation: “curve” is faceted

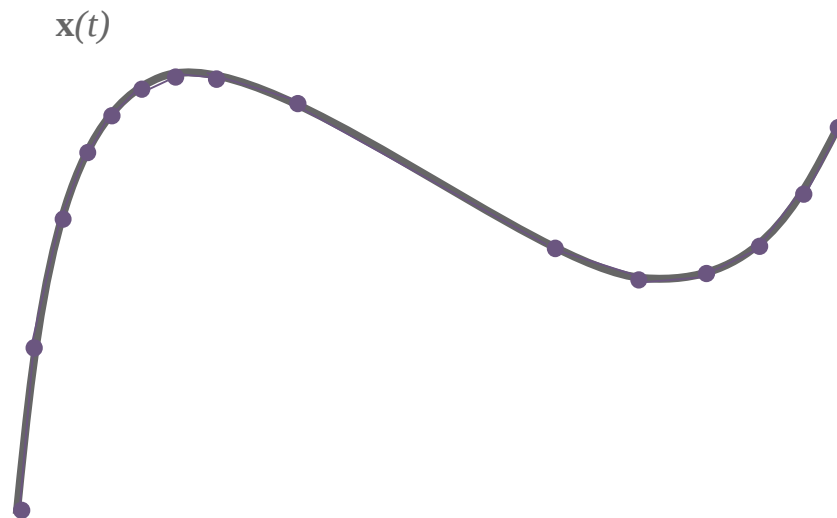
- ▶ Too many points?

- ▶ Slow to draw too many line segments



Adaptive Sampling

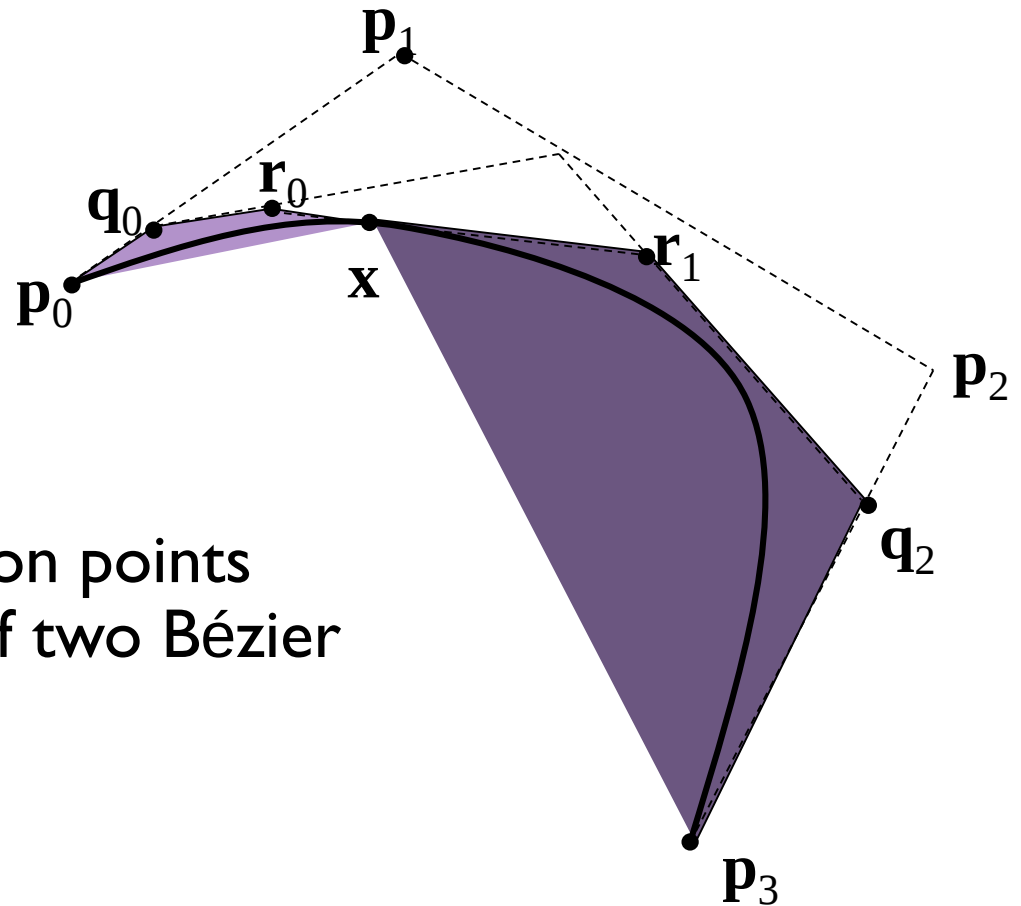
- ▶ Use only as many line segments as you need
 - ▶ Fewer segments where curve is mostly flat
 - ▶ More segments where curve bends
 - ▶ Segments never smaller than a pixel



