
Computer Graphics

11 - Curves

Yoonsang Lee
Hanyang University

Spring 2026

Final Exam Announcement

- Date & time: **June 22 (Mon), 10:00 - 11:30 AM**
- Place: IT.BT 508 + one additional room (TBA)
- Scope: Lecture, Lab, Lab Check **8 ~ 13**, **Project 2 & 3**
- You may refer to past exam problems from previous Computer Graphics courses:
<https://cgrhyu.github.io/3-courses.html>
- **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 (please do not be late).
 - This policy is to prevent early finishers from sharing exam information with late arrivals.
- Bring your **student ID card** to the exam.

Final Exam Announcement – Additional Notes

- **Language of Answers**

- All answers should be written in English in principle.
- If writing in English is difficult, you may answer in Korean instead.
- However, this may result in some deduction of points.

- **Questions During the Exam**

- No questions regarding exam problems will be accepted during the exam.
- If you are unsure about a problem, use the provided information and write your best answer.

Final Exam Announcement – Electronic Devices

- **No digital or electronic devices** are allowed during the exam.
- This includes (but is not limited to): smartphones, smartwatches, tablets, laptops, wireless earphones, and smart glasses.
- All devices must be **powered off and kept in your bag**.
 - Do not wear any electronic devices during the exam.
- Use of any device during the exam will be treated as **academic misconduct**.
- If you are unsure whether a device is allowed, **do not bring it**.

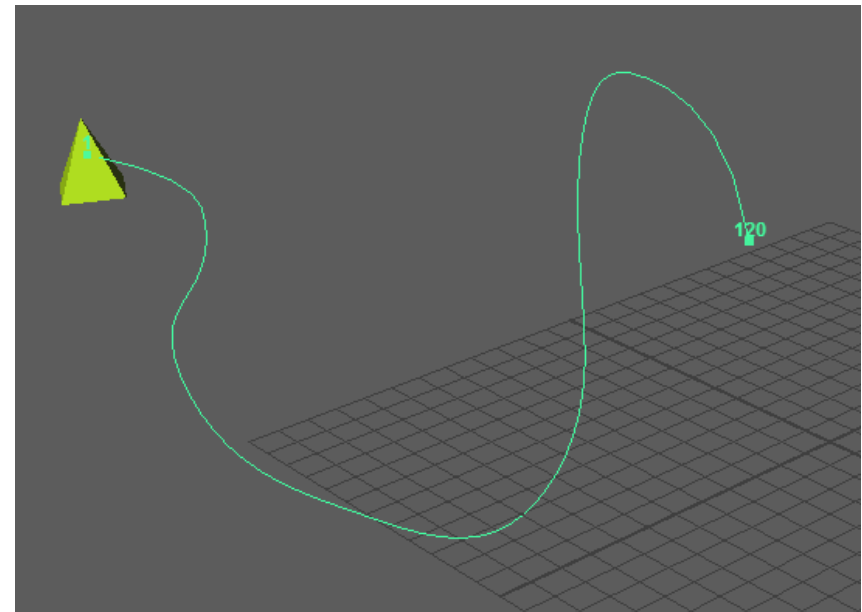
Outline

- Intro: Motivation and Curve Representation
- Polynomial Curves
 - Interpolation & Runge's Phenomenon
 - Why Cubic?
 - Formulation
- Hermite Curve
- Bezier Curve
- Spline Overview

Intro: Motivation and Curve Representation

Motivation: Why Do We Need Curves?

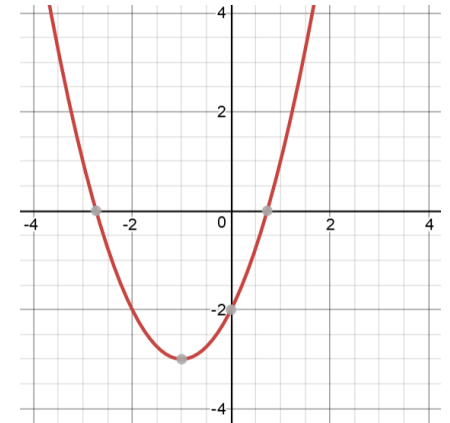
- **Smoothness**
 - no discontinuity
- In many CG applications, we need **smooth shape** and **smooth motion**.



Curve Representations

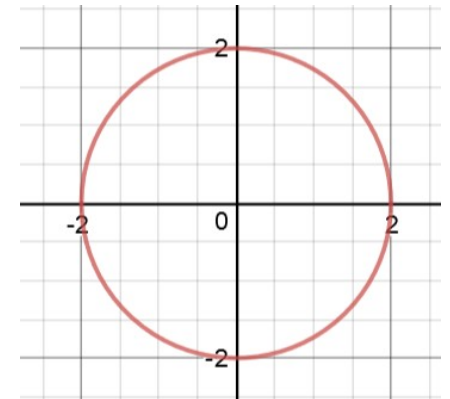
- **Explicit : $y = f(x)$**

- ex) $y = x^2 + 2x - 2$
- Pros) Easy to compute and plot points (just plug in x)
- Cons) Cannot represent curves where a single x corresponds to multiple y -values
 - Examples: circles, vertical lines, ...



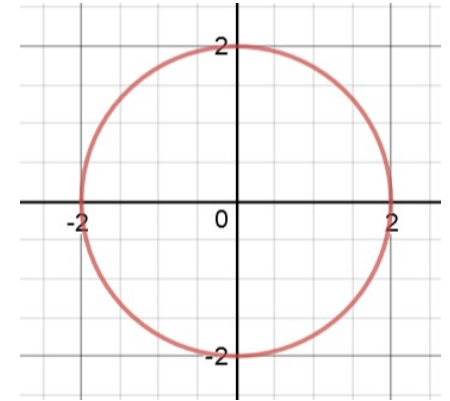
- **Implicit : $f(x, y) = 0$**

- 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 compute and plot points (just plug in t)
 - The parameter t acts as a **“local coordinate”** for points on the curve
- In computer graphics, parametric representation is usually the most suitable.



Polynomial Curves

Polynomial Curves

- *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 through given data points**.

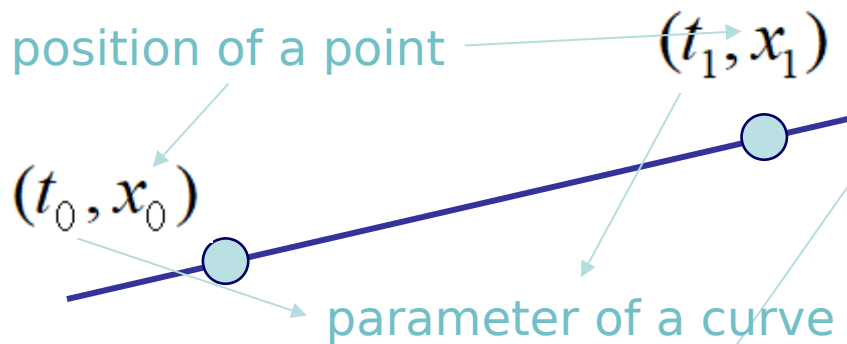
Polynomial Interpolation

- Linear interpolation with a polynomial of degree one

- Input: two nodes

- Output: Linear polynomial

How can we find a_0 and a_1 ?



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



$$\begin{aligned} a_1 t_0 + a_0 &= x_0 \\ a_1 t_1 + a_0 &= x_1 \end{aligned}$$

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

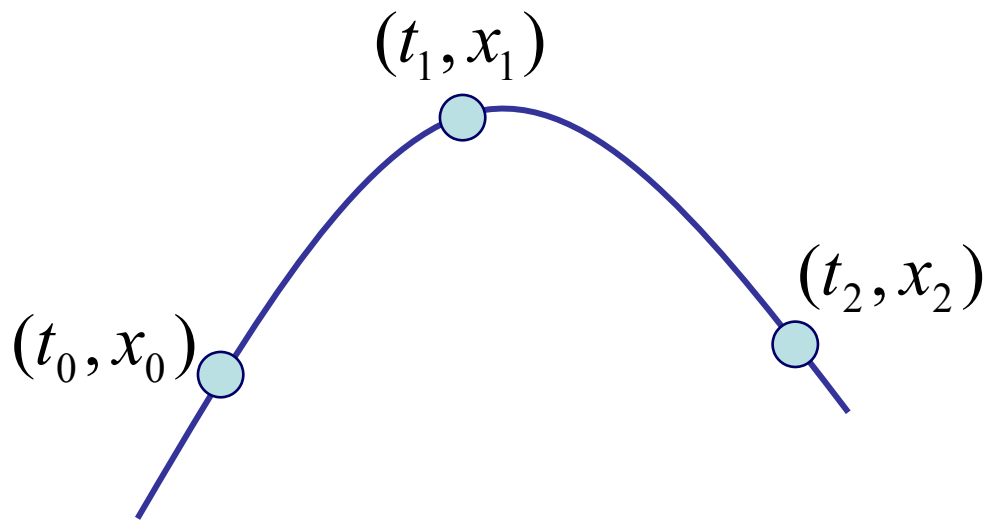
We can compute the value of a_0 & a_1 because we have **2 equations (=2 data points)** for **2 unknowns!**

