NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

Module 3:
Bezier Techniques

Learning Objectives

- Understand the definition of Bezier curves
- Understand what make Bezier curves popular
- Learn typical algorithms of Bezier curves

NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

Sources

- Textbook (Chapter 3.5: Bezier curves)

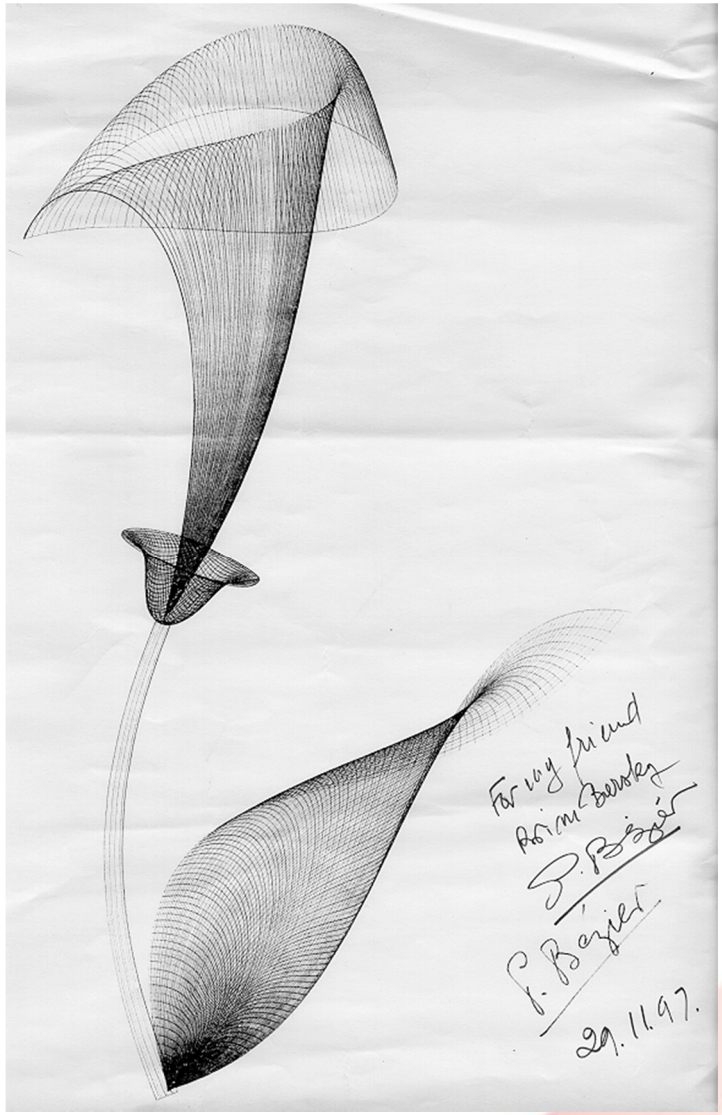
- Joy's On-Line Geometric Modeling Notes (Bernstein polynomial, Bezier curves and patches)

  http://graphics.idav.ucdavis.edu/education/CAGDNotes/homepage.html

- Shene's Computing with Geometry Notes (Unit 5. Bezier curves)

  http://www.cs.mtu.edu/~shene/COURSES/cs3621/NOTES/notes.html

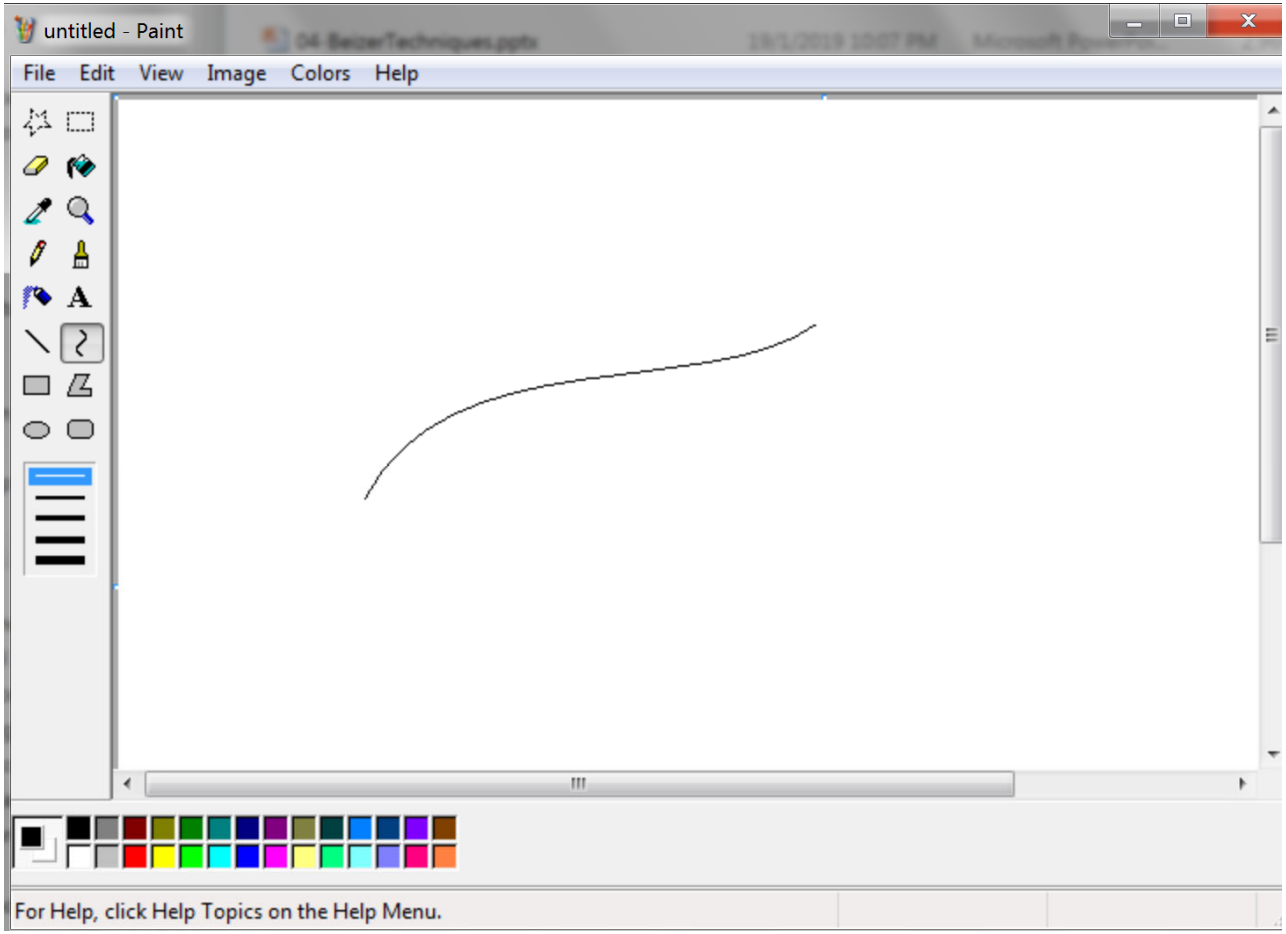
Outline



NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

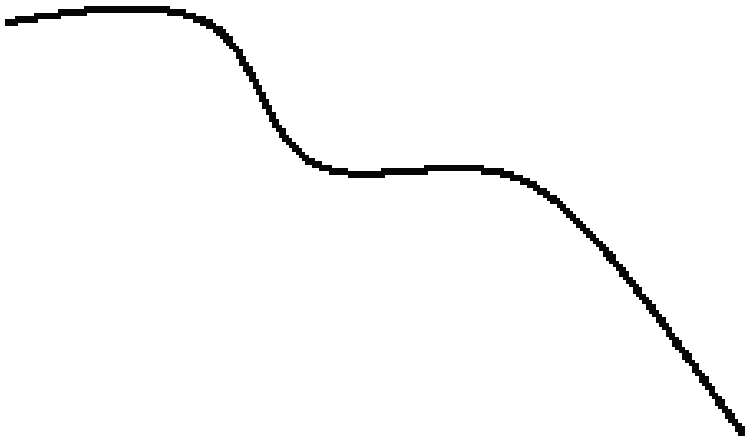
Introduction

- MS-Paint



Introduction

- Problem: How can we efficiently and effectively design, represent and manipulate curves that can be used to interpolate or fit data points?
  - Flexible model
  - Easy to compute
  - Intuitive, user friendly interface is possible



- Our choice:
  - Parametric
    - Polynomial
      - Bezier

Why parametric curves?

- A curve can be approximated by a polygon/polyline.
  - doesn't work well when scaled
  - needs lots of points to make the curve look smooth
  - inconvenient to have to specify so many points
- A curve can be represented implicitly: $f(x,y) = 0$.
  - difficult to plot
- A parametric curve can tell where the fly is at time t. It describes the motion of a point through space at time t.

NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

Parametric polynomial curves

- The functions x(t) and y(t) are both polynomials.

$$x = x(t) = a_0 + a_1 t + \cdots + a_n t^n$$
$$y = y(t) = b_0 + b_1 t + \cdots + b_n t^n$$

or in vector form

$$P(t) = P_0 + P_1 t + \cdots + P_n t^n, \quad \text{where } P_i = \begin{pmatrix} a_i \\ b_i \end{pmatrix}$$

- Eg. cubic

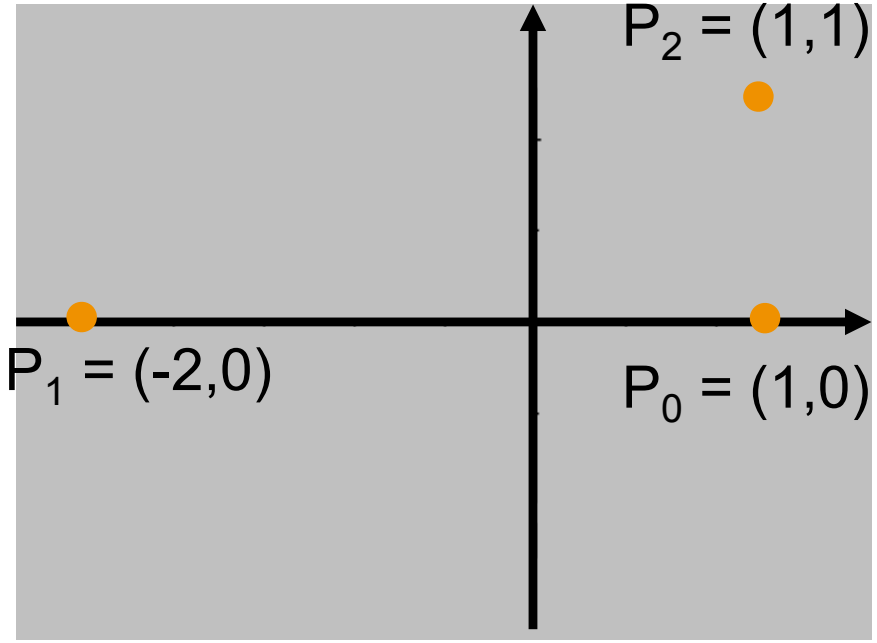
$$P(t) = \begin{pmatrix} 1 \\ 2 \end{pmatrix} + \begin{pmatrix} 1 \\ -1 \end{pmatrix} t + \begin{pmatrix} 2 \\ 0 \end{pmatrix} t^2 + \begin{pmatrix} 3 \\ 2 \end{pmatrix} t^3 = \begin{pmatrix} 1 + t + 2t^2 + 3t^3 \\ 2 - t + 2t^3 \end{pmatrix}$$

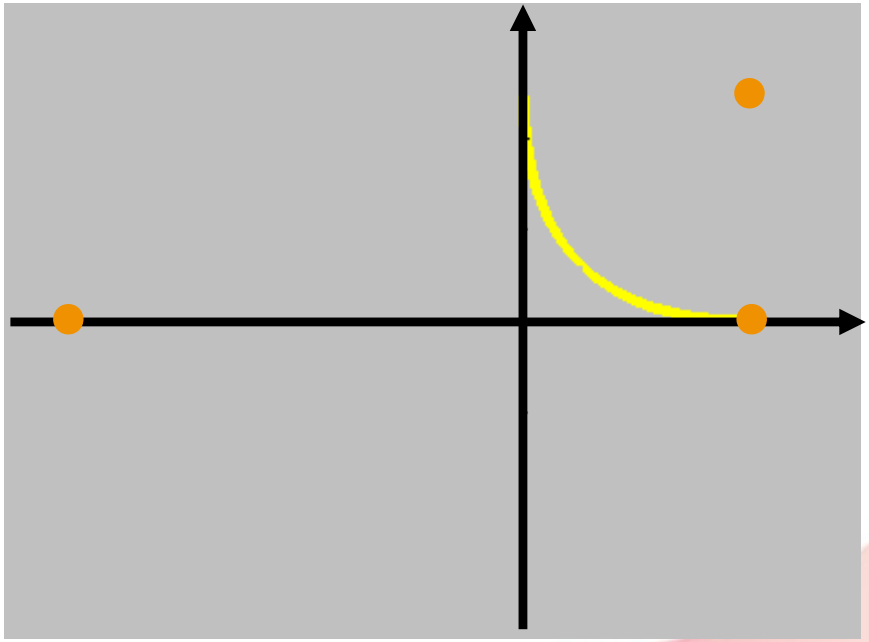
- Polynomial curves have advantages: easy to compute, differentiable, …

