DMET B402 Computer Graphics

Turker Ince

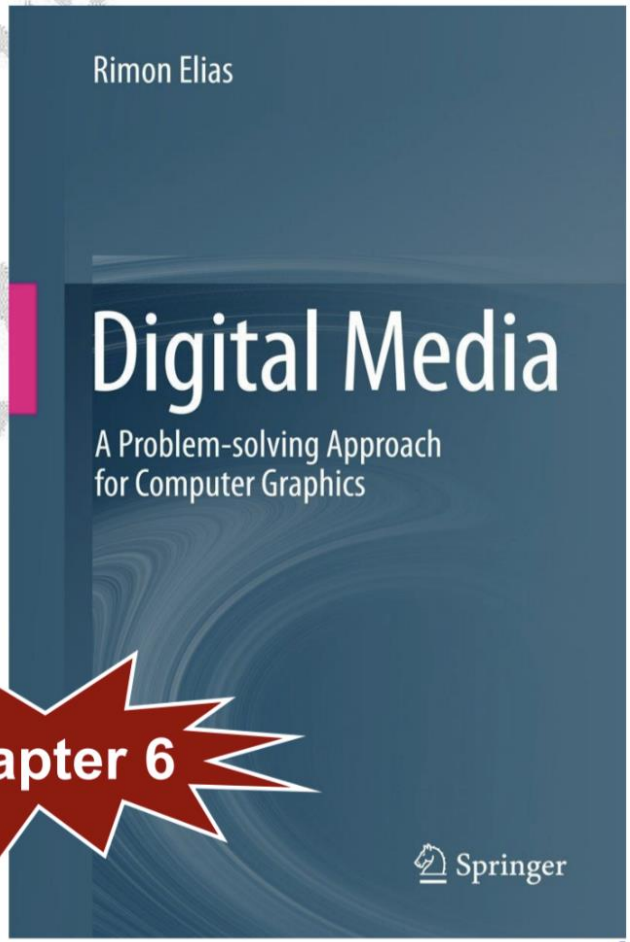
Lecture 11: Curves

Due Credits

- The presented material is mostly based on Assoc. Prof. Dr. Rimon Elias (GUC).
- CENG 477 course material @METU

Contents

- Curves:
  - Polynomial equations
  - Representations of curves
  - Degree
  - Interpolating versus approximating curves
  - Bézier curves
  - B-spline curves
  - NURBS curves



Chapter 6

Polynomial Equations

- A polynomial is used to represent lines and curves mathematically.

Polynomial equations:

- A line may be represented as:

First-degree (linear) polynomial → $ax + by = c$

- A curve may be represented as:

Quadratic polynomial → $y = ax^2 + bx + c$

- Also, a curve may be represented as:

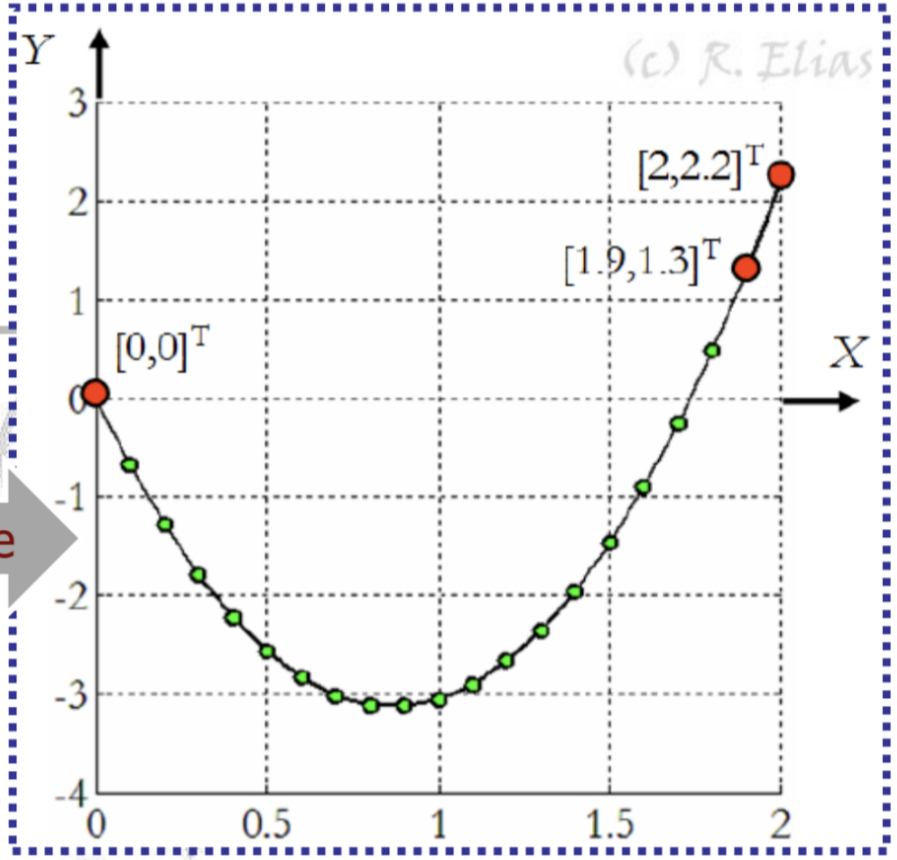
Cubic polynomial → $y = ax^3 + bx^2 + cx + d$

A polynomial used to represent curves may take the form $y=f(x)$ where $f(x)$ consists of powers of x only.