If $t_0=0$ and $t_1=1$, then $a_0=x_0$ and $a_1=x_1-x_0$

$$\rightarrow x(t) = (x_1 - x_0)t + x_0 = (1-t)x_0 + tx_1$$

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**)

$$a_2 t_0^2 + a_1 t_0 + a_0 = x_0$$

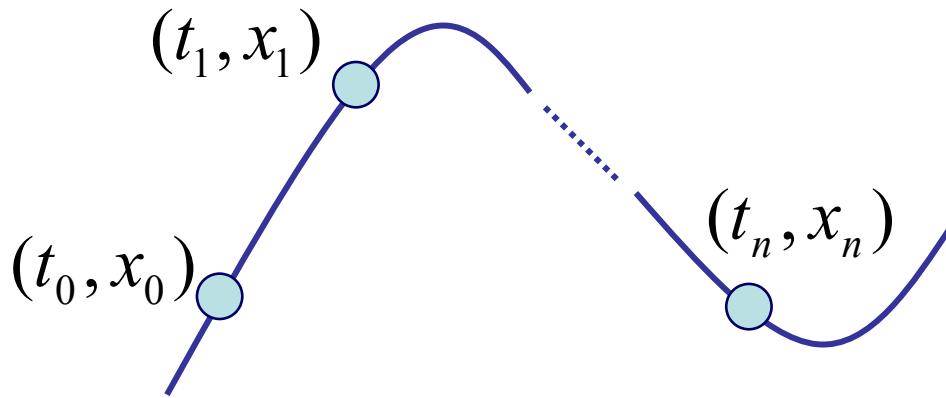
$$a_2 t_1^2 + a_1 t_1 + a_0 = x_1$$

$$a_2 t_2^2 + a_1 t_2 + a_0 = x_2$$

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

Polynomial Interpolation

- Polynomial interpolation of degree n



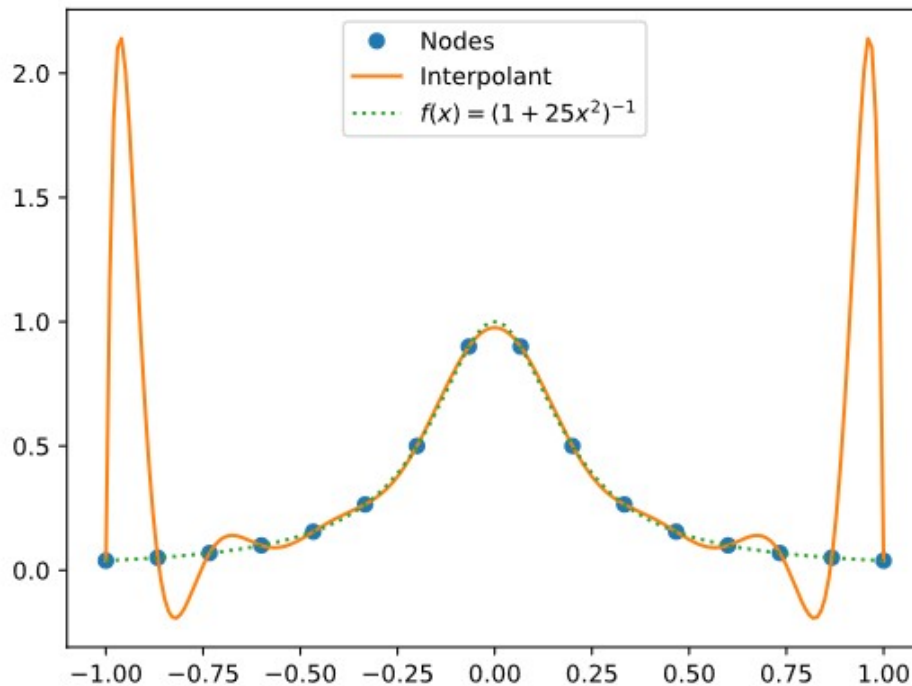
$$x(t) = a_n t^n + \dots + a_1 t + a_0$$

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

- How to find the value of unknowns $a_n, \dots, a_0$?
 - Gauss Elimination, Gauss-Jordan Elimination, Matrix Inverse (if invertible), ...

Problem of Higher-Degree Polynomial Interpolation

- Oscillations at the ends – Runge's Phenomenon



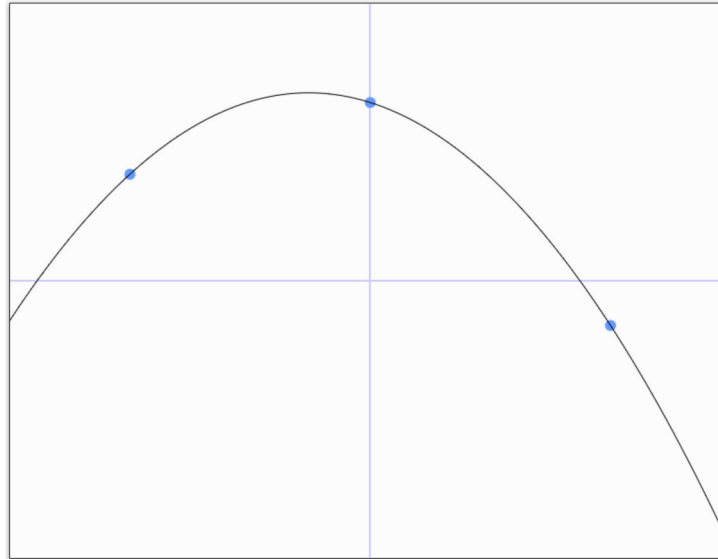
- Using higher-degree polynomial interpolation for curves is a bad idea.

[Demo] Polynomial Interpolation

Interpolation Polynomial

Click and drag the **control points** and the polynomial curve will interpolate to satisfy them.

Polynomial Degree: (parabola)



<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)

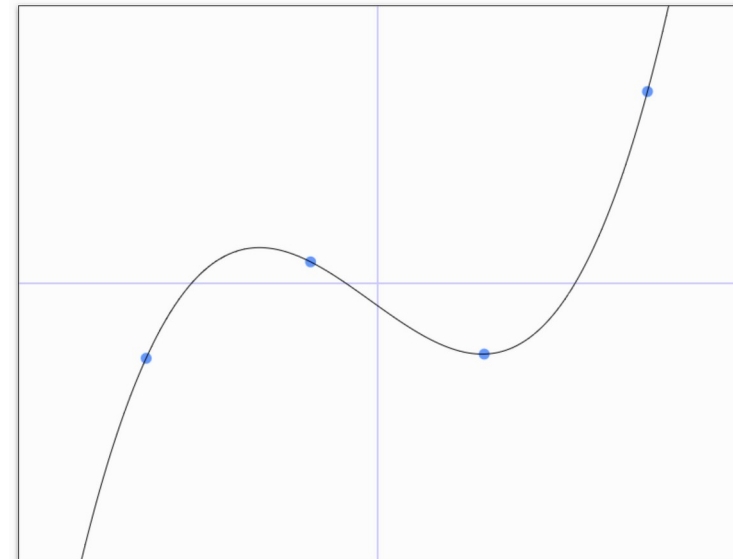
$$x(t) = a_x t^3 + b_x t^2 + c_x t + d_x$$

$$y(t) = a_y t^3 + b_y t^2 + c_y t + d_y$$

$$z(t) = a_z t^3 + b_z t^2 + c_z t + d_z$$

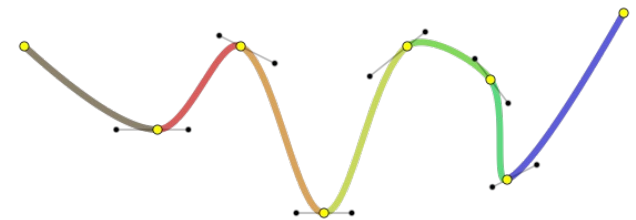
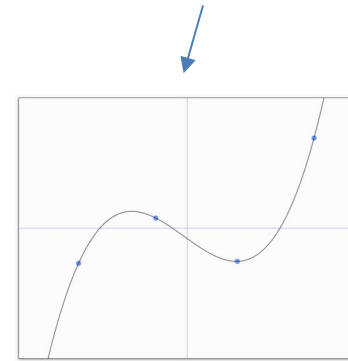
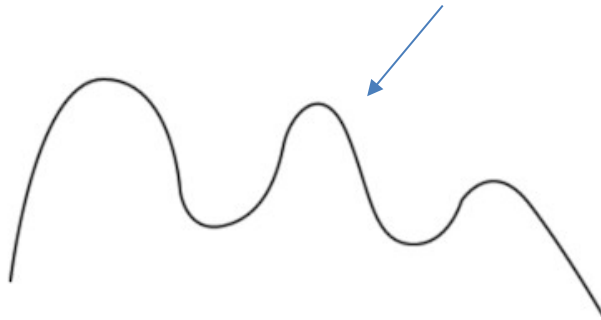
or

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



Complex Curve from Cubic Polynomials?

- How can we build complex curves using cubic polynomials?



- Answer → **Spline: *piecewise* polynomial**
- At this moment, let's just focus on a single piece of polynomial.

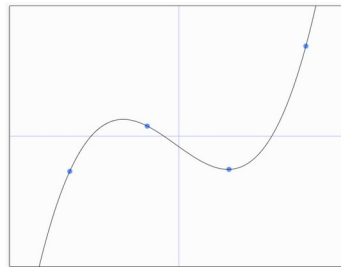
Defining a Single Piece of Cubic Polynomial

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

- 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)**.

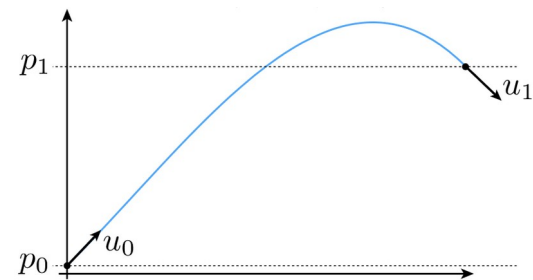
For example,

- 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**:

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

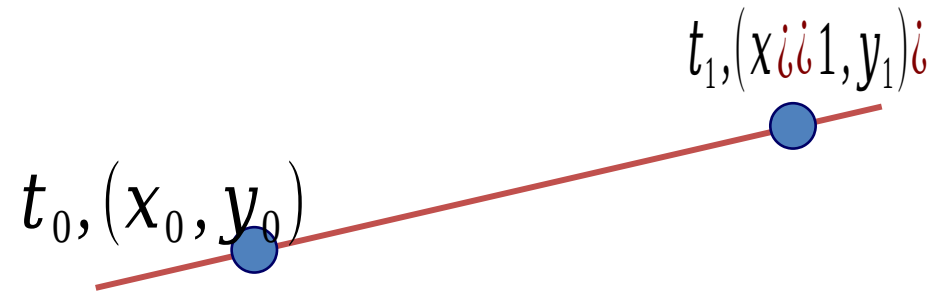
- coefficients: **a, b, c, d**
- variable: **t**

- 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: **$\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3$**
- Basis functions control each point's influence on the curve.