What's wrong with standard polynomials for design?

$$P(t) = P_0 + P_1 t + P_2 t^2, \quad t \in [0,1]$$

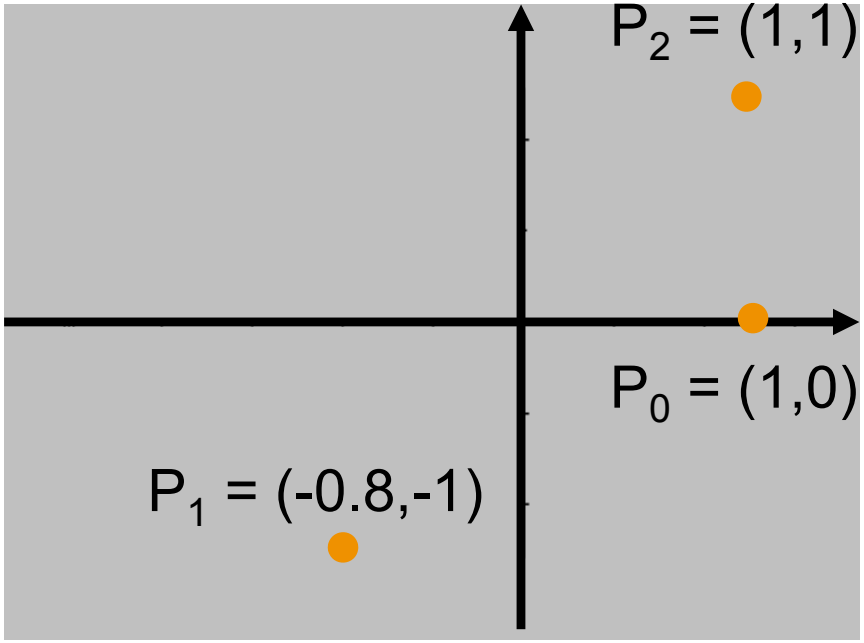
where P_0 = (1,0), P_1= (-2,0), P_2 = (1,1)

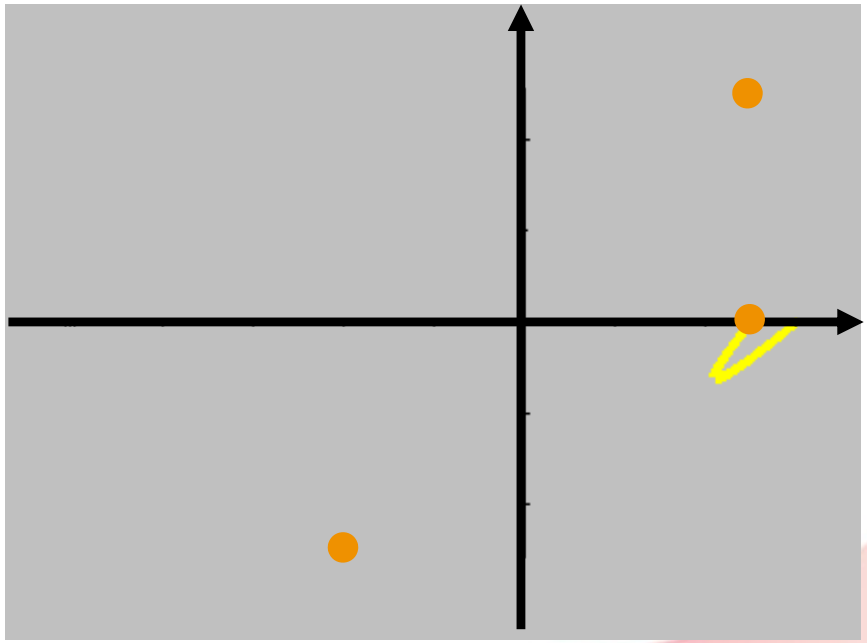




What's wrong with standard polynomials for design?

- When P_1 is changed into P_1 = (-0.8,-1), how about the curve?



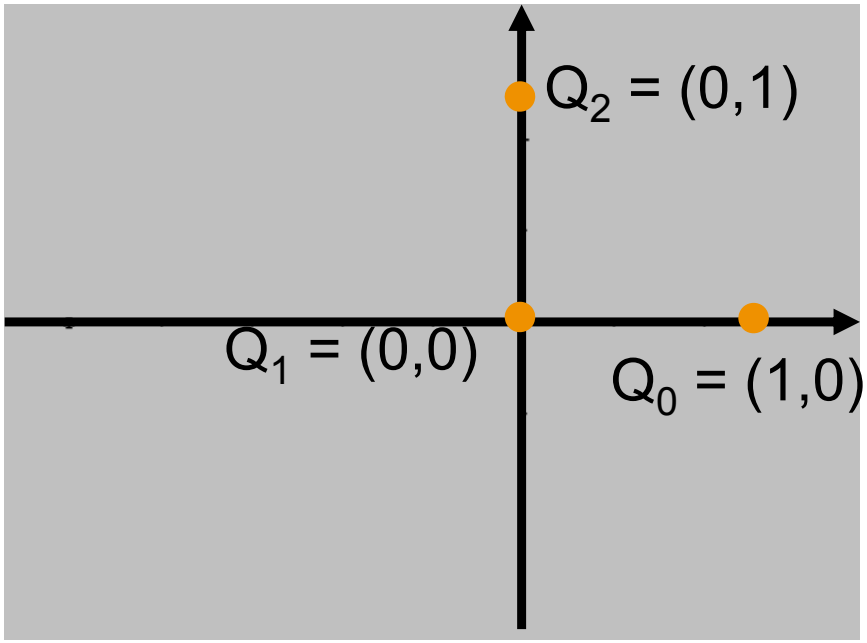


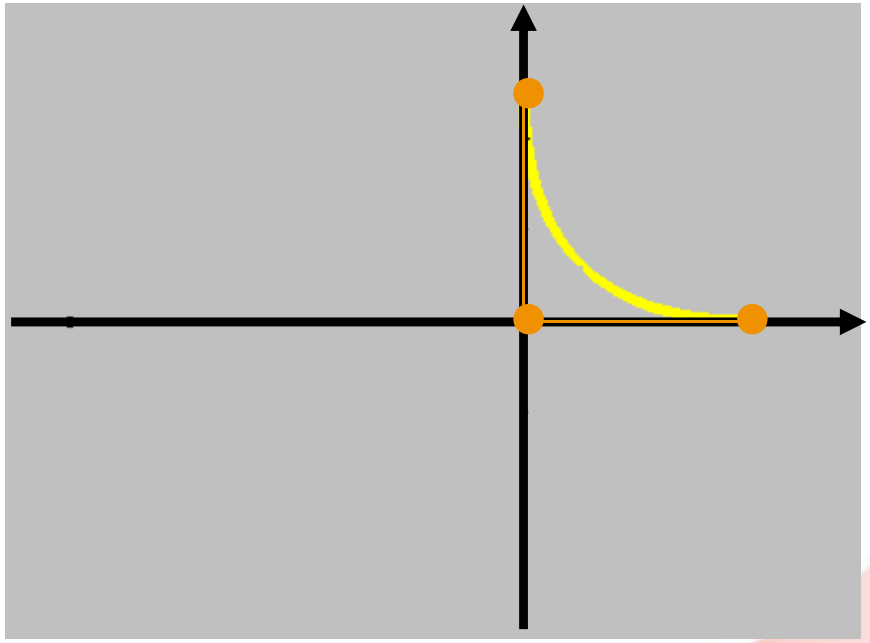
NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

Bezier's solution

Assume $P_0 = (1,0)$, $P_1 = (-2,0)$, $P_2 = (1,1)$. We rewrite

$$P(t) = P_0 + P_1 t + P_2 t^2 = \left(1 - 2t + t^2, t^2\right)$$
$$= (1,0)(1-t)^2 + (0,0)\,2(1-t)t + (0,1)\,t^2$$



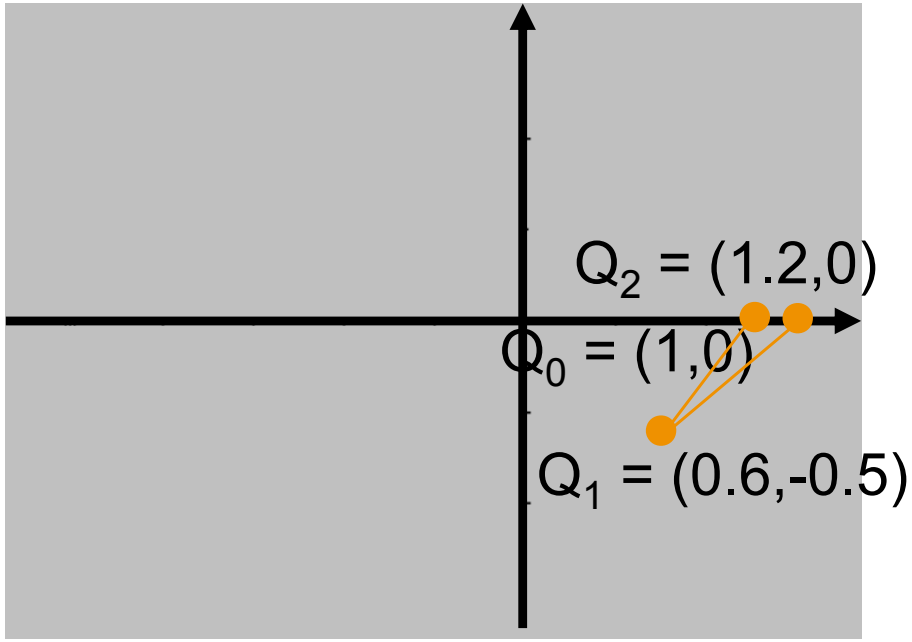


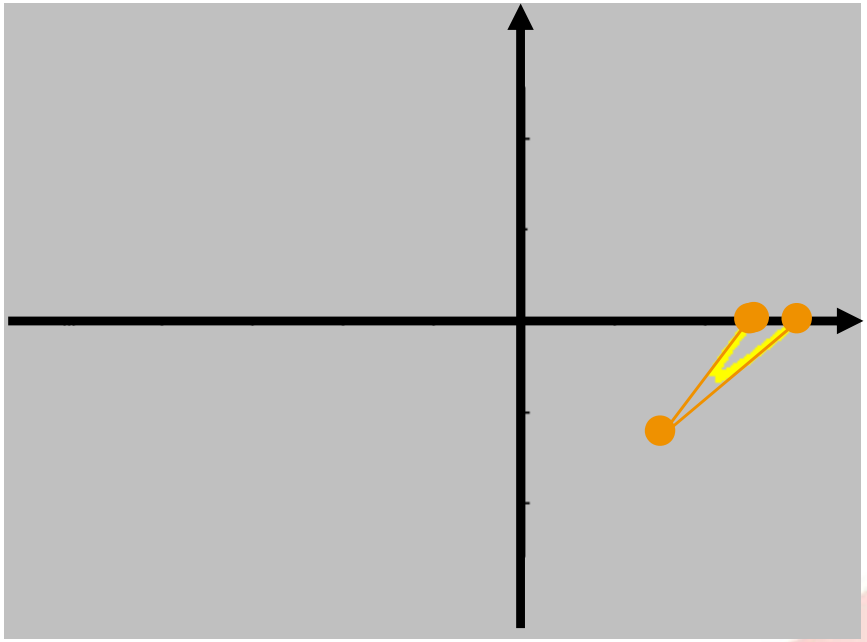
NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

Bezier's solution

Assume P_0 = (1,0), P_1= (-0.8,-1), P_2 = (1,1). We rewrite

$$P(t) = P_0 + P_1 t + P_2 t^2 = \left(1 - 0.8t + t^2, -t + t^2\right)$$
$$= (1,0)(1-t)^2 + (0.6,-0.5)\,2(1-t)t + (1.2,0)\,t^2$$





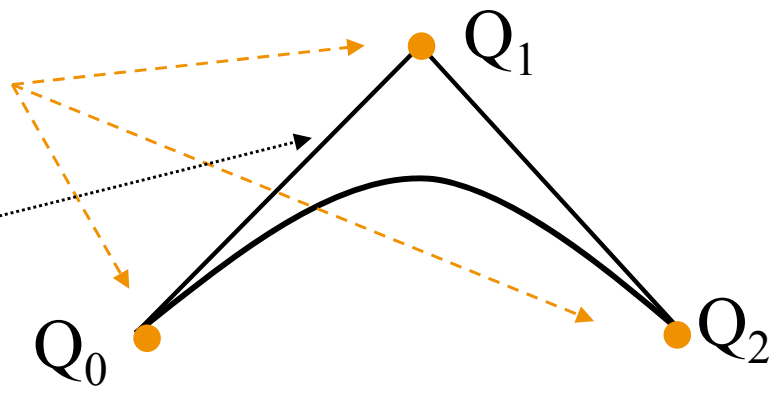
NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

Bezier's solution

- In the previous example, Bezier's solution uses { (1-t)2, 2(1-t)t, t2 } (instead of the power bases { 1, t, t2 }) as the basis functions, which makes the coefficients mean something. What?

