Computer Graphics

11 – Curves

Yoonsang Lee
Spring 2022

Final Exam Announcement

- Date & time: **Jun 15**, 09:30 - 10:30 am
- Place: IT.BT, 508
- Scope: Lecture 8 ~ 13

- **You cannot leave until 30 minutes after the start of the exam** even if you finish the exam earlier.

- That means, **you cannot enter the room after 30 minutes from the start of the exam** (do not be late, never too late!).

- Please bring your student ID card to the exam.

Topics Covered

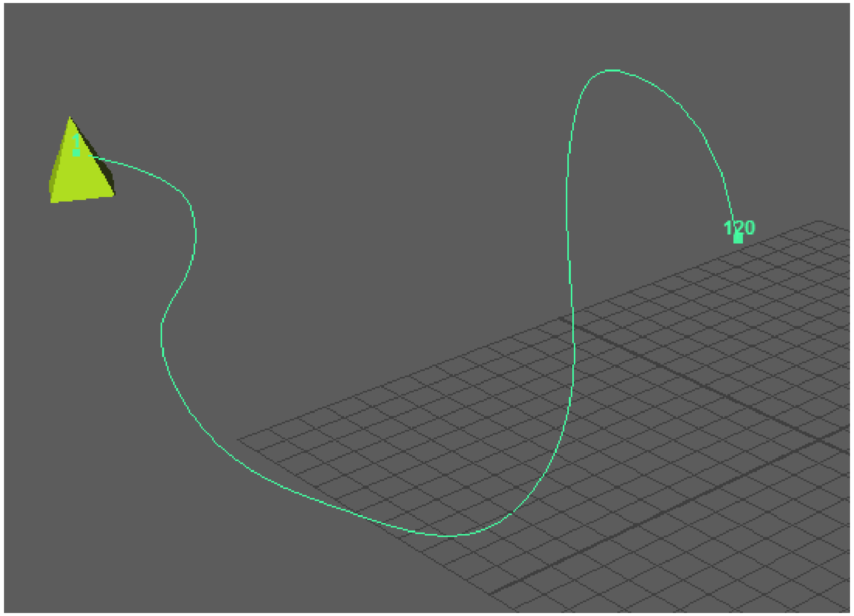
- Intro: Motivation and Curve Representation
- Polynomial Curve
  - Polynomial Interpolation
  - More Discussion on Polynomials
- Hermite Curve
- Bezier Curve
- (Very short) Intro to Spline

Intro: Motivation and Curve Representation

Motivation: Why Do We Need Curve?

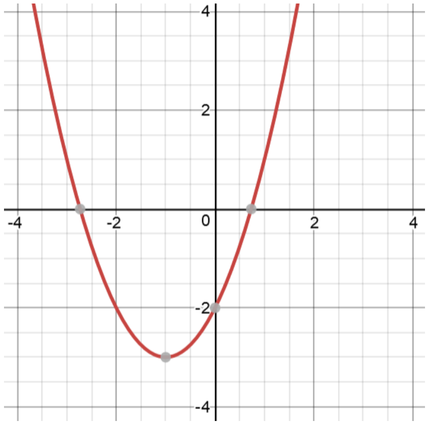
- **Smoothness**
  - no discontinuity

- In many CG applications, we need **smooth shape** and **smooth movement.**

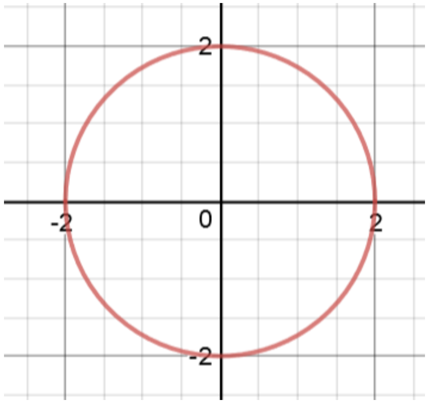


Curve Representations

- Non-parametric
  - **Explicit : y = f(x)**
    - ex) $y = x^2 + 2x - 2$
    - Pros) Easy to generate points
    - Cons) Cannot express vertical lines!

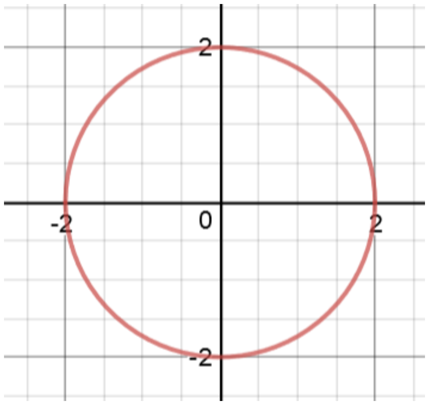


  - **Implicit : f(x, y) = 0**
    - ex) $x^2 + y^2 – 2^2 = 0$
    - Pros) Easy to test if a point is inside or outside
    - Cons) Inconvenient to generate points



Curve Representations

- **Parametric : (x, y) = (f(t), g(t))**
  - ex) $(x, y) = (2\cos(t), 2\sin(t))$
  - Each point on a curve is expressed as a function of additional parameter **t**
  - Pros) Easy to generate points
  - The parameter **t** acts as a "local coordinate" for points on the curve



- For computer graphics, the parametric representation is the most suitable.

Polynomial Curve

Polynomial Curve

- *Polynomials* are usually used to describe curves in computer graphics.
  - Simple
  - Efficient
  - Easy to manipulate

- A polynomial of *degree n*:

$$x(t) = a_n t^n + a_{n-1} t^{n-1} + \cdots + a_1 t + a_0$$

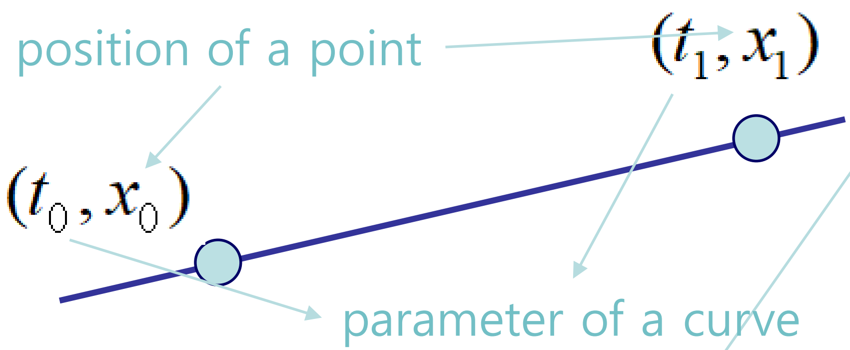
Polynomial Interpolation

- One way to make a smooth curve using polynomials is polynomial interpolation.

- Polynomial interpolation determines a specific smooth polynomial curve **passing though given data points.**

Polynomial Interpolation

- Linear interpolation with a polynomial of degree one
  - Input: two nodes
  - Output: Linear polynomial



$$x(t) = a_1 t + a_0$$

How to find a_0 and a_1?

→
$$a_1 t_0 + a_0 = x_0$$
$$a_1 t_1 + a_0 = x_1$$

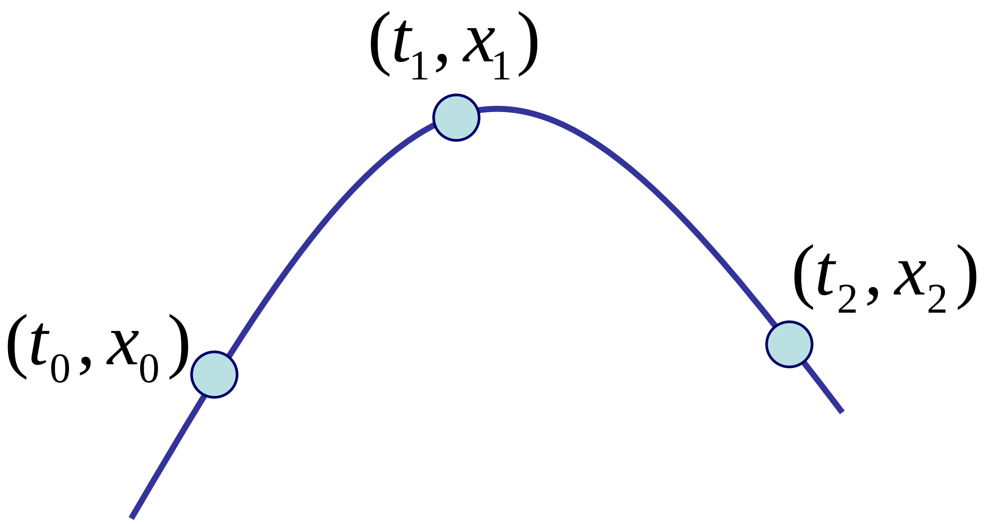
$$\begin{pmatrix} 1 & t_0 \\ 1 & t_1 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \end{pmatrix} = \begin{pmatrix} x_0 \\ x_1 \end{pmatrix}$$

We can compute the value of a_0 & a_1 because we have **2 equations (=2 data points)** for **2 unknowns**!