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

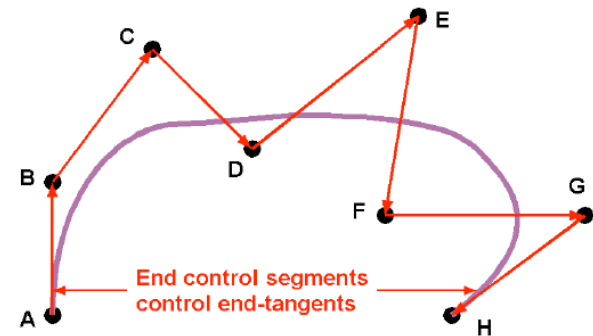
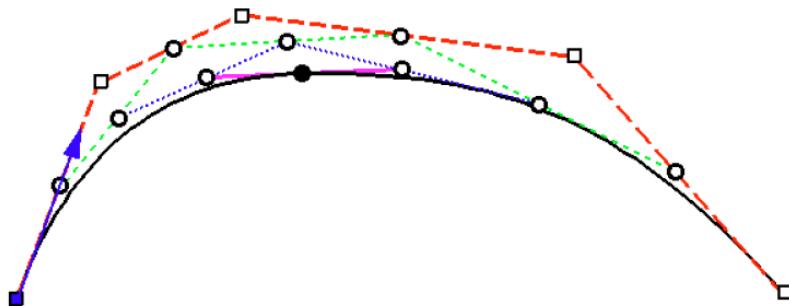
- ▶ Use De Casteljau construction to split Bézier segment in two
- ▶ For each part
 - ▶ If “flat enough”: draw line segment
 - ▶ Else: continue recursion
- ▶ Curve is flat enough if hull is flat enough
 - ▶ Test how far the approximating control points are from a straight segment
 - ▶ If less than one pixel, the hull is flat enough

Lecture Overview

- ▶ Polynomial Curves
 - ▶ Introduction
 - ▶ Polynomial functions
- ▶ Bézier Curves
 - ▶ Introduction
 - ▶ Drawing Bézier curves
 - ▶ Longer 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

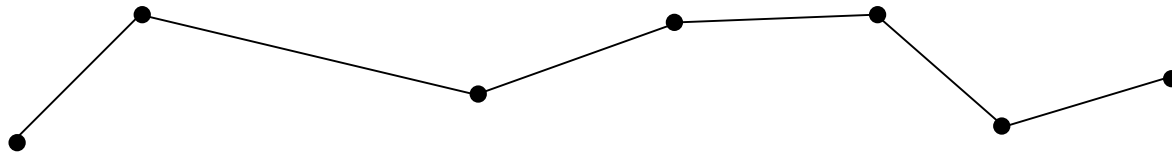


- ▶ Hard to control and hard to work with
 - ▶ Intermediate points don't have obvious effect on shape
 - ▶ Changing any control point changes the whole curve
 - ▶ Want *local support*: each control point only influences nearby portion of curve

