

Computergrafik

Matthias Zwicker
Universität Bern
Herbst 2016

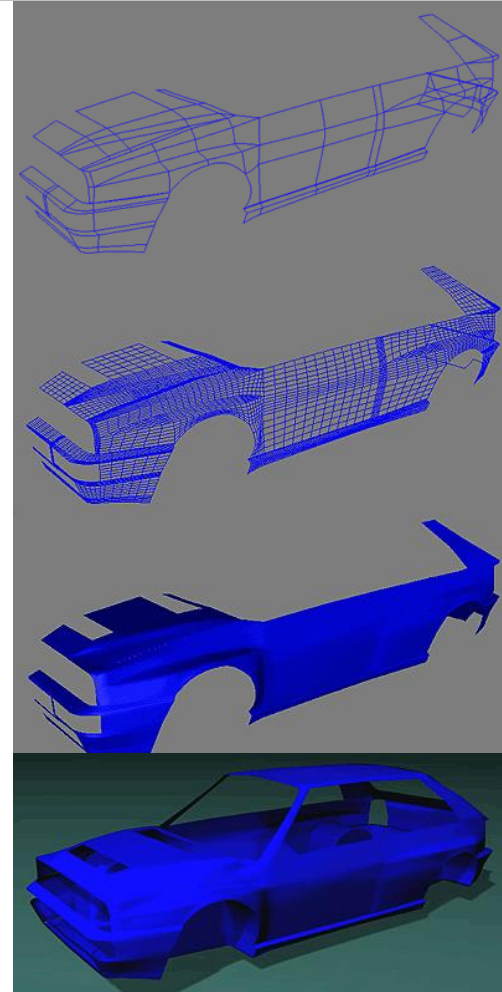
Today

Curves

- Introduction
- Polynomial curves
- Bézier curves
- Drawing Bézier curves
- Piecewise curves

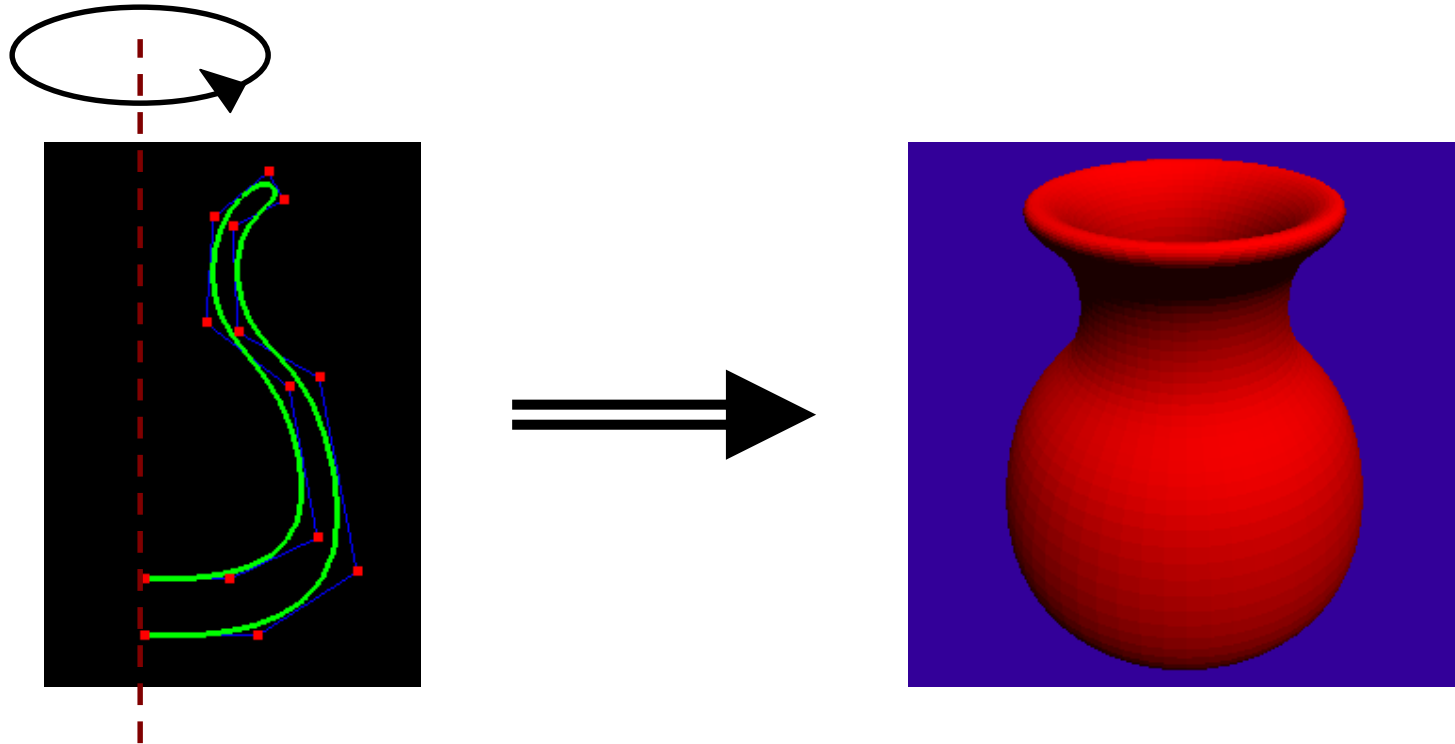
Modeling

- Creating 3D objects
- How to construct complicated surfaces?
- Goal
 - Specify objects with few **control points**
 - Resulting object should be visually pleasing (smooth)
- Start with curves, then generalize to surfaces



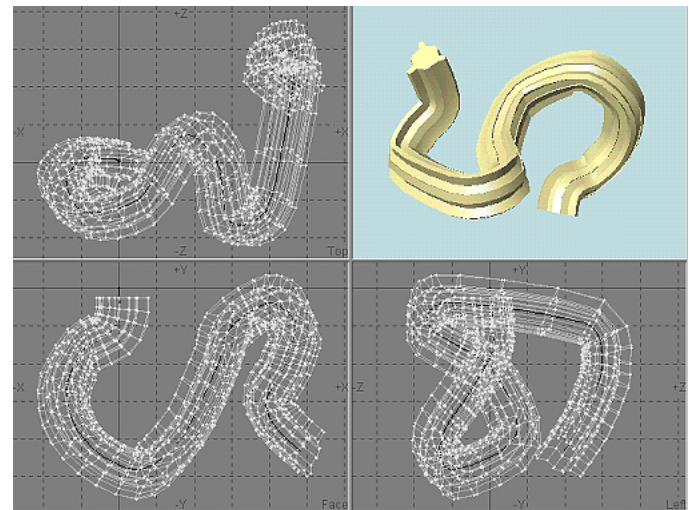
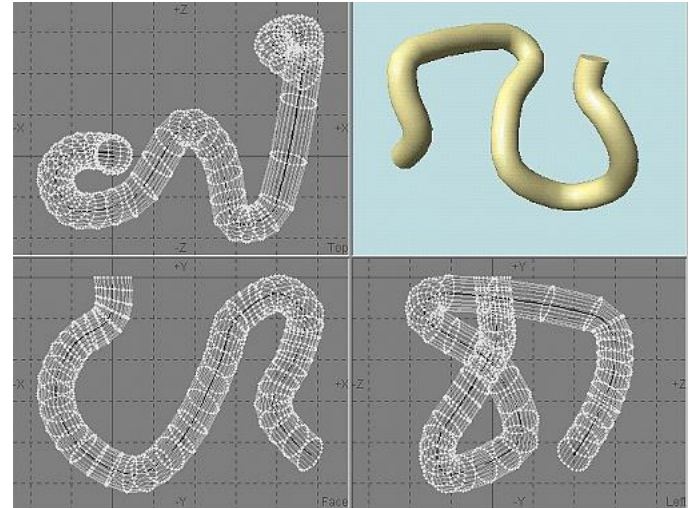
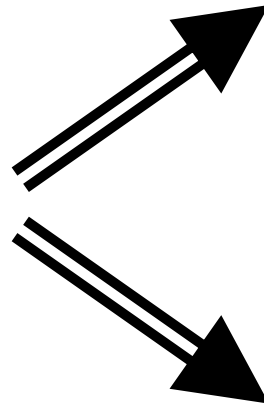
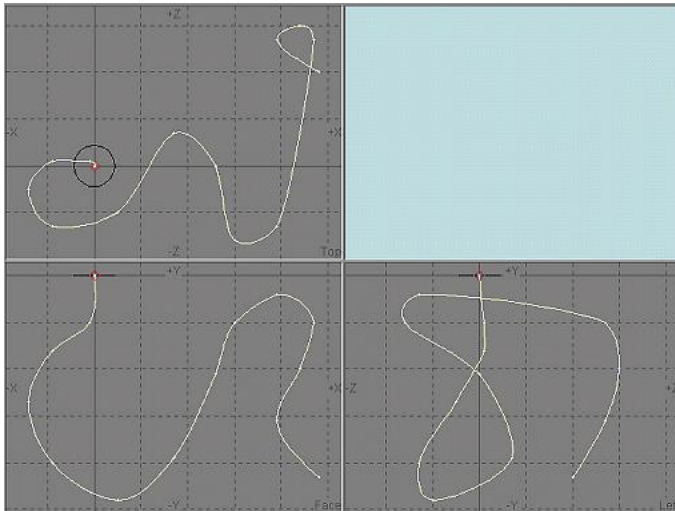
Usefulness of curves

- Surface of revolution



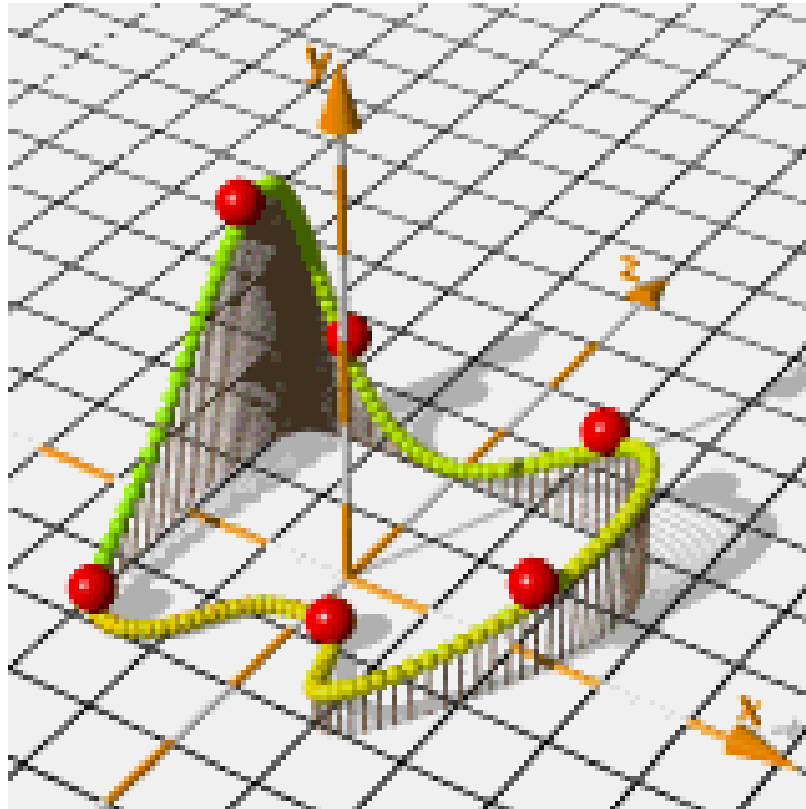
Usefulness of curves

- Extruded/swept surfaces



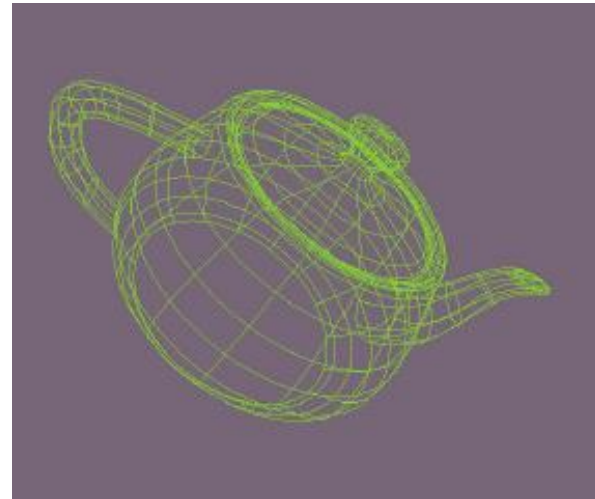
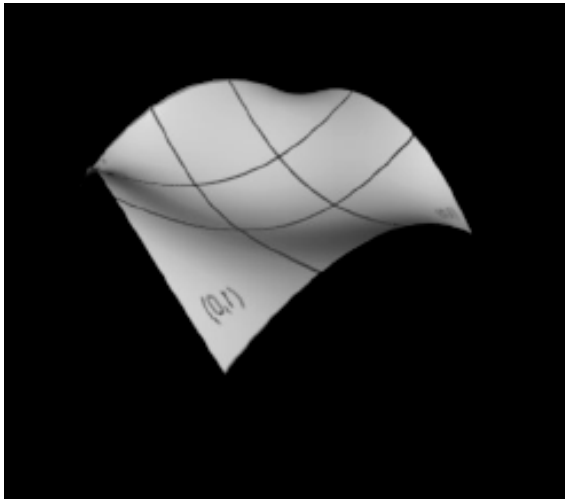
Usefulness of curves