If $t_0=0$ and $t_1=1$, then $a_0=x_0$ and $a_1=x_1-x_0$
→ $x(t) = (x_1-x_0)t + x_0 = (1-t)x_0 + tx_1$

Polynomial Interpolation

- Quadratic interpolation with a polynomial of degree two



$$x(t) = a_2 t^2 + a_1 t + a_0$$

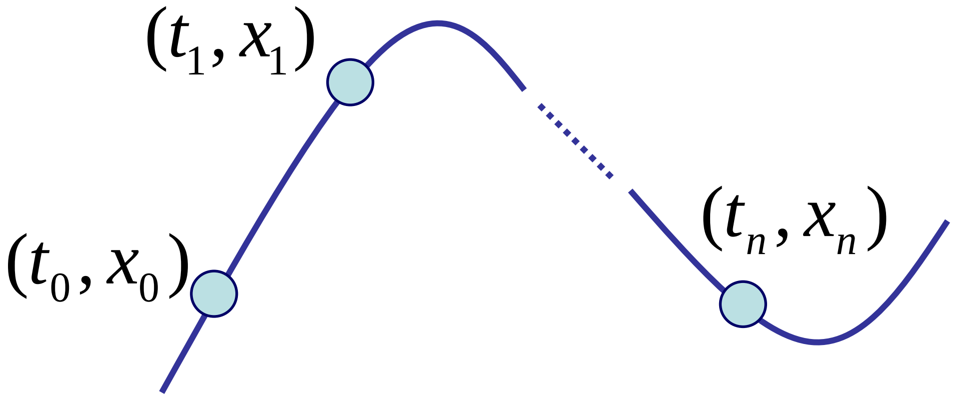
(we need **3 points** to get the value of **3 unknowns**)

$$a_2 t_0^2 + a_1 t_0 + a_0 = x_0$$
$$a_2 t_1^2 + a_1 t_1 + a_0 = x_1$$
$$a_2 t_2^2 + a_1 t_2 + a_0 = x_2$$

$$\begin{pmatrix} 1 & t_0 & t_0^2 \\ 1 & t_1 & t_1^2 \\ 1 & t_2 & t_2^2 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \end{pmatrix} = \begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix}$$

Polynomial Interpolation

- Polynomial interpolation of degree n



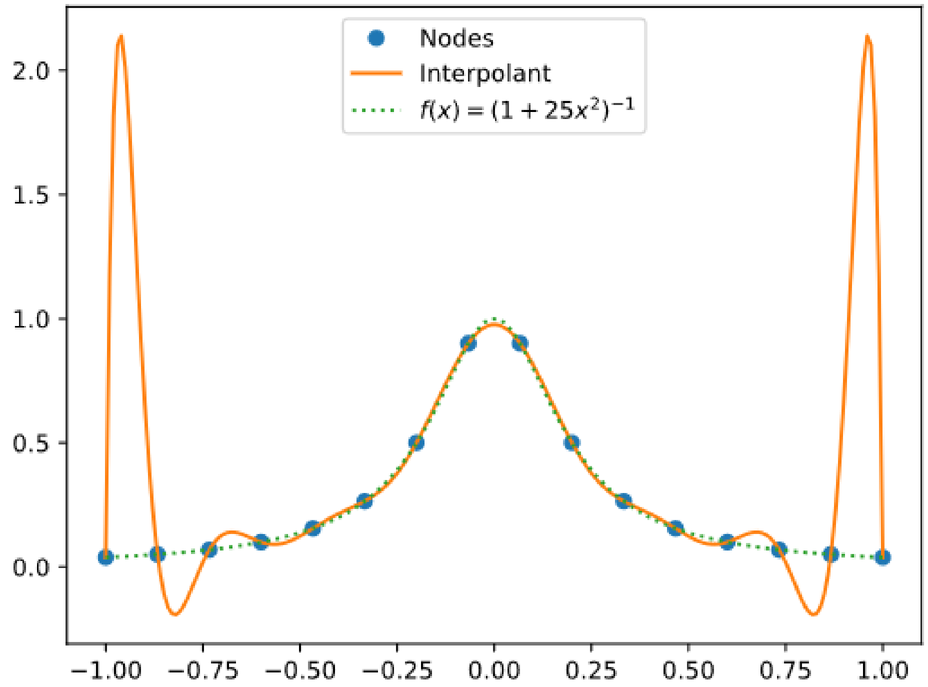
$$x(t) = a_n t^n + \cdots + a_1 t + a_0$$

$$\begin{pmatrix} 1 & \cdots & t_0^{n-1} & t_0^n \\ 1 & \cdots & t_1^{n-1} & t_1^n \\ \vdots & \ddots & \vdots & \vdots \\ 1 & \cdots & t_n^{n-1} & t_n^n \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{pmatrix}$$

- How to find the value of unknowns a_n, ..., a_0?
- Several methods:
  - Solving linear system, Lagrange's, Newton's method, ...

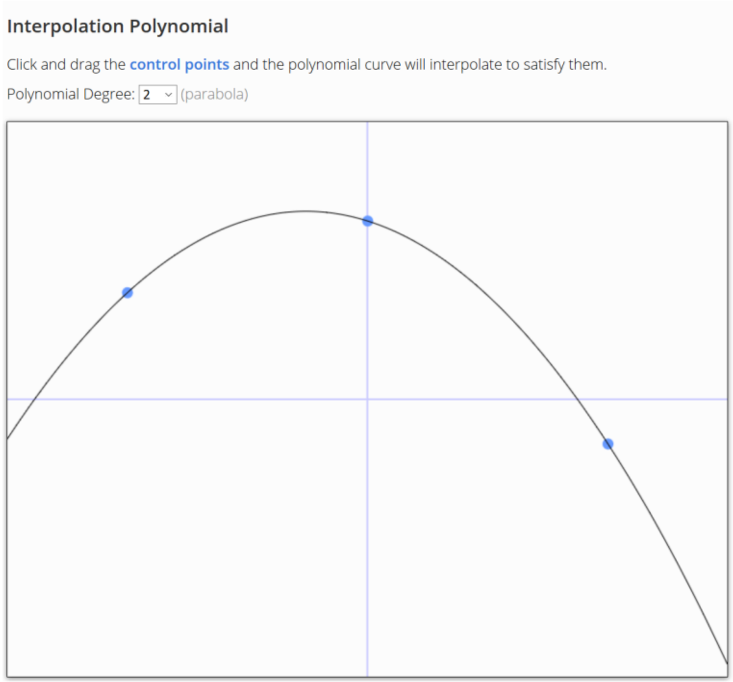
Problem of Higher-Degree Polynomial Interpolation

- Oscillations at the ends – Runge's Phenomenon



- Using higher-degree polynomial interpolation for curves is a bad idea.

[Practice] Polynomial Interpolation



https://www.benjoffe.com/code/demos/interpolate

- Drag points and observe changes of the curve.
- Increase polynomial degree and drag points.

Cubic Polynomials

- Cubic (degree of 3) polynomials are commonly used in computer graphics because...

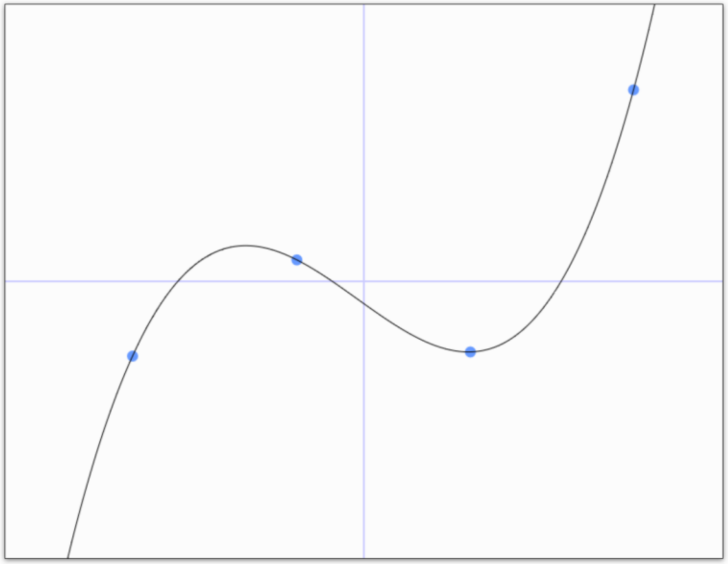
- The lowest-degree polynomials representing a 3D curve.

- Can avoid unwanted wiggles of higher-degree polynomials (Runge's Phenomenon)

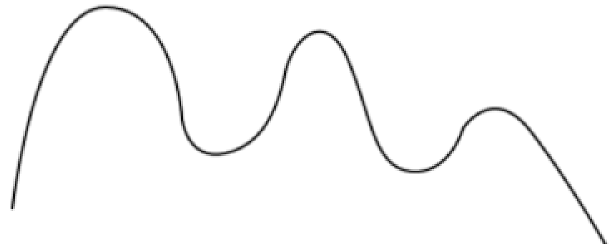
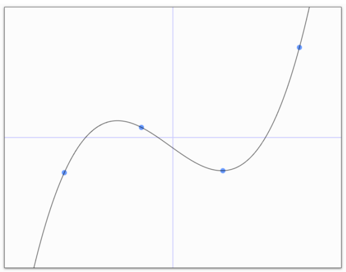
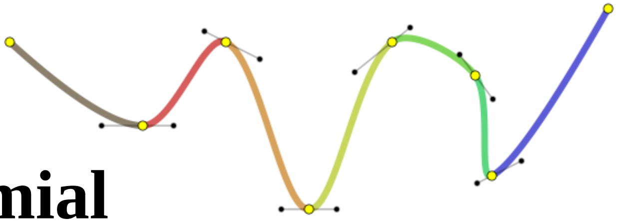
$$x(t) = a_x t^3 + b_x t^2 + c_x t + d_x$$
$$y(t) = a_y t^3 + b_y t^2 + c_y t + d_y$$
$$z(t) = a_z t^3 + b_z t^2 + c_z t + d_z$$

or

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



Then, how can make complex curves using such a low degree polynomial?

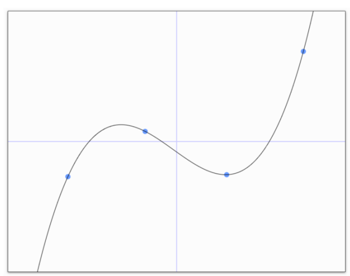
- How to make  ?
- using 
- Answer → **Spline**: ***piecewise* polynomial** 
- At this moment, let's just think about a single piece of polynomial.