Trivial Example: Linear Polynomial



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 **basis matrix** **geometry vector**

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

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

power basis vector

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

Trivial Example: Linear Polynomial

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

$$\begin{aligned}\mathbf{p}(t) &= (\mathbf{p}_1 - \mathbf{p}_0)t + \mathbf{p}_0 \\ &= \underbrace{(1 - t)}_{\text{basis functions}} \mathbf{p}_0 + \underbrace{t}_{\text{basis functions}} \mathbf{p}_1\end{aligned}$$

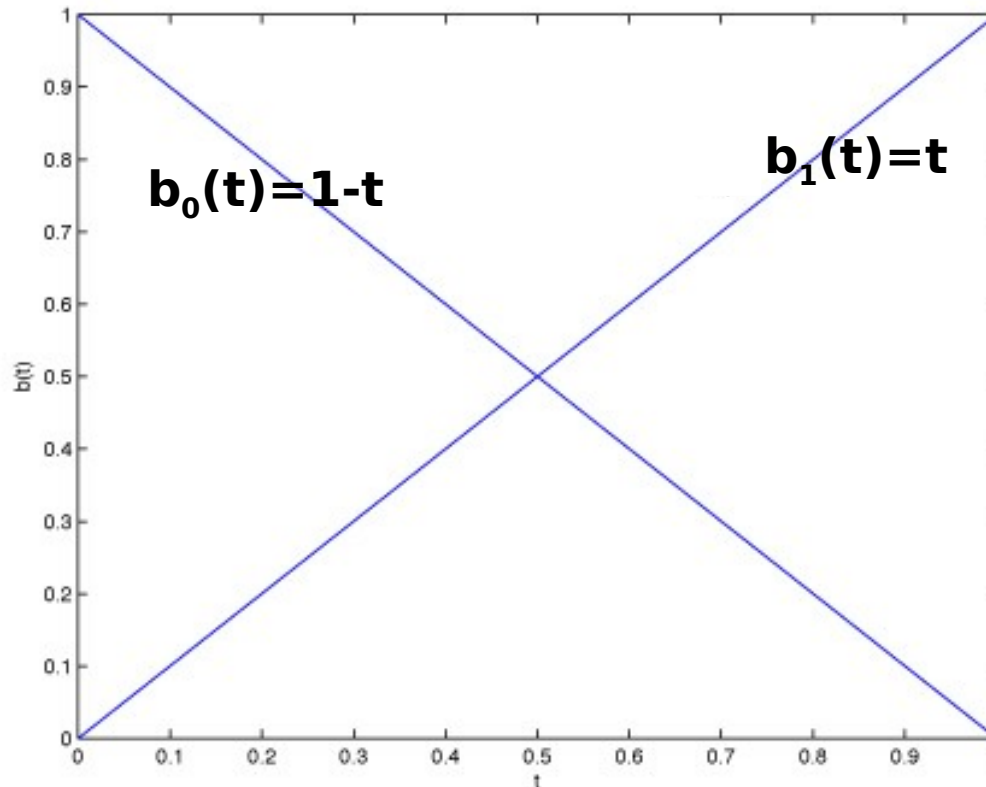
- interpretation in matrix viewpoint

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

Meaning of Basis Functions

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

- Contribution of each point as t changes



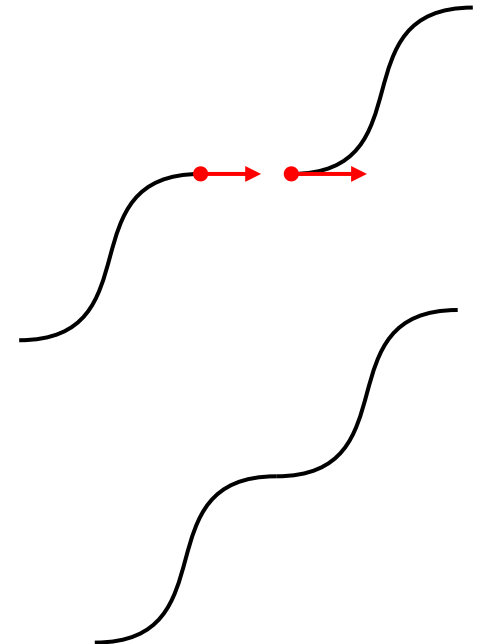
Quiz 1

- Go to <https://www.slido.com/>
- Join #cg-ys
- Click "Polls"
- Submit your answer in the following format:
 - **Student ID: Your answer**
 - e.g. **2024123456: 4.0**
- Note that your quiz answer must be submitted **in the above format** to be counted for attendance.

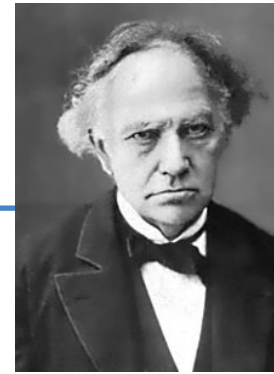
Hermite Curve

Hermite Curve

- A Hermite curve is typically a cubic polynomial expressed in Hermite form.
- Motivation:
 - In spline design, smooth connection between curve segments is essential.
 - Hermite splines address this by specifying:
 - Position of the endpoints
 - 1st derivatives at the endpoints

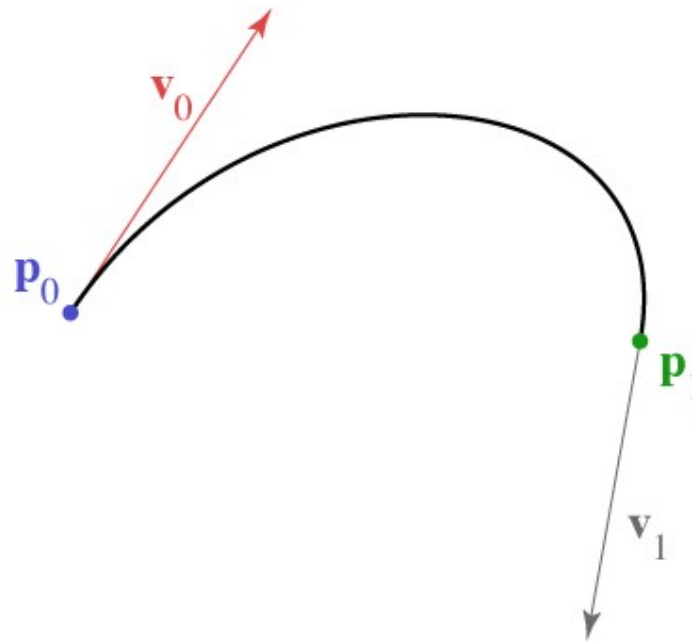


Hermite Curve



Charles Hermite
(1822-1901)

- Constraints: endpoints and their tangents (derivatives)



Hermite curve

- Solve constraints to find coefficients

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

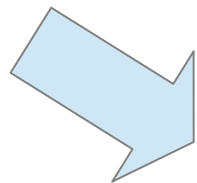
$$x'(t) = 3at^2 + 2bt + c$$

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

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

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

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



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

Hermite curve

- Solve constraints to find coefficients

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

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

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

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

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

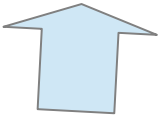
$$x'(1) = x'_1 = 3a + 2b + c$$

$$d = x_0$$

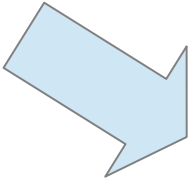
$$c = x'_0$$

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

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

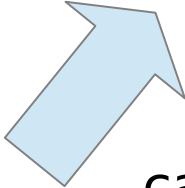


$$\begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x'_0 \\ x'_1 \end{bmatrix}$$



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

calculate A^{-1}



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

$$\begin{bmatrix} \mathbf{p}_0 \\ \mathbf{p}_1 \\ \mathbf{v}_0 \\ \mathbf{v}_1 \end{bmatrix} = \begin{bmatrix} x_0 & y_0 \\ x_1 & y_1 \\ x'_0 & y'_0 \\ x'_1 & y'_1 \end{bmatrix}$$

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

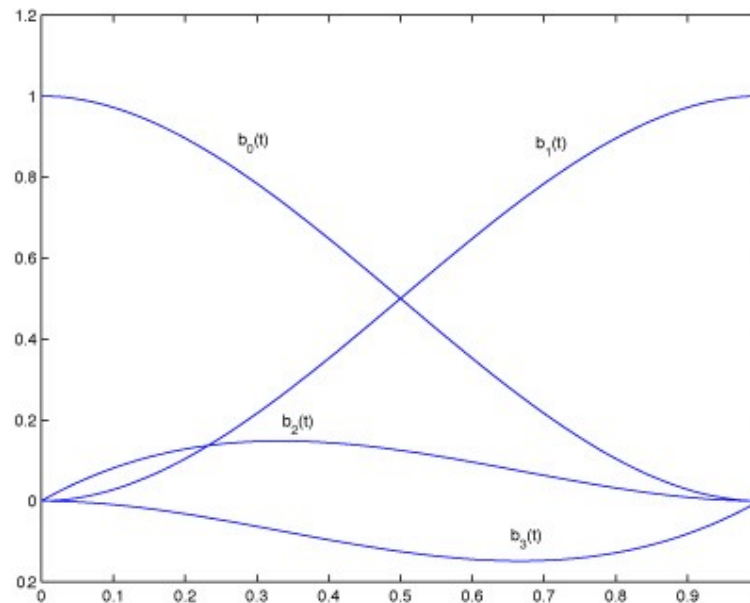
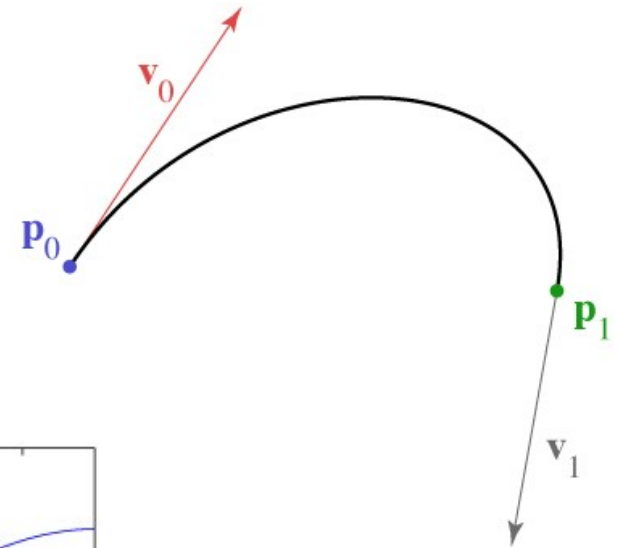
$$\mathbf{p}(t) = \mathbf{a}t^3 + \mathbf{b}t^2 + \mathbf{c}t + \mathbf{d}$$