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



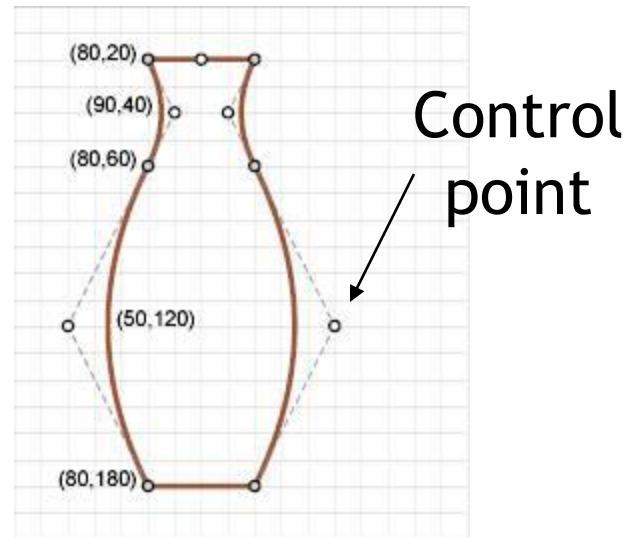
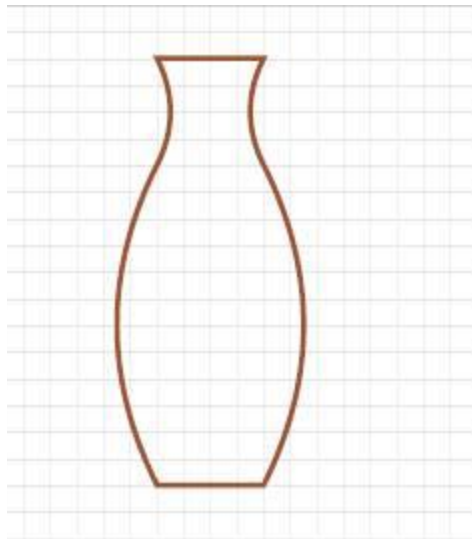
Usefulness of curves

- Generalize to surface patches using “grids of curves”, next class



How to represent curves

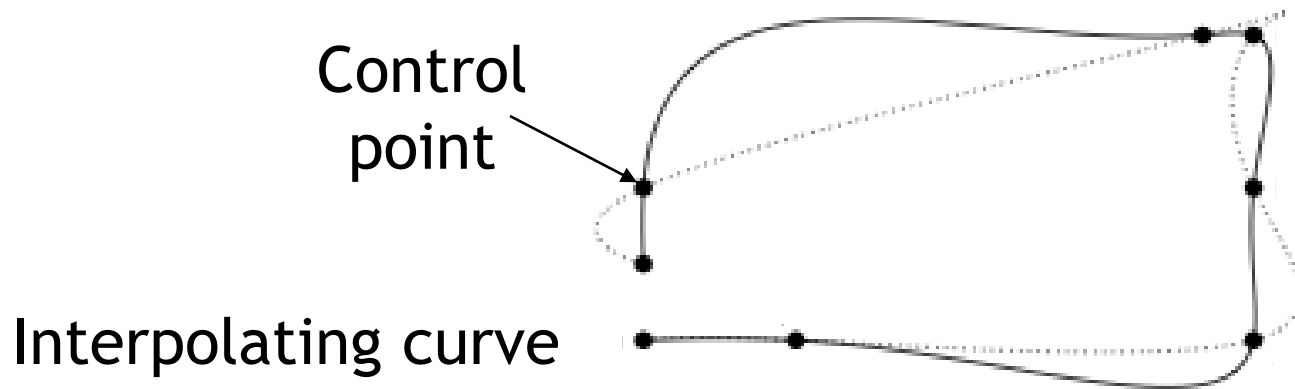
- Specify every point along curve?
 - Hard to get precise, smooth results
 - Too much data, too hard to work with
- Idea: specify curves using small numbers of **control points**
- Mathematics: use **polynomials** to represent curves



Interpolating polynomial curves

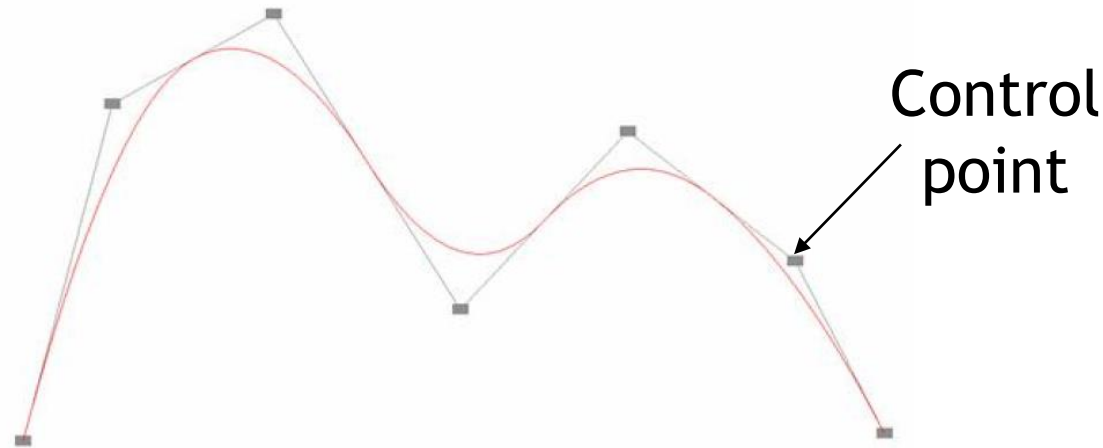
http://en.wikipedia.org/wiki/Polynomial_interpolation

- Curve goes through all control points
- Seems most intuitive
- Surprisingly, not usually the best choice
 - Hard to predict behavior
 - Overshoots, wiggles
 - Hard to get “nice-looking” curves



Approximating polynomial curves

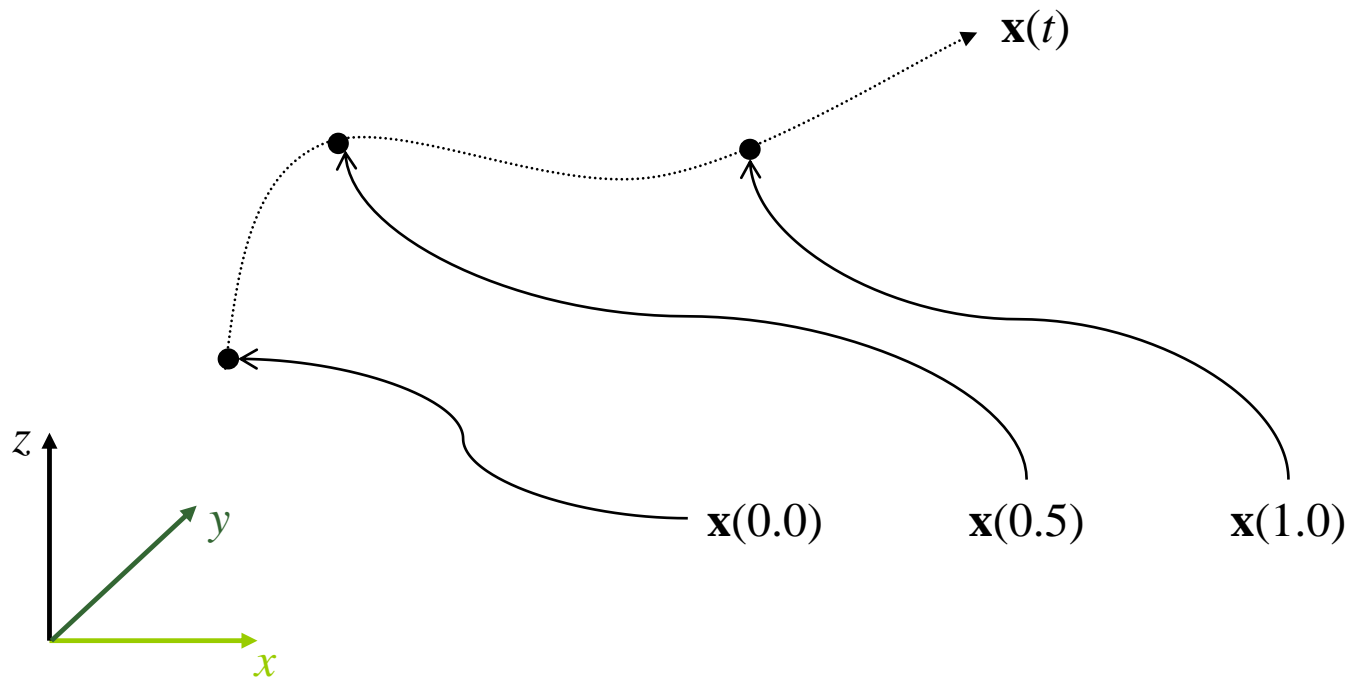
- Curve is “influenced” by control points



- Various types & techniques based on polynomial functions
 - Bézier curves, B-splines, NURBS
- Focus on Bézier curves

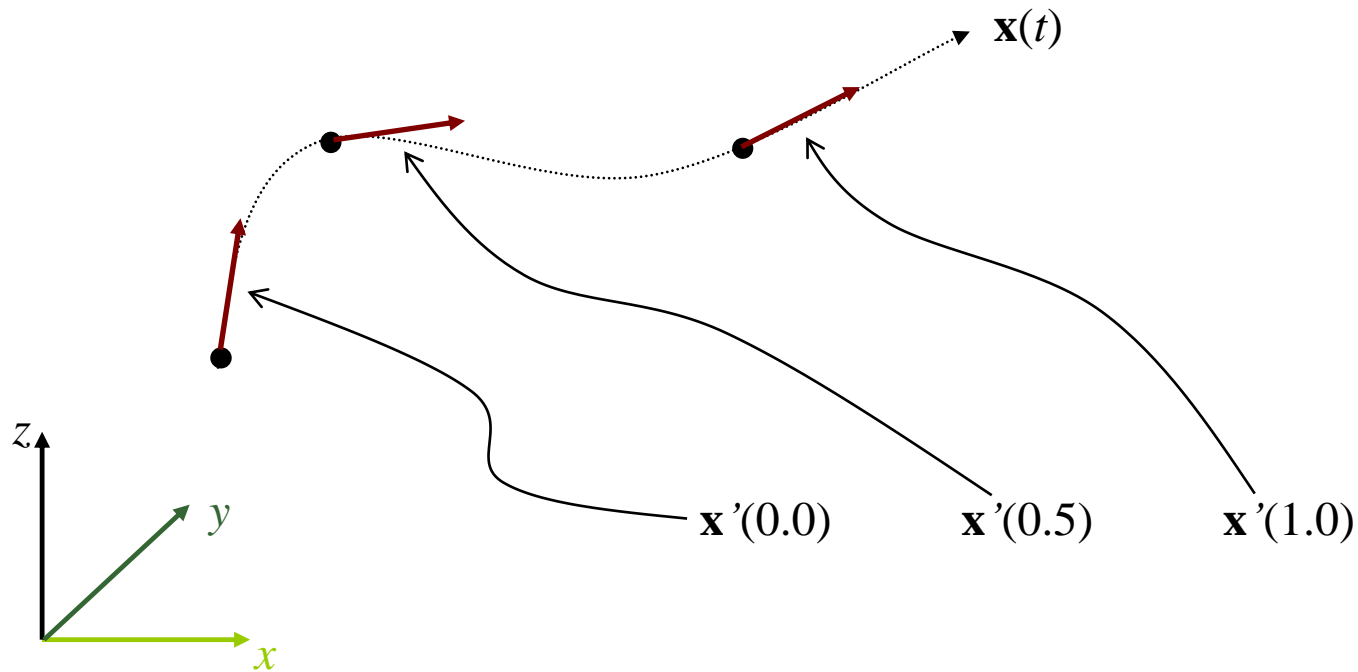
Mathematical definition

- A vector valued function of one variable $\mathbf{x}(t)$
 - Given t , compute a 3D point $\mathbf{x}=(x,y,z)$
 - May interpret as three functions $x(t)$, $y(t)$, $z(t)$
 - “Moving a point along the curve”



Tangent vector

- Derivative $\mathbf{x}'(t) = \frac{d\mathbf{x}}{dt} = (x'(t), y'(t), z'(t))$
- A vector that points in the direction of movement
- Length of $\mathbf{x}'(t)$ corresponds to speed



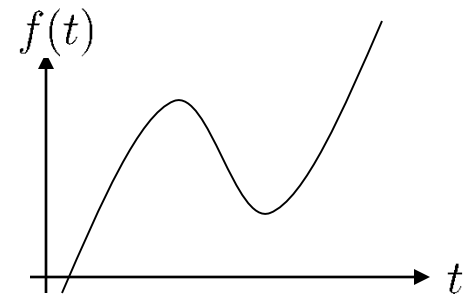
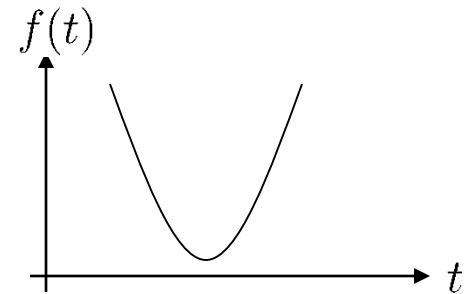
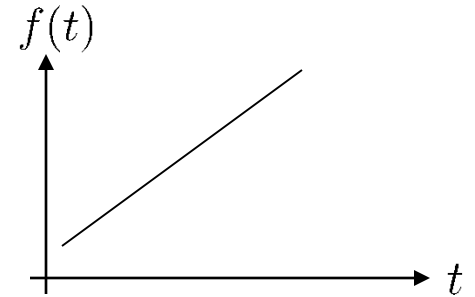
Today