Defining a Single Piece of Cubic Polynomial

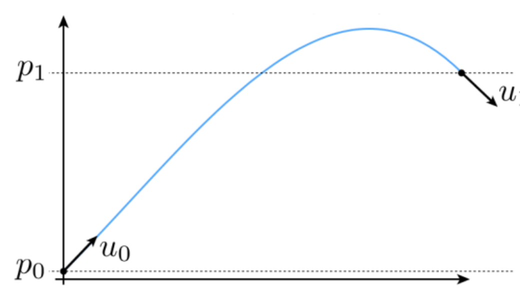
$$\mathbf{p}(t) = \mathbf{a}t^3 + \mathbf{b}t^2 + \mathbf{c}t + \mathbf{d}$$

- Goal: Defining a specific curve (finding **a**, **b**, **c**, **d**) as we want (using data points or *conditions* given by you)

- **4 unknowns**, so we need **4 equations (conditions or constraints)**. For example,
  - 4 data points



  - position and derivative of two end points



  - ...

Formulation of a Single Piece of Polynomial

- A polynomial can be formulated in two ways:

- With **coefficients** and **variable**:

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

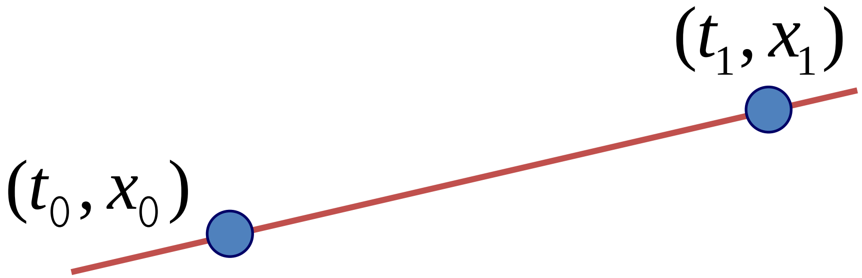
  - coefficients: **a**, **b**, **c**, **d**
  - variable: t

- With ***basis functions*** and **points**:

$$\mathbf{p}(t) = b_0(t)\mathbf{p}_0 + b_1(t)\mathbf{p}_1 + b_2(t)\mathbf{p}_2 + b_3(t)\mathbf{p}_3$$

  - *basis functions:* $b_0(t)$, $b_1(t)$, $b_2(t)$, $b_3(t)$
  - points: $\mathbf{p}_0$, $\mathbf{p}_1$, $\mathbf{p}_2$, $\mathbf{p}_3$

Trivial Example: Linear Polynomial



$$x(t) = a_1 t + a_0$$

Trivial Example: Linear Polynomial

- Formulation with coefficients and variable:

$$x(t) = (x_1 - x_0)t + x_0$$

$$y(t) = (y_1 - y_0)t + y_0$$

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

  - Matrix formulation

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

$$p(t) = \begin{bmatrix} x(t) & y(t) \end{bmatrix}$$

basis matrix

$$\begin{bmatrix} p_0 \\ p_1 \end{bmatrix} = \begin{bmatrix} x_0 & y_0 \\ x_1 & y_1 \end{bmatrix}$$

Trivial Example: Linear Polynomial

- Formulation with basis functions and points:
  - regroup expression by **p** rather than *t*

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

basis functions

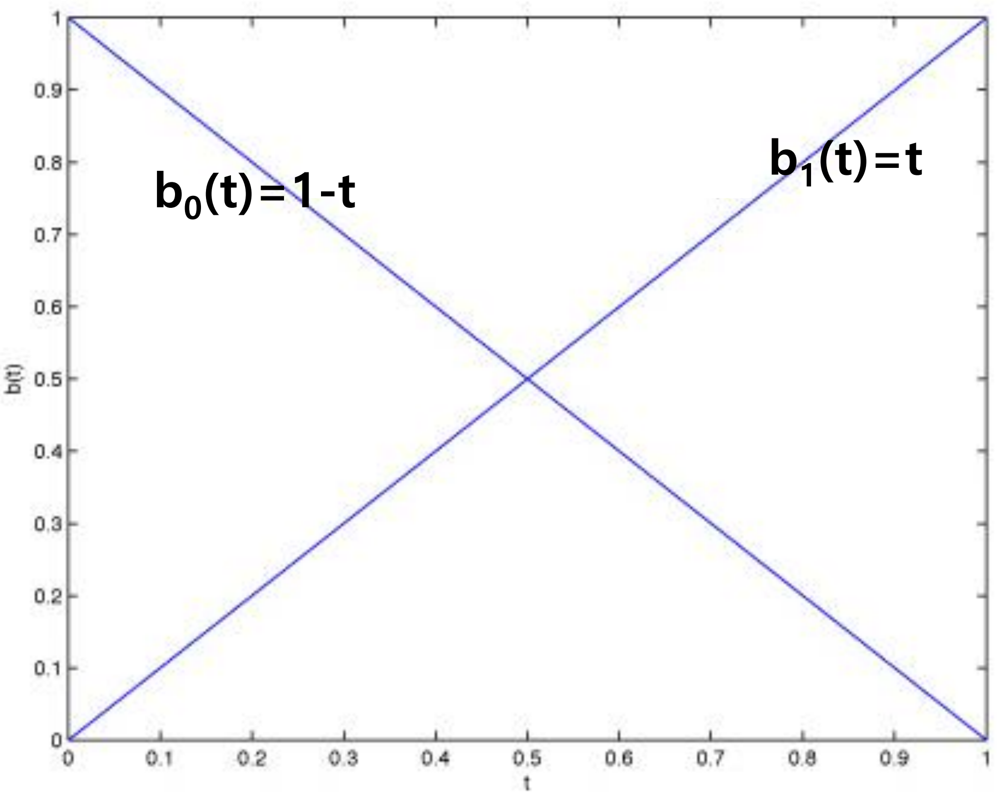
  - interpretation in matrix viewpoint

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

Meaning of Basis Functions

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

- Contribution of each point as *t* changes



Quiz #1

- Go to https://www.slido.com/
- Join #**cg-ys**
- Click “Polls”

- Submit your answer in the following format:
  - **Student ID: Your answer**
  - **e.g. 2017123456: 4)**

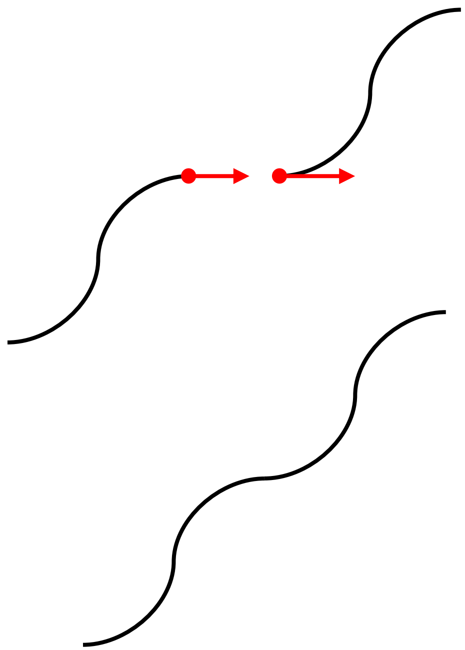
- Note that you must submit all quiz answers in the above format to be checked for “attendance”.

Hermite Curve

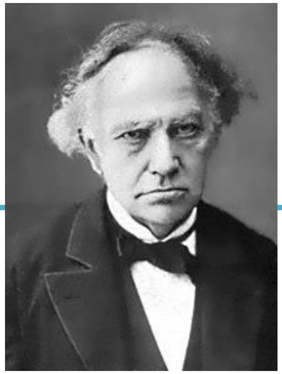
Hermite Curve

- A Hermite curve is a cubic polynomial defined in Hermite form.

- In splines, we want curve pieces that connect smoothly.
- In Hermite spline, you can do this by specifying
  - position of the endpoints
  - 1st derivatives at the endpoints

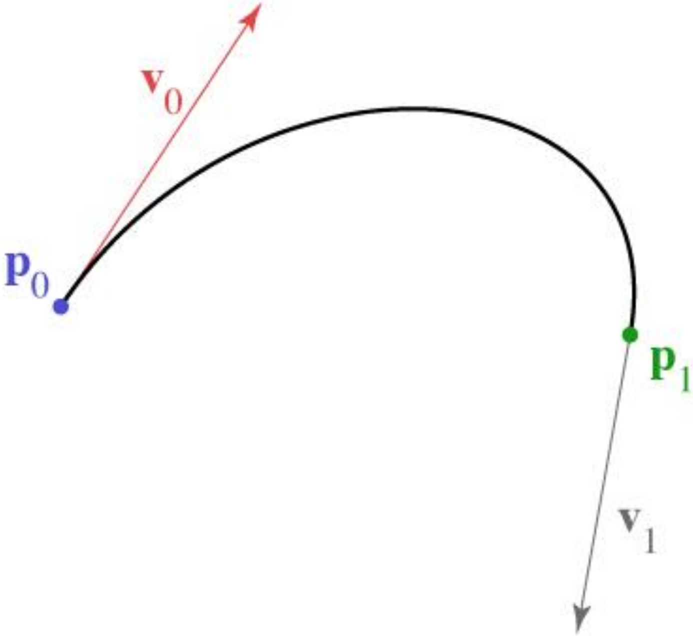


Hermite Curve



Charles Hermite
(1822-1901)

- A cubic polynomial.
- Constraints: endpoints and their tangents (derivatives)



Hermite curve

- Solve constraints to find coefficients

$$x(t) = at^3 + bt^2 + ct + d$$
$$x'(t) = 3at^2 + 2bt + c$$
$$x(0) = x_0 = d$$
$$x(1) = x_1 = a + b + c + d$$
$$x'(0) = x'_0 = c$$
$$x'(1) = x'_1 = 3a + 2b + c$$

$$\begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} x_0 \\ x_1 \\ x'_0 \\ x'_1 \end{bmatrix}$$

Hermite curve

- Solve constraints

to find coefficients

$$x(t) = at^3 + bt^2 + ct + d$$

$$x'(t) = 3at^2 + 2bt + c$$

$$x(0) = x_0 = d$$

$$x(1) = x_1 = a + b + c + d$$

$$x'(0) = x'_0 = c$$

$$x'(1) = x'_1 = 3a + 2b + c$$

$$\begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} x_0 \\ x_1 \\ x'_0 \\ x'_1 \end{bmatrix}$$

calculate A^{-1}

$$\begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x'_0 \\ x'_1 \end{bmatrix}$$

$$d = x_0$$

$$c = x'_0$$

$$a = 2x_0 - 2x_1 + x'_0 + x'_1$$

$$b = -3x_0 + 3x_1 - 2x'_0 - x'_1$$

Hermite curve

- Matrix form is much simpler

$$\mathbf{p}(t) = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{p}_0 \\ \mathbf{p}_1 \\ \mathbf{v}_0 \\ \mathbf{v}_1 \end{bmatrix}$$