Coefficients ≡ "**Control points**"

Polygon ≡ "**Control polygon**"



- As a result, this model provides an intuitive interface for users to control / edit the shape.

History

- Bézier curves and surfaces were developed in the 1960s, independently, by Pierre Bézier and Paul de Casteljau. These curves and surfaces were initially used as the basic form in computer modeling to define car bodies in auto-manufacturing.
  - Pierre Bezier: (1 Sep 1910 – 25 Nov 1999)
    a French engineer at Renault

  - Paul de Casteljau: (19 Nov 1930 –)
    a French engineer at Citroen

NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

History

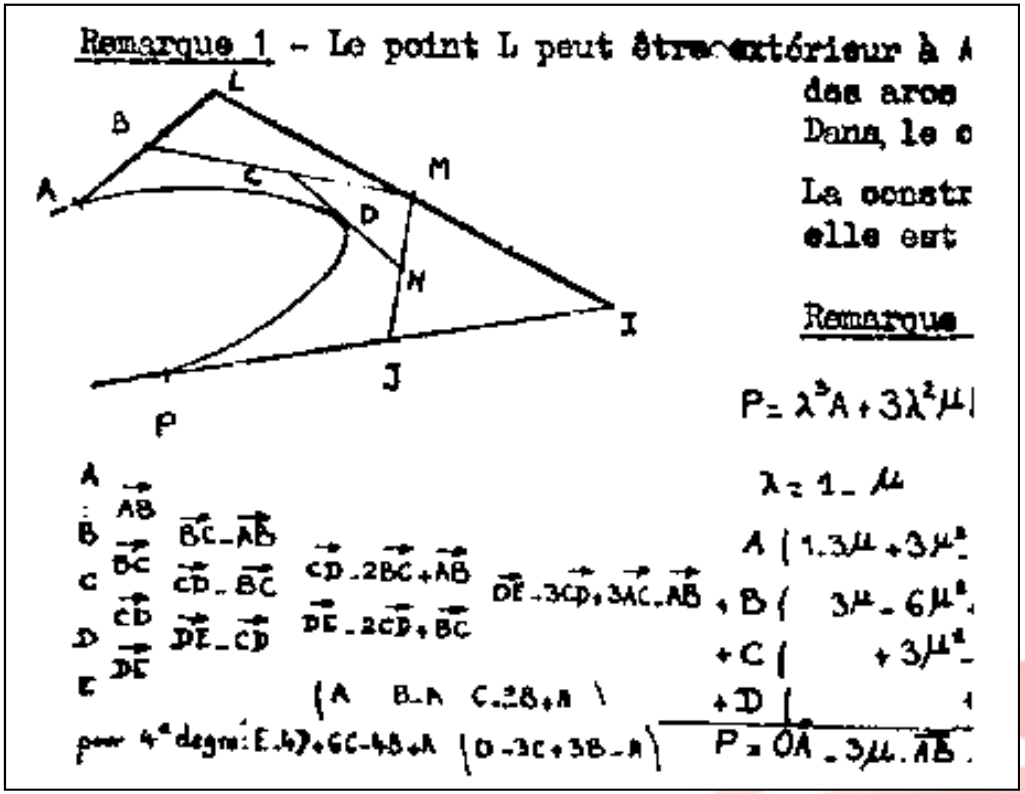
The scheme originally proposed by Bezier is:

$$P(t) = \sum_{j=0}^{n} A_j^n(t) a_j \qquad 0 \le t \le 1,,$$

$$A_0^n(t) = 1,\ A_j^n(t) = \frac{(-t)^j}{(j-1)!}\frac{d^{j-1}}{dt^{j-1}}\frac{(1-t)^{n-1}-1}{t}, \quad j = 1,2,\cdots,n,$$

$$a_0 = P_0,\ a_j = P_j - P_{j-1}, \qquad j = 1,2,\cdots,n.$$

De Casteljau's description is:



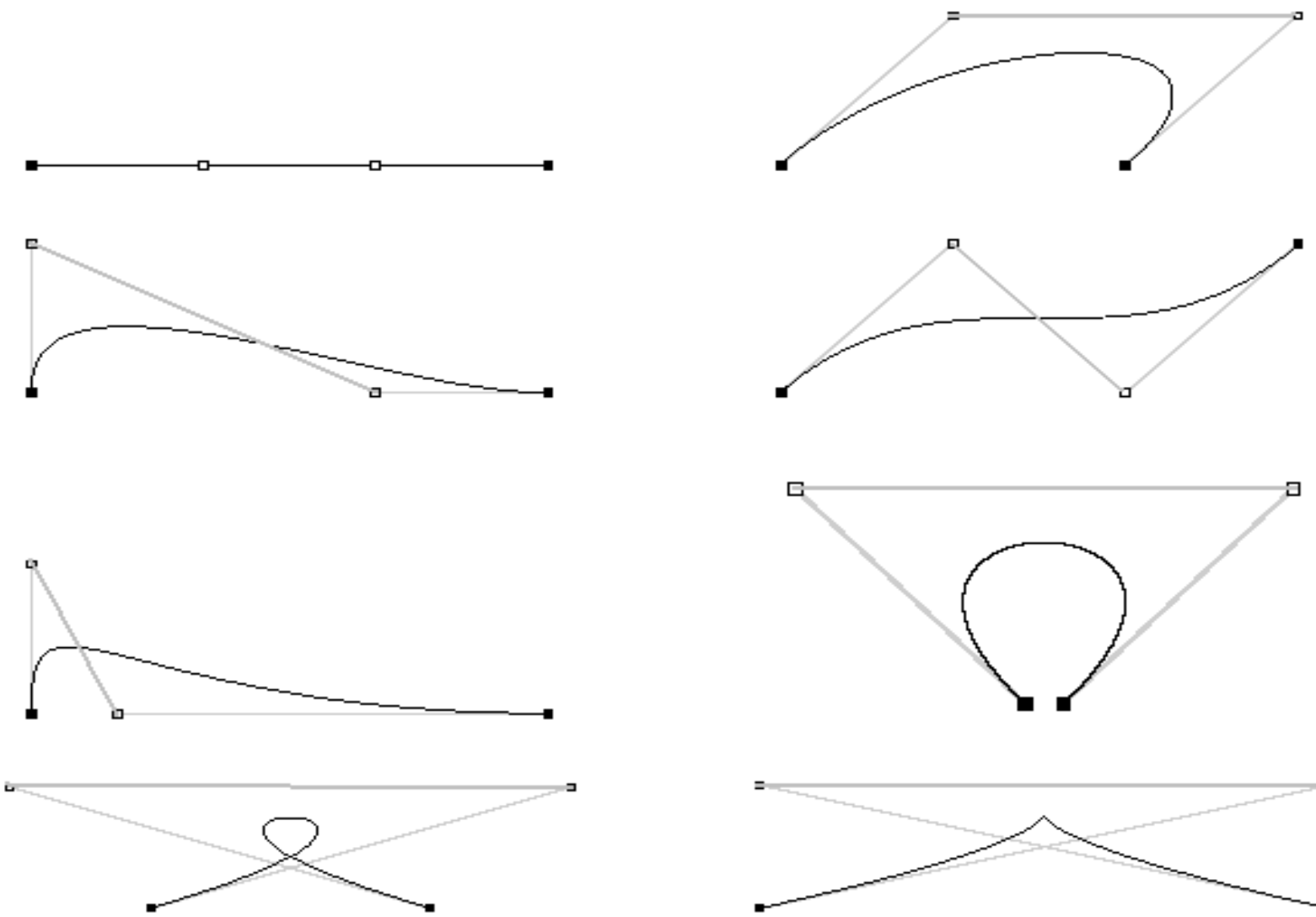
Outline

Bezier curve

- A Bezier curve is defined by a polyline
  - called a control polygon, vertices called control points
  - the control points are ordered to form a polyline
- Any number of the control points may be used, but 4 and 3 are most popular.
  - order of the curve = number of control points
  - degree of the curve = order of the curve - 1
- The Bézier curve approximates the polyline. It passes through its first and last control points, but, in general, no others.

Demo: Bezier curves

Examples: cubic Bezier curves



Mathematical formulation

Given a set of ordered points P_0, P_1, …, P_n, the degree n Bezier curve is a curve segment defined by parametric equation:

$$P(t) = P_0 B_0^n(t) + P_1 B_1^n(t) + \cdots + P_n B_n^n(t) = \sum_{i=0}^{n} P_i B_i^n(t), \quad t \in [0,1]$$

where the *basis* functions or *blending* functions

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

are the Bernstein polynomials, satisfying $\sum_{i=0}^{n} B_i^n(t) \equiv 1$.

If you consider $B_i^n(t)$ is a variable mass assigned to point P_i, then the above formula defines the mass center of these points at time t. As t varies between 0 and 1, a curve is swept out by the center of masses. This curve is the Bezier curve.

Explicit form for lower degrees

- We can say a Bezier curve is a linear combination of the basis functions:
  - Linear (n=1):

$$P(t) = P_0 B_0^1(t) + P_1 B_1^1(t) = P_0(1-t) + P_1 t$$

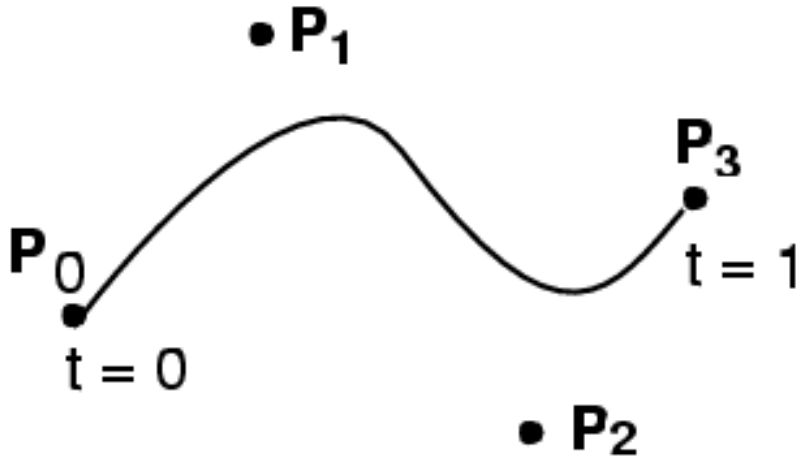
  - Quadratic (n=2):

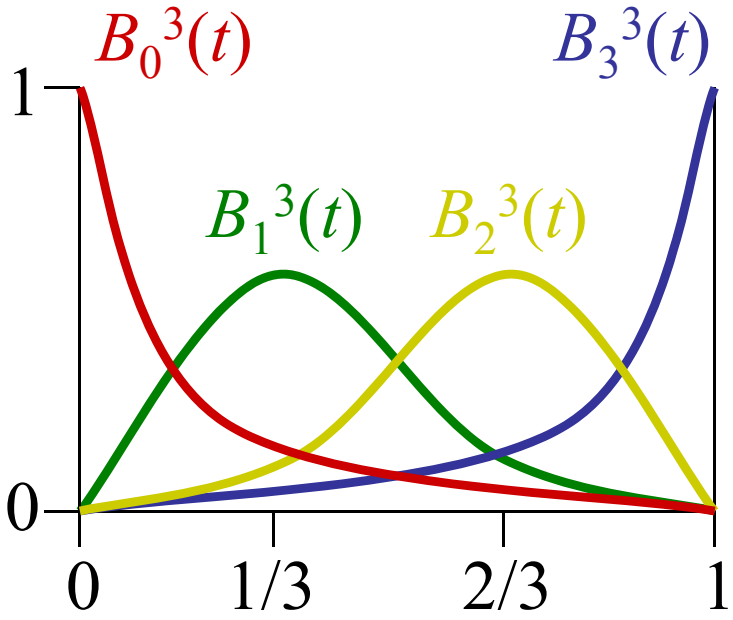
$$P(t) = P_0 B_0^2(t) + P_1 B_1^2(t) + P_2 B_2^2(t)$$
$$= P_0(1-t)^2 + P_1 2(1-t)t + P_2 t^2$$

  - Cubic (n=3):

$$P(t) = P_0 B_0^3(t) + P_1 B_1^3(t) + P_2 B_2^3(t) + P_3 B_3^3(t)$$
$$= P_0(1-t)^3 + P_1 3(1-t)^2 t + P_2 3(1-t)t^2 + P_3 t^3$$

Example: cubic Bezier curve





$$Q(t) = P_0(1-t)^3 + P_1 3(1-t)^2 t + P_2 3(1-t)t^2 + P_3 t^3$$

What're the Bernstein polynomials?