Piecewise Curves

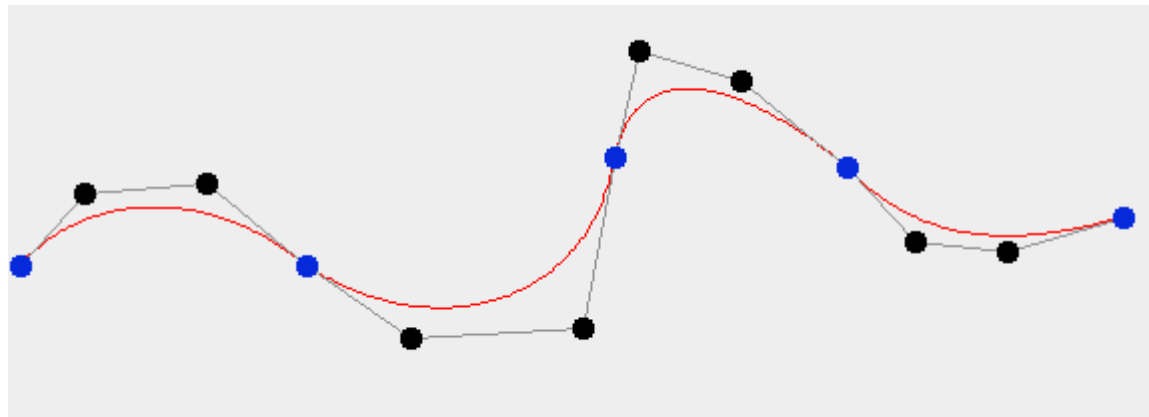
- ▶ Sequence of line segments

- ▶ *Piecewise linear* curve



- ▶ Sequence of cubic curve segments

- ▶ *Piecewise cubic* curve (here piecewise Bézier)



Global Parameterization

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

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

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

- ▶ Alternate solution: u defined from 0 to 1

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

Piecewise Bézier curve

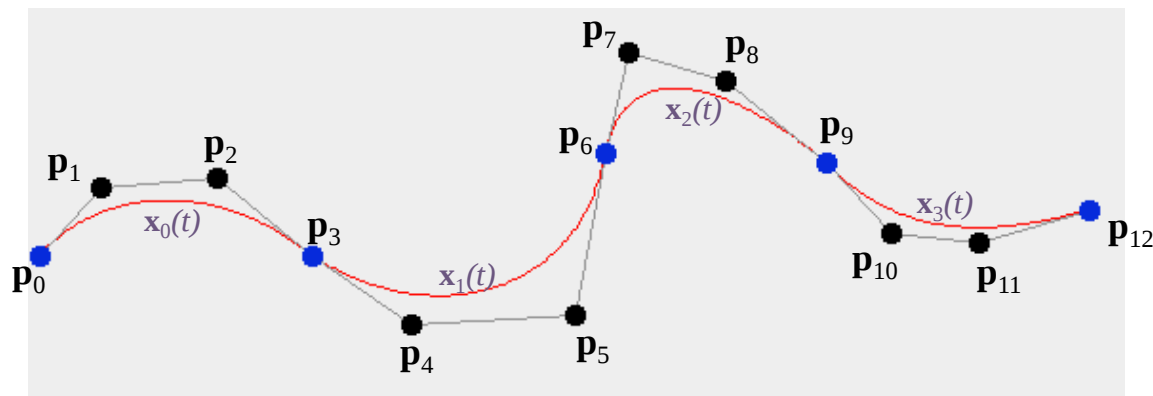
- Given $3N + 1$ points $\mathbf{p}_0, \mathbf{p}_1, \dots, \mathbf{p}_{3N}$
- Define N Bézier segments:

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

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

$\vdots$

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

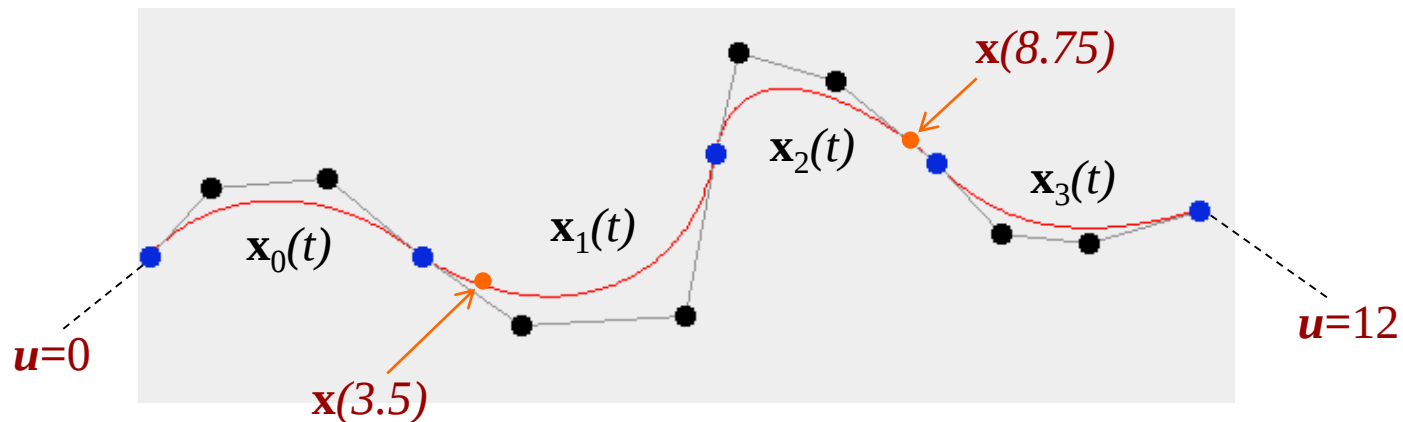


Piecewise Bézier Curve

- ▶ Parameter in $0 \leq u \leq 3N$

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

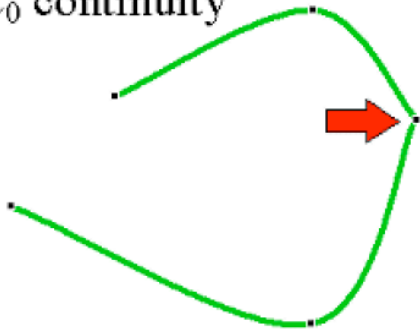
$$\mathbf{x}(u) = \mathbf{x}_i\left(\frac{1}{3}u - i\right), \text{ where } i = \left\lfloor \frac{1}{3}u \right\rfloor$$