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

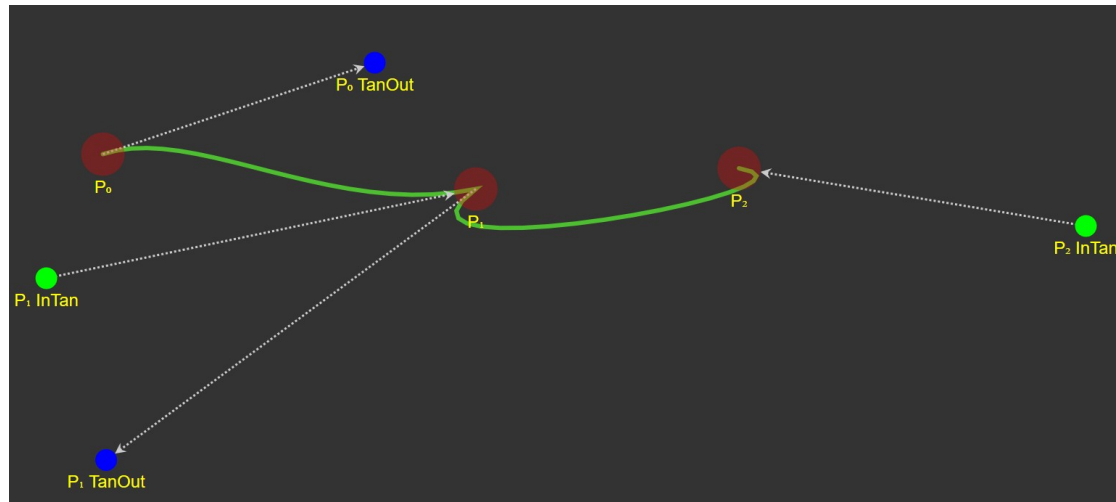
$$\mathbf{p}(t) = b_0(t)\mathbf{p}_0 + b_1(t)\mathbf{p}_1 + b_2(t)\mathbf{p}_2 + b_3(t)\mathbf{p}_3$$

Hermite curve

- Hermite basis functions



[Demo] Hermite Curve



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

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

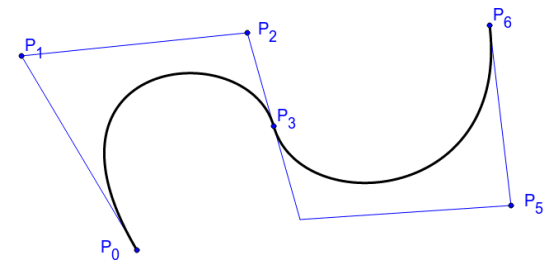
Quiz 2

- Go to <https://www.slido.com/>
- Join #cg-ys
- Click "Polls"
- Submit your answer in the following format:
 - **Student ID: Your answer**
 - e.g. **2024123456: 4.0**
- Note that your quiz answer must be submitted **in the above format** to be counted for attendance.

Bezier Curve

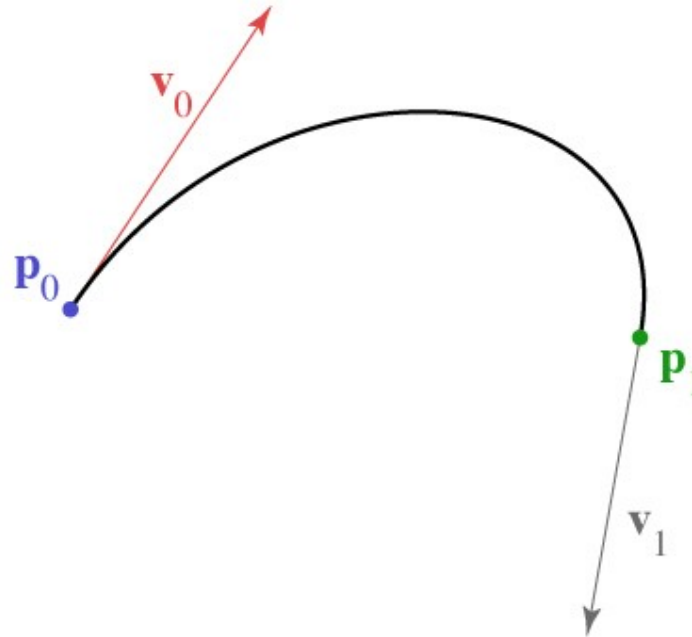
Bezier Curve

- A Bezier curve is typically a cubic polynomial expressed in Bezier form.
- Motivation:
 - In splines, we want curve segments that connect smoothly.
 - In Bezier splines, this is done by properly placing *control points*.