Degree

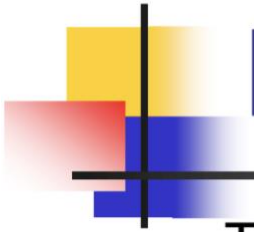
Equation	Degree
Linear	1
Quadratic	2
Cubic	3

A degree 2 curve



- The **degree** of a polynomial equation is the largest exponent in the equation.
- An equation of a curve may be of degree 2 or higher.

- Degree is a mathematical property of a curve that controls how many vertices are available for modeling.
- Degree is the curve's degree of freedom to bend.



Representations of Curves

- There are different ways to write an equation for a curve.

1. Explicit representation (as in the previous slides)
2. Implicit representation
3. Parametric representation

Representations of Curves: Explicit Representation

1. Explicit representation of curves:

- The basic definition of a curve in 2D is by considering a polynomial function that takes the form:

$$y = f(x)$$

- Consider this cubic polynomial:

$$y = ax^3 + bx^2 + cx + d$$

where the coefficients *a*, *b*, *c*, *d* are constants.

- Having this polynomial with a range for *x*, a curve can be drawn.

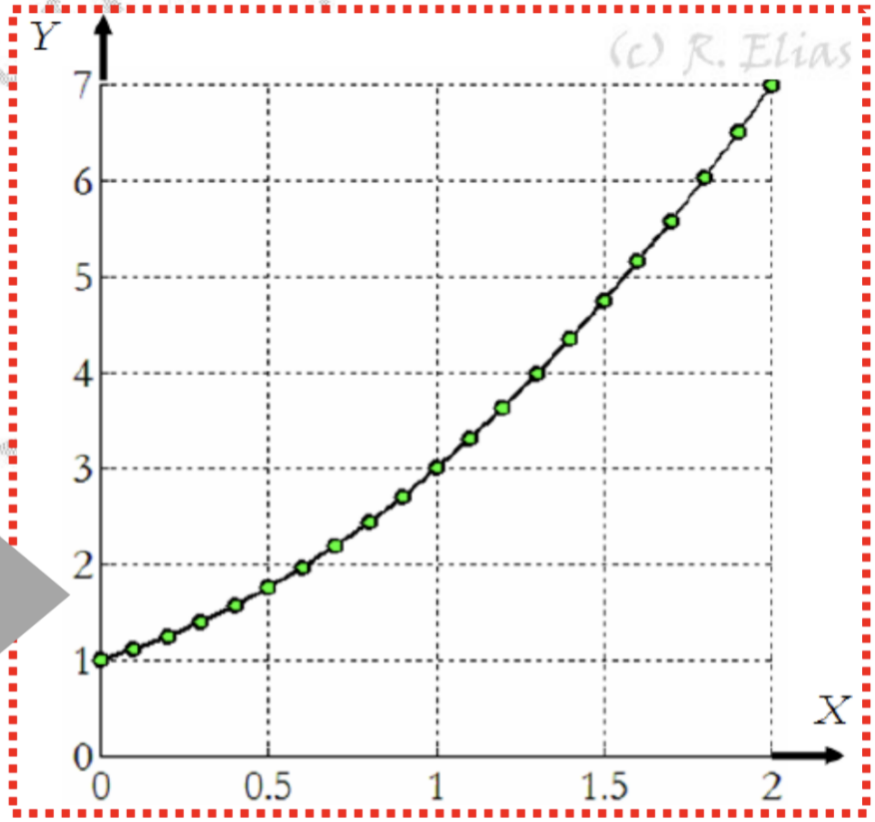
Representations of Curves: Explicit Representation

- **Example:** Given the quadratic polynomial:

$$y = ax^2 + bx + c$$

where *x* values range from 0 to 2, draw the curve that is represented by this polynomial equation.

Output



Representations of Curves: Explicit Representation

- If the parameters or coefficients (i.e., *a*, *b*, ..) are not available, the curve can be represented explicitly by considering control points. For example, if three points (2 of them are end points) are specified for a quadratic curve, the curve can be drawn.
- For a quadratic curve, three simultaneous equations can be solved to get one solution:

Given $[x_0, y_0]^T, [x_1, y_1]^T$ and $[x_2, y_2]^T$

$$y_0 = ax_0^2 + bx_0 + c$$
$$y_1 = ax_1^2 + bx_1 + c$$
$$y_2 = ax_2^2 + bx_2 + c$$

$$a = \frac{x_0(y_1 - y_2) + y_0(x_2 - x_1) + x_1 y_2 - x_2 y_1}{(x_1 - x_0)(x_2 - x_0)(x_2 - x_1)}$$

$$b = \frac{y_2 - y_0}{x_2 - x_0} - (x_2 + x_0)a$$

$$c = y_0 - ax_0^2 - bx_0$$

can be applied to higher degrees

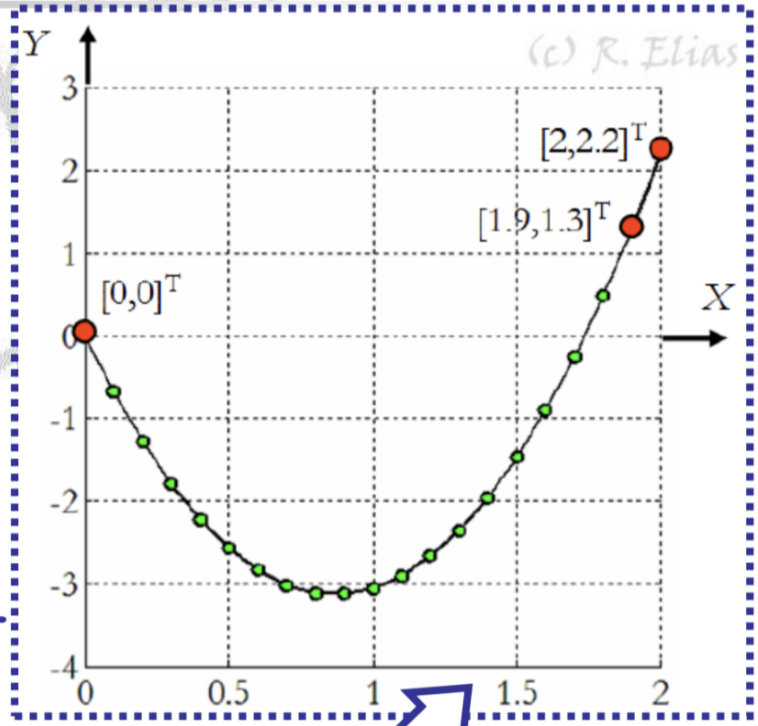
Representations of Curves: Explicit Representation