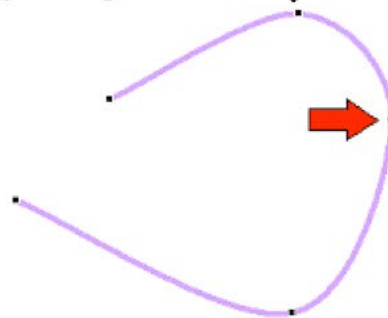
Parametric Continuity

- ▶ 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

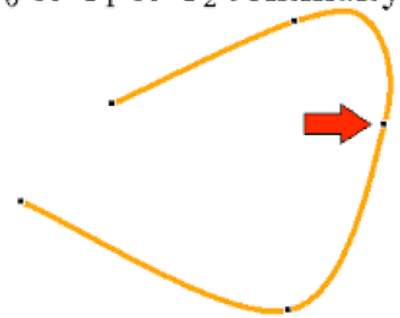
C_0 continuity



C_0 & C_1 continuity

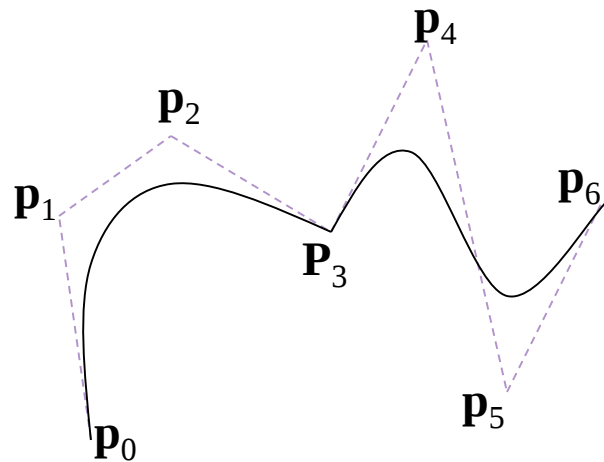
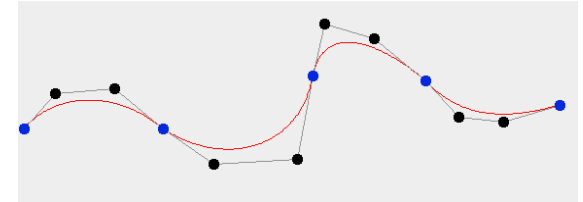


C_0 & C_1 & C_2 continuity

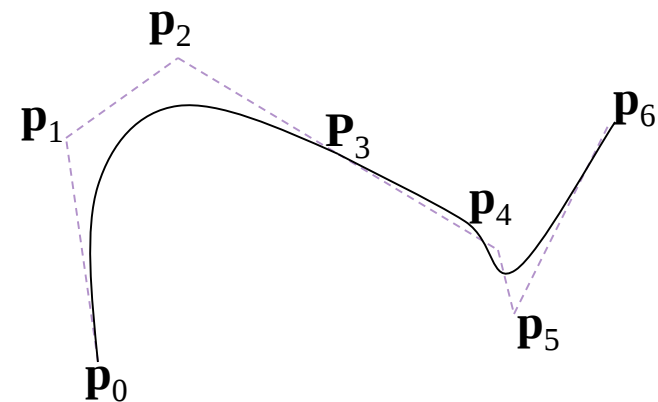


Piecewise Bézier Curve

- ▶ $3N+1$ 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 and rarely necessary