Curves

- Introduction
- Polynomial curves
- Bézier curves
- Drawing Bézier curves
- Piecewise curves

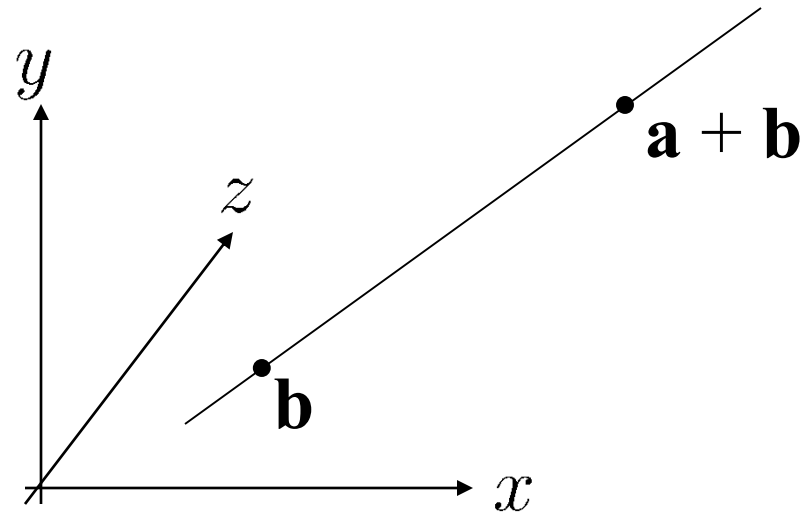
Polynomial functions

- **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)



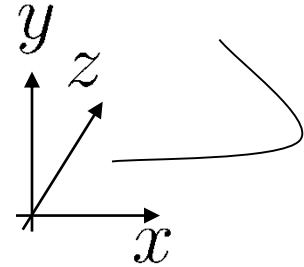
Polynomial curves

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

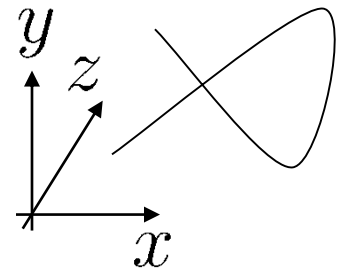


Polynomial curves

- 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

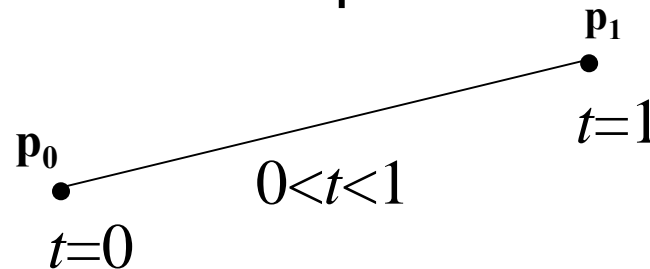
- Polynomial coefficients a, b, c, d etc. can be interpreted as 3D control points
 - Remember a, b, c, d have x, y, z components each
- Unfortunately, polynomial coefficients don't intuitively describe shape of curve
- Main objective of curve representation is to come up with intuitive control points
 - Position of control points predicts shape of curve

Control points

- How many control points?
 - Two points define a line (1st order)
 - Three points define a quadratic curve (2nd order)
 - Four points define a cubic curve (3rd order)
 - $k+1$ points define a k -order curve
- Let's start with a line...

First order curve

- Based on **linear interpolation (LERP)**
http://en.wikipedia.org/wiki/Linear_interpolation
 - 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 curve)
 - $t=0$ corresponds to \mathbf{p}_0
 - $t=1$ corresponds to \mathbf{p}_1
 - $t=0.5$ corresponds to midpoint



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

Linear interpolation

- Three different ways to write it
 - Equivalent, but different properties become apparent
 - Advantages for different operations, see later

1. Weighted sum of control points

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

2. Polynomial in t

$$\mathbf{x}(t) = (\mathbf{p}_1 - \mathbf{p}_0)t + \mathbf{p}_0 t^0$$

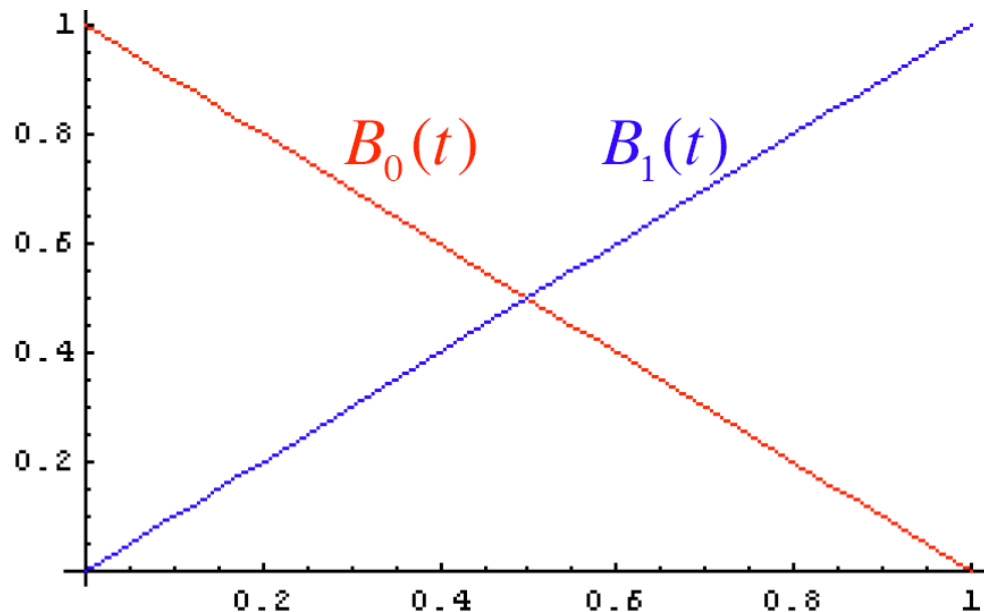
3. Matrix form

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

Weighted sum of control points

$$\begin{aligned}\mathbf{x}(t) &= (1-t)\mathbf{p}_0 + t\mathbf{p}_1 \\ &= B_0(t)\mathbf{p}_0 + B_1(t)\mathbf{p}_1, \text{ where } B_0(t) = 1-t \text{ and } B_1(t) = t\end{aligned}$$

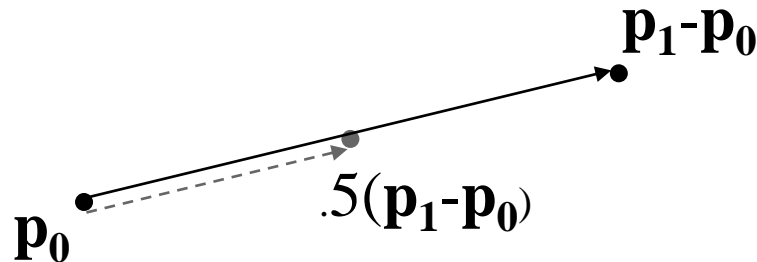
- Weights $B_0(t)$, $B_1(t)$ are functions of t
 - Sum is always 1, for any value of t
 - Also known as **basis** or **blending functions**



Linear polynomial

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

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

Tangent

- For a straight line, the tangent is constant

$$\mathbf{x}'(t) = \mathbf{p}_1 - \mathbf{p}_0$$

- Weighted average

$$\mathbf{x}(t) = \mathbf{p}_0(1 - t) + \mathbf{p}_1t \rightarrow \mathbf{x}'(t) = (-1)\mathbf{p}_0 + (+1)\mathbf{p}_1$$

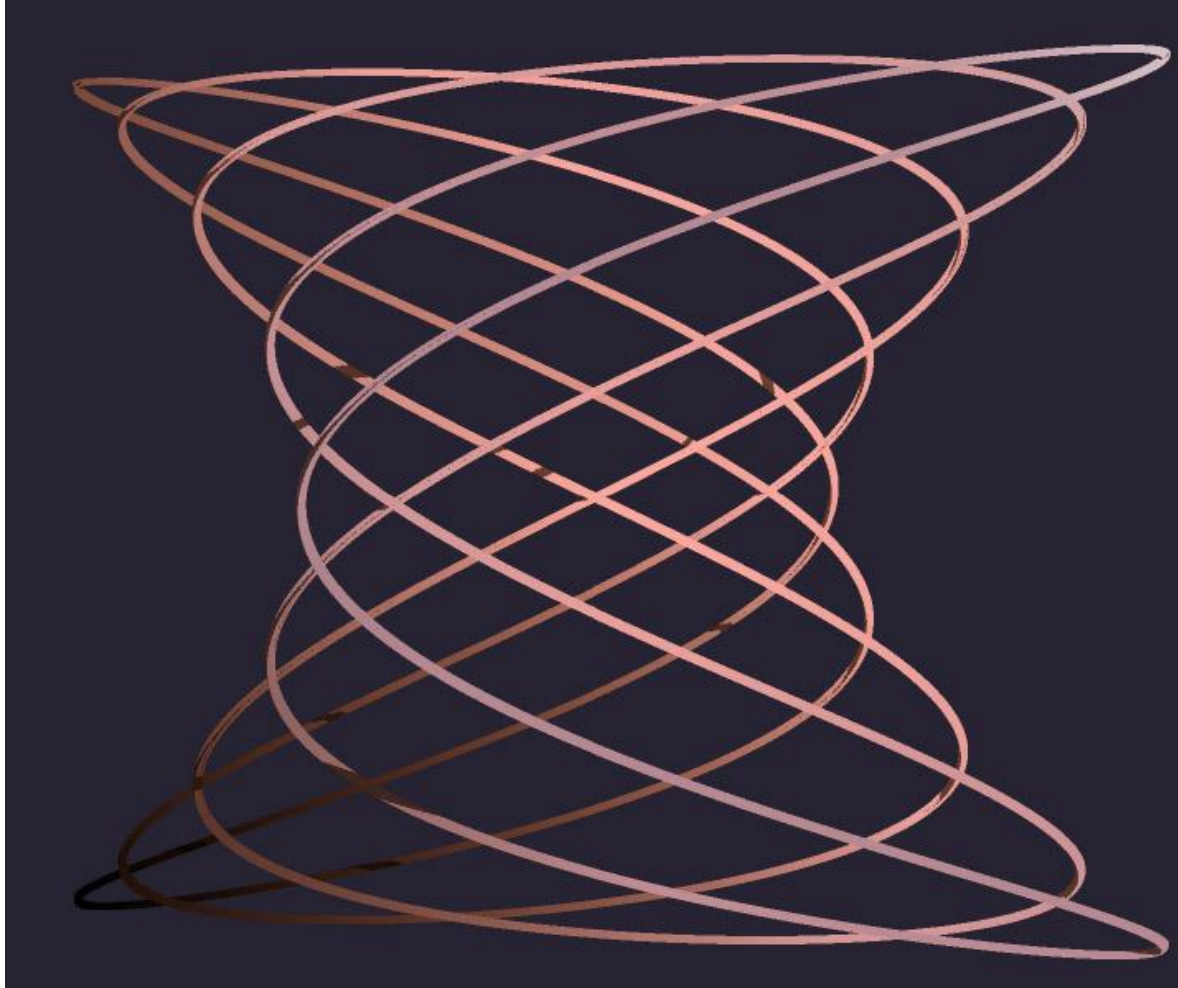
- Polynomial

$$\mathbf{x}(t) = (\mathbf{p}_1 - \mathbf{p}_0)t + \mathbf{p}_0 \rightarrow \mathbf{x}'(t) = 0t + (\mathbf{p}_1 - \mathbf{p}_0)$$

- Matrix form

$$\mathbf{x}'(t) = \begin{bmatrix} \mathbf{p}_0 & \mathbf{p}_1 \end{bmatrix} \begin{bmatrix} -1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

Lissajou curves



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

Today

Curves

- Introduction
- Polynomial curves
- Bézier curves
- Drawing Bézier curves
- Piecewise curves