Recall: Hermite curve

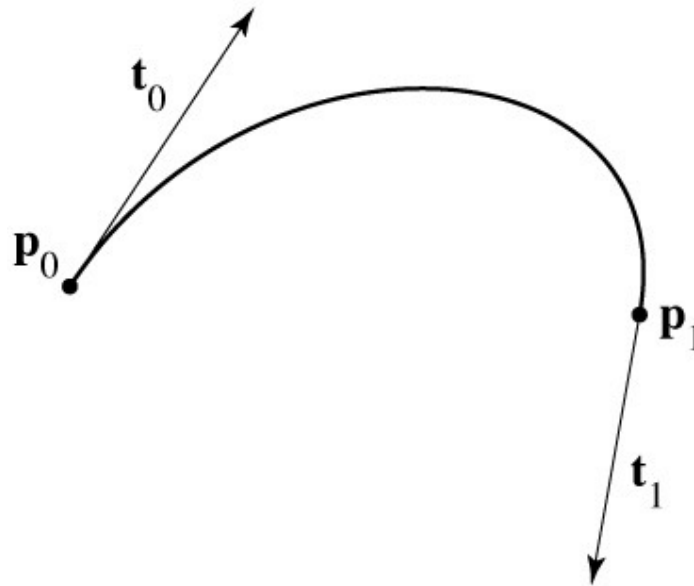
- Constraints: endpoints and tangents



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

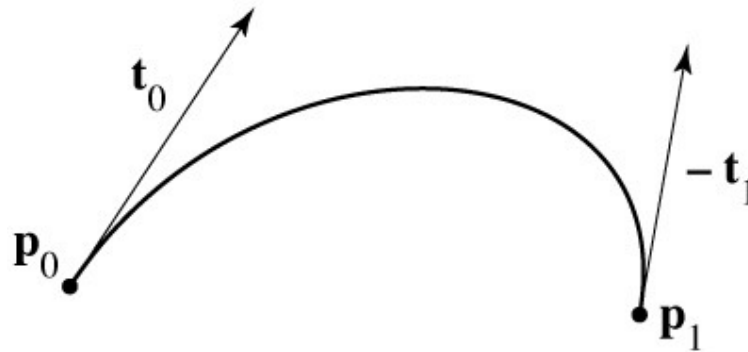
Hermite to Bézier

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



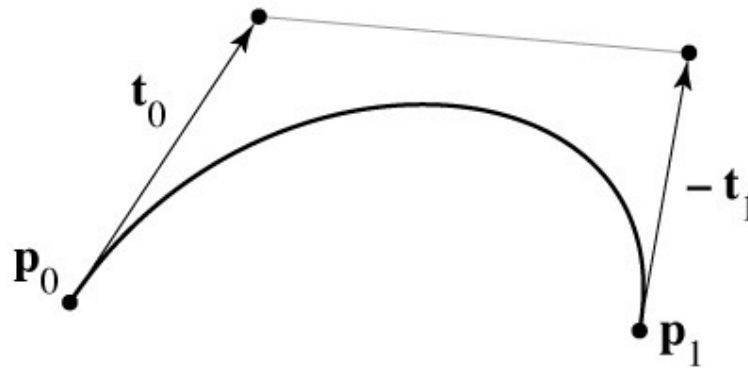
Hermite to Bézier

- 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

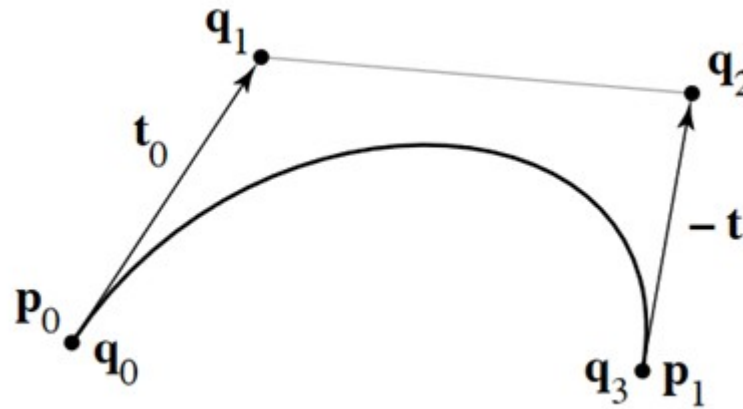


Hermite to Bézier



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

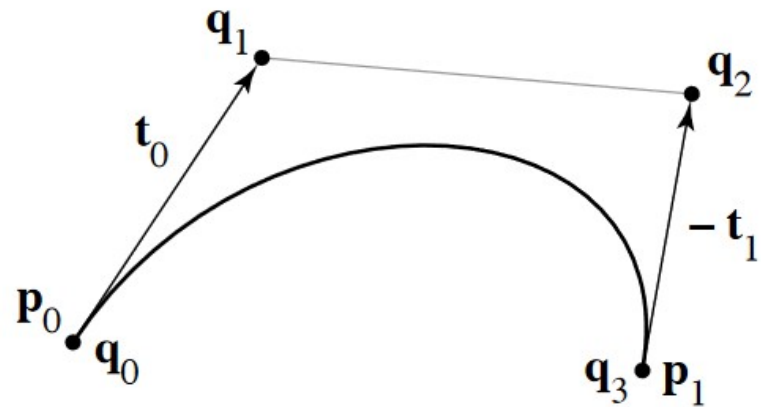
Hermite to Bézier

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

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

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

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



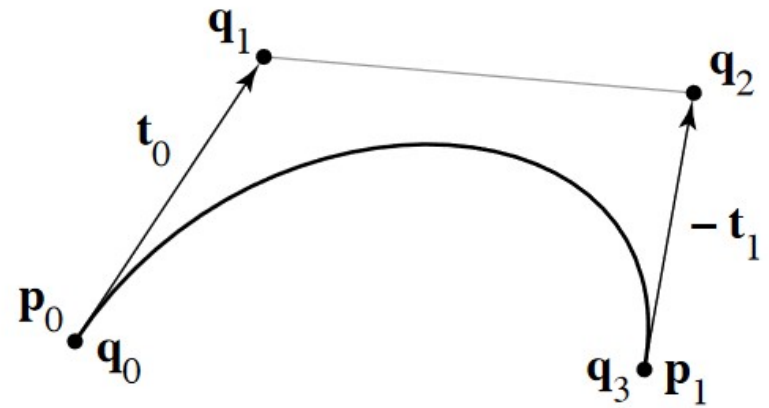
Hermite to Bézier

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

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

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

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



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

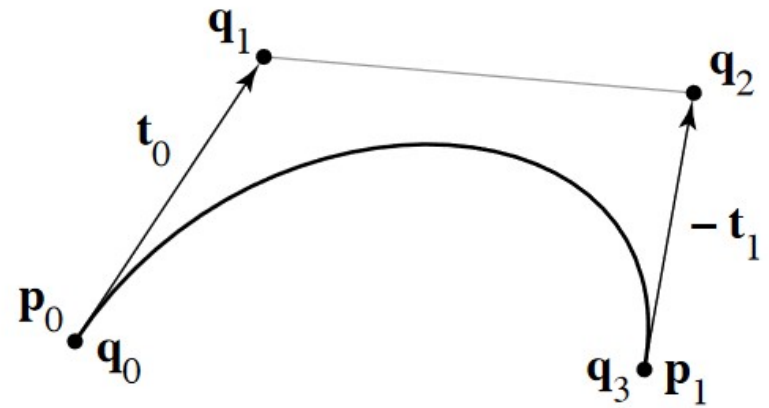
Hermite to Bézier

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

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

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

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



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

Hermite basis matrix

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

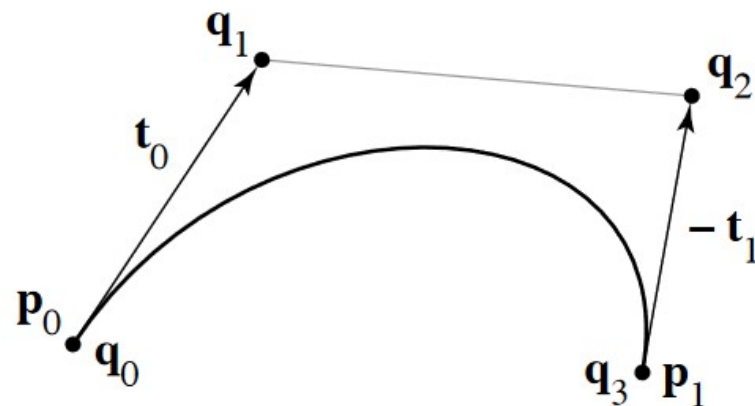
$$\mathbf{p}_0 = \mathbf{q}_0$$

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

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

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

Hermite to Bézier



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

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
(n: degrees of polynomial, k: index of basis function)

Bezier Curve

- Bernstein basis functions

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

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

$$B_1^3(t) = 3t(1-t)^2$$

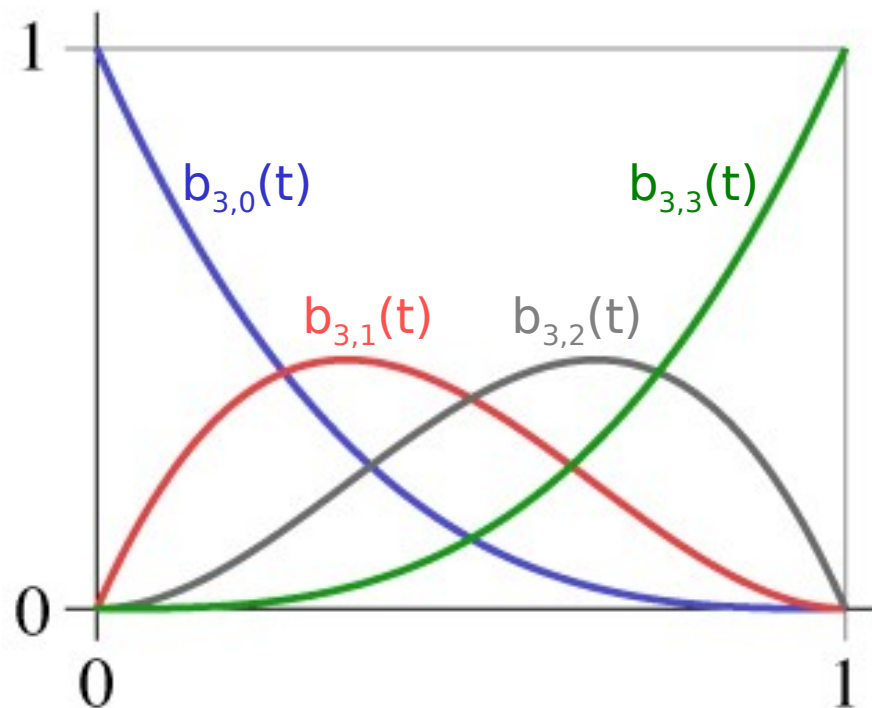
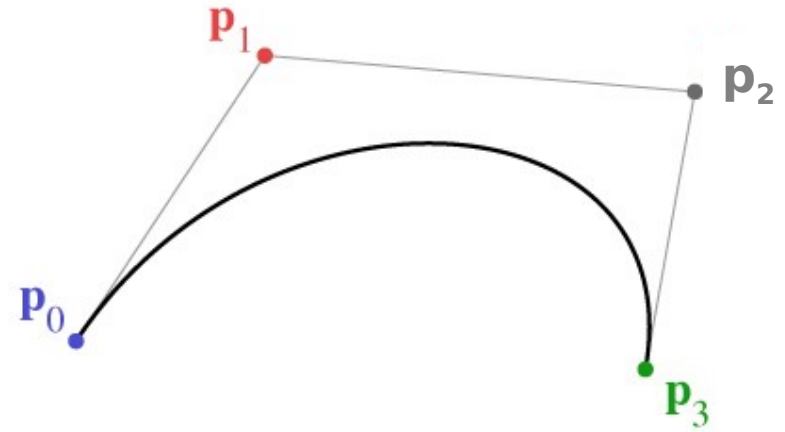
$$B_2^3(t) = 3t^2(1-t)$$

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

- 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

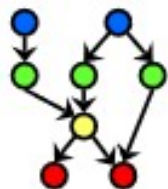


de Casteljau's Algorithm



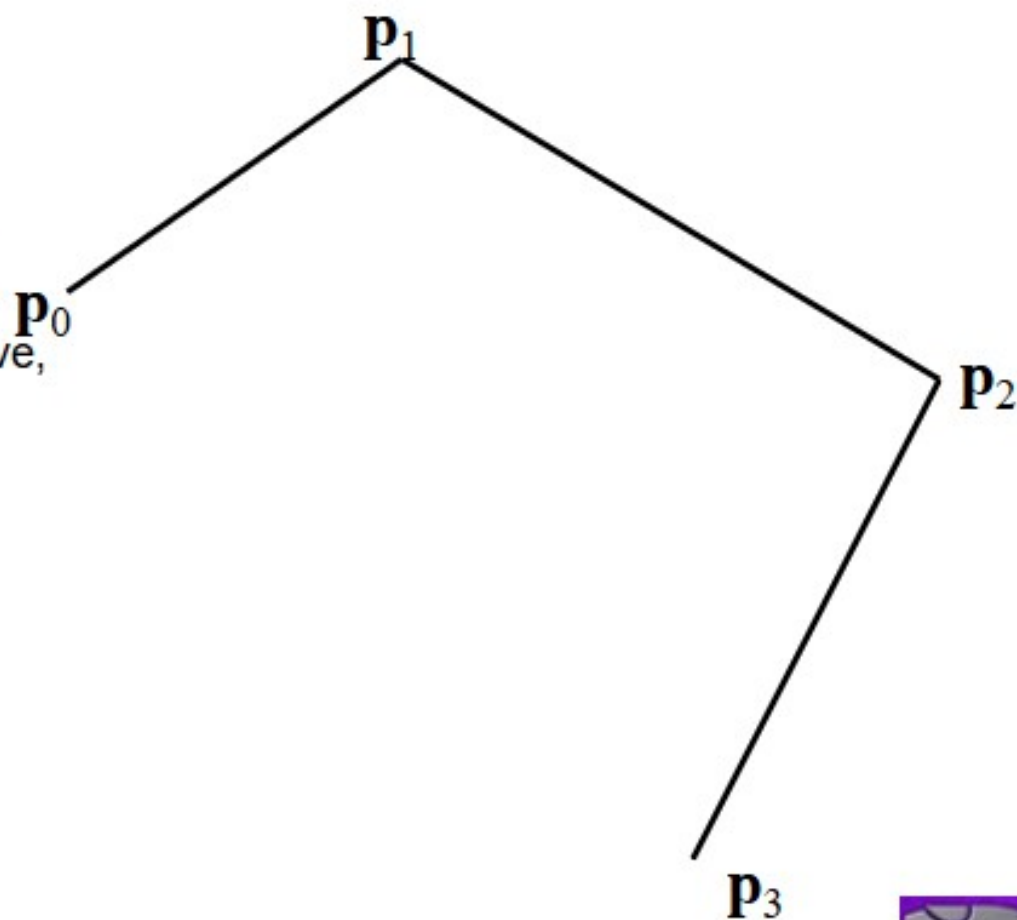
Paul de Casteljau
(1930-)
first developed the
'Bezier' curve using this
algorithm in 1959 while
working at Citroën, but
was not able to publish
them due to company
policy