C_1 discontinuous



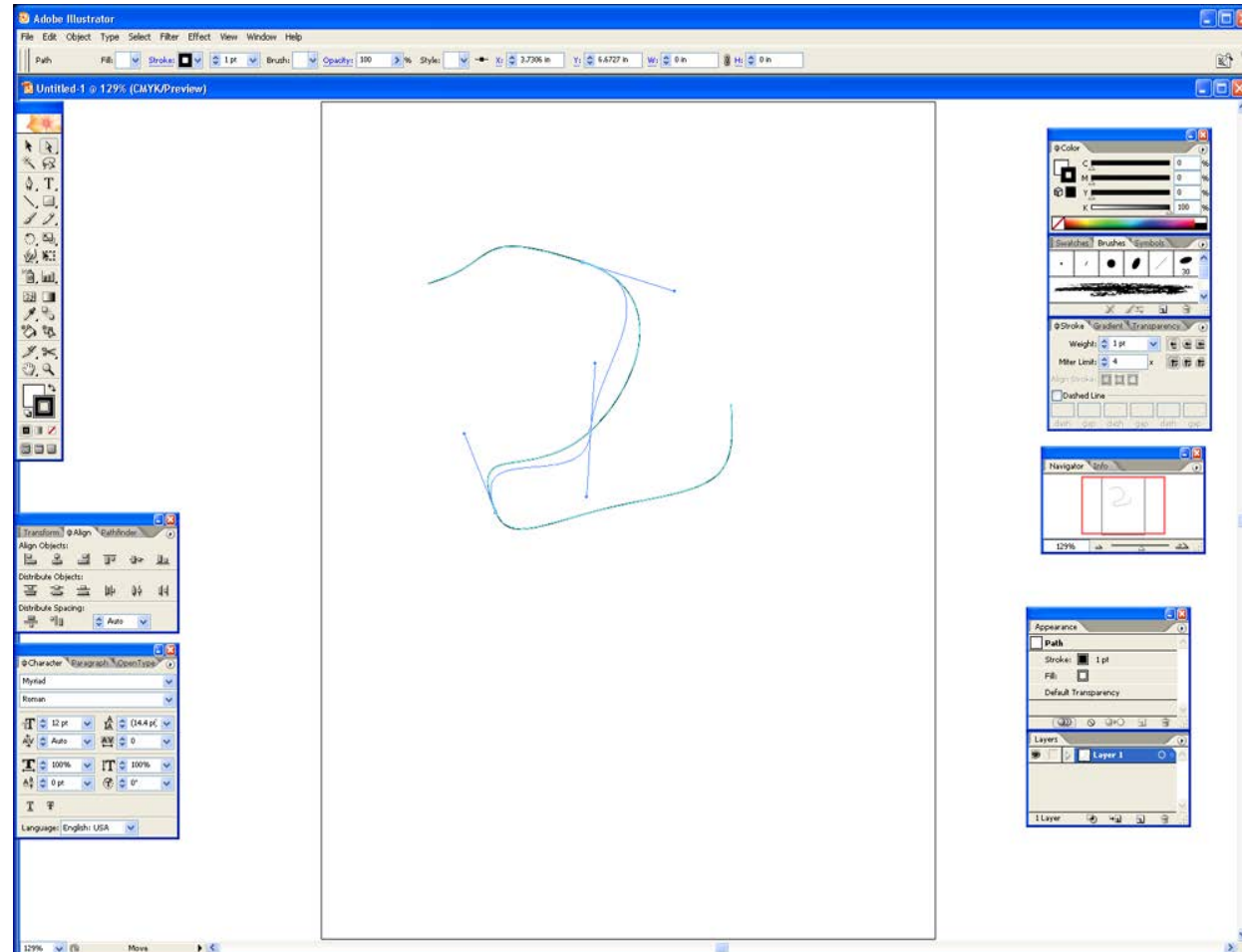
C_1 continuous

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
- ▶ Solutions
 - ▶ User interface using “Bézier handles” to ascertain C^1 continuity
 - ▶ Generalization to B-splines or NURBS