Recall

$$((1-t)+t)^n = \binom{n}{0}(1-t)^n t^0 + \binom{n}{1}(1-t)^{n-1}t^1 + \cdots + \binom{n}{n}(1-t)^0 t^n$$

- the Beinstein polynomials $B_i^n(t) = \binom{n}{i}(1-t)^{n-i}t^i$ are the terms of the expansion of $((1-t)+t)^n$.
- partition of unity: $\sum_{i=0}^{n} B_i^n(t) \equiv 1.$
- binomial coefficient ("*n*-choose-*i*"):

$$\binom{n}{i} = \frac{n!}{(n-i)!i!} = \frac{n(n-1)(n-2)\cdots 2\cdot 1}{[(n-i)(n-i-1)\cdots 1]\cdot[i(i-1)\cdots 1]}$$

- lerps of lower degrees: $B_i^n(t) = (1\text{-}t)B_i^{n\text{-}1}(t) + tB_{i\text{-}1}^{n\text{-}1}(t)$

NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

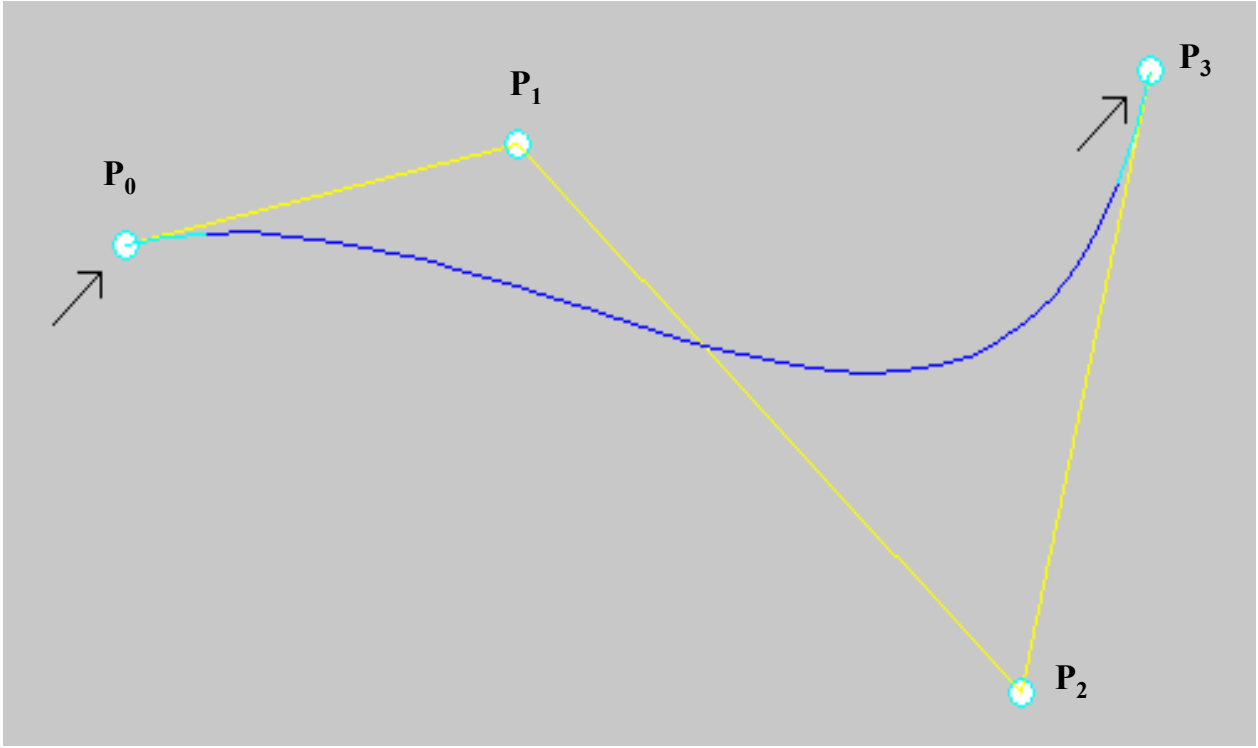
Outline

Properties

- Endpoint interpolation
- Co-tangency
- Affine invariance
- Convex hull
- Linear precision
- Variation diminishing

Endpoint interpolation

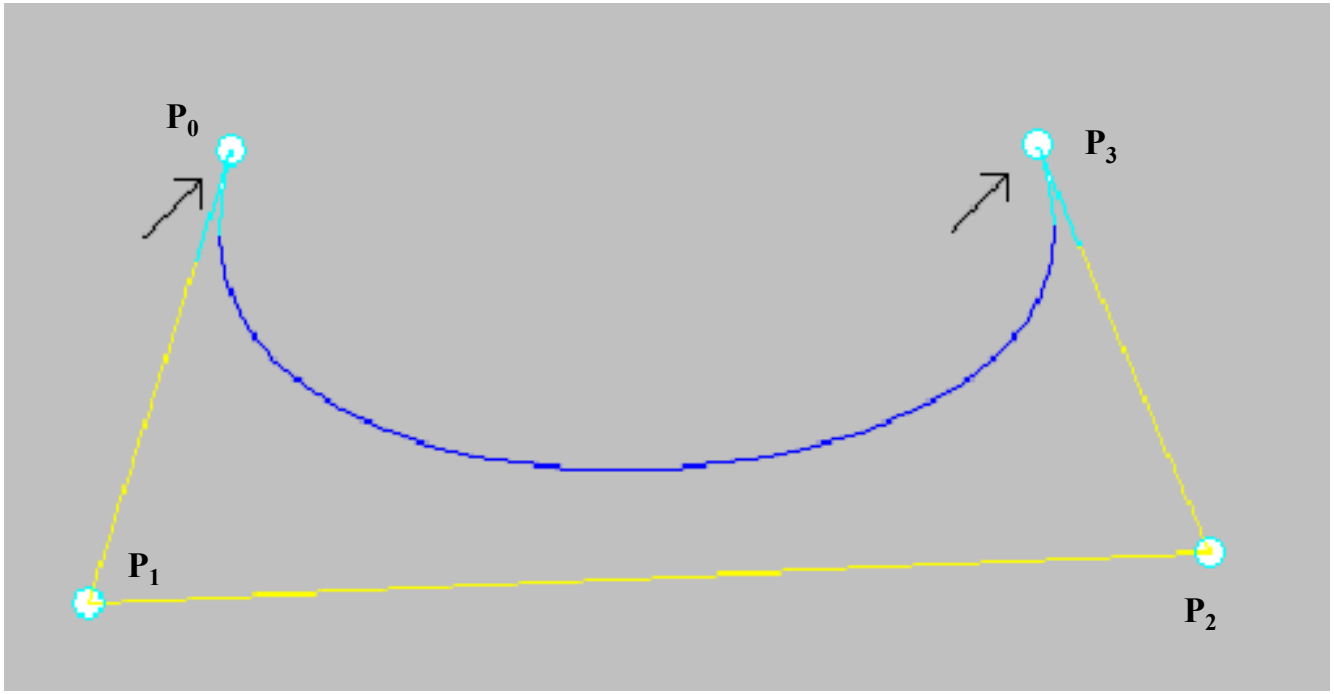
- The Bezier curve always passes through the end points of the control polygon.



- Why? $P(0) = P_0, \; P(1) = P_n$

Co-tangency

- The ends of the Bezier curve are always tangential to the first and last legs of the control polygon.



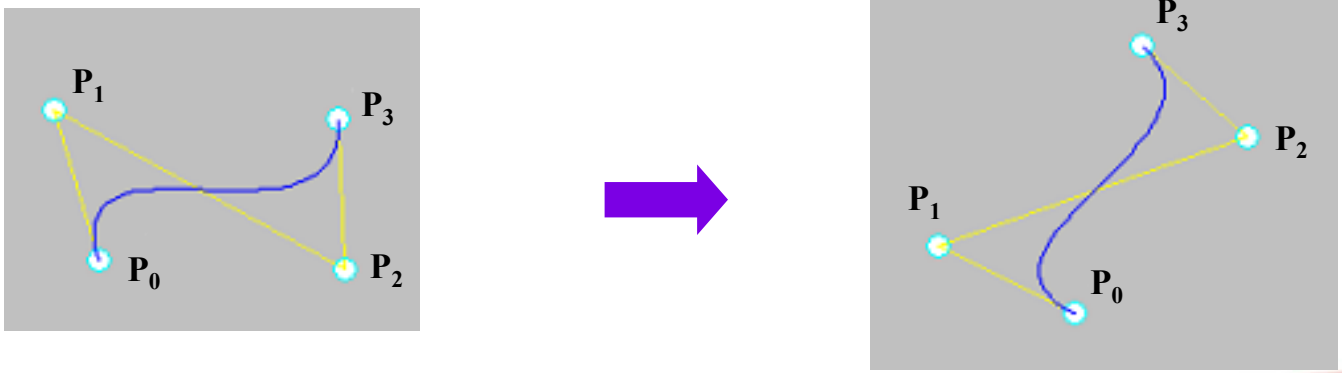
- Why?

$$P'(0) = n(P_1 - P_0), \quad P'(1) = n(P_n - P_{(n-1)})$$

NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

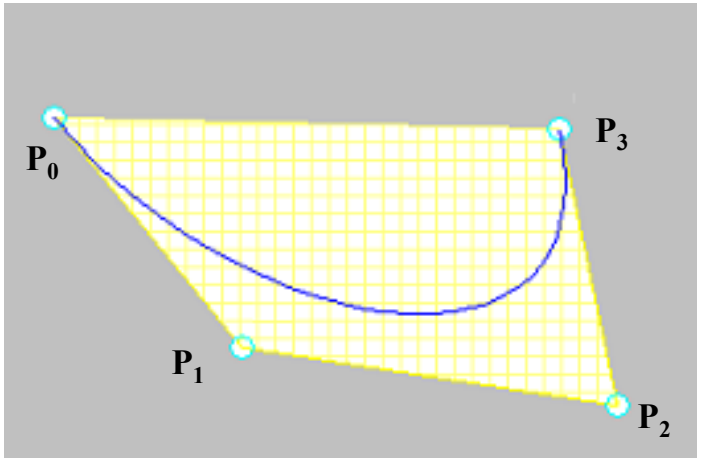
Affine invariance

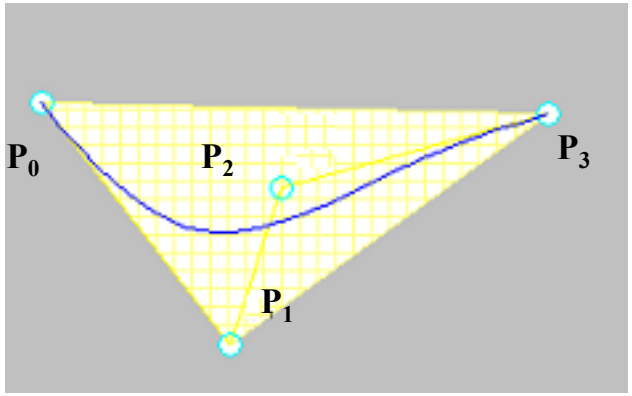
- Any linear transformation (translation, rotation, scaling) of the control points defines a new curve that is just the transformation of the original curve.
- Implication: If the control polygon is rotated or translated, then the shape of the curve will not change



Convex hull

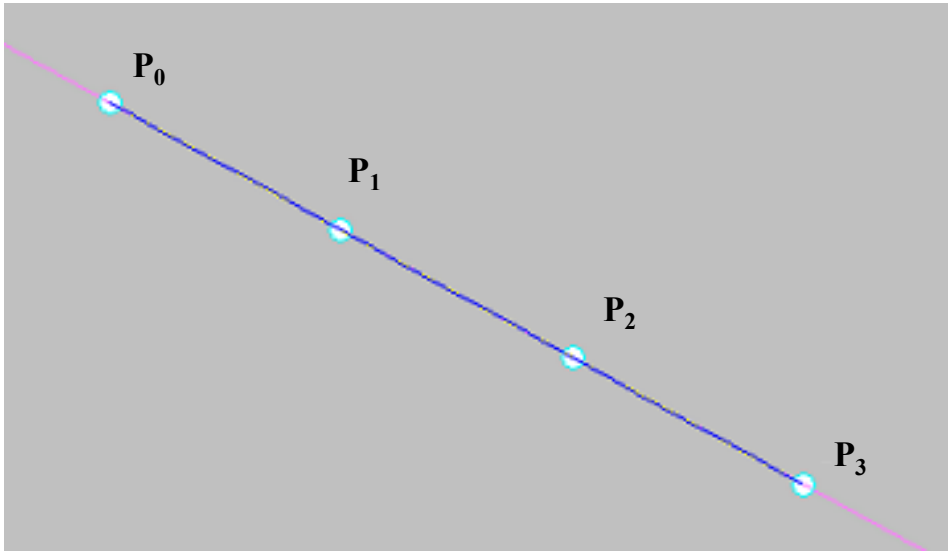
- The Bezier curve always remains within the convex hull of its control polygon.
- The convex hull is the tightest convex polygon that completely surrounds the control points.