  - coefficients = rows
  - basis functions = columns

Hermite basis matrix

$$\begin{bmatrix} p_0 \\ p_1 \\ v_0 \\ v_1 \end{bmatrix} = \begin{bmatrix} x_0 & y_0 \\ x_1 & y_1 \\ x'_0 & y'_0 \\ x'_1 & y'_1 \end{bmatrix}$$

Coefficients = rows

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

$$\begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \begin{bmatrix} \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \end{bmatrix} \begin{bmatrix} \mathbf{p}_0 \\ \mathbf{p}_1 \\ \mathbf{p}_2 \\ \mathbf{p}_3 \end{bmatrix}$$

$$\mathbf{p}(t) = b_0(t)\mathbf{p}_0 + b_1(t)\mathbf{p}_1 + b_2(t)\mathbf{p}_2 + b_3(t)\mathbf{p}_3$$

Basis functions = columns

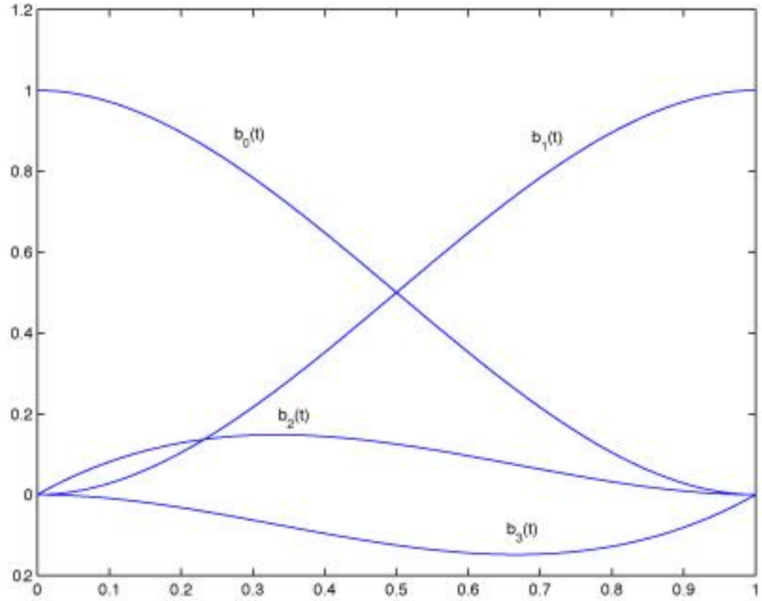
$$\mathbf{p}(t) = \mathbf{a}t^3 + \mathbf{b}t^2 + \mathbf{c}t + \mathbf{d}$$

$$\begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \begin{bmatrix} \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \end{bmatrix} \begin{bmatrix} \mathbf{p}_0 \\ \mathbf{p}_1 \\ \mathbf{p}_2 \\ \mathbf{p}_3 \end{bmatrix}$$

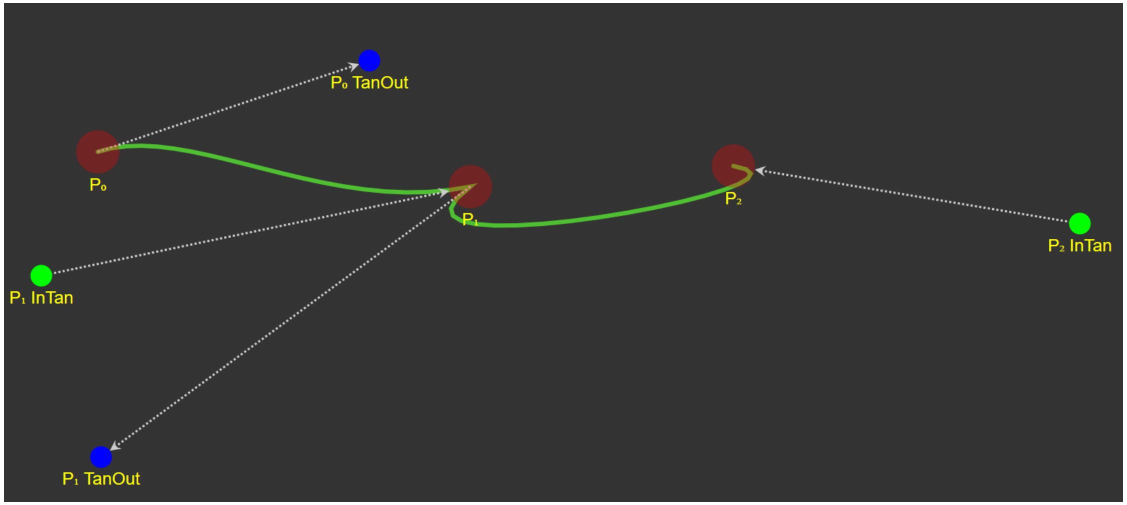
$$\mathbf{p}(t) = b_0(t)\mathbf{p}_0 + b_1(t)\mathbf{p}_1 + b_2(t)\mathbf{p}_2 + b_3(t)\mathbf{p}_3$$

Hermite curve

- Hermite basis functions



[Practice] Hermite Curve Online Demo



https://codepen.io/liorda/pen/KrvBwr

- Change the position of end points and their derivatives by dragging

Quiz #2

- Go to https://www.slido.com/
- Join #**cg-ys**
- Click “Polls”

- Submit your answer in the following format:
  - **Student ID: Your answer**
  - **e.g. 2017123456: 4)**

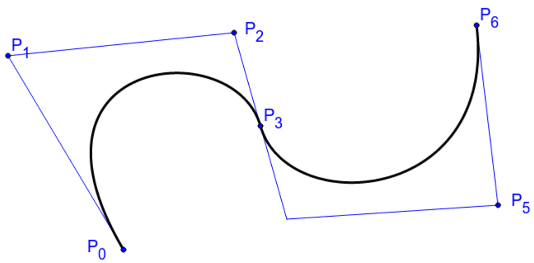
- Note that you must submit all quiz answers in the above format to be checked for “attendance”.

Bezier Curve

Bezier Curve

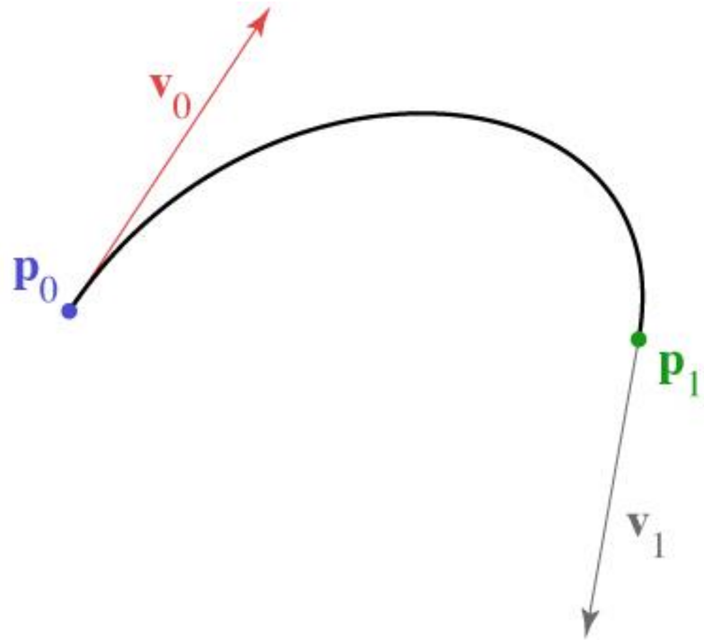
- A Bezier curve is a polynomial defined in Bezier form.
  - We'll see a cubic Bezier curve example in the following slides.
  - But note that Bezier curves are not limited to using a third-degree polynomial.

- In Bezier spline, you can connect curve pieces smoothly by carefully specifying *control points*.



Recall: Hermite curve

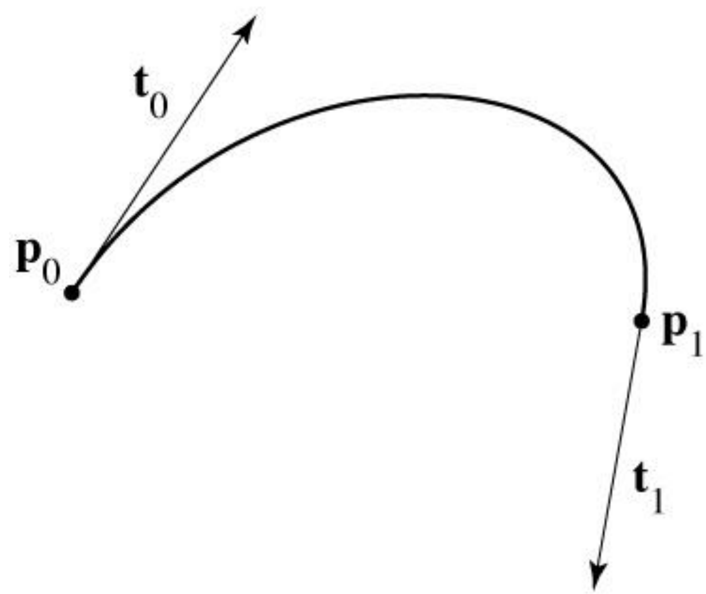
- Constraints: endpoints and tangents (derivatives)



$$\mathbf{p}(t) = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{p}_0 \\ \mathbf{p}_1 \\ \mathbf{v}_0 \\ \mathbf{v}_1 \end{bmatrix}$$