- **Example:** Consider a quadratic curve whose endpoints are $[0, 0]^T$ and $[2, 2.2]^T$. If the point $[1.9, 1.3]^T$ is on the curve, get its polynomial equation and draw the curve.

$$a = \frac{x_0(y_1-y_2)+y_0(x_2-x_1)+x_1y_2-x_2y_1}{(x_1-x_0)(x_2-x_0)(x_2-x_1)}$$
$$= \frac{0\times(1.3-2.2)+0\times(2-1.9)+1.9\times2.2-2\times1.3}{(1.9-0)(2-0)(2-1.9)} = 4.1579$$

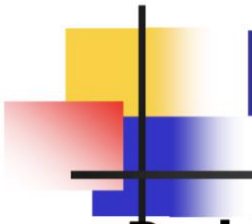
$$b = \frac{y_2-y_0}{x_2-x_0} - (x_2+x_0)a$$
$$= \frac{2.2-0}{2-0} - (2+0)\times 4.1579 = -7.2158$$

$$c = y_0 - ax_0^2 - bx_0$$
$$= 0 - 4.1579\times 0^2 - (-7.2158)\times 0 = 0$$



$$y = 4.1579x^2 - 7.2158x$$

Representations of Curves: Explicit Representation



Problems

- It is impossible to get multiple values of *y* for the same *x*. So circles and ellipses must be represented as a sequence of curve segments.
- Describing curves with vertical tangents is difficult. (It is difficult to represent a slope of infinity.)

Curves

- Representations
  - Explicit: y = f(x)
    - Essentially a function plot over some interval x \in [a, b]
    - Simple to compute, simple to check whether point lies on a curve
    - Cannot represent closed curves
  - Implicit: f(x, y) = 0
    - Solution of an equation system (ax + by + c = 0 for line in 2D)
    - Simple to check whether point lies on a curve
    - Can represent closed curves
  - Parametric: x = x(t), y = y(t)
    - Describe position on the curve by a parameter (c(t) = (1-t)a + tb for line in 2D)
    - Hard to check whether point lies on a curve
    - Can represent closed curves
    - Simple to render; just change t and get new x, y to render

Representations of Curves: Implicit Representation

2. Implicit representation of curves:

- The *implicit representation* of curves combines the variables (i.e., *x* and *y* in 2D space or *x*, *y* and *z* in 3D space) in an equation.
- Generally, the implicit representation functions take the forms:

$$f(x, y) = 0 \quad \text{or} \quad f(x, y, z) = 0$$

- Examples:

Linear $ax + by + c = 0$

Higher-degree $ax^3 + by^2 + 2cxy + 2dx + 2ey + f = 0$

- The entire equation must be solved in order to calculate *x* and *y*.

Representations of Curves: Implicit Representation

Problems

- The equation may have more solutions than we want.
- Example:

$$x^2 + y^2 = 1$$

  - This is an implicit representation of a circle.
  - If you want to model half a circle, you have to add a constraint like x>0.
  - This cannot be contained in an implicit equation.

Representations of Curves: Parametric Representation

3. Parametric representation of curves:

- The parametric representation of curves writes short and easily-solved curve equations that translate one variable into values for the others.

Linear parametric curves

- In 2D space

$$\begin{aligned} x(t) &= a_x t + b_x \\ y(t) &= a_y t + b_y \end{aligned} \equiv \begin{aligned} x(t) &= \underbrace{\begin{bmatrix} t & 1 \end{bmatrix}}_{\mathbf{t}} \underbrace{\begin{bmatrix} a_x \\ b_x \end{bmatrix}}_{\mathbf{x}} = \mathbf{t} \bullet \mathbf{x}^T \\ y(t) &= \underbrace{\begin{bmatrix} t & 1 \end{bmatrix}}_{\mathbf{t}} \underbrace{\begin{bmatrix} a_y \\ b_y \end{bmatrix}}_{\mathbf{y}} = \mathbf{t} \bullet \mathbf{y}^T \end{aligned}$$

- In 3D, we add:

$$z(t) = a_z t + b_z \equiv z(t) = \underbrace{\begin{bmatrix} t & 1 \end{bmatrix}}_{\mathbf{t}} \underbrace{\begin{bmatrix} a_z \\ b_z \end{bmatrix}}_{\mathbf{z}} = \mathbf{t} \bullet \mathbf{z}^T$$

Representations of Curves: Parametric Representation

Quadratic parametric curves

- In 2D:

$$\begin{aligned} x(t) &= a_x t^2 + b_x t + c_x \\ y(t) &= a_y t^2 + b_y t + c_y \end{aligned}$$