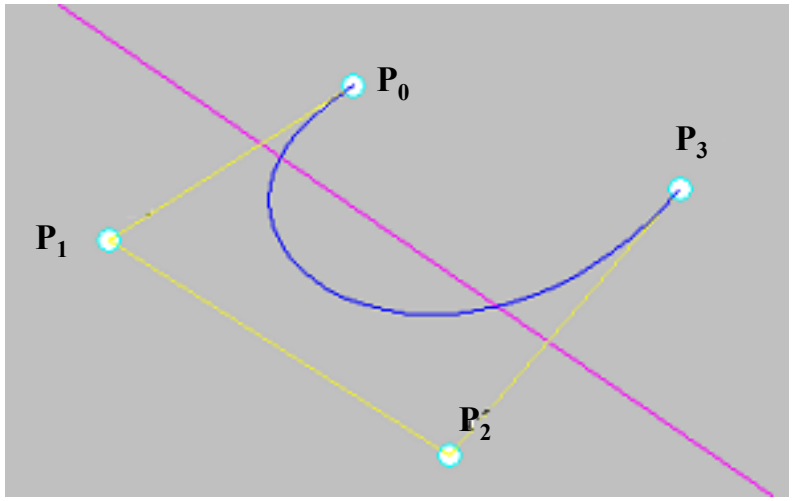
Linear precision

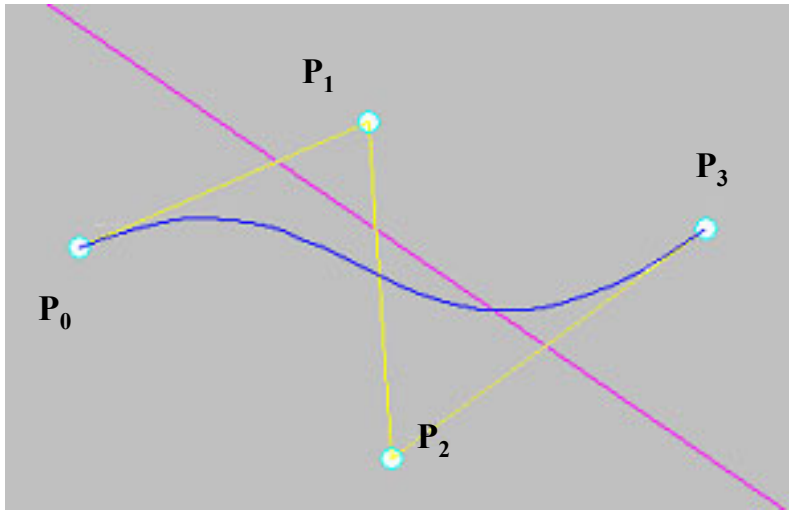
- If the control points of the Bezier curve lie upon a straight line and are equally spaced along the line, then the curve is the line segment connecting the two end points.



Variation diminishing

- The Bezier curve does not intersect an arbitrary line more times than the control polygon.
- Implication: The Bezier curve does not “wiggle” any more than the control polygon.

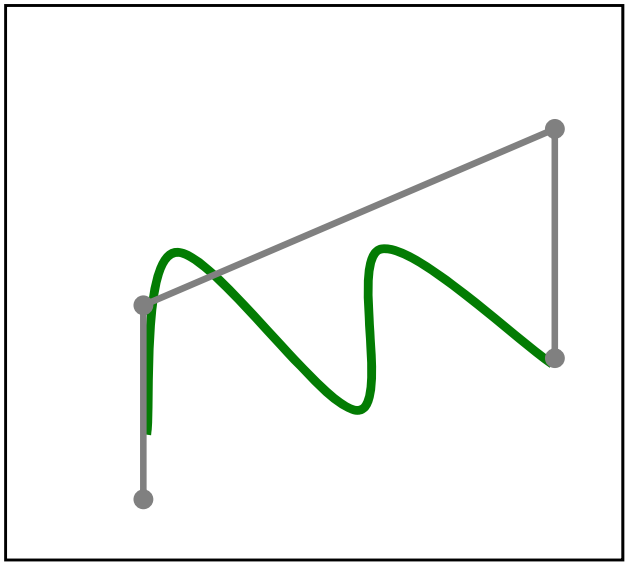




NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

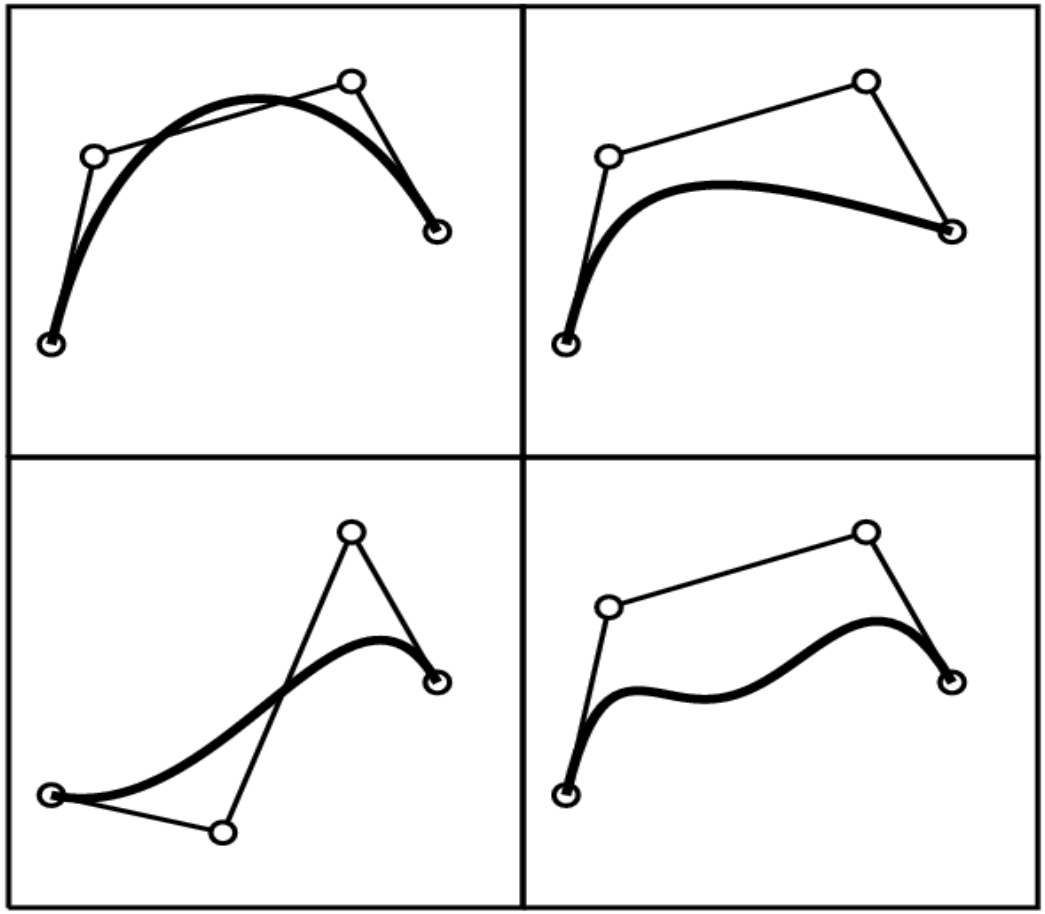
Question for you

Refer to the figure. Someone draws the green curve as the Bezier curve defined by the black control polygon. Please show the mistakes in this drawing.



Question for you

- Which of the curves MUST not be the cubic Bezier curve defined by the respective control polygon?



NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

Outline

Algorithms

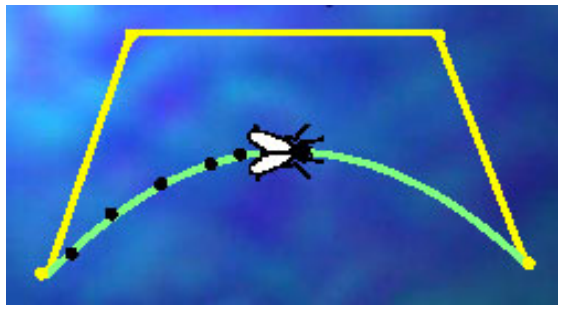
- de Casteljau algorithm
- Subdivision algorithm
- Degree elevation
- Differentiation
- Tessellation

NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

4.1 de Casteljau algorithm

- Problem: where is the fly at time *t*?

  Mathematically, how can we evaluate the point on the Bezier curve corresponding to parameter *t* ?



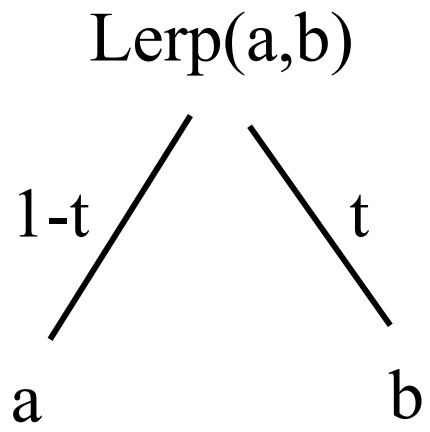
- Brute force solution: directly substitute *t* into the formula to evaluate *P(t).* For example,

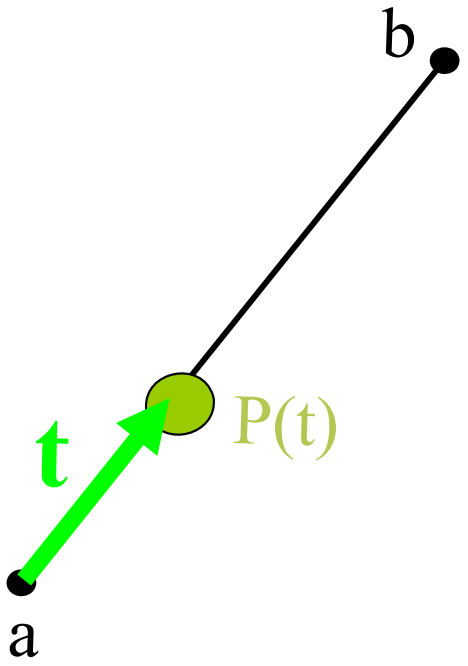
  However, this is probably the worst method of evaluating a point on the curve. Numerical instability, caused by raising small values to high powers, generates errors.

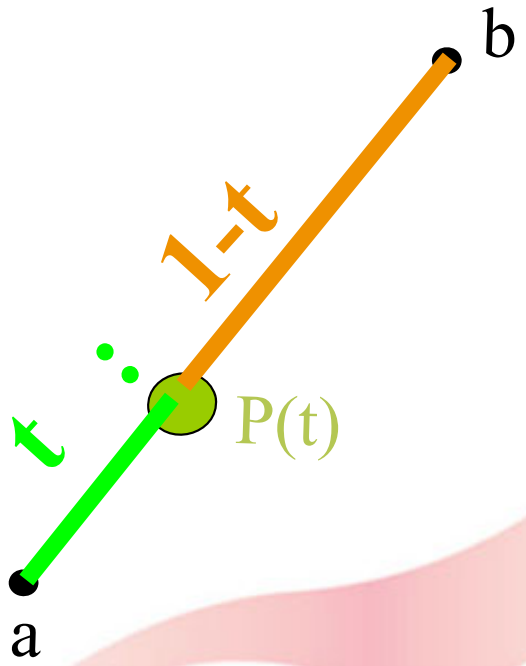
- Better solution: de Casteljau's algorithm !

Basic process: Lerp

- Lerp – linear interpolation: $P(t) = (1-t)\,a + t\,b = a + t\,(b-a)$
- Notation: Lerp(a,b) = $(1-t)\,a + t\,b$
- Lerp(a,b) linearly interpolates a and b. As t varies between 0 and 1, Lerp(a,b) gradually changes from a to b.