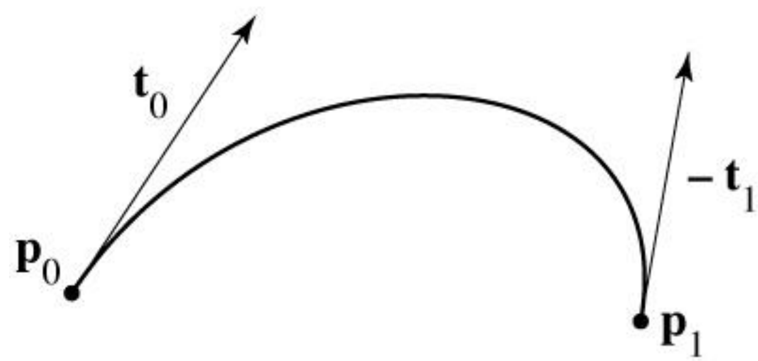
Hermite to Bézier

- Mixture of points and vectors is awkward
- Specify tangents as differences of points



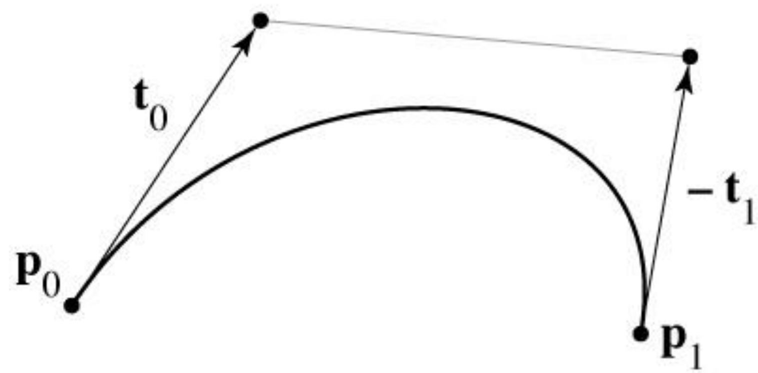
Hermite to Bézier

- Mixture of points and vectors is awkward
- Specify tangents as differences of points



Hermite to Bézier

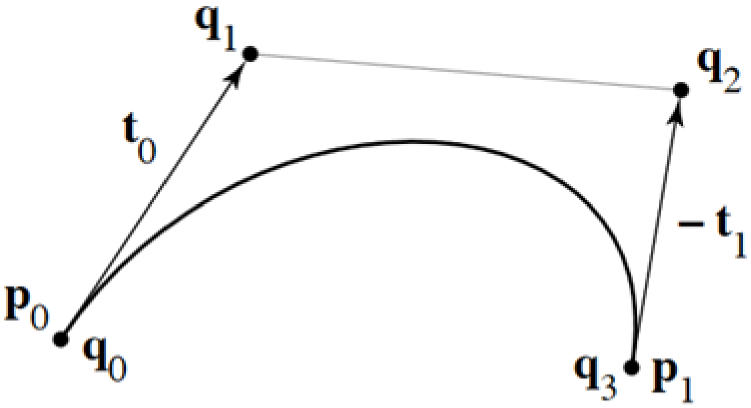
- Mixture of points and vectors is awkward
- Specify tangents as differences of points



Hermite to Bézier

Pierre Bézier (1910-1999) widely published research on this curve while working at Renault

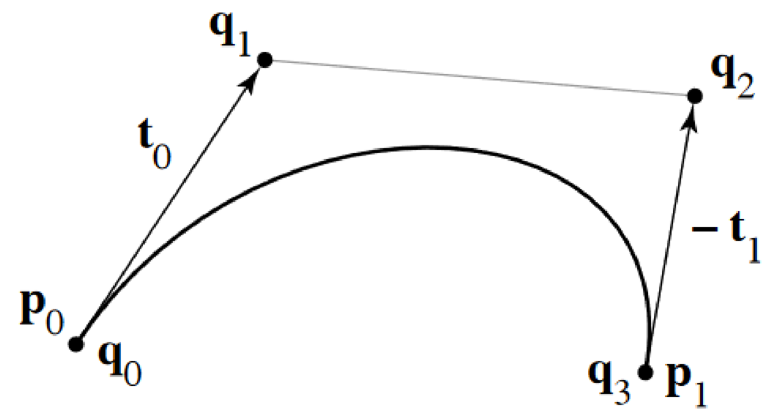
- Mixture of points and vectors is awkward
- Specify tangents as differences of points



q_0, q_1, q_2, q_3
: control points

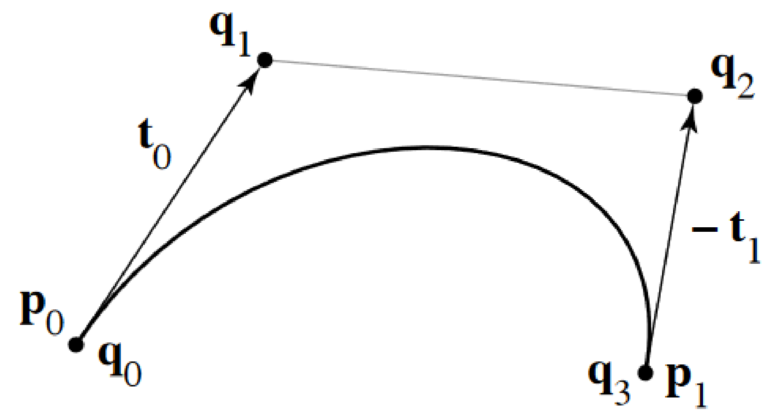
– note derivative is defined as 3 times offset t

Hermite to Bézier



$$\mathbf{p}_0 = \mathbf{q}_0$$
$$\mathbf{p}_1 = \mathbf{q}_3$$
$$\mathbf{v}_0 = 3(\mathbf{q}_1 - \mathbf{q}_0)$$
$$\mathbf{v}_1 = 3(\mathbf{q}_3 - \mathbf{q}_2)$$

Hermite to Bézier



$$\mathbf{p}_0 = \mathbf{q}_0$$

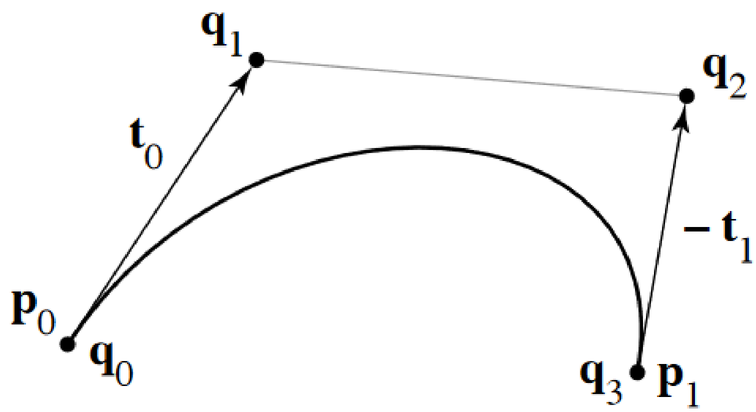
$$\mathbf{p}_1 = \mathbf{q}_3$$

$$\mathbf{v}_0 = 3(\mathbf{q}_1 - \mathbf{q}_0)$$

$$\mathbf{v}_1 = 3(\mathbf{q}_3 - \mathbf{q}_2)$$

$$\begin{bmatrix} \mathbf{p}_0 \\ \mathbf{p}_1 \\ \mathbf{v}_0 \\ \mathbf{v}_1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ -3 & 3 & 0 & 0 \\ 0 & 0 & -3 & 3 \end{bmatrix} \begin{bmatrix} \mathbf{q}_0 \\ \mathbf{q}_1 \\ \mathbf{q}_2 \\ \mathbf{q}_3 \end{bmatrix}$$

Hermite to Bézier



$$\mathbf{p}_0 = \mathbf{q}_0$$
$$\mathbf{p}_1 = \mathbf{q}_3$$
$$\mathbf{v}_0 = 3(\mathbf{q}_1 - \mathbf{q}_0)$$
$$\mathbf{v}_1 = 3(\mathbf{q}_3 - \mathbf{q}_2)$$

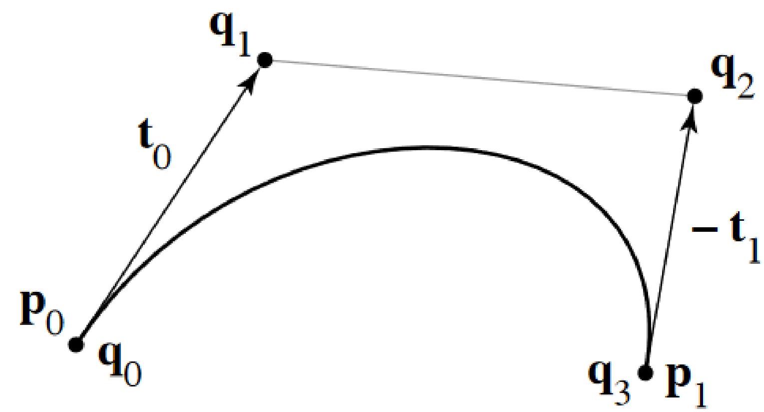
control points

$$\begin{bmatrix} \mathbf{a} \\ \mathbf{b} \\ \mathbf{c} \\ \mathbf{d} \end{bmatrix} = \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ -3 & 3 & 0 & 0 \\ 0 & 0 & -3 & 3 \end{bmatrix} \begin{bmatrix} \mathbf{q}_0 \\ \mathbf{q}_1 \\ \mathbf{q}_2 \\ \mathbf{q}_3 \end{bmatrix}$$

Hermite matrix

$$\begin{bmatrix} \mathbf{a} \\ \mathbf{b} \\ \mathbf{c} \\ \mathbf{d} \end{bmatrix} = \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{q}_0 \\ \mathbf{q}_1 \\ \mathbf{q}_2 \\ \mathbf{q}_3 \end{bmatrix}$$