$\equiv$

$$x(t) = \underbrace{\begin{bmatrix} t^2 & t & 1 \end{bmatrix}}_{\mathbf{t}} \underbrace{\begin{bmatrix} a_x \\ b_x \\ c_x \end{bmatrix}}_{\mathbf{x}} = \mathbf{t} \bullet \mathbf{x}^T$$

$$y(t) = \underbrace{\begin{bmatrix} t^2 & t & 1 \end{bmatrix}}_{\mathbf{t}} \underbrace{\begin{bmatrix} a_y \\ b_y \\ c_y \end{bmatrix}}_{\mathbf{y}} = \mathbf{t} \bullet \mathbf{y}^T$$

- In 3D, we add:

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

$\equiv$

$$z(t) = \underbrace{\begin{bmatrix} t^2 & t & 1 \end{bmatrix}}_{\mathbf{t}} \underbrace{\begin{bmatrix} a_z \\ b_z \\ c_z \end{bmatrix}}_{\mathbf{z}} = \mathbf{t} \bullet \mathbf{z}^T$$

Each dimension is treated independently, so we can add dimensions.

Representations of Curves: Parametric Representation

- A quadratic parametric spline may be written as:

$$\dot{\mathbf{p}}(t) = \mathbf{a}t^2 + \mathbf{b}t + \mathbf{c} \equiv \begin{aligned} x(t) &= a_x t^2 + b_x t + c_x \\ y(t) &= a_y t^2 + b_y t + c_y \end{aligned}$$

where:

- $\dot{\mathbf{p}}$ is the curve point we are trying to find;
- **a**, **b**, **c** are coefficients for the *x*- and *y*-directions; and
- *t* is the parameter (where $0 \leq t \leq 1$).
  - You can visualize parametric curves as being drawn by a point moving through space. At any time *t*, we can calculate the *x* and *y* values of the moving point.
  - The curve is actually a combination of two quadratic curves, one is *y=f(t)* and the other is *x=f(t)*. By varying t between 0 and 1, *x* and *y* will both vary and create the curve.

Representations of Curves: Parametric Representation

How to solve

- In order to solve this equation we can specify three points on the curve; two of them are endpoints.

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

- At the start point $\dot{\mathbf{p}}_0$ where t=0:

$$\dot{\mathbf{p}}_0(0) = \mathbf{c}$$

- At the endpoint $\dot{\mathbf{p}}_2$ where t=1:

$$\dot{\mathbf{p}}_2(1) = \mathbf{a} + \mathbf{b} + \mathbf{c}$$

- At the intermediate point $\dot{\mathbf{p}}_1$:

$$\dot{\mathbf{p}}_1(t) = \mathbf{a}t^2 + \mathbf{b}t + \mathbf{c}$$

$$\begin{aligned} \mathbf{c} &= \dot{\mathbf{p}}_0 \\ \mathbf{a} &= \frac{t(\dot{\mathbf{p}}_2 - \dot{\mathbf{p}}_0) + (\dot{\mathbf{p}}_0 - \dot{\mathbf{p}}_1)}{t(1-t)} \\ \mathbf{b} &= \dot{\mathbf{p}}_2 - \dot{\mathbf{p}}_0 - \mathbf{a} \end{aligned}$$

Representations of Curves: Parametric Representation

- We may write:

$$\begin{aligned} \mathbf{c} &= \dot{\mathbf{p}}_0 \\ \mathbf{a} &= \frac{t(\dot{\mathbf{p}}_2 - \dot{\mathbf{p}}_0) + (\dot{\mathbf{p}}_0 - \dot{\mathbf{p}}_1)}{t(1-t)} \\ \mathbf{b} &= \dot{\mathbf{p}}_2 - \dot{\mathbf{p}}_0 - \mathbf{a} \end{aligned}$$

$$\begin{bmatrix} \dot{\mathbf{p}}_0 \\ \dot{\mathbf{p}}_1 \\ \dot{\mathbf{p}}_2 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ t^2 & t & 1 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{a} \\ \mathbf{b} \\ \mathbf{c} \end{bmatrix}$$

$$\begin{bmatrix} \mathbf{a} \\ \mathbf{b} \\ \mathbf{c} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ t^2 & t & 1 \\ 1 & 1 & 1 \end{bmatrix}^{-1} \begin{bmatrix} \dot{\mathbf{p}}_0 \\ \dot{\mathbf{p}}_1 \\ \dot{\mathbf{p}}_2 \end{bmatrix}$$

Interpolation matrix

Can you get the interpolation matrix for cubic curves?

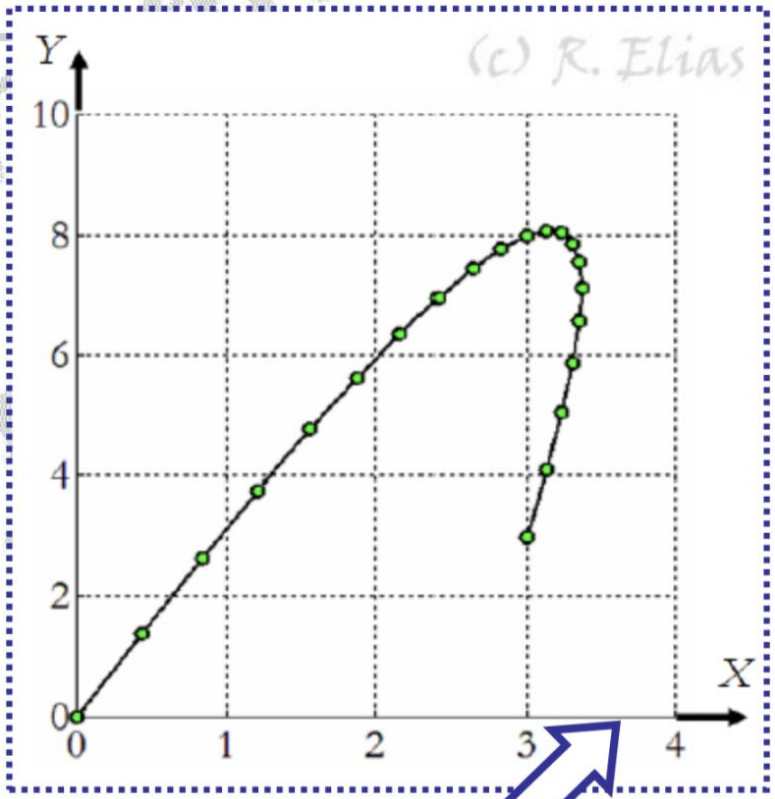
Representations of Curves: Parametric Representation