Bézier Handles

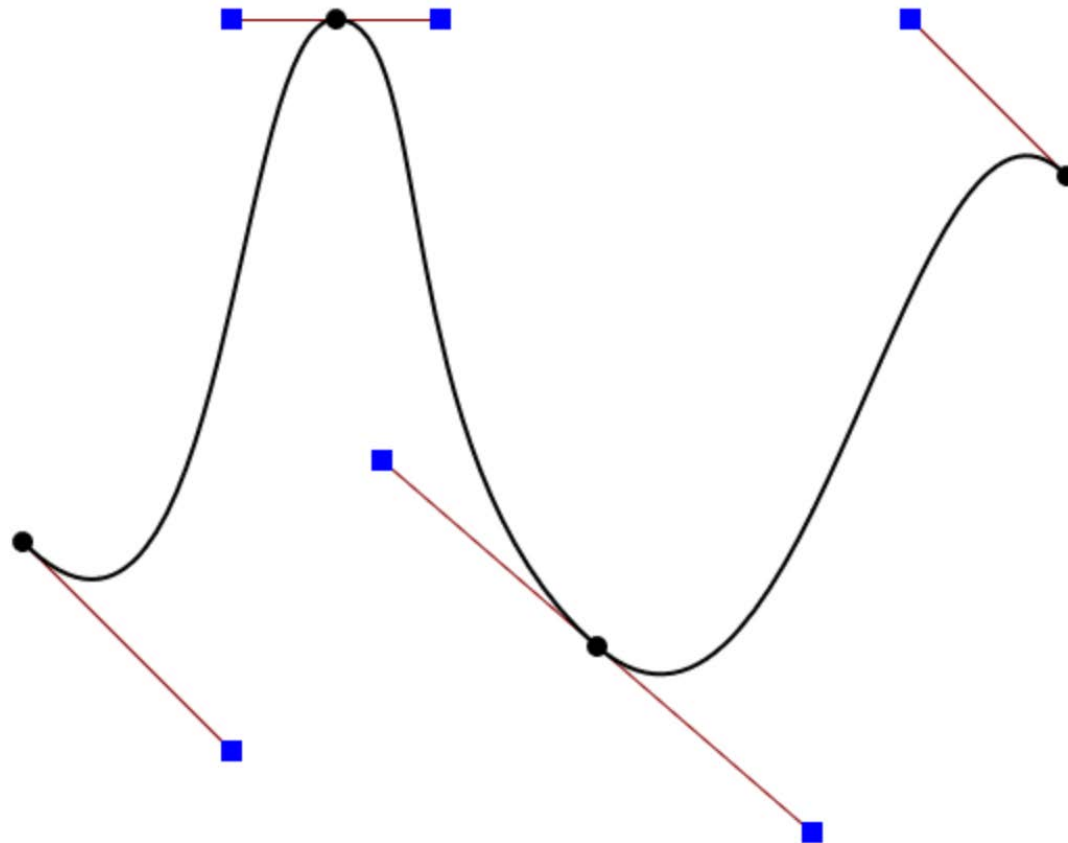
- ▶ Segment end points (interpolating) presented as curve control points
- ▶ Midpoints (approximating points) presented as “handles”
- ▶ Can have option to enforce C_1 continuity



Adobe Illustrator

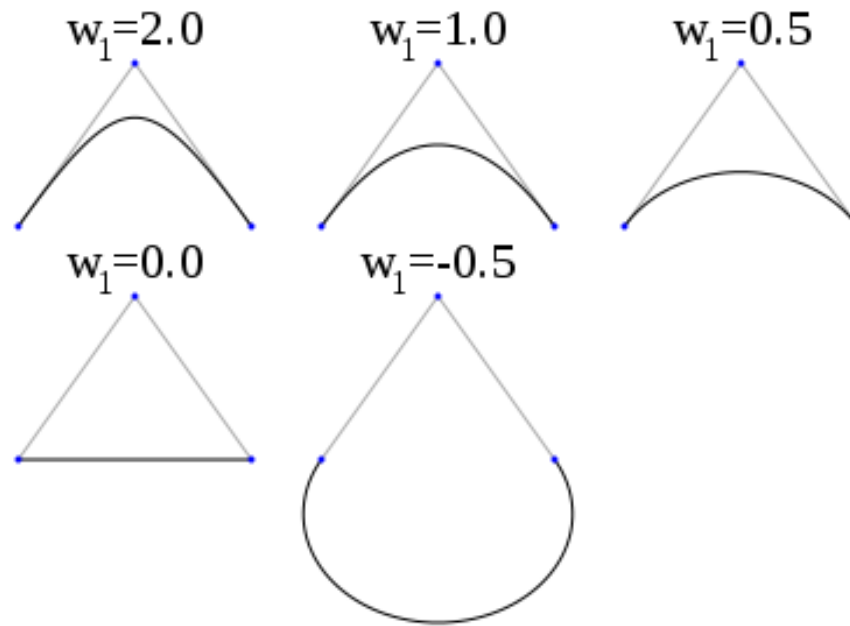
Demo: Bezier handles

- ▶ <http://math.hws.edu/eck/cs424/notes2013/canvas/bezier.html>



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)



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

- ▶ **Non Uniform Rational B-Splines**
- ▶ 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`
- ▶ Demos:
 - ▶ <http://bentonian.com/teaching/AdvGraph0809/demos/Nurbs2d/index.html>
 - ▶ <http://geometrie.foretnik.net/files/NURBS-en.swf>