Hermite to Bézier



$$\mathbf{p}_0 = \mathbf{q}_0$$

$$\mathbf{p}_1 = \mathbf{q}_3$$

$$\mathbf{v}_0 = 3(\mathbf{q}_1 - \mathbf{q}_0)$$

$$\mathbf{v}_1 = 3(\mathbf{q}_3 - \mathbf{q}_2)$$

$$\begin{bmatrix} \mathbf{a} \\ \mathbf{b} \\ \mathbf{c} \\ \mathbf{d} \end{bmatrix} = \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ -3 & 3 & 0 & 0 \\ 0 & 0 & -3 & 3 \end{bmatrix} \begin{bmatrix} \mathbf{q}_0 \\ \mathbf{q}_1 \\ \mathbf{q}_2 \\ \mathbf{q}_3 \end{bmatrix}$$

Bézier matrix

Bezier basis matrix

use notation '**p**' instead of '**q**'

$$\mathbf{p}(t) = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{p}_0 \\ \mathbf{p}_1 \\ \mathbf{p}_2 \\ \mathbf{p}_3 \end{bmatrix}$$

– note that these are the Bernstein polynomials

$$b_{n,k}(t) = \binom{n}{k} t^k (1-t)^{n-k}$$

and that defines Bézier curves for any degree

Bezier Curve

- Bernstein basis functions

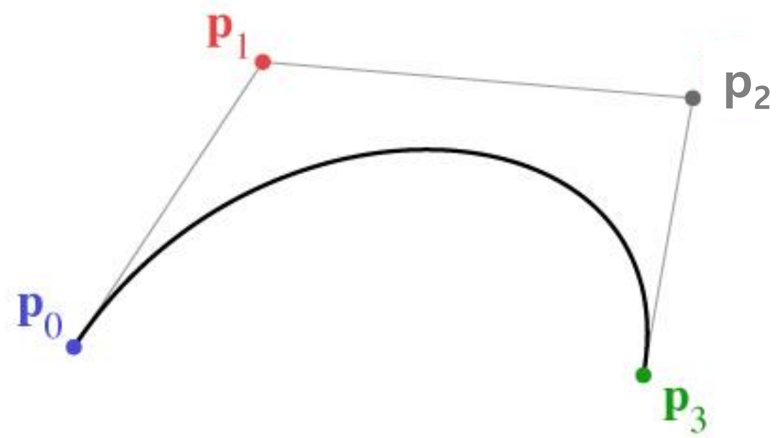
$$B_i^n(t) = \binom{n}{i}(1-t)^{n-i}t^i$$

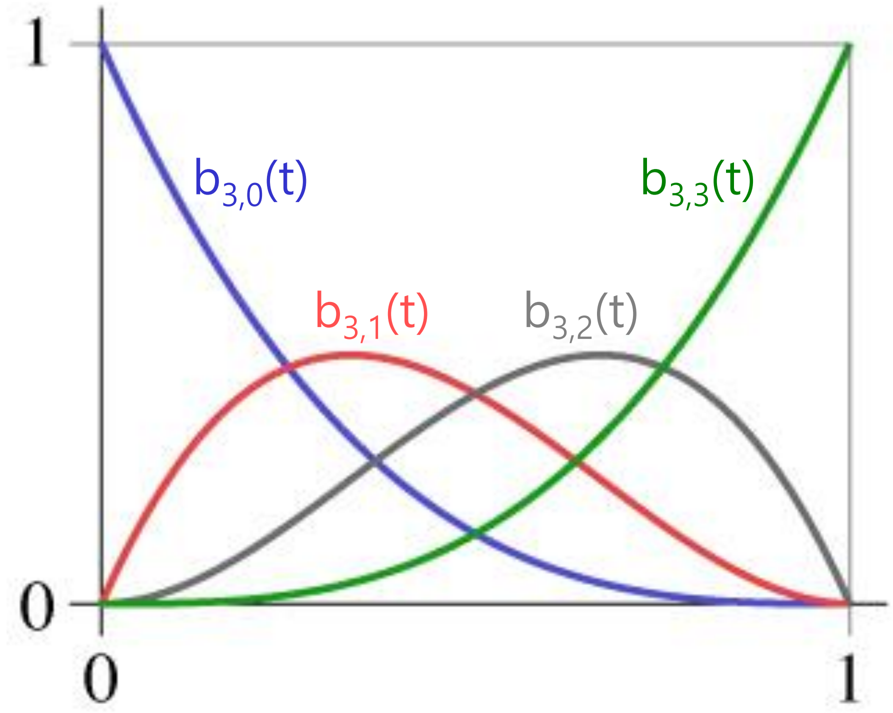
$$B_0^3(t) = (1-t)^3$$
$$B_1^3(t) = 3t(1-t)^2$$
$$B_2^3(t) = 3t^2(1-t)^1$$
$$B_3^3(t) = t^3$$

- Cubic Bezier curve: Cubic polynomial in Bernstein bases

$$\begin{aligned}\mathbf{p}(t) &= B_0^3(t)\mathbf{p}_0 + B_1^3(t)\mathbf{p}_1 + B_2^3(t)\mathbf{p}_2 + B_3^3(t)\mathbf{p}_3 \\ &= (1-t)^3\mathbf{p}_0 + 3t(1-t)^2\mathbf{p}_1 + 3t^2(1-t)\mathbf{p}_2 + t^3\mathbf{p}_3\end{aligned}$$

Bézier basis





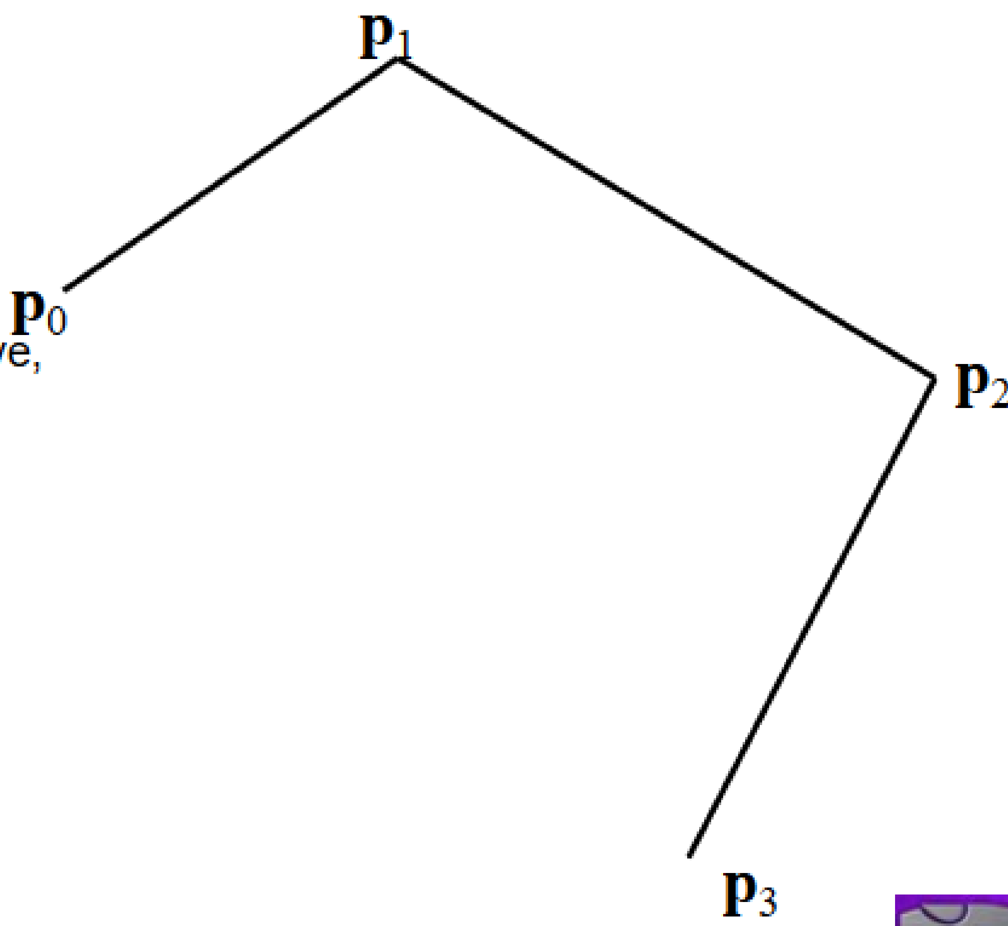
de Casteljau's Algorithm

Paul de Casteljau (1930-) first developed the 'Bezier' curve using this algorithm in 1959 while working at Citroën, but was not able to publish them due to company policy

- Another method to compute Bezier curve

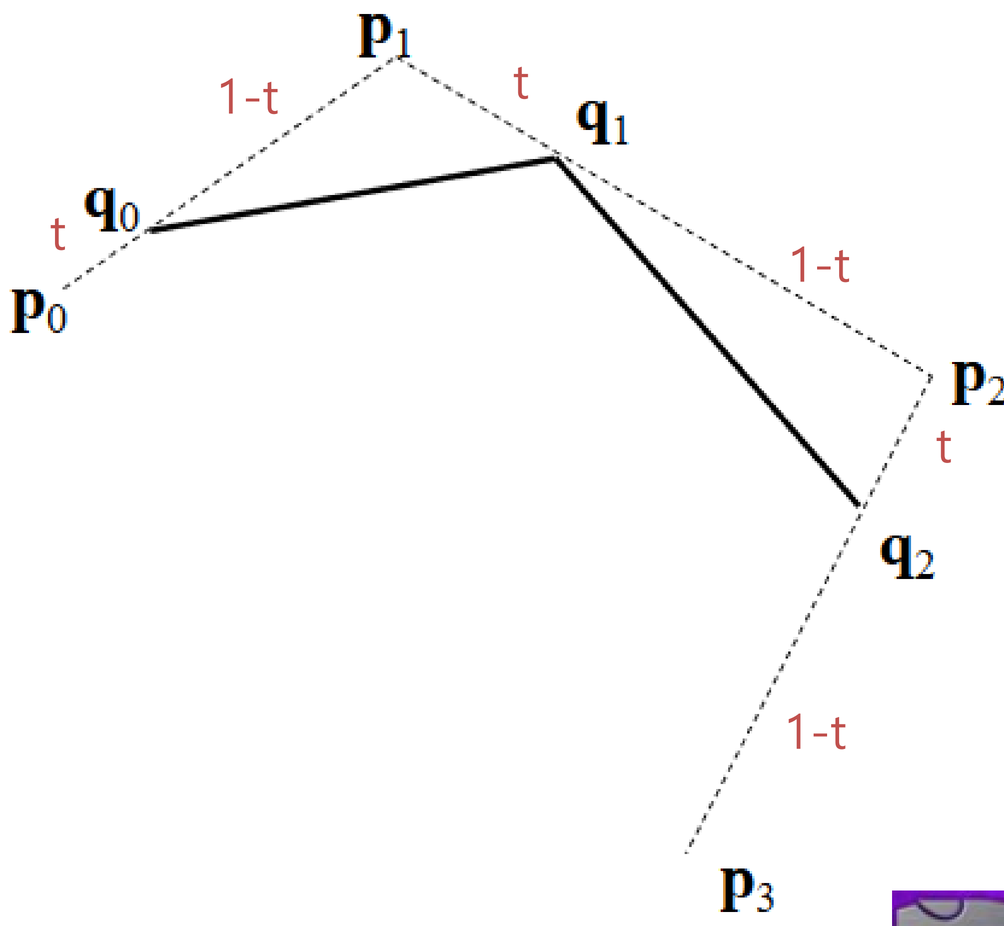
de Casteljau Algorithm

- We start with our original set of points
- In the case of a cubic Bezier curve, we start with four points