- **Example:** A quadratic curve whose endpoints are $[0, 0]^T$ and $[3, 3]^T$ is represented parametrically. If the point $[3, 8]^T$ is on the curve at $t = 0.5$, get the coefficients of the curve equation.
- **Answer:** The coefficients are

$$\mathbf{c} = \dot{\mathbf{p}}_0 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\mathbf{a} = \frac{t(\dot{\mathbf{p}}_2 - \dot{\mathbf{p}}_0) + (\dot{\mathbf{p}}_0 - \dot{\mathbf{p}}_1)}{t(1-t)} = \begin{bmatrix} -6 \\ -26 \end{bmatrix}$$

$$\mathbf{b} = \dot{\mathbf{p}}_2 - \dot{\mathbf{p}}_0 - \mathbf{a} = \begin{bmatrix} 9 \\ 29 \end{bmatrix}$$



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

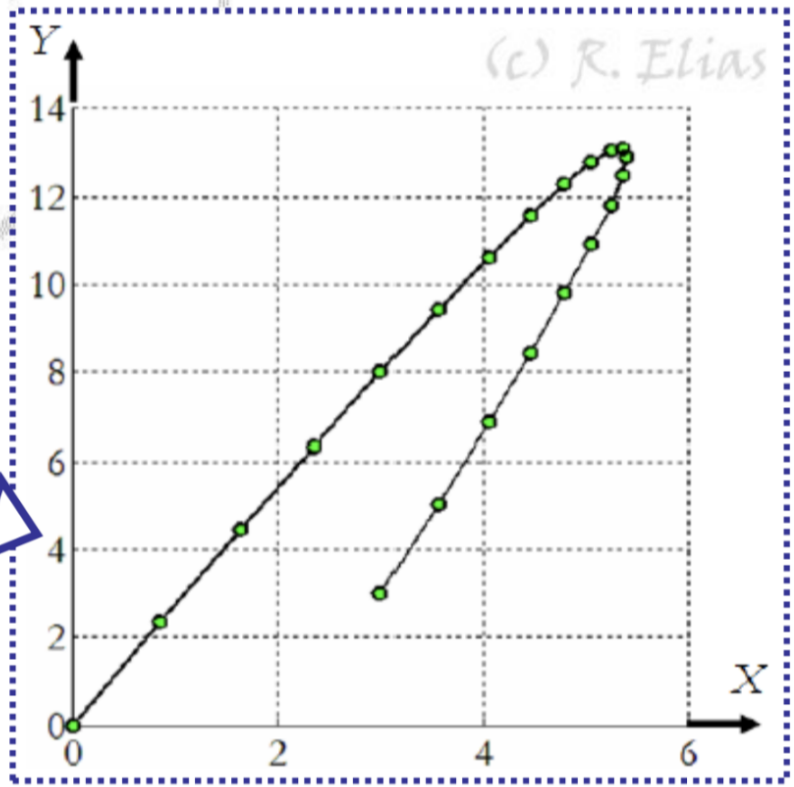
$$= -\begin{bmatrix} 6 \\ 26 \end{bmatrix} t^2 + \begin{bmatrix} 9 \\ 29 \end{bmatrix} t + \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Representations of Curves: Parametric Representation

- **Example:** Re-solve the previous example if the point $[3, 8]^T$ is on the curve where $t = 0.2$.

- **Answer:** The equation →

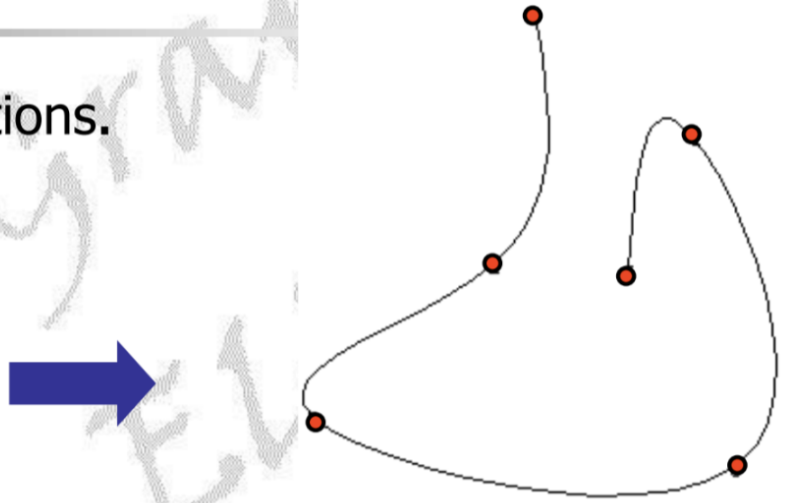
$$\begin{aligned}\dot{\mathbf{p}}(t) &= \mathbf{a}t^2 + \mathbf{b}t + \mathbf{c} \\ &= -\begin{bmatrix} 15 \\ 46.25 \end{bmatrix} t^2 + \begin{bmatrix} 18 \\ 49.25 \end{bmatrix} t + \begin{bmatrix} 0 \\ 0 \end{bmatrix}\end{aligned}$$