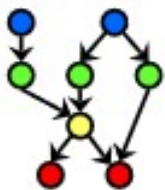
- Another method to compute Bezier curve



DE CASTELJAU ALGORITHM

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



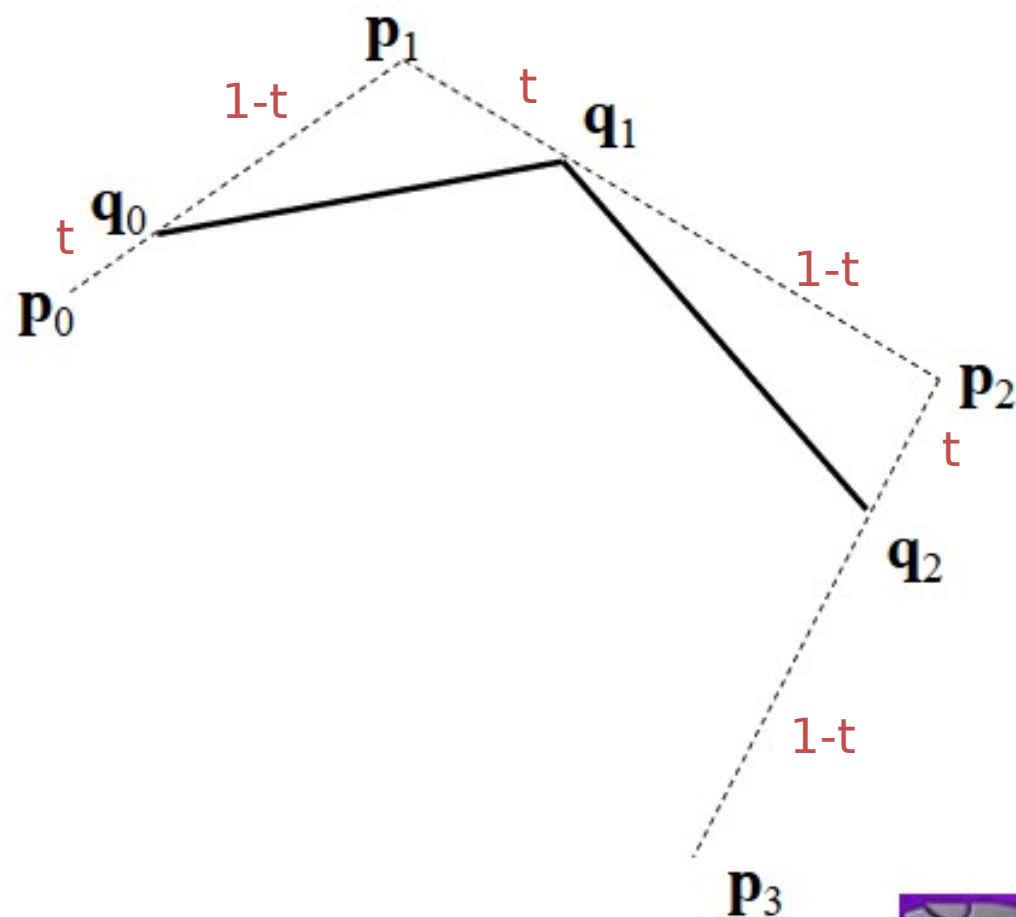


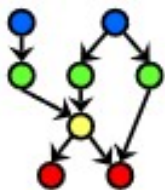
DE CASTELJAU ALGORITHM

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

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

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

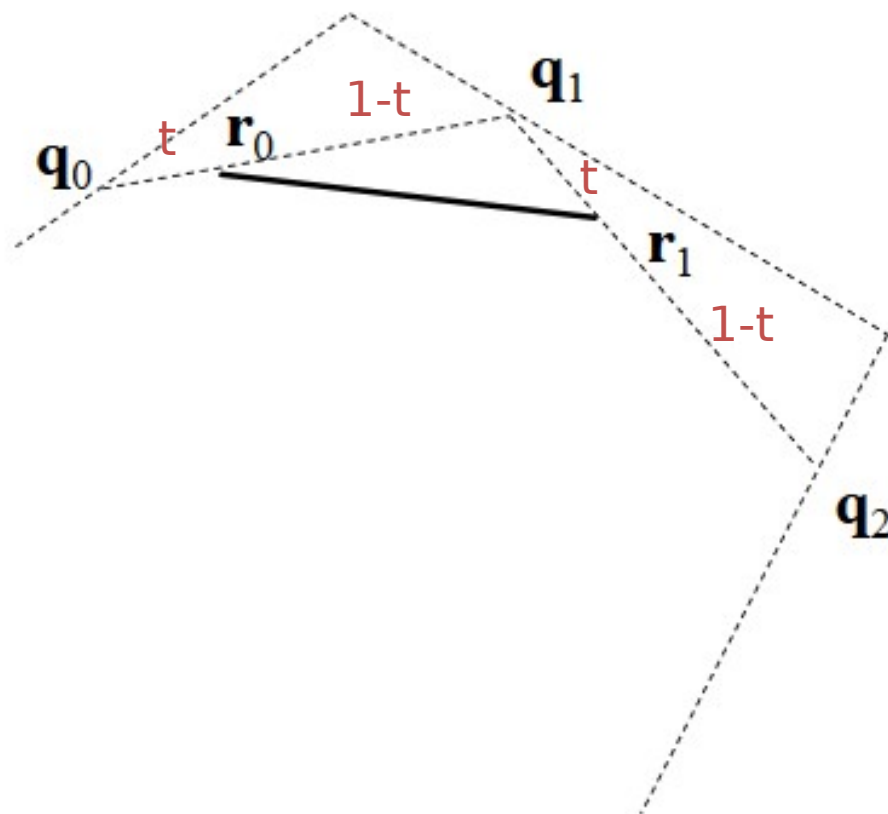


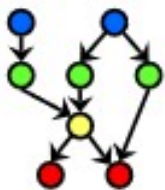


DE CASTELJAU ALGORITHM

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

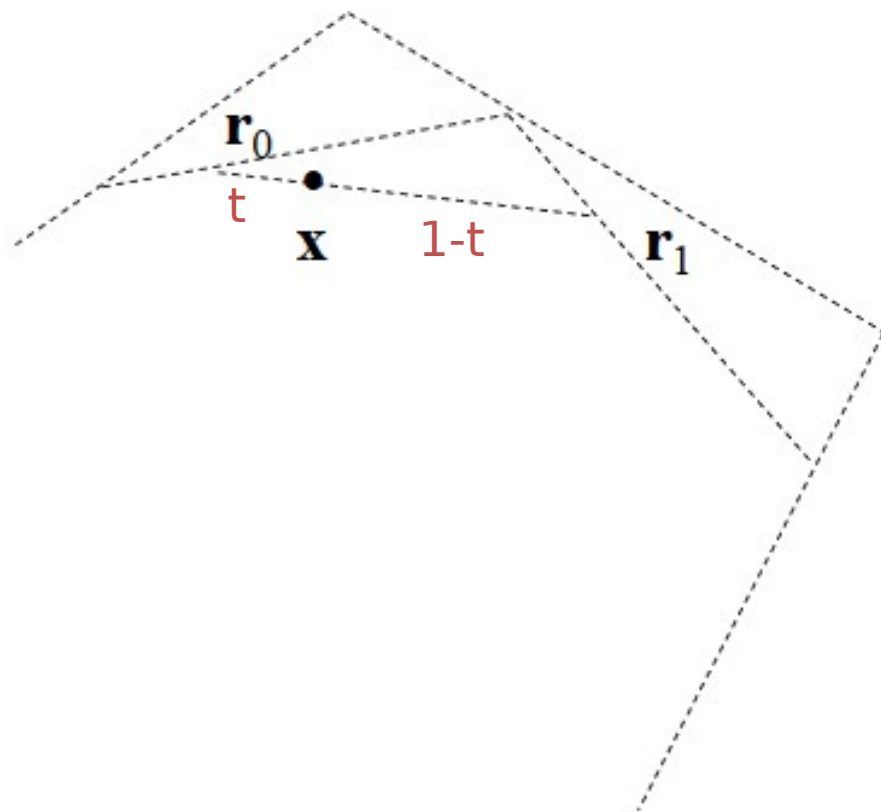
$$\mathbf{r}_1 = \text{Lerp}(t, \mathbf{q}_1, \mathbf{q}_2)$$