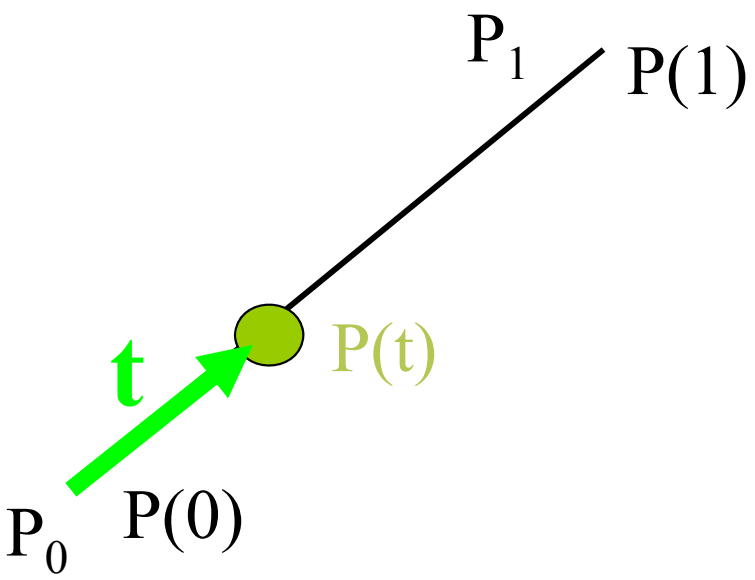






NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

Linear Bezier curve

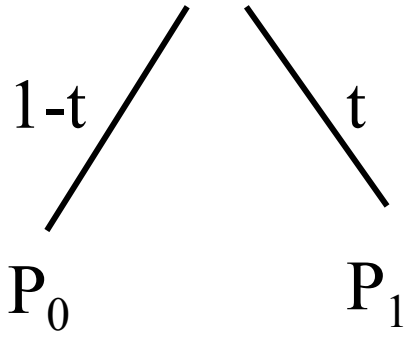


Linear (degree 1, order 2)

$P(0) = P_0, P(1) = P_1$

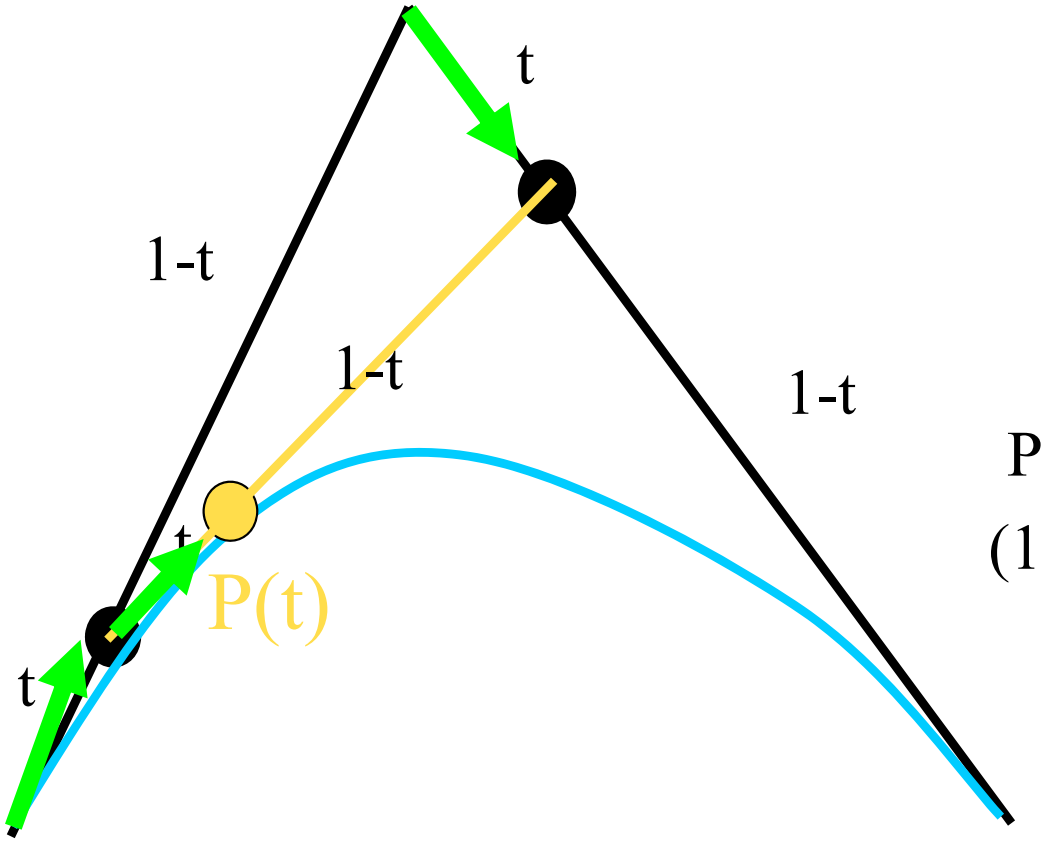
$P(t) = ?$

$$P(t) = (1-t)\,P_0 + t\,P_1 = P_0 + t(P_1-P_0)$$



easy to code up, very stable numerically

Quadratic Bezier curve



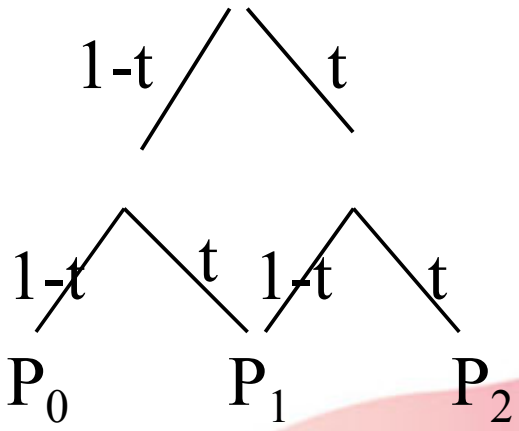
Quadratic: degree 2, order 3

$P(0) = P_0, P(1) = P_2$

$P(t) = ?$

$$P(t) = (1-t)^2 P_0 + 2t(1-t) P_1 + t^2 P_2 =$$

$$(1-t)\,[(1-t)P_0 + tP_1\,] + t\,[(1-t)P_1 + t\,P_2]$$



NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

General Bezier curves

The de Casteljau algorithm can be regarded as repeated linear interpolation:

Step 1. Divide each leg into parameter proportions

Step 2. Connect with new segments

Step 3. Repeat with new segments

Step 4. The final point is on the curve.

de Casteljau algorithm

Input: control polygon $P_0 P_1 \ldots P_n$ and parameter value t

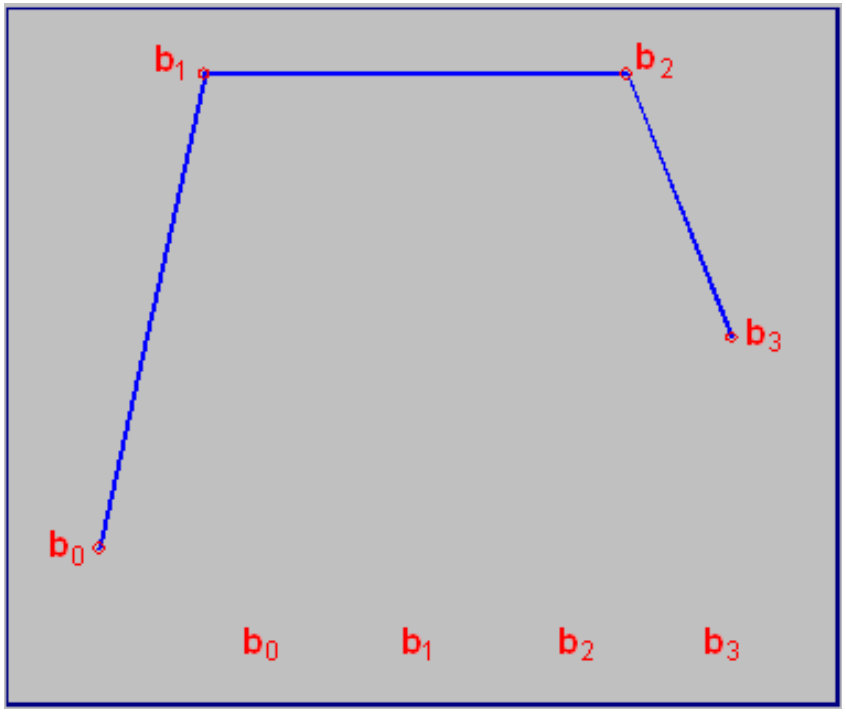
Output: P(t)

Procedure:

1. Initialize $P_0^0 = P_0, \quad P_1^0 = P_1, \cdots, P_n^0 = P_n$
2. for k from 1 to *n* do
3. for i from 0 to *n*-*k* do
4. $$P_i^k(t) = (1-t)P_i^{k-1} + tP_{i+1}^{k-1}$$
5. end-for
6. end-for
7. return $P(t) = P_0^n$

NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

Illustration of de Casteljau algorithm



NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

Illustration of de Casteljau algorithm

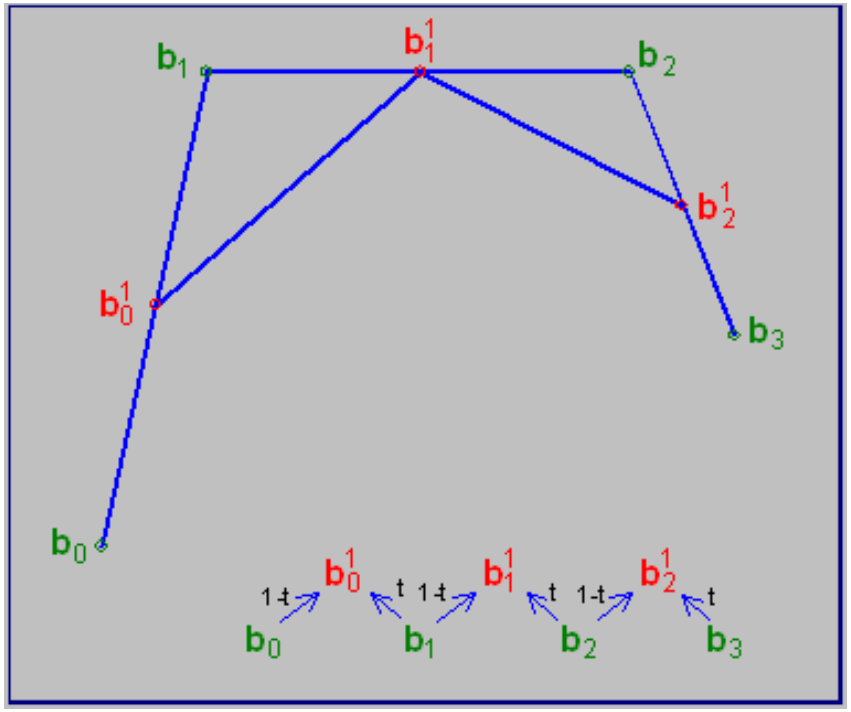


Illustration of de Casteljau algorithm

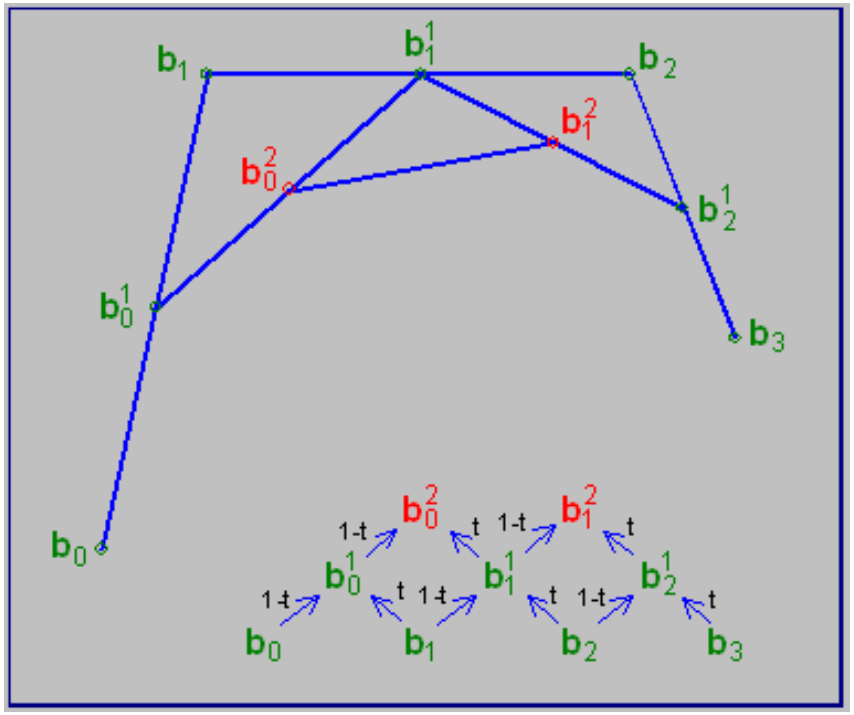
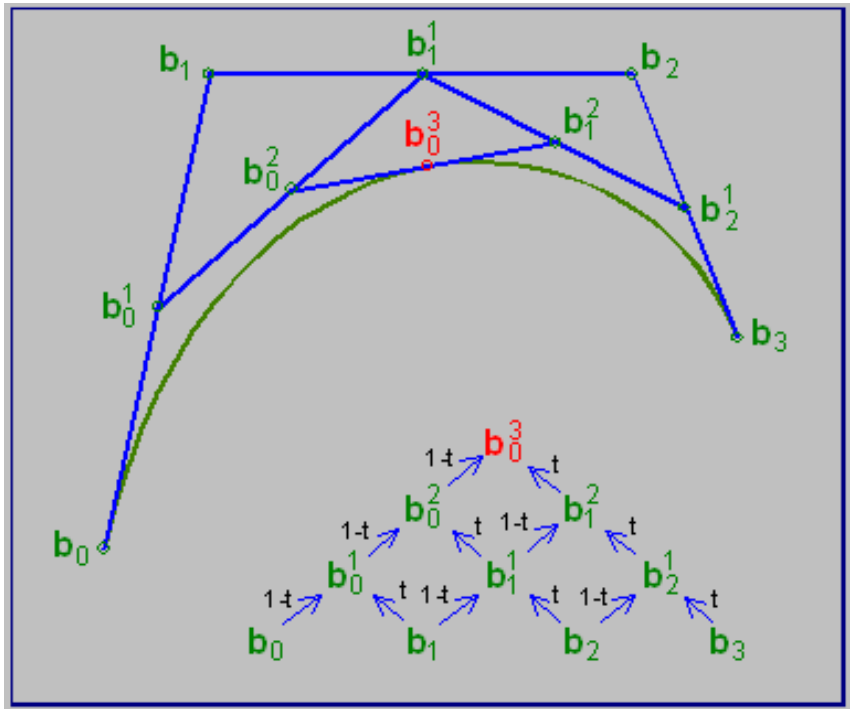


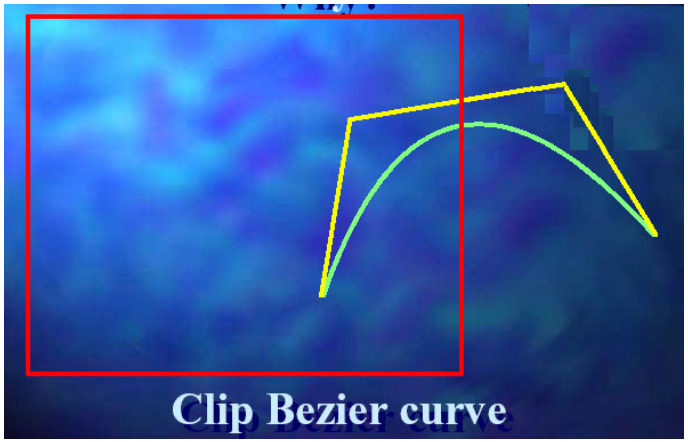
Illustration of de Casteljau algorithm

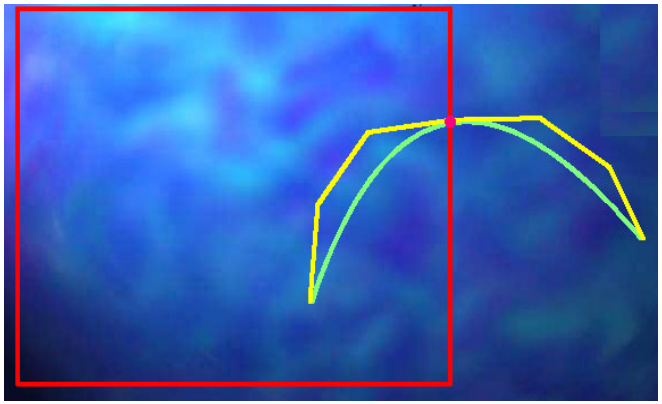


NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

4.2 Subdivision algorithm

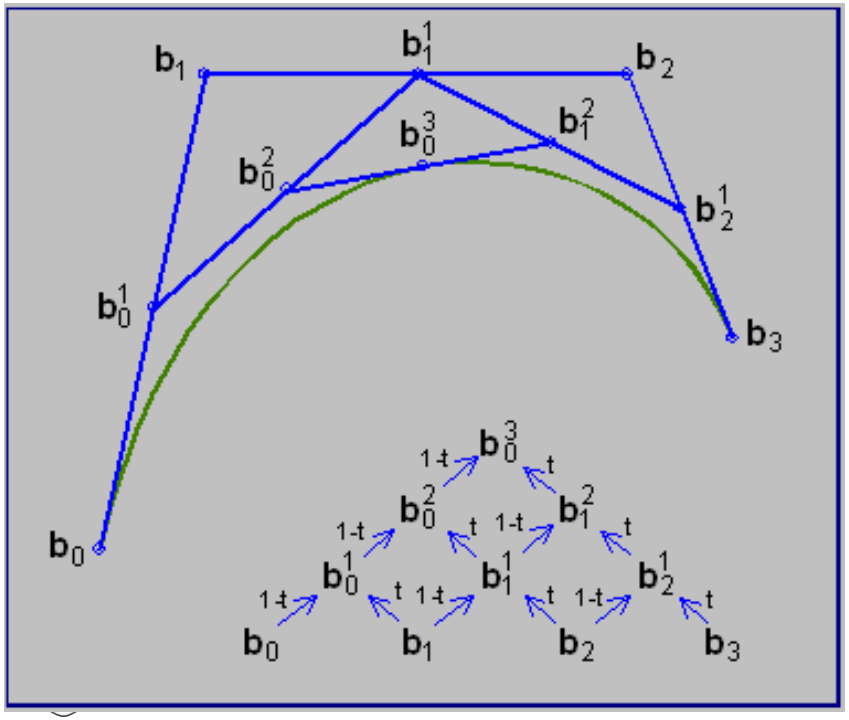
- Subdivision for a Bezier curve is to split it into two (or several) segments, each of which is also a Bezier curve. Subdivision makes divide-conquer possible. Uses include:
  - Design refinement: permit existing designs to be refined and modified.
  - Clip a curve to a boundary: visualization, other operations

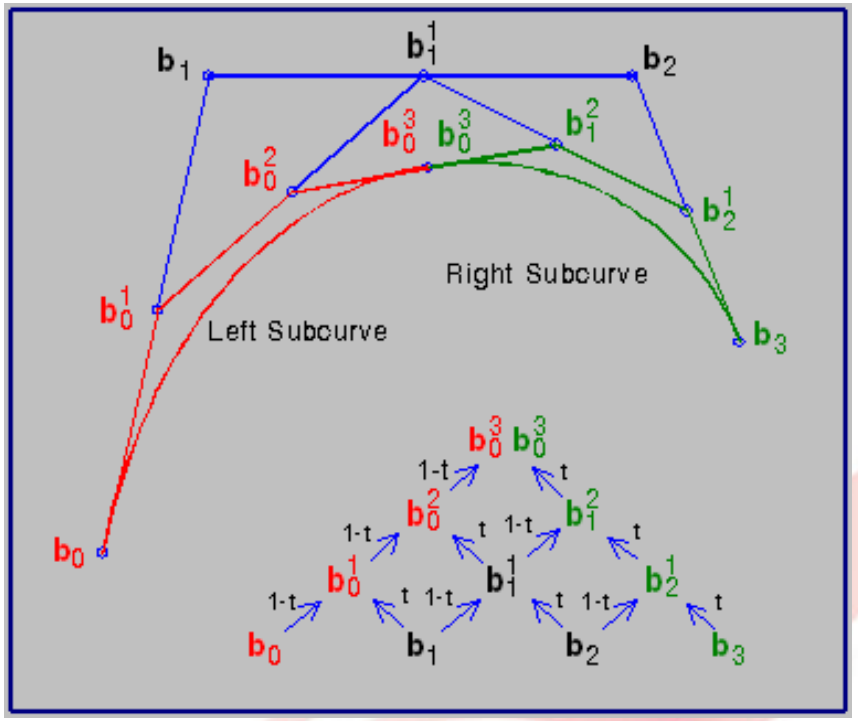




Subdivision via de Casteljau

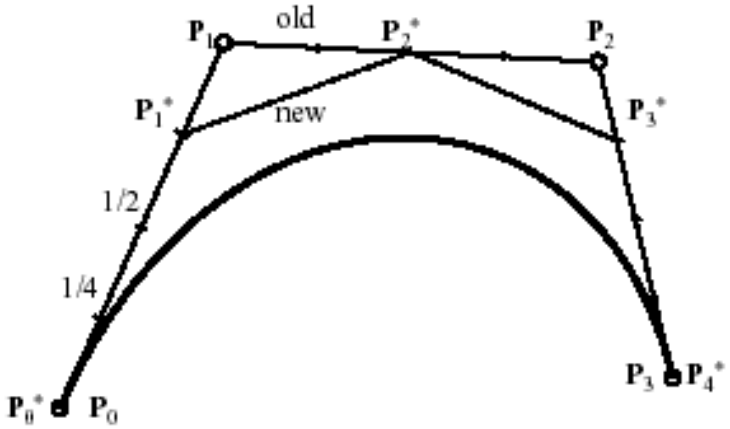
- Subdivision can be accomplished by de Casteljau algorithm. By using the left and right legs of the de Casteljau pyramid as control points, two separate Bézier curves are obtained that together replicate the original.





4.3 Degree elevation

- The algorithm to find the Bézier curve of the next higher degree is called *degree elevation*. Degree elevation is always possible.
- Given a Bézier curve of degree *n* with control points P_0, P_1, ..., P_n, it may be proved that the control points for the Bézier curve of degree *n*+1 can be found as follows:
  - *The endpoints are identical:* $P_0^* = P_0, \quad P_{n+1}^* = P_n$
  - *The others are* $P_i^* = \frac{i}{n+1} P_{i-1} + \left(1 - \frac{i}{n+1}\right) P_i, \quad i = 1,2,\cdots n$

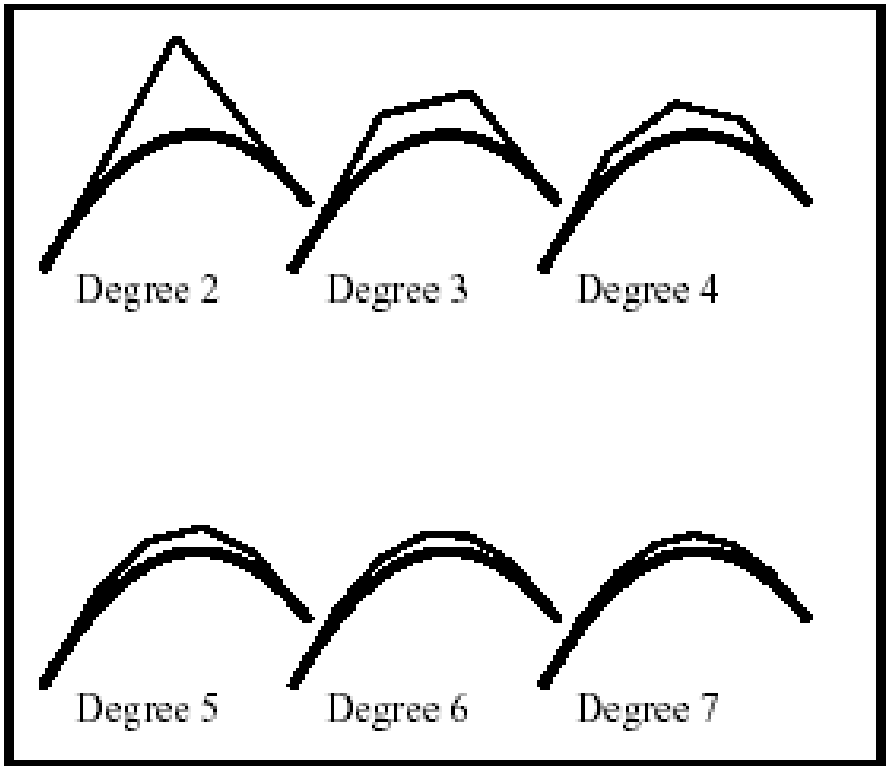


NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

Repeated degree elevation

- Degree elevation can be used to unify the degree of different degree curves. It can also be used to increase the degree of freedom for design.