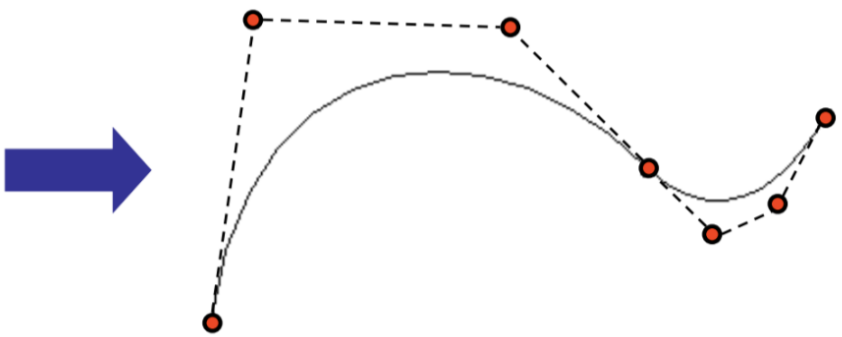
Interpolating versus Approximating Curves

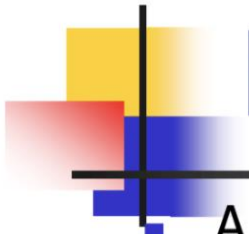
- Two types for spline representations.

1. **Interpolating curves** where the curve passes through the points describing it. (Refer to what was discussed.)



2. **Approximating curves** where the curve gets near the points describing it. Examples include Bézier, B-spline and NURBS curves.





Bézier Curves

- A Bézier curve is an approximating curve that is generated as a blend of a number of control points.

- A Bézier curve is controlled by two or more control points.
- In case of more than two control points:
  - Two of these points are the end points of the curve.
  - The rest of the points control the shape of the curve as a **blend** of the control points.

- The number of control points determines the degree of the curve.
  - Linear Bézier curves (given two points)
  - Quadratic Bézier curves (given three points)
  - Cubic Bézier curves (given four points)

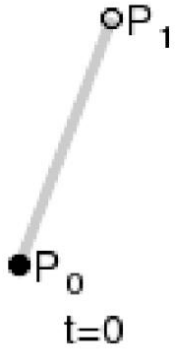
Linear Bézier Curves

- **Linear Bézier curves** (two control points $\dot{\mathbf{p}}_0$ and $\dot{\mathbf{p}}_1$):
  - This is just a representation of a parametric equation of a line.

$$\dot{\mathbf{p}}(t) = (1-t)\dot{\mathbf{p}}_0 + t\dot{\mathbf{p}}_1$$ where t $\in$ [0,1]

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

Geometry matrix for linear Bézier curves



Source: Wikipedia

Quadratic Bézier Curves

- **Quadratic Bézier curves** (three control points $\dot{\mathbf{p}}_0$, $\dot{\mathbf{p}}_1$ and $\dot{\mathbf{p}}_2$):
  - A point $\dot{\mathbf{p}}_{01}$ is obtained at t along the linear Bézier curve connecting $\dot{\mathbf{p}}_0$ and $\dot{\mathbf{p}}_1$.

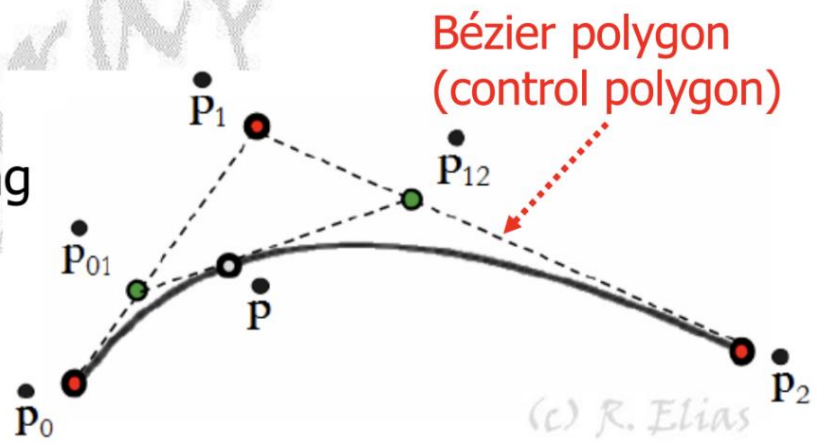
$$\dot{\mathbf{p}}_{01}(t) = (1-t)\dot{\mathbf{p}}_0 + t\dot{\mathbf{p}}_1$$