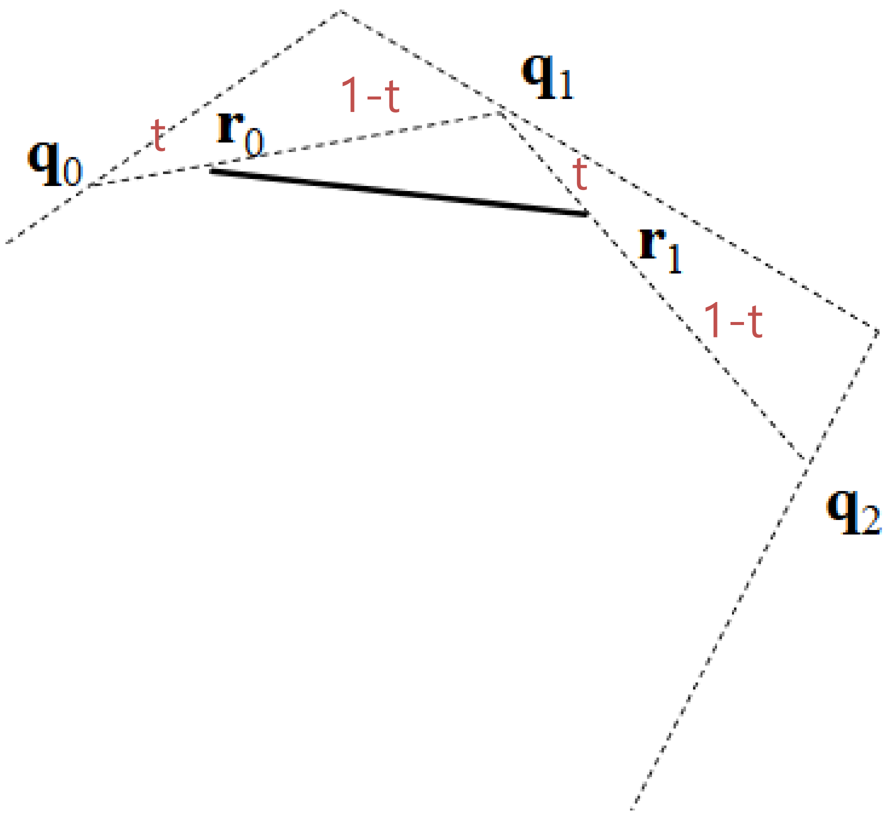
de Casteljau Algorithm

$$\mathbf{q}_0 = Lerp(t, \mathbf{p}_0, \mathbf{p}_1)$$
$$\mathbf{q}_1 = Lerp(t, \mathbf{p}_1, \mathbf{p}_2)$$
$$\mathbf{q}_2 = Lerp(t, \mathbf{p}_2, \mathbf{p}_3)$$



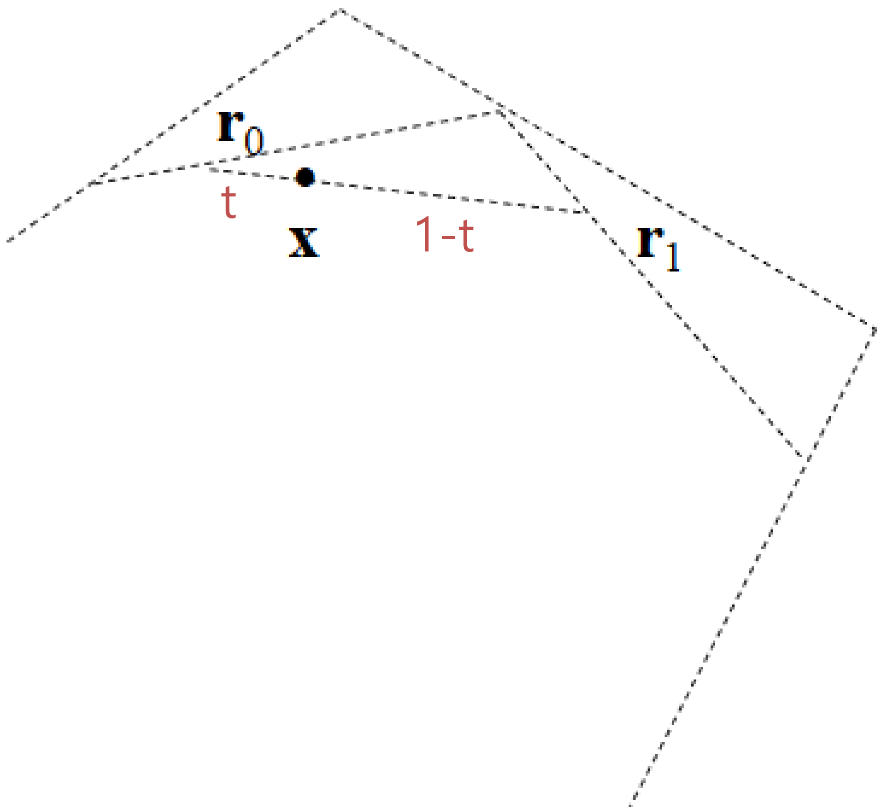
de Casteljau Algorithm

$$\mathbf{r}_0 = Lerp(t, \mathbf{q}_0, \mathbf{q}_1)$$
$$\mathbf{r}_1 = Lerp(t, \mathbf{q}_1, \mathbf{q}_2)$$

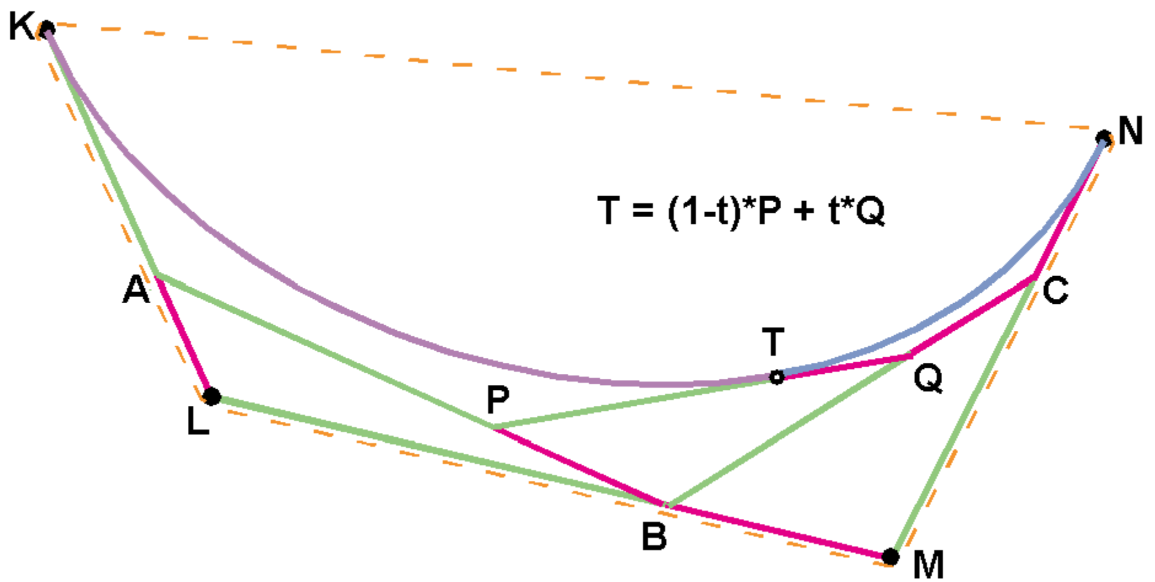


de Casteljau Algorithm

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



de Casteljau’s Algorithm



$$P = (1-t)*A + t*B \qquad Q = (1-t)*B + t*C$$

$$T = (1-t)(1-t)*A + (1-t)t*B + t(1-t)*B + t*t*C$$

$$A = (1-t)*K + t*L \qquad B = (1-t)*L + t*M \qquad C = (1-t)*M + t*N$$

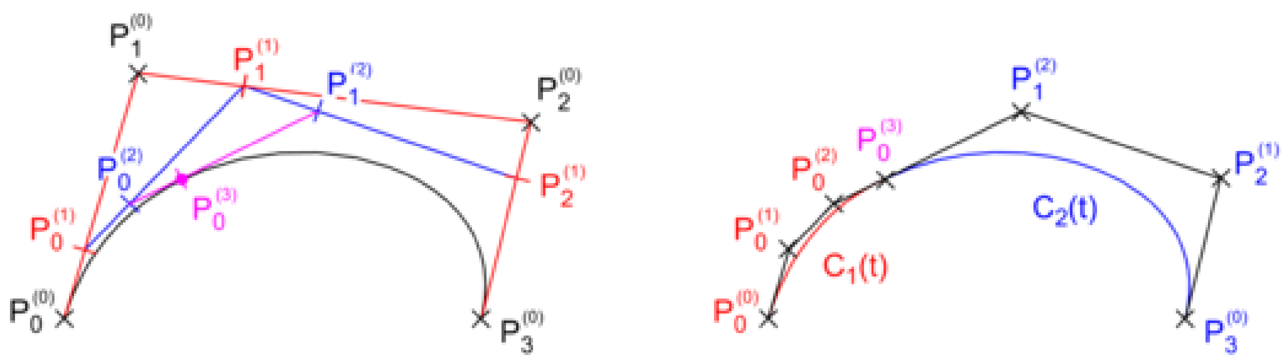
$$T = (1-t)^3*K+(1-t)^2t*L + 2(1-t)^2t*L+2(1-t)t^2*M + (1-t)t^2*M+t^3*N$$