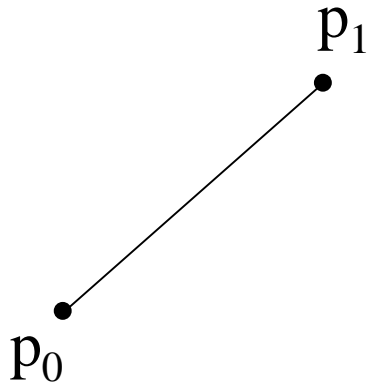
Bézier curves

http://en.wikipedia.org/wiki/B%C3%A9zier_curve

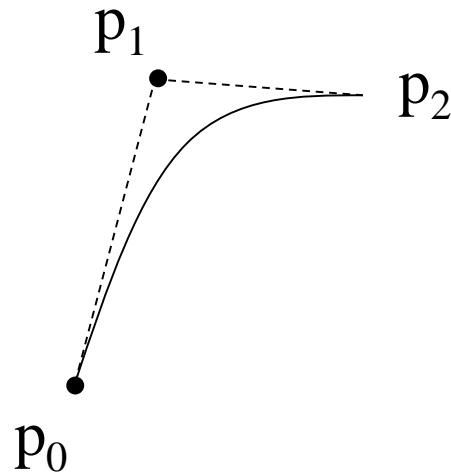
- A particularly intuitive way to define **control points** for **polynomial curves**
- Developed for CAD (computer aided design) and manufacturing
 - Before games, before movies, CAD was the big application for CG
- Pierre Bézier (1962), design of auto bodies for Peugeot, http://en.wikipedia.org/wiki/Pierre_B%C3%A9zier
- Paul de Casteljau (1959), for Citroen

Bézier curves

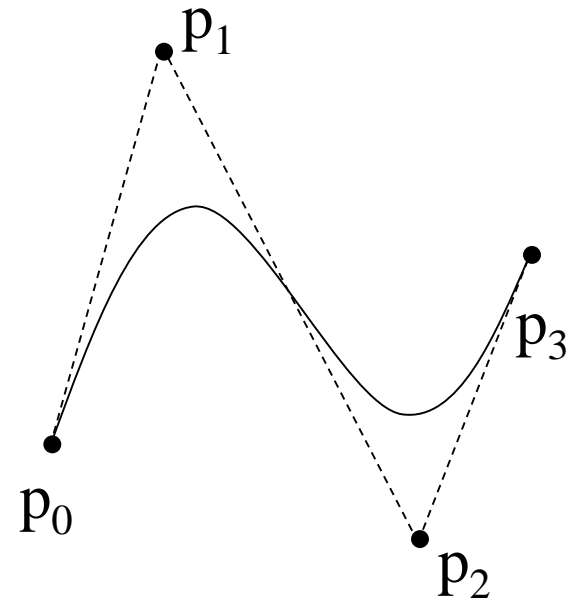
- Higher order extension of linear interpolation
- Control points p_0, p_1, \dots



Linear



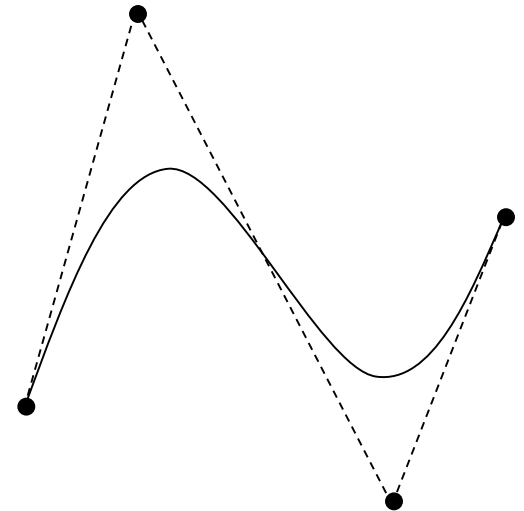
Quadratic



Cubic

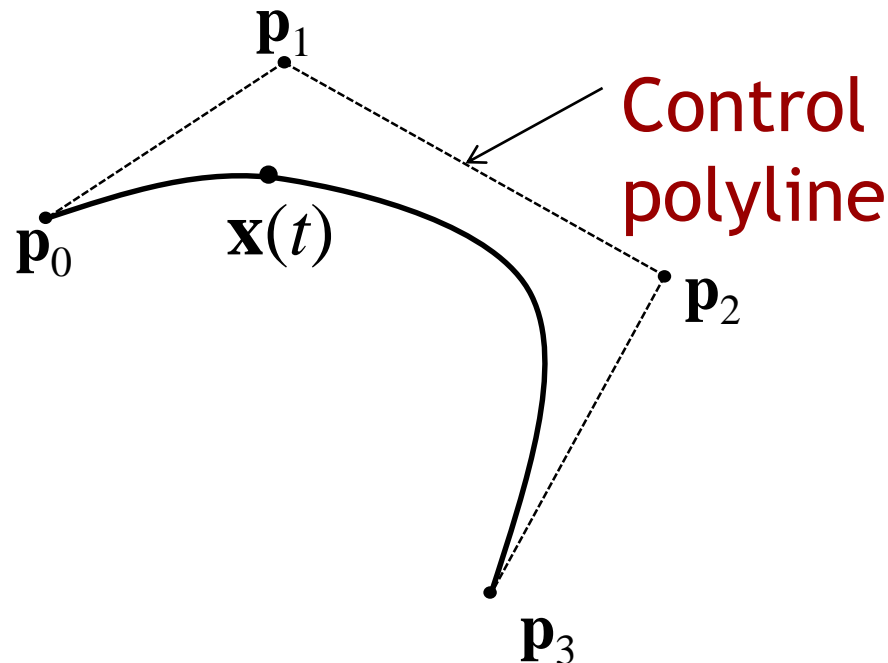
Bézier curves

- Intuitive control over curve given control points
 - Endpoints are interpolated, intermediate points are approximated
 - Convex Hull property
 - Variation-diminishing property
- Many demo applets online
 - <http://ibiblio.org/e-notes/Splines/Intro.htm>
 - <http://www.gris.uni-tuebingen.de/edu/projects/grdev/doc/html/Overview.html>
 - <http://www.theparticle.com/applets/nyu/BezierApplet/>
 - <http://www.sunsite.ubc.ca/LivingMathematics/V001N01/UBCEexamples/Bezier/bezier.html>



Cubic Bézier curve

- Cubic polynomials, most common case
- Defined by 4 control points
- Two interpolated **endpoints**
- Two **midpoints** control the tangent at the endpoints



Bézier Curve formulation

- Three alternatives, analogous to linear case
 1. Weighted average of control points
 2. Cubic polynomial function of t
 3. Matrix form
- Algorithmic construction
 - de Casteljau algorithm

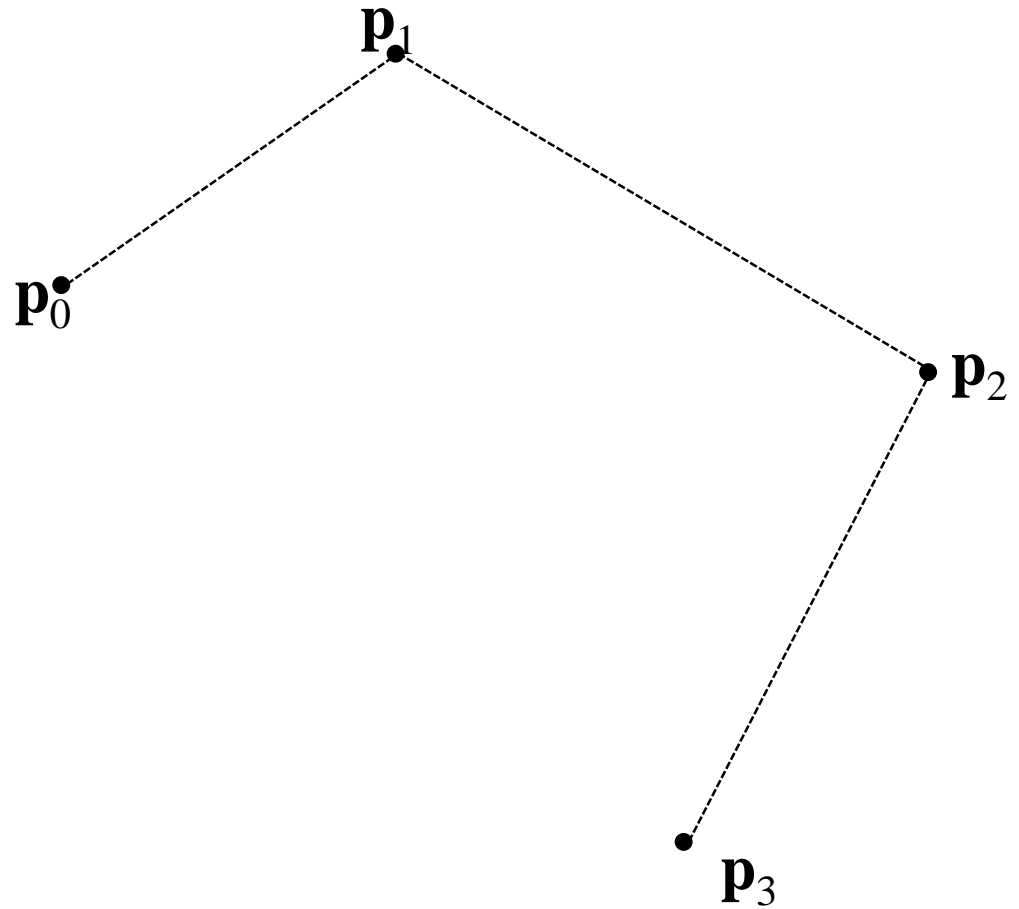
de Casteljau Algorithm

http://en.wikipedia.org/wiki/De_Casteljau's_algorithm

- A **recursive** series of **linear interpolations**
 - Works for any order, not only cubic
- Not terribly efficient to evaluate
 - Other forms more commonly used
- Why study it?
 - Intuition about the geometry
 - Useful for subdivision (later today)

de Casteljau Algorithm

- Given the control points
- A value of t
- Here $t \approx 0.25$

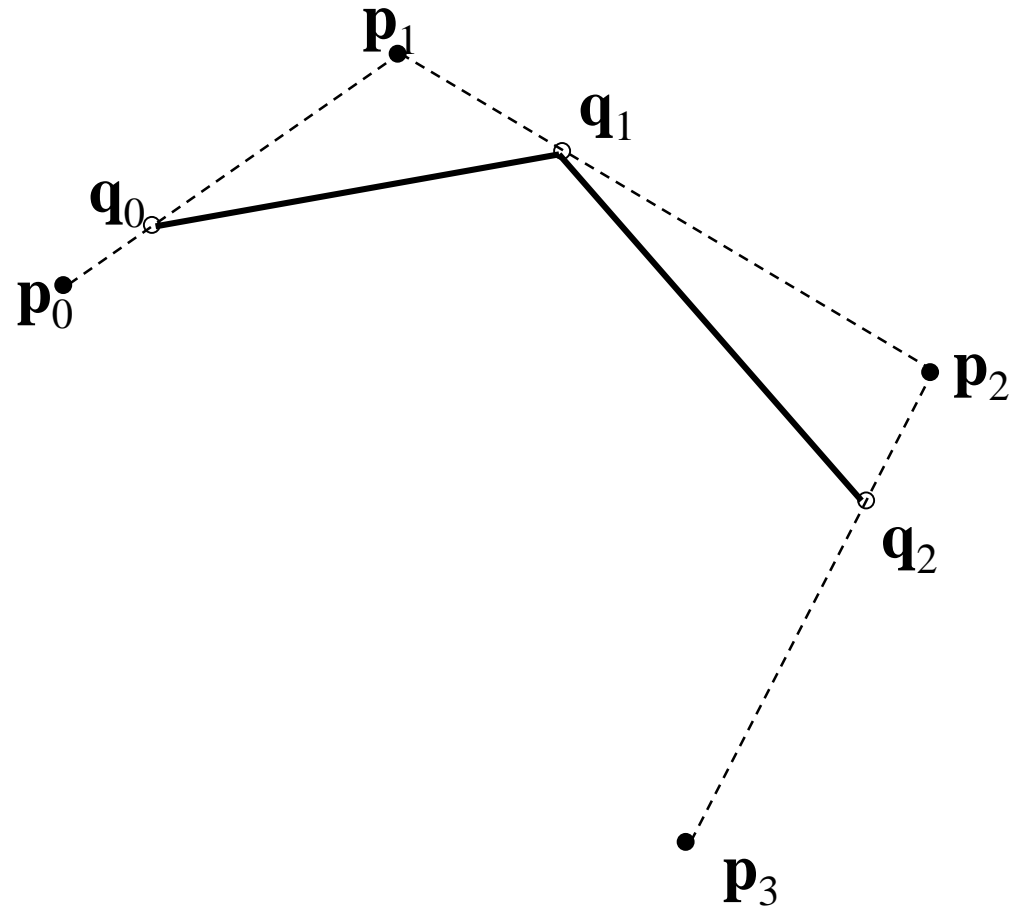


de Casteljau Algorithm

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

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

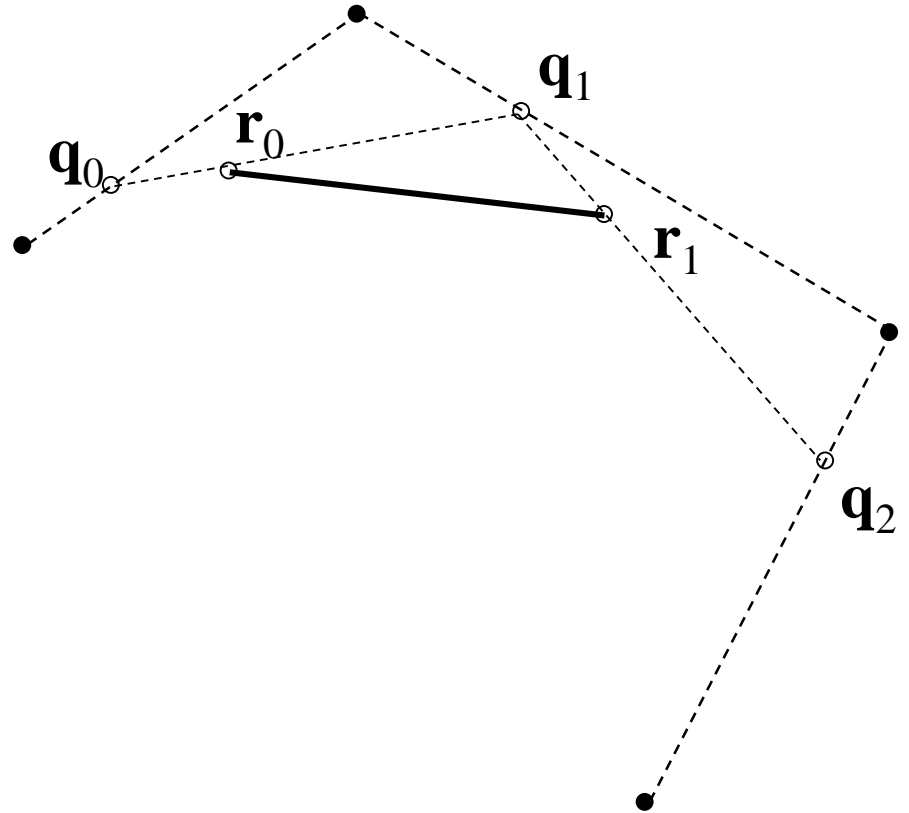
$$\mathbf{q}_2(t) = \text{Lerp}(t, \mathbf{p}_2, \mathbf{p}_3)$$