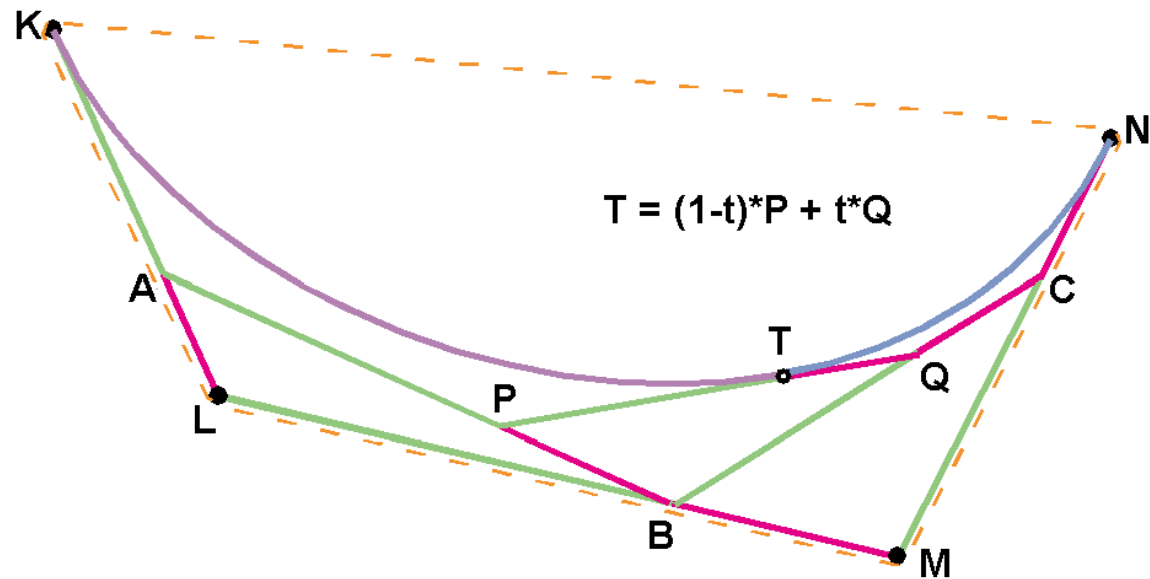


DE CASTELJAU ALGORITHM

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



de Casteljau's Algorithm



$$T = (1-t)*P + t*Q$$

$$P = (1-t)*A + t*B$$

$$Q = (1-t)*B + t*C$$

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

$$A = (1-t)*K + t*L$$

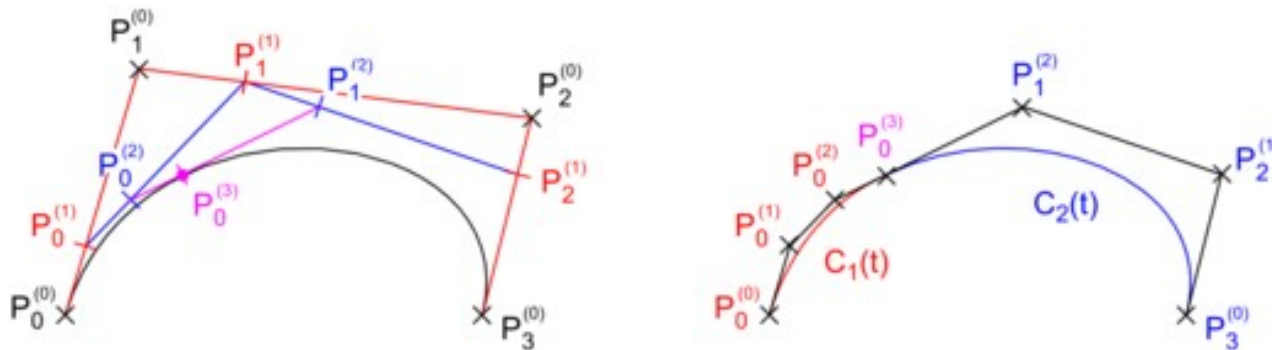
$$B = (1-t)*L + t*M$$

$$C = (1-t)*M + t*N$$

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

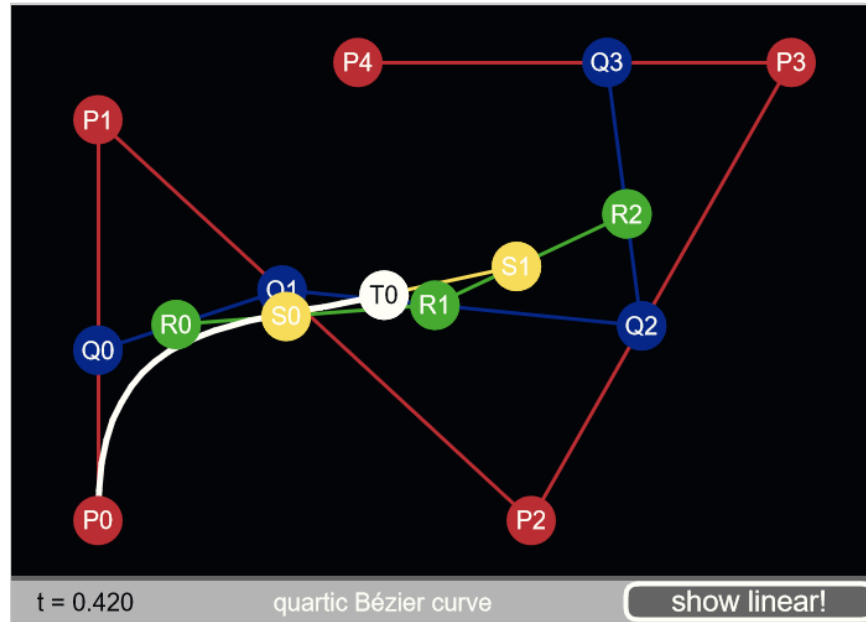
de Casteljau's Algorithm

- Nice recursive algorithm to compute a point on a Bezier curve
- Additionally, it subdivides a Bezier curve into two segments



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

[Demo] de Casteljau's Algorithm



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

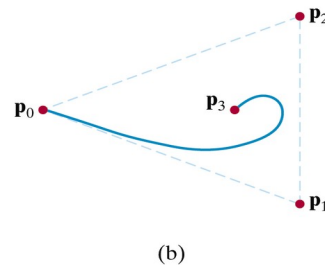
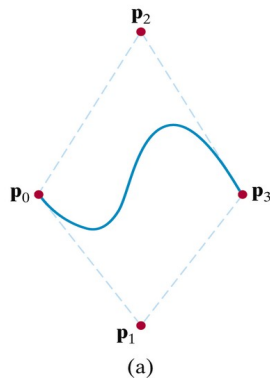
- Move red points
- Also check the subdivision demo

Displaying Curves

- To display a curve, compute a set of points on a curve and connecting the points with line segments.
- Brute-force
 - Evaluate $\mathbf{p}(t)$ for incrementally spaced values of t .
- Finite difference
 - Produces the same sampled points, but more efficiently.
 - Compute initial differences once and generate points using only additions.
 - See <http://www.drdoobs.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.

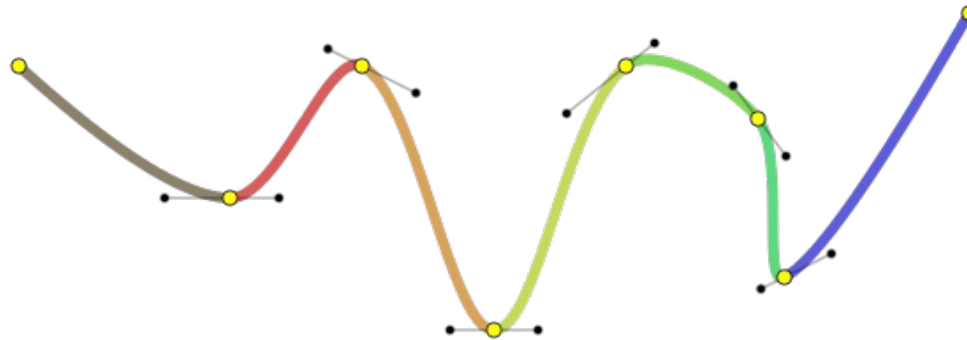
Quiz 3

- Go to <https://www.slido.com/>
- Join #cg-ys
- Click "Polls"
- Submit your answer in the following format:
 - **Student ID: Your answer**
 - e.g. **2024123456: 4.0**
- Note that your quiz answer must be submitted **in the above format** to be counted for attendance.

Spline Overview

Spline

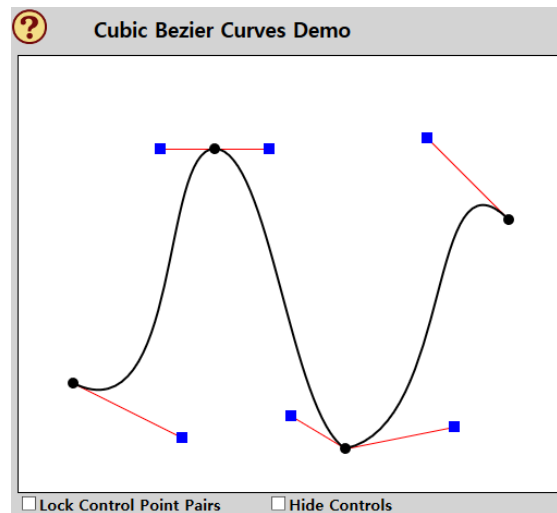
- Spline: ***piecewise polynomial***
 - A spline combines multiple polynomial segments to represent a longer smooth curve.



- 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?

Continuity

- Let's try another Bezier demo: Bezier spline



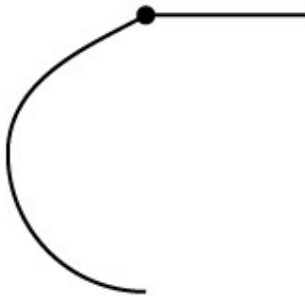
<http://math.hws.edu/graphicsbook/demos/c2/cubic-bezier.html>

- How to “smooth” the spline?