https://people.eecs.berkeley.edu/~sequin/CS284/LECT06/L3.htm

de Casteljau's Algorithm

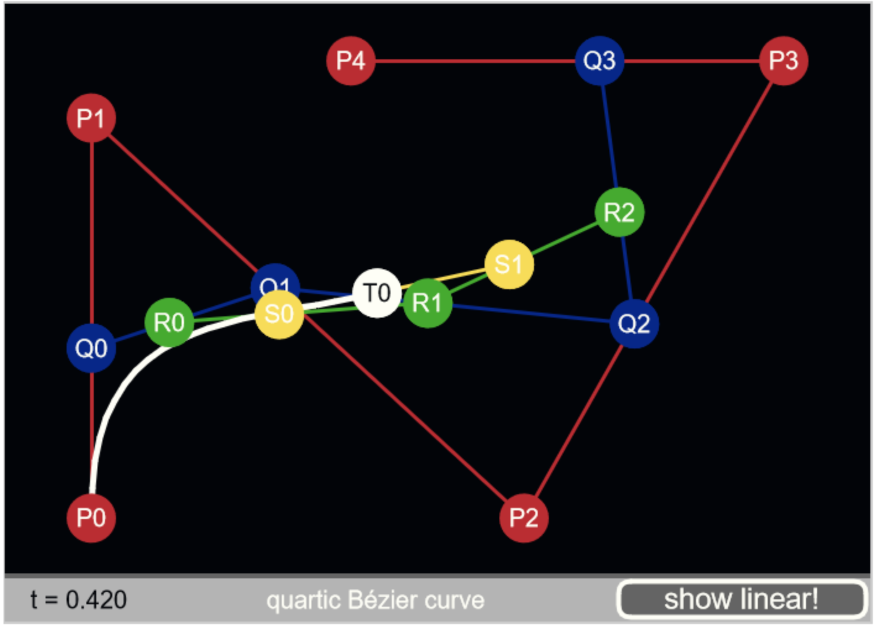
- Nice recursive algorithm to compute a point on a Bezier curve

- Additionally, it subdivide a Bezier curve into two segments



- You can draw a curve with a sufficient number of subdivided control points
  - "Subdivision" method for displaying curves

[Practice] de Casteljau's Algorithm



http://www.malinc.se/m/DeCasteljauAndBezier.php

- Move red points
- Also check the subdivision demo

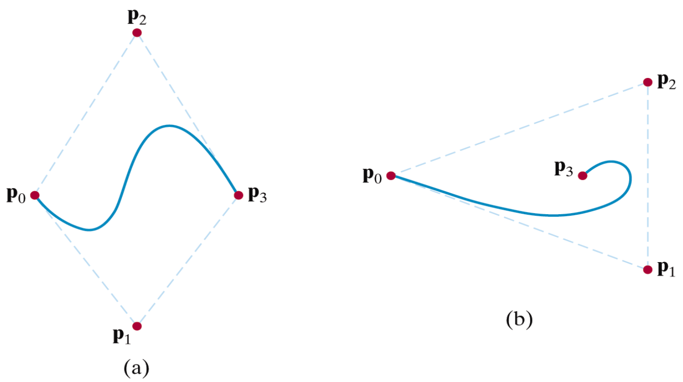
Displaying Curves

- To display a curve, we need to generate a list of line segments to draw.
  - What we can compute is a set of points on a curve
  - Connecting them with line segments would be good approximation for the curve
- Brute-force
  - Evaluate **p**(t) for incrementally spaced values of t
- Finite difference
  - The same idea, but much more efficient
  - See http://www.drdobbs.com/forward-difference-calculation-of-bezier/184403417
- Subdivision
  - Use de Casteljau’s algorithm

Properties of Bezier Curve

- Intuitively controlled by control points

- The curve is contained in the *convex hull* of control points.



Convex hull: Minimal-sized convex polygon containing all points

- End point interpolation.

Quiz #3

- Go to https://www.slido.com/
- Join #**cg-ys**
- Click “Polls”

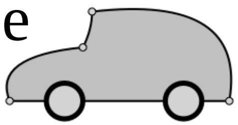
- Submit your answer in the following format:
  - **Student ID: Your answer**
  - **e.g. 2017123456: 4)**

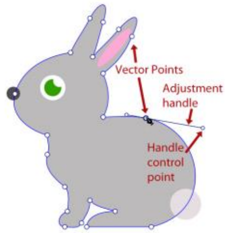
- Note that you must submit all quiz answers in the above format to be checked for “attendance”.

Bezier Spline

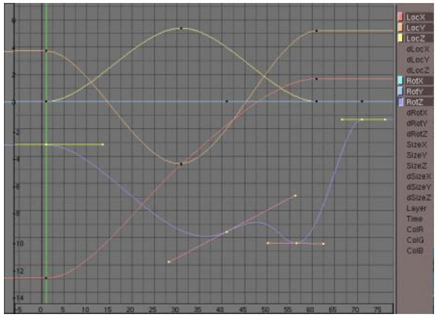
- Bezier spline, piecewise Bezier polynomials, is very widely used. For example,

- To draw shapes in graphic tools such as Adobe Illustrator

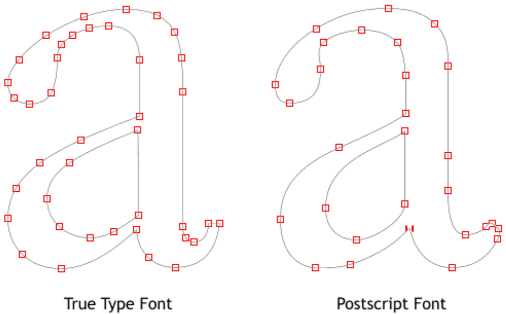




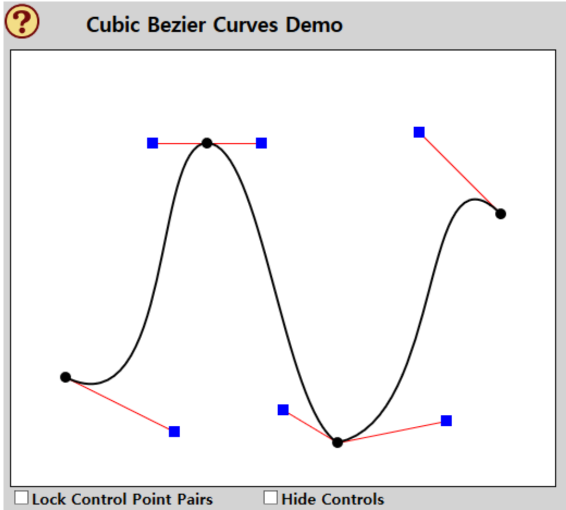
- To define animation paths in 3D authoring tools such as Blender and Maya



- TrueType fonts use quadratic Bezier spline, PostScript fonts use cubic Bezier spline



[Practice] Bezier Spline

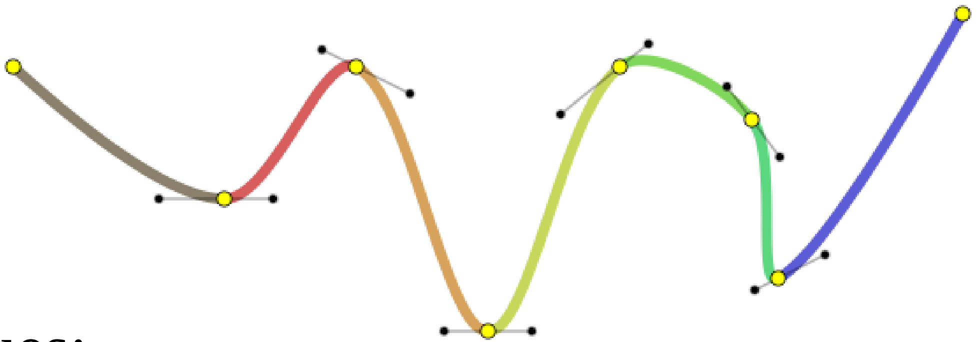


http://math.hws.edu/graphicsbook/demos/c2/cubic-bezier.html

- How to “smooth” the spline?

Spline

- Spline: ***piecewise* polynomial**



- Three issues:
  - How to connect these pieces *continuously*?
  - How easy is it to *"control"* the shape of a spline?
  - Does a spline have to *pass through* specific points?

- For details, see *11-reference-splines.pdf*

Next Time

- Lab for this lecture (next Monday):
  - Lab assignment 11

- Next lecture:
  - 12 - More Lighting, Texture

- Acknowledgement: Some materials come from the lecture slides of
  - Prof. Jehee Lee, SNU, http://mrl.snu.ac.kr/courses/CourseGraphics/index_2017spring.html
  - Prof. Taesoo Kwon, Hanyang Univ., http://calab.hanyang.ac.kr/cgi-bin/cg.cgi
  - Prof. Steve Marschner, Cornell Univ., http://www.cs.cornell.edu/courses/cs4620/2014fa/index.shtml
  - Prof. William H. Hsu, Kansas State Univ. http://slideplayer.com/slide/4635444/