de Casteljau Algorithm

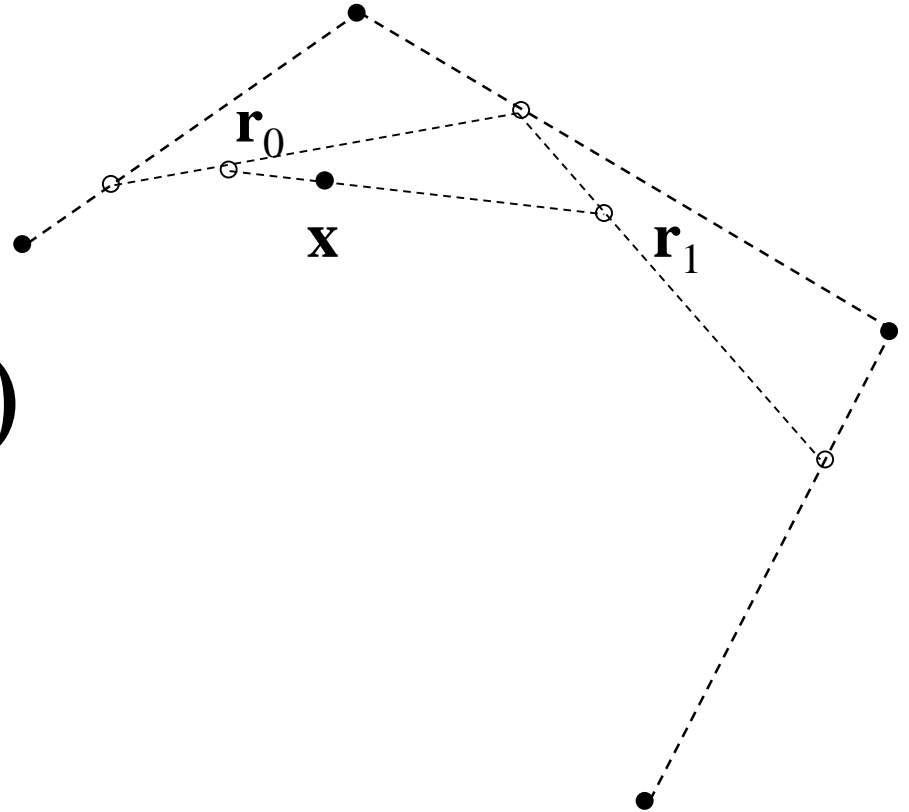
$$\mathbf{r}_0(t) = \text{Lerp}(t, \mathbf{q}_0(t), \mathbf{q}_1(t))$$

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

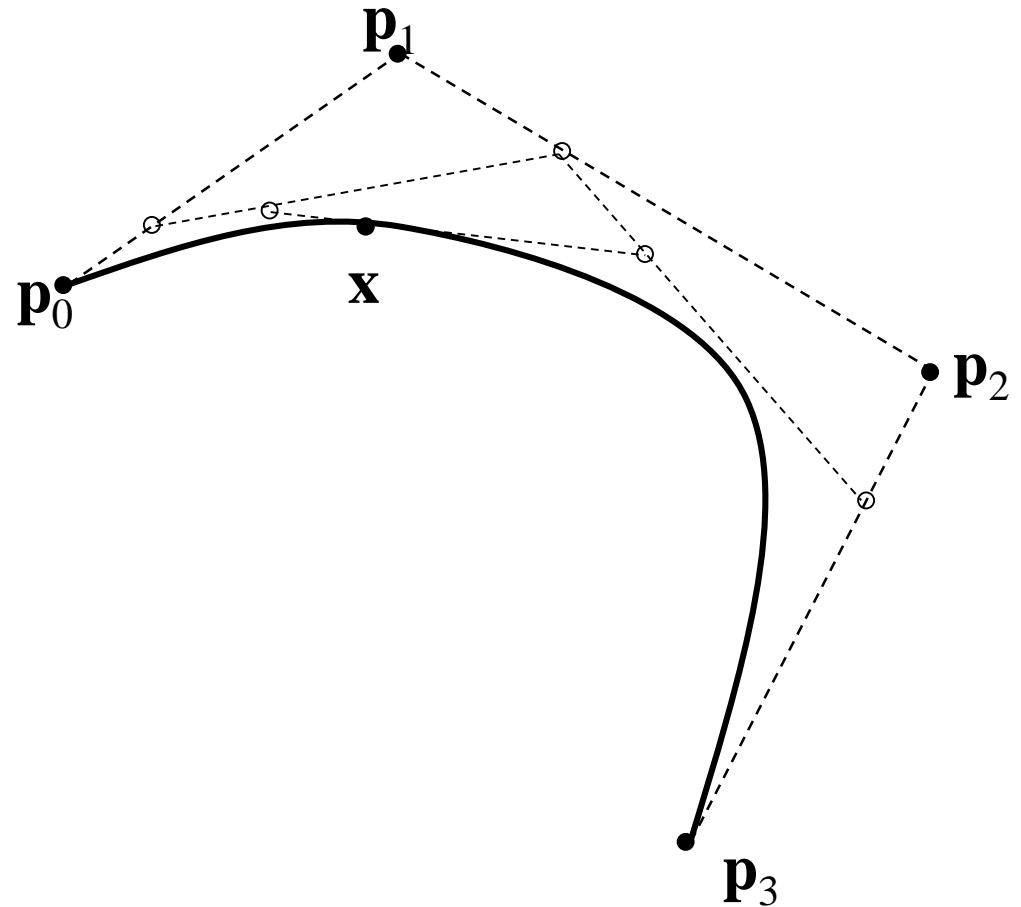


de Casteljau Algorithm

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



de Casteljau algorithm

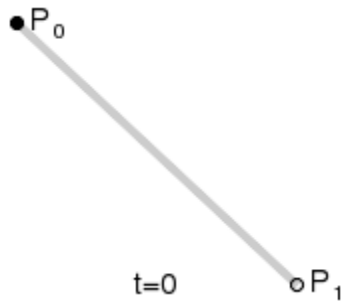


- Applets

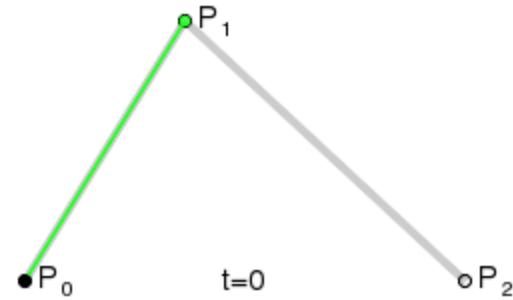
- <http://www2.mat.dtu.dk/people/J.Gravesen/cagd/decast.html>
- <http://www.caffeineowl.com/graphics/2d/vectorial/bezierintro.html>

de Casteljau Algorithm

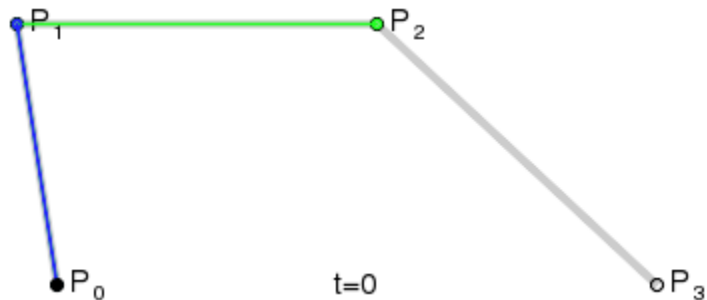
http://en.wikipedia.org/wiki/De_Casteljau's_algorithm



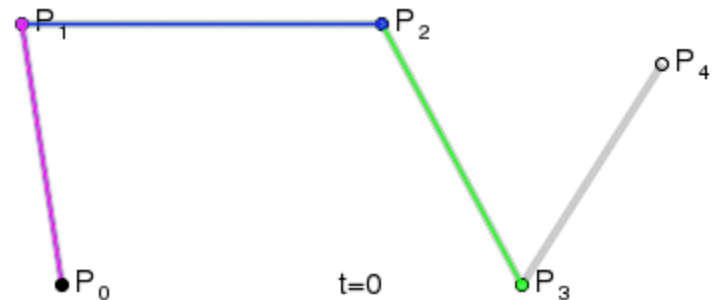
Linear



Quadratic



Cubic



Quartic

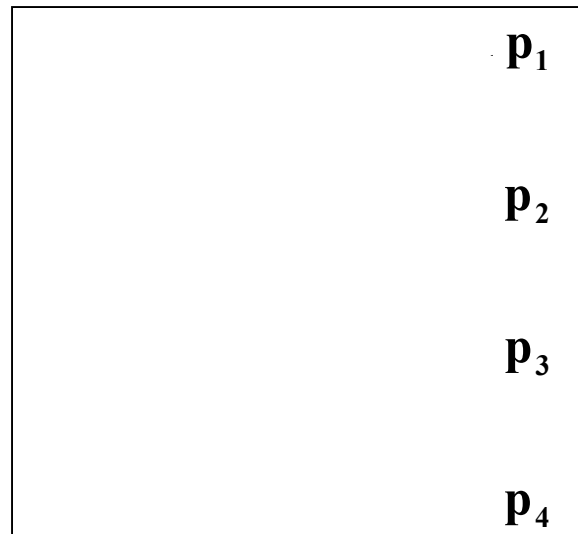
Recursive linear interpolation

\mathbf{p}_0

\mathbf{p}_1

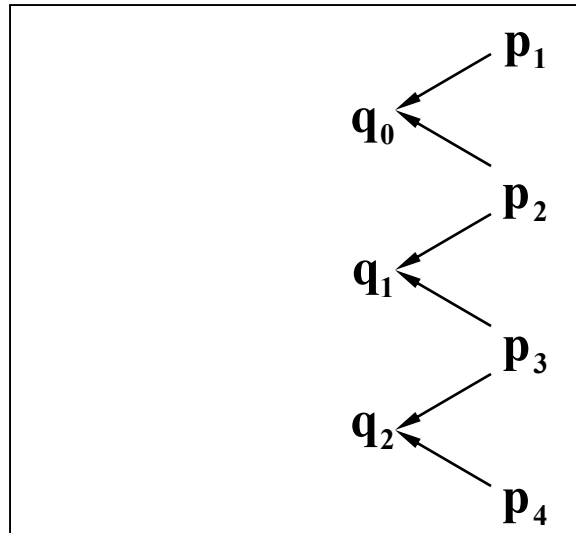
\mathbf{p}_2

\mathbf{p}_3



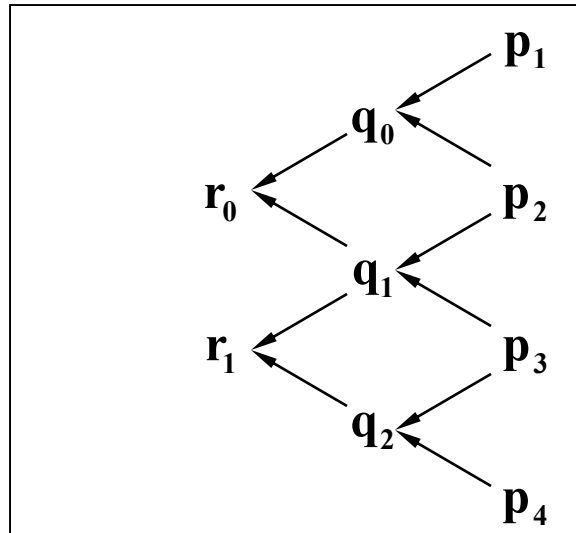
Recursive linear interpolation

$$\begin{array}{l} \mathbf{q}_0 = \text{Lerp}(t, \mathbf{p}_0, \mathbf{p}_1) \\ \mathbf{q}_1 = \text{Lerp}(t, \mathbf{p}_1, \mathbf{p}_2) \\ \mathbf{q}_2 = \text{Lerp}(t, \mathbf{p}_2, \mathbf{p}_3) \end{array} \begin{array}{l} \mathbf{p}_0 \\ \mathbf{p}_1 \\ \mathbf{p}_2 \\ \mathbf{p}_3 \end{array}$$