- When degree elevation is repeated, the control polygon will converge to the curve.



NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

4.4 Differentiation

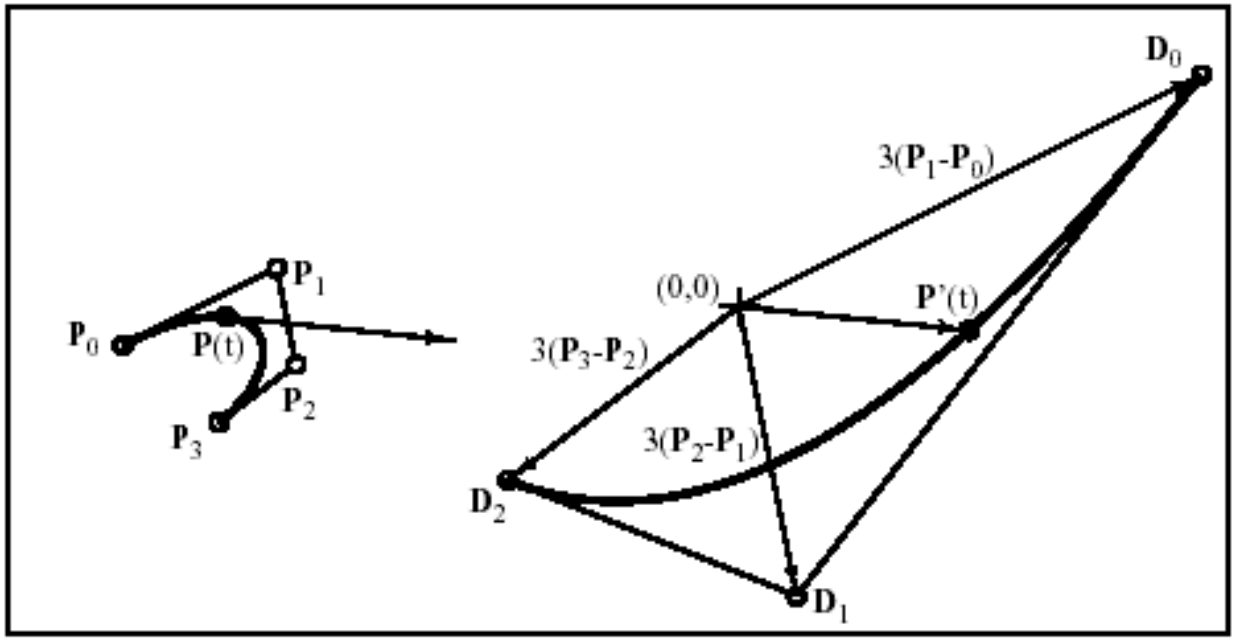
Derivative of a degree n Bezier curve $P(t)$ is

$$P'(t) = n\sum_{i=0}^{n-1}(P_{i+1} - P_i)B_i^{n-1}(t)$$

- That is, the parametric derivative of a degree n Bezier curve can be determined geometrically from its control points. It can be expressed as a curve of degree n-1 with control points $D_i = n(P_{i+1}-P_i)$.
- It is easy to obtain that $P'(0) = n(P_1-P_0)$ and $P'(1) = n(P_n-P_{n-1})$.
- The first derivative curve is known as a *hodograph*.

Bezier curve and hodograph

- The first derivative of a parametric curve provides us with a tangent vector.
- This differentiation can be repeated to obtain higher derivatives.



NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

Example: power basis → Bernstein

Geometric approach: Given a curve $P(t)$ in any form, we use its positions and differentiations at two endpoints to determine the Bezier control points.

$$P_0 = P(0), \quad P_n = P(1)$$
$$n(P_1 - P_0) = P'(0), \quad n(P_n - P_{n-1}) = P'(1)$$
$$\ldots\ldots$$

Example: convert degree 2 curve $Q(t) = Q_0 + Q_1 t + Q_2 t^2$ into the Bezier representation.

Solution: $P_0 = Q(0) = Q_0, \quad P_2 = Q(1) = Q_0 + Q_1 + Q_2$

$$2(P_1 - P_0) = Q'(0) = Q_1 + 2Q_2 t\big|_{t=0} = Q_1 \quad \Rightarrow \quad P_1 = P_0 + 0.5Q_1$$

Therefore, $Q(t) = Q_0 B_0^2(t) + (Q_0 + 0.5Q_1) B_1^2(t) + (Q_0 + Q_1 + Q_2) B_2^2(t)$

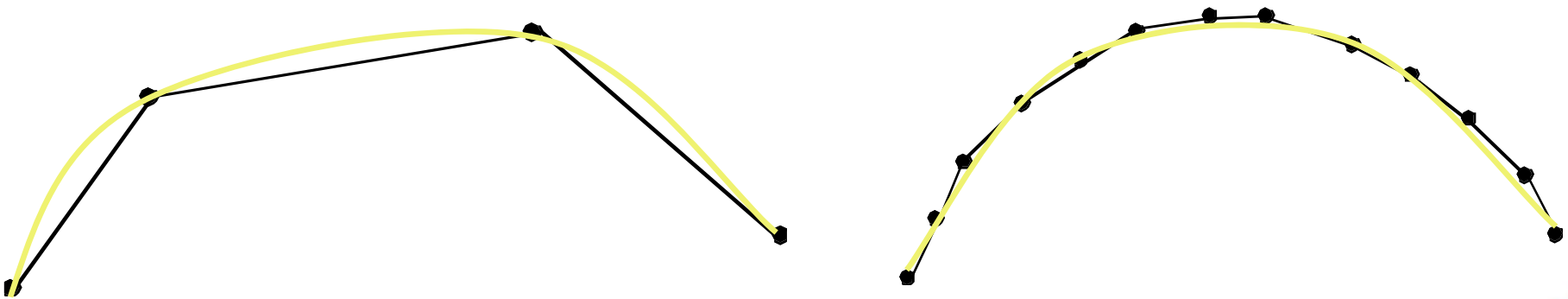
4.5 Tessellation

- Flatness of a curve
- Adaptive rendering
- Uniform sampling

NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

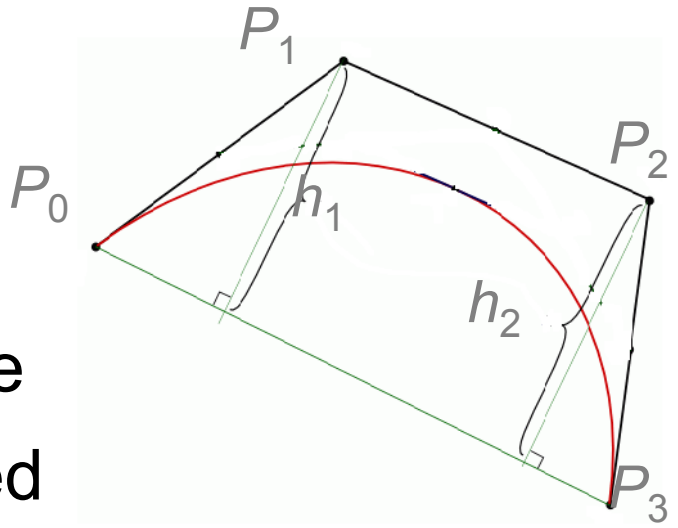
Flatness

- Motivation: In many applications, the curve needs to be approximated by line segments. The less flat the curve is, the more line segments are needed. Therefore the question is **how to measure whether a curve is flat or not**.



Estimate flatness

- Method: use convex hull property
  - Connect two endpoints P_0P_n as the base line.
  - Compute the distance h_i of all other control points P_i against the base line
  - Compute h = max{h_i }. Then h is used as a bound of the flatness of the curve.
  - If h is smaller than given tolerance, the curve is flat. We just use the base line for the curve.

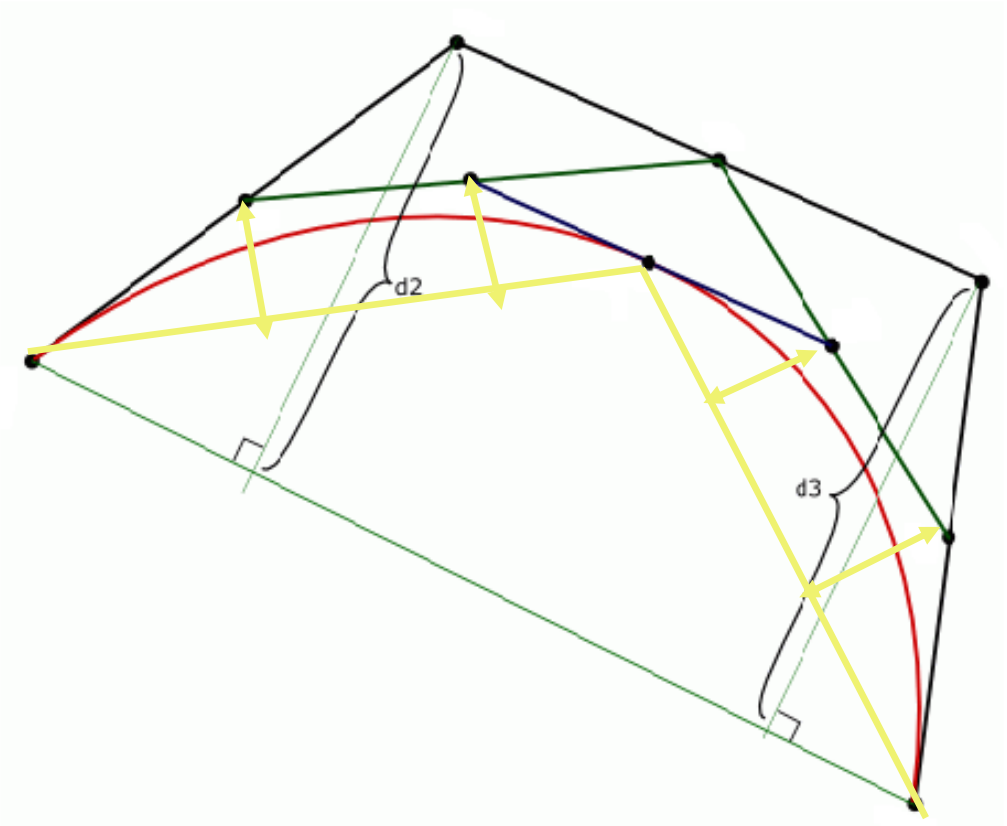


Distance formula: $$h_i = \frac{|P_0P_n \times P_0P_i|}{|P_0P_n|}$$

NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

Adaptive tessellation

- A tessellation algorithm:
  - keep breaking curve into sub-curves
  - stop when each sub-curve is nearly collinear
  - draw the base line



Adaptive tessellation algorithm

```
void TessellateBezierCurve (BezierCurve *bez)
{
        double height = maxDistance (bez);
        if (height < ε) {
                outputLine(bez->P0, bez->Pn);
                return;
                }
        else {
                midSubdivide (bez, leftBez, rightBez);
                TessellateBezierCurve (leftBez);
                TessellateBezierCurve (rightBez);
        }
}
```

maxDistance (BezierCurve *bez) : compute the maximal distance of the control points against the base line $P_0 P_n$.

midSubdivide (BezierCurve *bez, BezierCurve *leftBez, BezierCurve *rightBez): subdivide a Bezier curve into two sub-curves at the midpoint (t=0.5).

Outline

Homework

Q: Analyze whether the following two Bezier curves have the same shape or not.

Bezier curve 1:

$$\mathbf{r}_1(t) = \begin{pmatrix} -52 \\ -22 \end{pmatrix} B_0^3(t) + \begin{pmatrix} -52 \\ 36 \end{pmatrix} B_1^3(t) + \begin{pmatrix} 18 \\ 40 \end{pmatrix} B_2^3(t) + \begin{pmatrix} -6 \\ -60 \end{pmatrix} B_3^3(t), \quad t \in [0,1]$$

Bezier curve 2:

$$\mathbf{r}_2(s) = \begin{pmatrix} -6 \\ -60 \end{pmatrix} B_0^4(s) + \begin{pmatrix} 12 \\ 15 \end{pmatrix} B_1^4(s) + \begin{pmatrix} -17 \\ 38 \end{pmatrix} B_2^4(s) + \begin{pmatrix} -52 \\ 21.5 \end{pmatrix} B_3^4(s) + \begin{pmatrix} -52 \\ -22 \end{pmatrix} B_4^4(s), \quad s \in [0,1]$$

Outline

Summary

- Bezier curve formulation
- Bezier curve properties and their implications
- Algorithms
  - de Casteljau algorithm
  - Subdivision algorithm
  - Degree elevation
  - Differentiation
  - Bezier curve tessellation

End