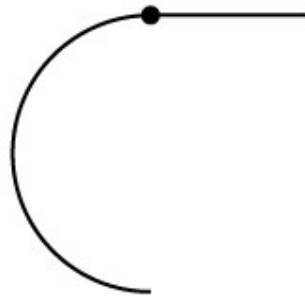
Continuity

- Smoothness can be described by degree of continuity
 - zero-order (C^0): position matches from both sides
 - first-order (C^1): position and 1st derivative (velocity) match from both sides
 - second-order (C^2): position and 1st & 2nd derivatives (velocity & acceleration) match from both sides

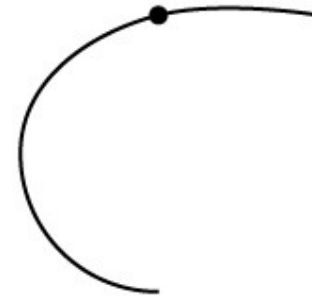
zero order



first order

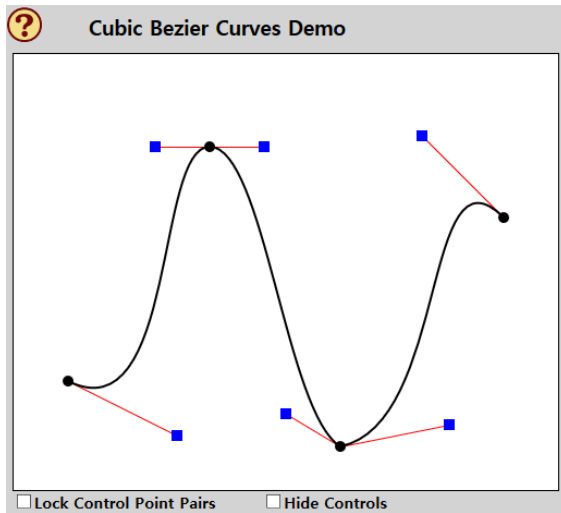


second order



Control

- Let's say you want to make a specific shape using these two curves. Which one is more controllable?

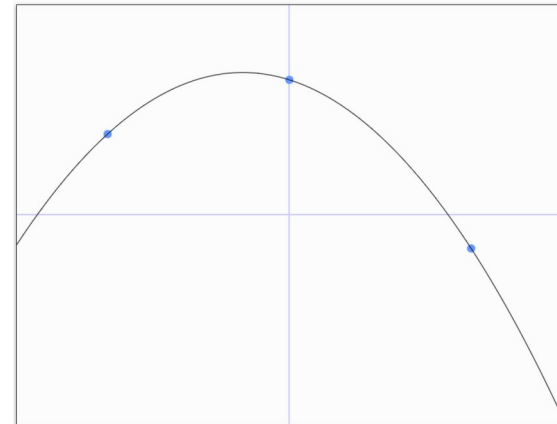


<http://math.hws.edu/graphicsbook/demos/c2/cubic-bezier.html>

Interpolation Polynomial

Click and drag the **control points** and the polynomial curve will interpolate to satisfy them.

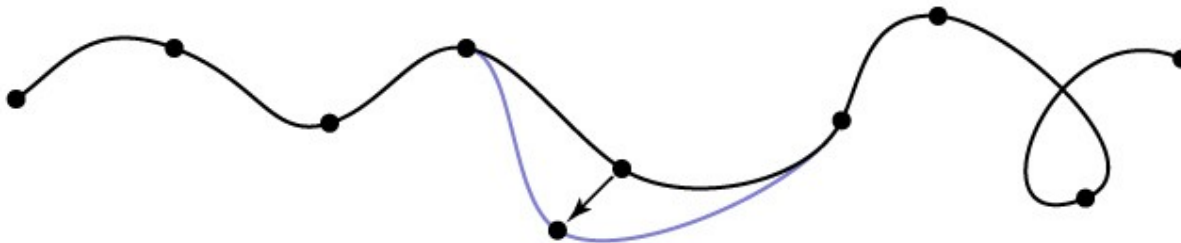
Polynomial Degree: (parabola)



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

Control

- Local control
 - changing control point only affects a **limited part** of spline
 - without this, splines are very difficult to use
 - many likely formulations lack this
 - natural spline
 - polynomial fits



Interpolation / Approximation

- Interpolation: passes through points



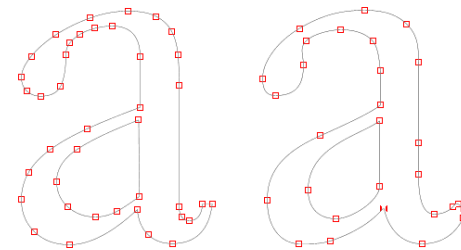
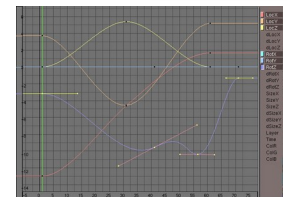
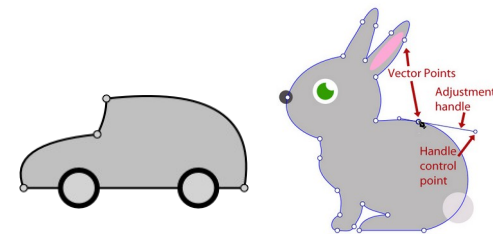
- Approximation: guided by points



- Interpolation properties are preferable, but not mandatory.

Bezier Spline

- Continuity: can be C^0 or C^1
- Local controllability
- Interpolation: pass through only the two end points
- Bezier spline is very widely used:
 - 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

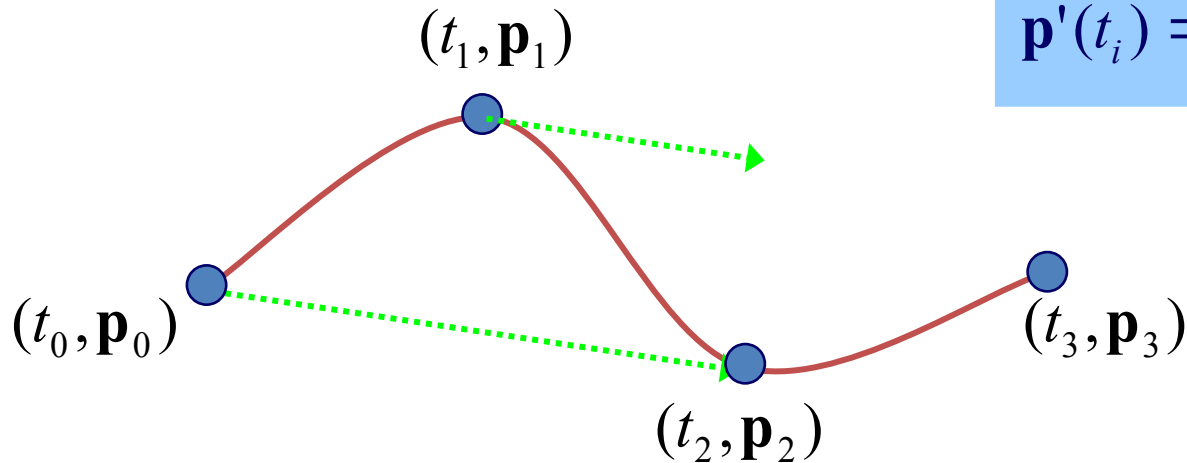


True Type Font

Postscript Font

Catmull-Rom Spline

- One Hermite curve between two consecutive control points.
- Define end point derivatives using adjacent control points.

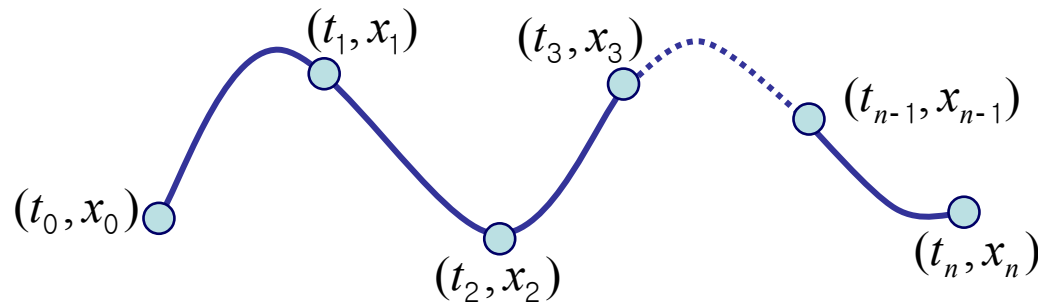


$$\mathbf{p}'(t_i) = \frac{\mathbf{p}_{i+1} - \mathbf{p}_{i-1}}{2}$$

- C^1 continuity, local controllability, interpolation

Natural Cubic Splines

- We want to achieve higher continuity (at least C^2)
- $4n$ unknowns
 - coefficients of n cubic polynomial segments
- $4n$ equations
 - $2n$ equations for end point interpolation
 - $(n-1)$ equations for tangential continuity
 - $(n-1)$ equations for second derivative continuity
 - 2 equations: $x''(t_0) = x''(t_n) = 0$



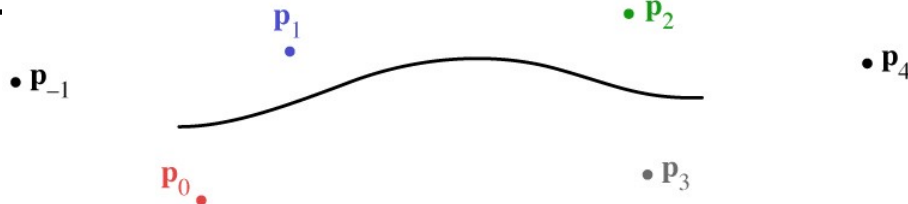
- C^2 continuity, no local controllability, interpolation

B-splines (brief intro)

- Use 4 points, but approximate only middle two



- Draw curve with overlapping segments
 - 0-1-2-3 1-2-3-4 2-3-4-5 3-4-5-6, etc



- C^2 continuity, local controllability,

Lab Session

- Now, let's start the lab today.