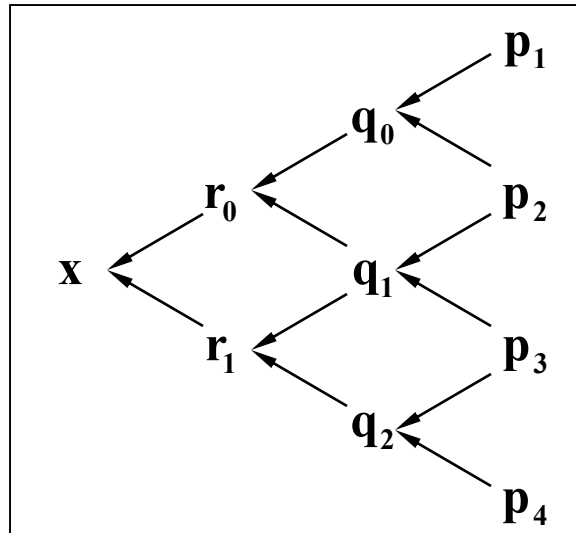
Recursive linear interpolation

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



Recursive linear interpolation

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



Expand the LERPs

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

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

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

Expand the LERPs

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

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

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

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

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

Expand the LERPs

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

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

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

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

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

Expand the LERPs

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

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

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

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

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

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

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

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

Weighted average of control points

- Regroup

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

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

Weighted average of control points

- Regroup

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

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

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

Bernstein polynomials

Cubic Bernstein polynomials

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

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

The cubic *Bernstein polynomials* :

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

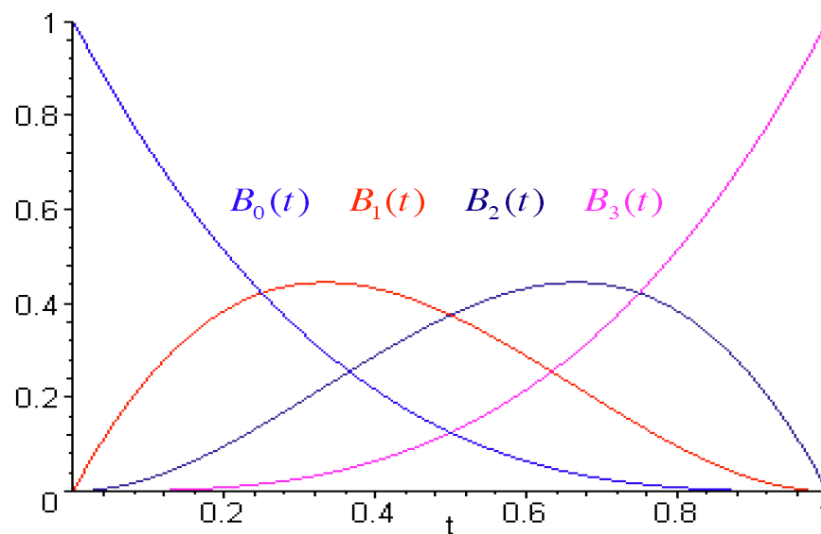
$$B_1(t) = 3t^3 - 6t^2 + 3t$$

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

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

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

Bernstein Cubic Polynomials

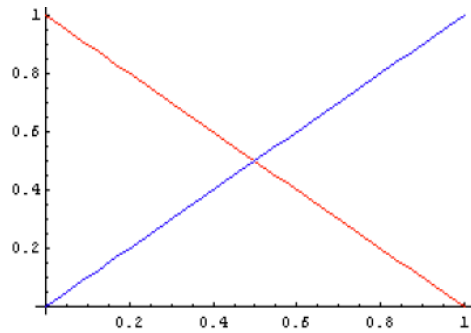


- **Partition of unity**, at each t always add to 1
- **Endpoint interpolation**, B_0 and B_3 go to 1

General Bernstein polynomials

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

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



General Bernstein polynomials

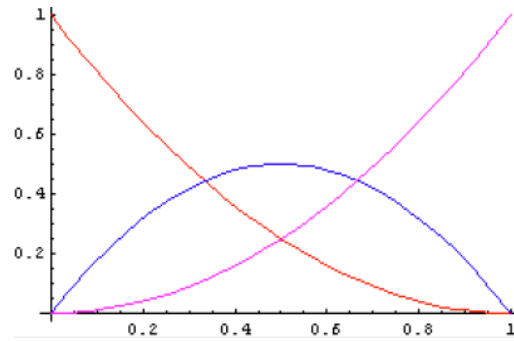
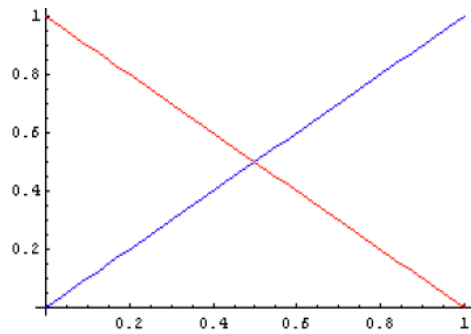
$$B_0^1(t) = -t + 1$$

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

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

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

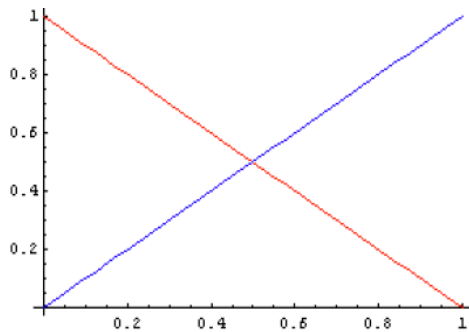
$$B_2^2(t) = t^2$$



General Bernstein polynomials

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

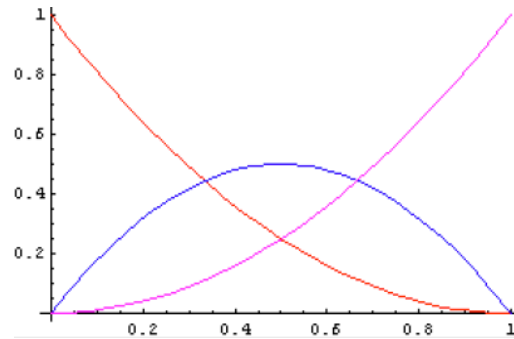
$$B_1^1(t) = t$$



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

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

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

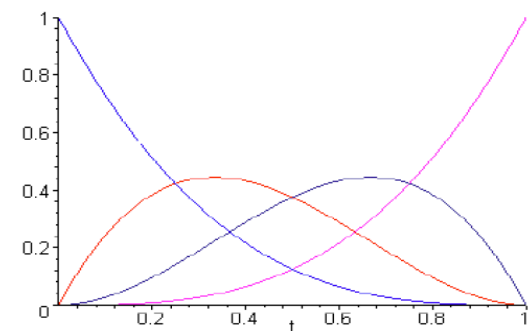


$$B_0^3(t) = -t^3 + 3t^2 - 3t + 1$$

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

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

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



General Bernstein polynomials

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

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

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

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

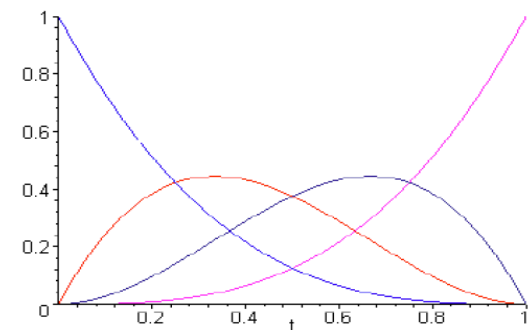
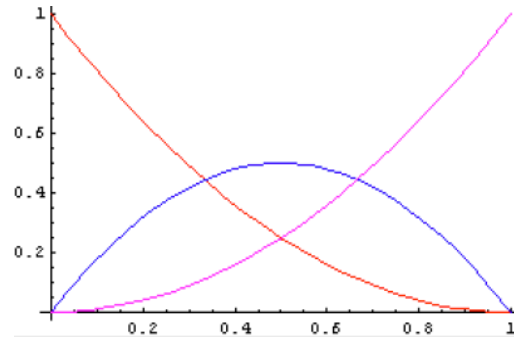
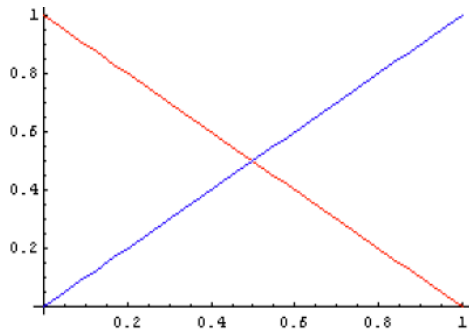
$$B_2^2(t) = t^2$$

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

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

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

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



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

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

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

Partition of unity, endpoint interpolation

General Bézier curves

- n th-order Bernstein polynomials form n th-order Bézier curves
- Bézier curves are weighted sum of control points using n th-order Bernstein polynomials

Bernstein polynomials
of order n :

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

Bézier curve of order n :

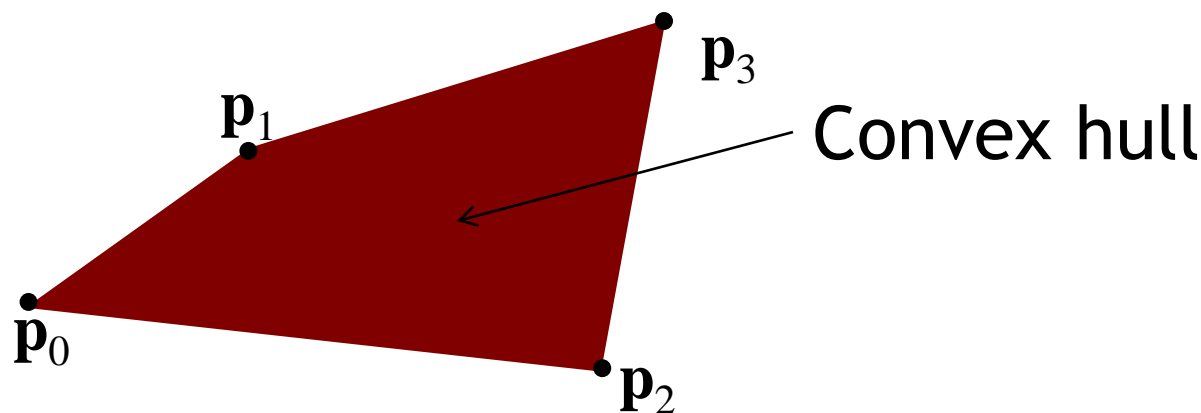
$$\mathbf{x}(t) = \sum_{i=0}^n B_i^n(t) \mathbf{p}_i$$

Bézier curve properties

- Convex hull property
- Variation diminishing property
- Affine invariance

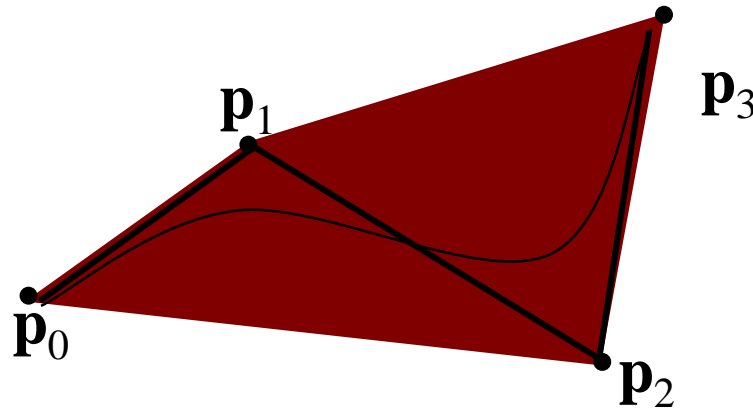
Convex hull, convex combination

- **Convex hull** of a set of points
 - Smallest polyhedral volume such that
 - (i) all points are in it
 - (ii) line connecting any two points in the volume lies completely inside it (or on its boundary)
- **Convex combination** of the points
 - Weighted average of the points, where weights all between 0 and 1, sum up to 1
- Any convex combination always lies within the convex hull



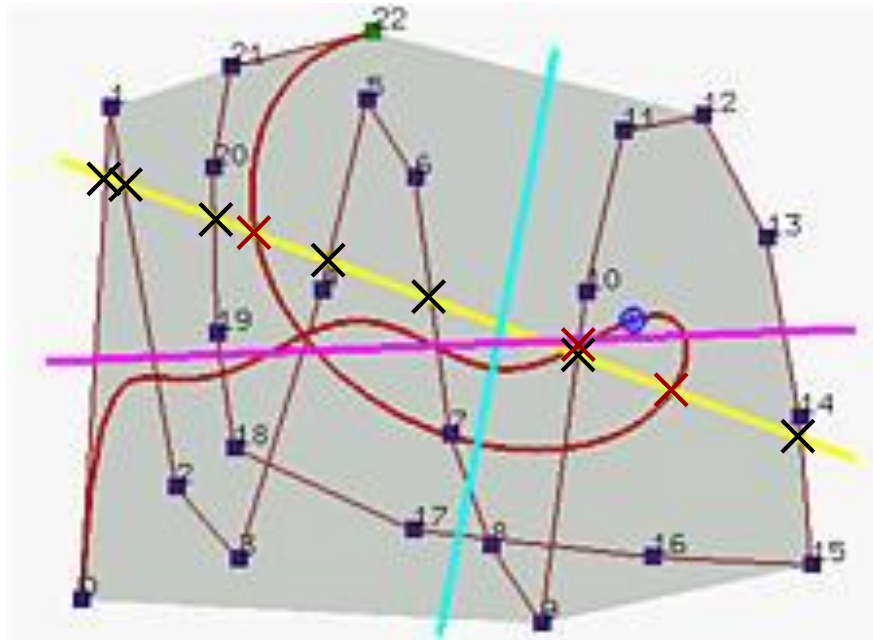
Convex hull property

- Bézier curve is a **convex combination** of the control points
 - Bernstein polynomials add to 1 at each value of t
- Curve is always **inside the convex hull** of control points
- Makes curve predictable
- Allows efficient culling, intersection testing, adaptive tessellation



Variation diminishing property

- If the curve is in a plane, this means no straight line intersects a Bézier curve more times than it intersects the curve's control polyline
- “Curve is not more wiggly than control polyline”



Yellow line: 7 intersections with control polyline
3 intersections with curve

Affine invariance

- Two ways to transform Bézier curves
 1. Transform the control points, then compute resulting point on curve
 2. Compute point on curve, then transform it
- Either way, get the same transform point!
 - Curve is defined via affine combination of points (convex combination is special case of an affine combination)
 - Invariant under affine transformations
 - Convex hull property always remains

Cubic polynomial form

Start with Bernstein form:

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

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 :

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

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 :

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

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

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 (**a,b,c,d**)
- Not much geometric intuition

Cubic matrix form

$$\mathbf{x}(t) = \begin{bmatrix} \vec{\mathbf{a}} & \vec{\mathbf{b}} & \vec{\mathbf{c}} & \mathbf{d} \end{bmatrix} \begin{bmatrix} t^3 \\ t^2 \\ t \\ 1 \end{bmatrix} \quad \begin{aligned} \vec{\mathbf{a}} &= (-\mathbf{p}_0 + 3\mathbf{p}_1 - 3\mathbf{p}_2 + \mathbf{p}_3) \\ \vec{\mathbf{b}} &= (3\mathbf{p}_0 - 6\mathbf{p}_1 + 3\mathbf{p}_2) \\ \vec{\mathbf{c}} &= (-3\mathbf{p}_0 + 3\mathbf{p}_1) \\ \mathbf{d} &= (\mathbf{p}_0) \end{aligned}$$

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

- Can construct other cubic curves by just using different basis matrix \mathbf{B}
- Hermite, Catmull-Rom, B-Spline, ...

Cubic matrix form

- 3 parallel equations, in 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
 - Precompute \mathbf{C}
 - Take advantage of existing 4x4 matrix hardware support

Today

Curves

- Introduction
- Polynomial curves
- Bézier curves
- Drawing Bézier curves
- Piecewise curves

Drawing Bézier curves

- Generally no low-level support for drawing smooth curves
 - I.e., GPU draws only **straight** line segments
- Need to break curves into line segments or individual pixels
- Approximating curves as series of line segments called **tessellation**
- Tessellation algorithms
 - Uniform sampling
 - Adaptive sampling
 - Recursive subdivision

Uniform sampling

- Approximate curve with $N-1$ straight segments

- N chosen in advance

- Evaluate $\mathbf{x}_i = \mathbf{x}(t_i)$ where $t_i = \frac{i}{N}$ for $i = 0, 1, \dots, N$

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

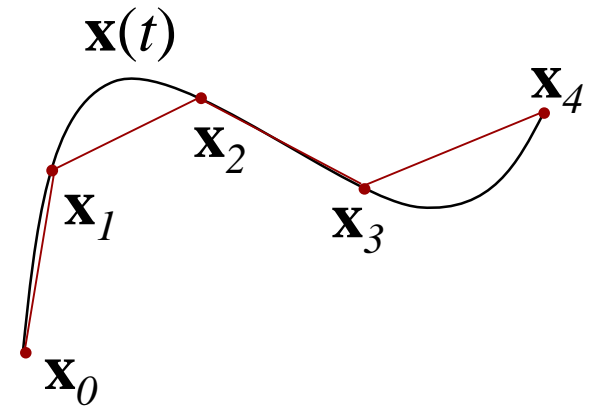
- Connect the points with lines

- Too few points?

- Bad approximation
 - “Curve” is faceted

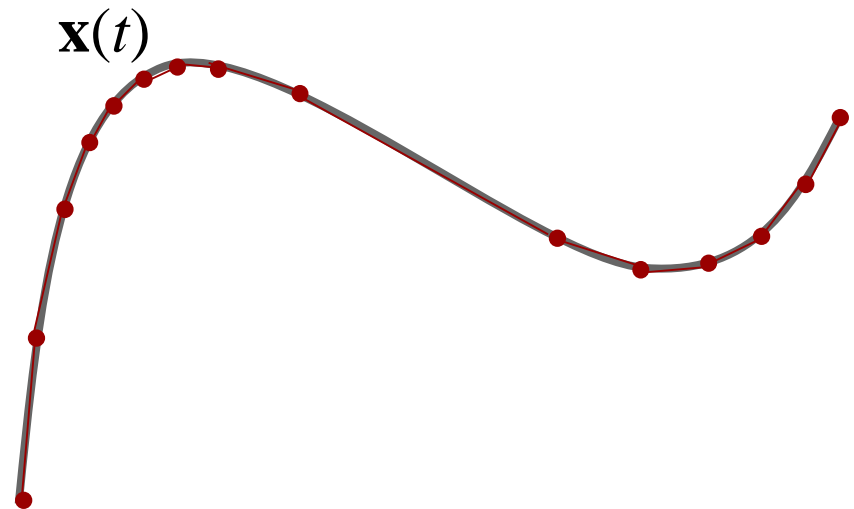
- Too many points?

- Slow to draw too many line segments
 - Segments may draw on top of each other



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
- Various schemes for sampling, checking results, deciding whether to sample more



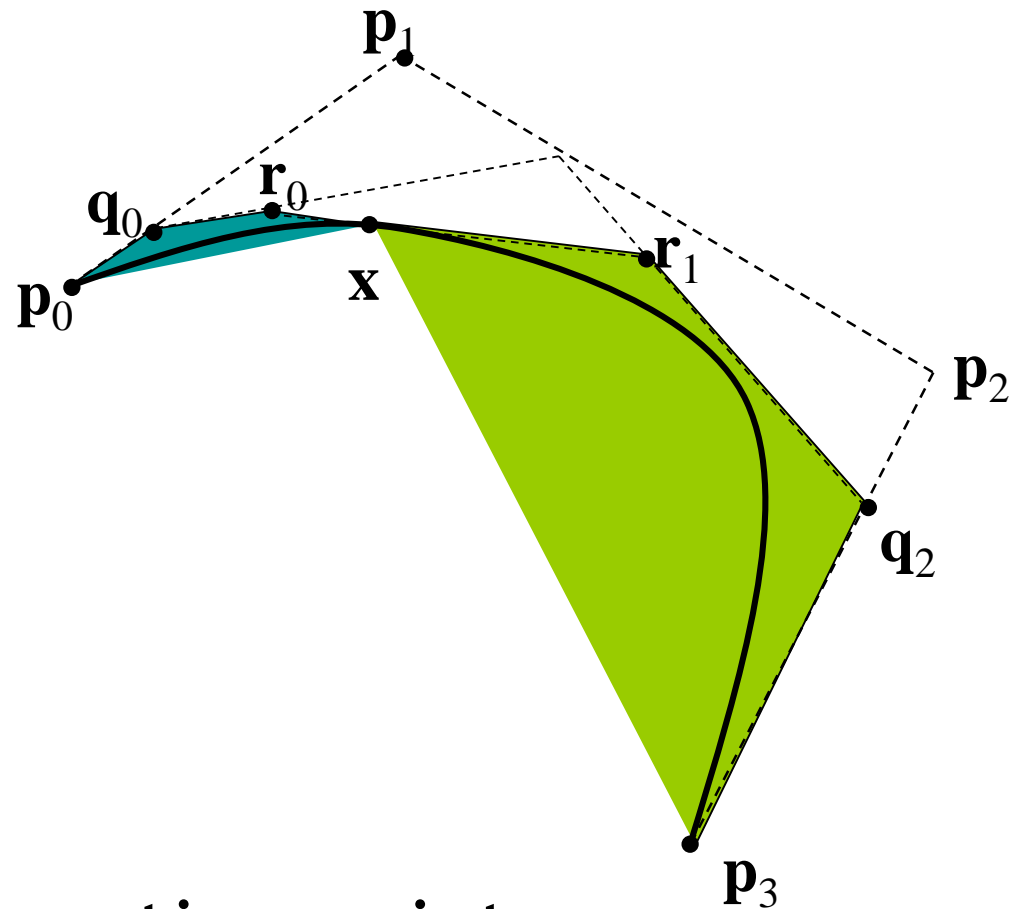
Recursive Subdivision

- Any cubic (or k -th order) **curve segment** can be expressed as a cubic (or k -th order) Bézier curve

“Any piece of a cubic (or k -th order) curve is itself a cubic (or k -th order) curve”

- Therefore, any Bézier curve can be **subdivided** into smaller Bézier curves

de Casteljau subdivision



- de Casteljau construction points are the control points of two Bézier sub-segments (p_0, q_0, r_0, x) and (x, r_1, q_2, p_3)

Adaptive subdivision algorithm

1. Use de Casteljau construction to split Bézier segment in middle ($t=0.5$)
2. For each half
 - If “flat enough”: draw line segment
 - Else: recurse from 1. for each half
- Curve is flat enough if hull is flat enough
- Test how far away midpoints are from straight segment connecting start and end
 - If about a pixel, then hull is flat enough

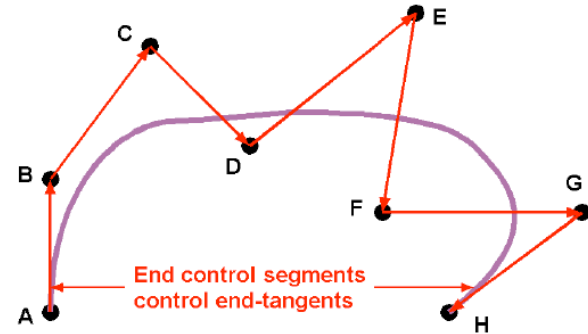
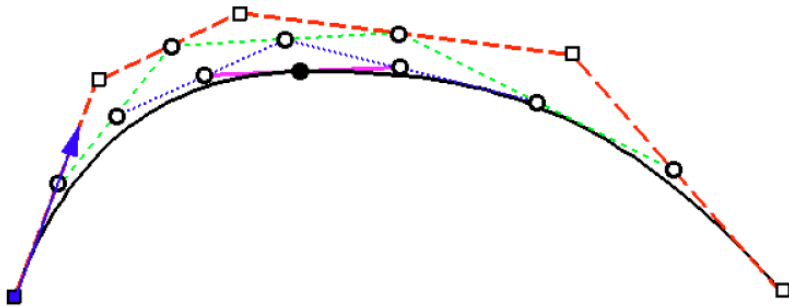
Today

Curves

- Introduction
- Polynomial curves
- Bézier curves
- Drawing Bézier curves
- Piecewise curves

More control points

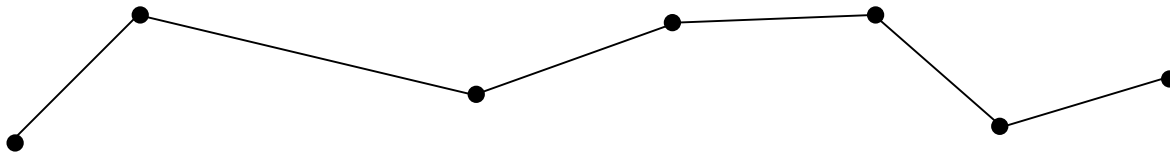
- Cubic Bézier curve limited to 4 control points
 - Cubic curve can only have one inflection
 - Need more control points for more complex curves
- $k-1$ order Bézier curve with k control points



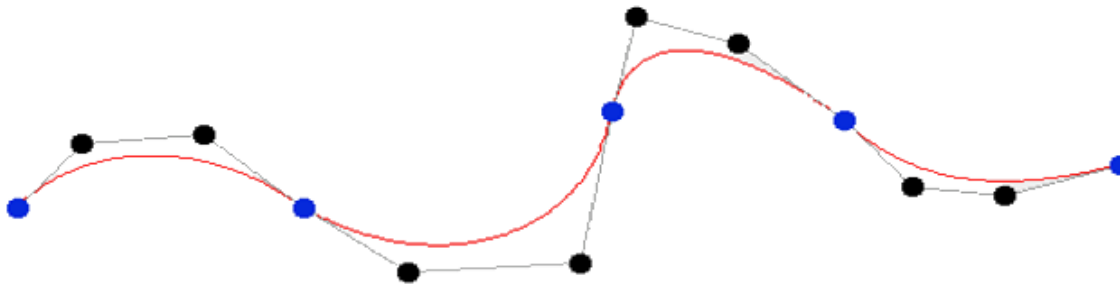
- Hard to control and hard to work with
 - Intermediate points don't have obvious effect on shape
 - Changing any control point changes the whole curve
- Want **local support**
 - Each control point only influences nearby portion of curve

Piecewise curves (splines)

- Sequence of simple (low-order) curves, end-to-end
 - Piecewise polynomial curve, or splines
[http://en.wikipedia.org/wiki/Spline_\(mathematics\)](http://en.wikipedia.org/wiki/Spline_(mathematics))
- Sequence of line segments
 - Piecewise linear curve (linear or first-order spline)



- Sequence of cubic curve segments
 - Piecewise cubic curve, here piecewise Bézier (cubic spline)



Piecewise cubic Bézier curve

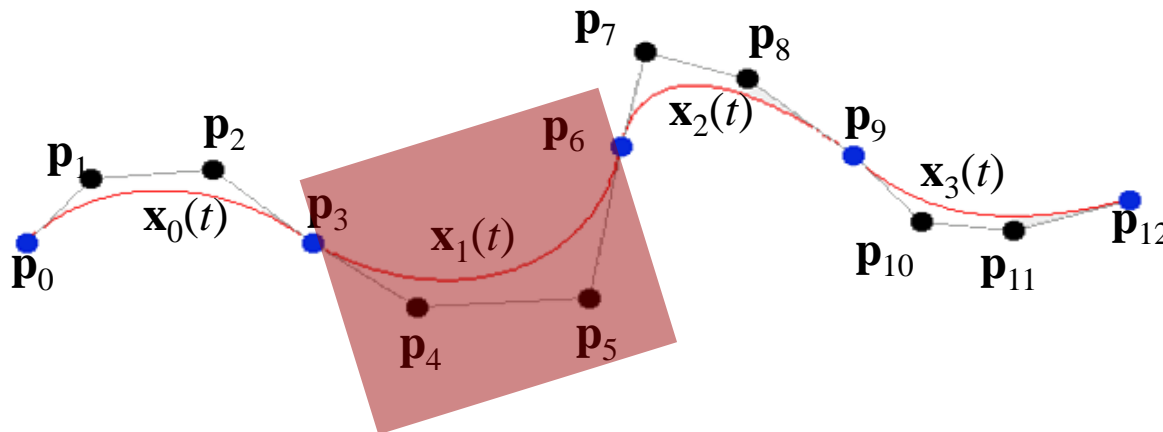
- Given $3N + 1$ points $\mathbf{p}_0, \mathbf{p}_1, \dots, \mathbf{p}_{3N}$
- Define N Bézier segments:

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

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

\vdots

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

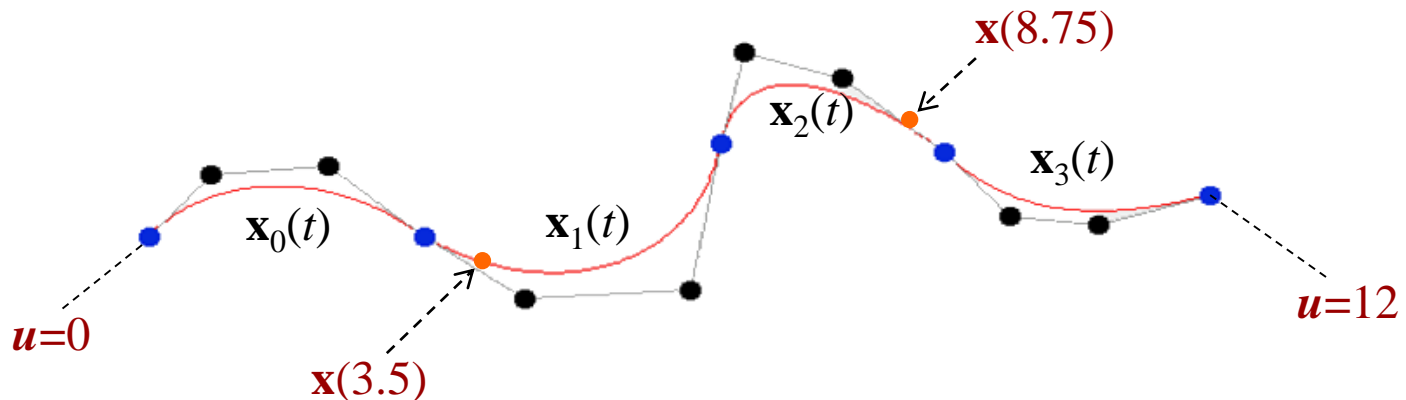


Piecewise cubic Bézier curve

- Global parameter u , $0 \leq u \leq 3N$

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

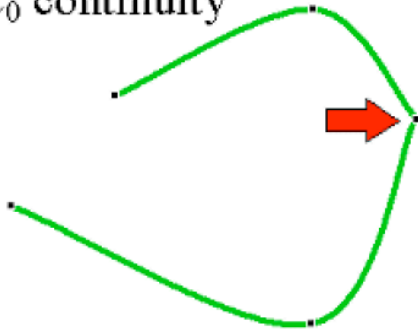
$$\mathbf{x}(u) = \mathbf{x}_i\left(\frac{1}{3}u - i\right), \text{ where } i = \left\lfloor \frac{1}{3}u \right\rfloor$$



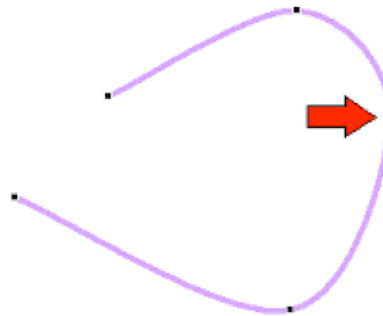
Continuity

- Want smooth curves
- C^0 continuity
 - No gaps
 - Segments match at the endpoints
- C^1 continuity: first derivative is well defined
 - No corners
 - Tangents/normals are C^0 continuous (no jumps)
- C^2 continuity: second derivative is well defined
 - Tangents/normals are C^1 continuous
 - Important for high quality reflections on surfaces

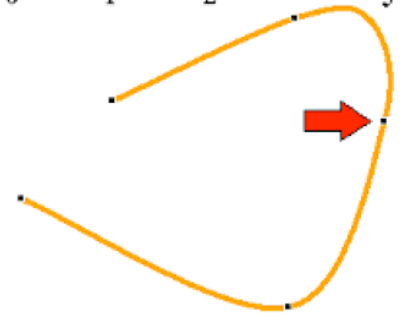
C_0 continuity



C_0 & C_1 continuity

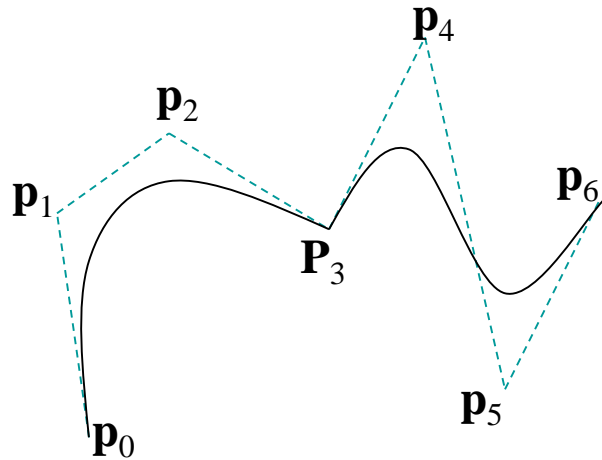
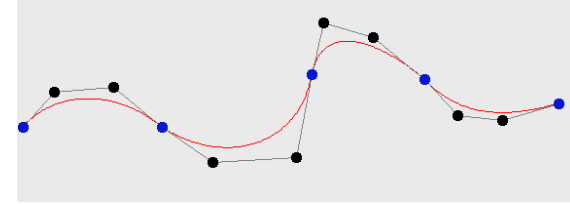


C_0 & C_1 & C_2 continuity

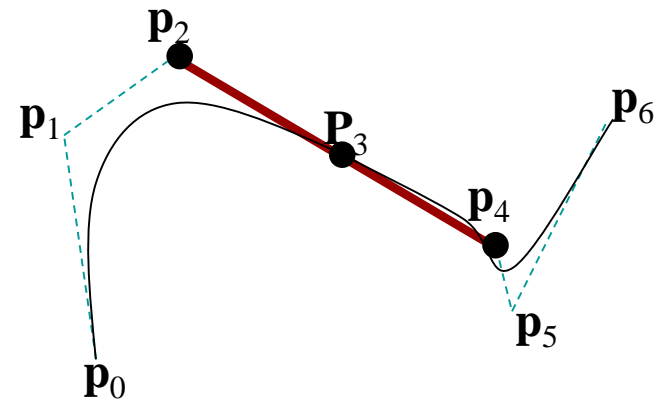


Piecewise cubic Bézier curve

- C^0 continuous by construction
- C^1 continuous at segment endpoints \mathbf{p}_{3i} **if** $\mathbf{p}_{3i} - \mathbf{p}_{3i-1} = \mathbf{p}_{3i+1} - \mathbf{p}_{3i}$
- C^2 is harder to get



C^0 continuous



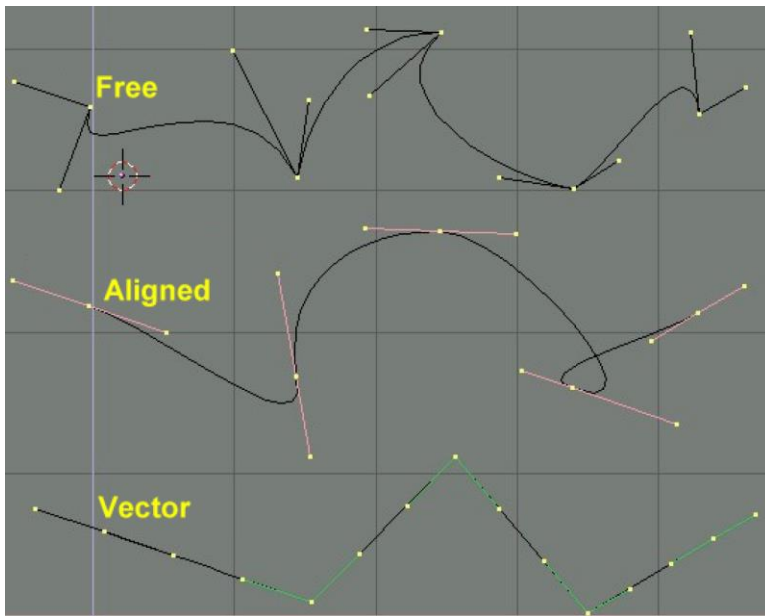
C^1 continuous

Piecewise cubic Bézier curves

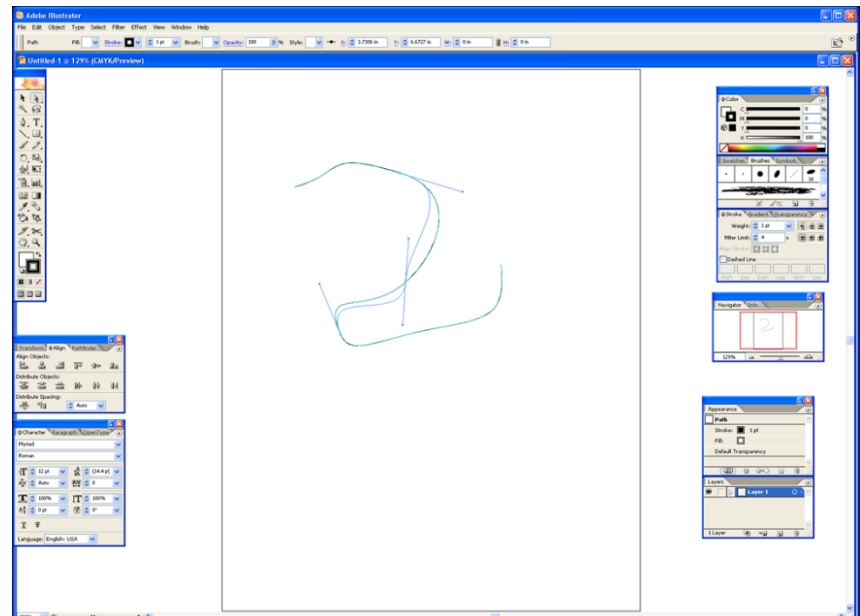
- Used often in 2D drawing programs
- Inconveniences
 - Must have 4 or 7 or 10 or 13 or ... (1 plus a multiple of 3) control points
 - Some points interpolate (endpoints), others approximate (handles)
 - Need to impose constraints on control points to obtain C^1 continuity
 - C^2 continuity more difficult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

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



[www.blender.org]



Adobe Illustrator

Next time

- B-splines and NURBS
- Extending curves to surfaces