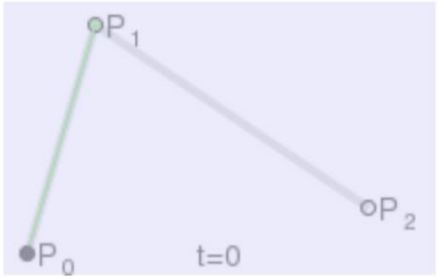
  - Similarly,

$$\dot{\mathbf{p}}_{12}(t) = (1-t)\dot{\mathbf{p}}_1 + t\dot{\mathbf{p}}_2$$

  - Then

$$\dot{\mathbf{p}}(t) = (1-t)\dot{\mathbf{p}}_{01} + t\dot{\mathbf{p}}_{12}$$

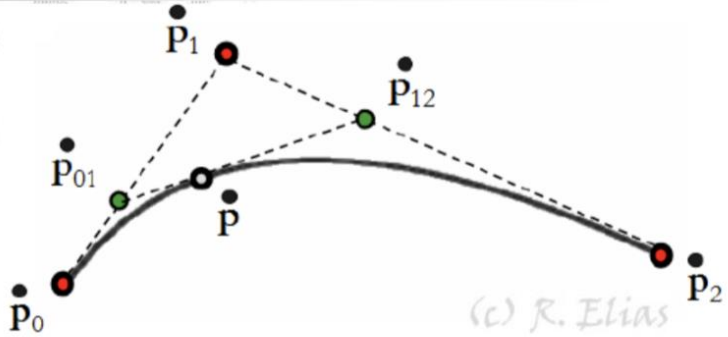




Source: Wikipedia

Quadratic Bézier Curves

$$\dot{\mathbf{p}}(t) = (1-t)\dot{\mathbf{p}}_{01} + t\dot{\mathbf{p}}_{12}$$



- Or

$$\begin{aligned}\dot{\mathbf{p}}(t) &= (1-t)\dot{\mathbf{p}}_{01} + t\dot{\mathbf{p}}_{12} \\ &= (1-t)\left[(1-t)\dot{\mathbf{p}}_0 + t\dot{\mathbf{p}}_1\right] + t\left[(1-t)\dot{\mathbf{p}}_1 + t\dot{\mathbf{p}}_2\right]\end{aligned}$$

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

///

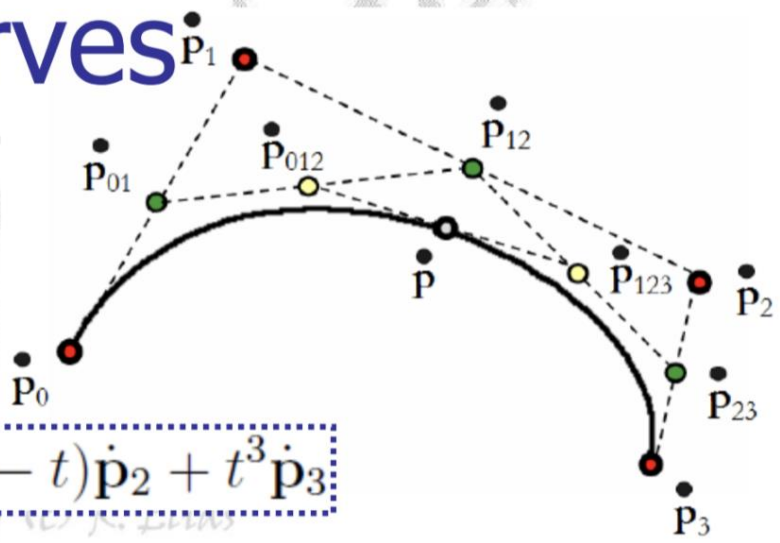
$$\dot{\mathbf{p}}(t) = \begin{bmatrix} t^2 & t & 1 \end{bmatrix} \begin{bmatrix} 1 & -2 & 1 \\ -2 & 2 & 0 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} \dot{\mathbf{p}}_0 \\ \dot{\mathbf{p}}_1 \\ \dot{\mathbf{p}}_2 \end{bmatrix}$$

Geometry matrix for quadratic Bézier curves

Cubic Bézier Curves

Cubic Bézier curves (four control points $\dot{\mathbf{p}}_0, \dot{\mathbf{p}}_1, \dot{\mathbf{p}}_2$ and $\dot{\mathbf{p}}_3$):

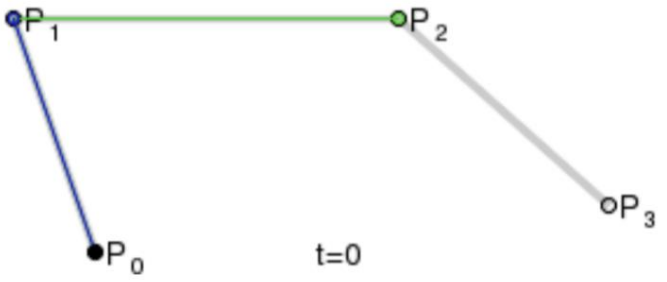
- Similar procedure:



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

Geometry matrix for cubic Bézier curves

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



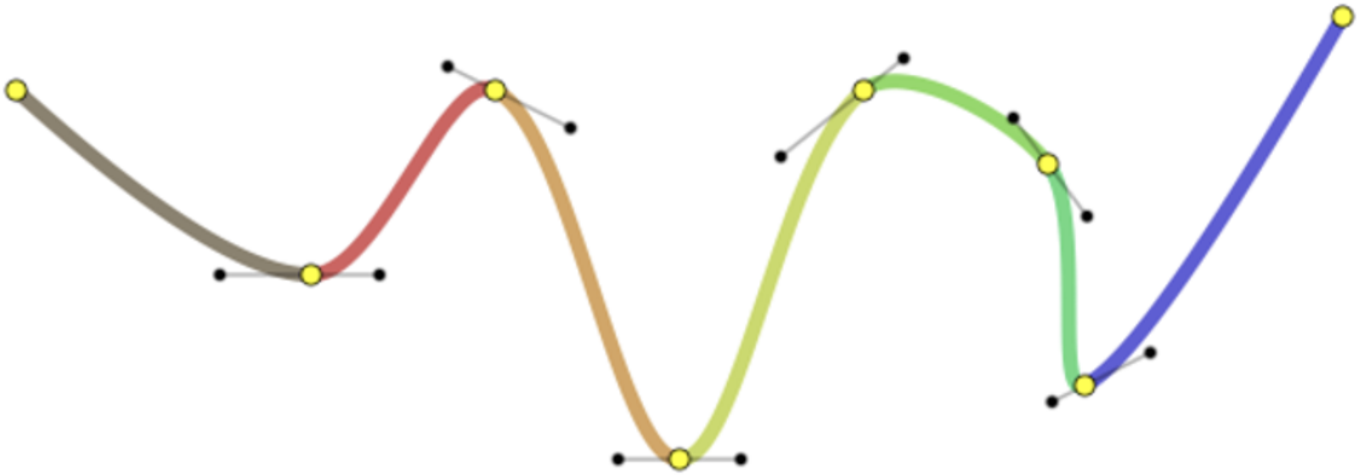
Source: Wikipedia

Higher Order Bézier Curves

- More complex Bézier curves that are higher than third degree curves can be constructed as a sequence of lower-degree Bézier curve segments.
- In this case, the joints should always look smooth.
- Splitting the curve into many lower-degree curve segments brings us to the concept of **B-splines**.

Curves

- Spline is a smooth aggregation of multiple different curves
- Spline can be defined piecewise by multiple polynomials



- Curve is defined by a single polynomial
- Curve is a special spline

B-spline Curves

- Unlike the previous curves discussed where each control point contributes to all points of the curve, a B-spline control point affects only a portion of the curve around it (and may be shared between segments).

- This is referred to as the ***local control*** property where control points away from a given curve point contributes nothing to its location.

- The local control property is achieved using polynomial ***basis*** or blending **functions**; hence, comes the letter "**B**" in **B**-spline.

B-spline Curves

- In this case, a curve point is estimated as a weighted sum of basis functions.

$$\dot{\mathbf{p}}(t) = \sum_{i=0}^{n} \dot{\mathbf{p}}_i N_{i,p}(t)$$

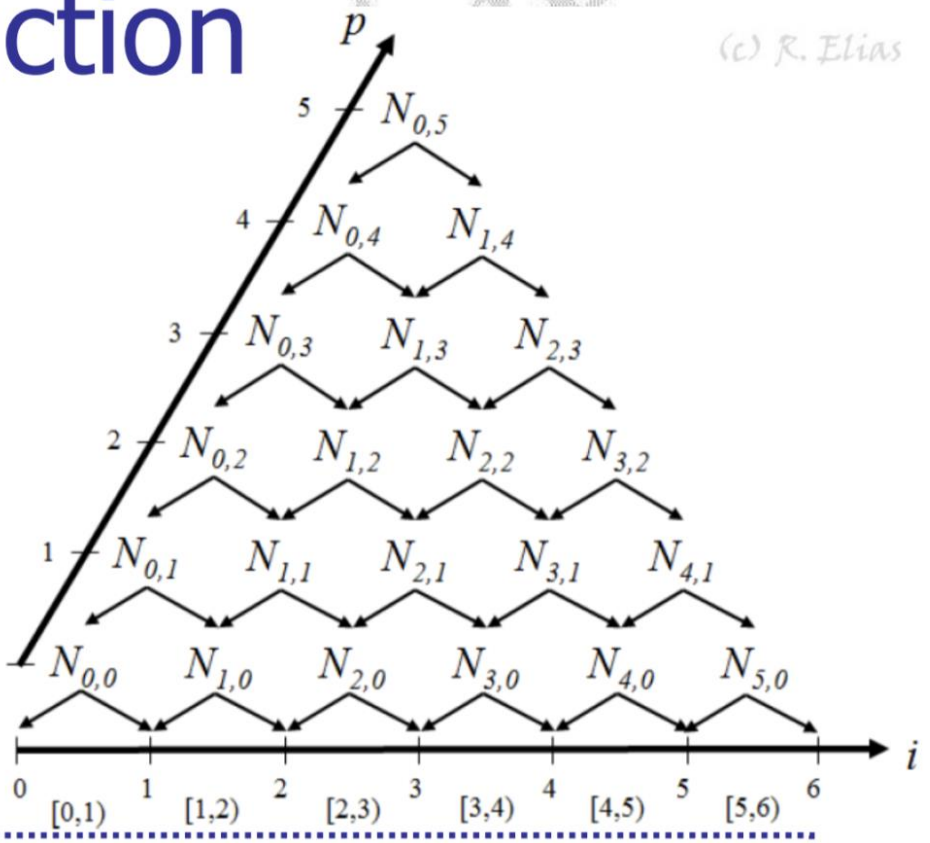
  - $\dot{\mathbf{p}}(t)$ is the curve point;
  - $\dot{\mathbf{p}}_i$ is the control point i such that $i \in \{0, 1, 2, \ldots, n\}$;
  - $n + 1$ is the number of control points;
  - p is the curve degree; and
  - $N_{i,p}(t)$ is the basis function.

Notice that you are adding the contributions of all control points; however, points far away from the current location will contribute nothing (zero).

The Basis Function

- A **basis function** specifies how strong or weak a control point contributes to a curve point at parameter *t*.

- The i^{th} basis function $N_{i,p}(t)$ of degree p is expressed by the Cox-de Boor recursion formula:



Base case

Recursive case

$$N_{i,0}(t) = \begin{cases} 1, & \text{if } t_i \le t < t_{i+1}; \\ 0, & \text{otherwise}, \end{cases}$$

$$N_{i,p}(t) = \frac{t-t_i}{t_{i+p}-t_i} N_{i,p-1}(t) + \frac{t_{i+p+1}-t}{t_{i+p+1}-t_{i+1}} N_{i+1,p-1}(t)$$

Quadratic B-spline Curves

- Consider the control points $\dot{\mathbf{p}}_0,\ \dot{\mathbf{p}}_1,\ \ldots,\ \dot{\mathbf{p}}_n$.

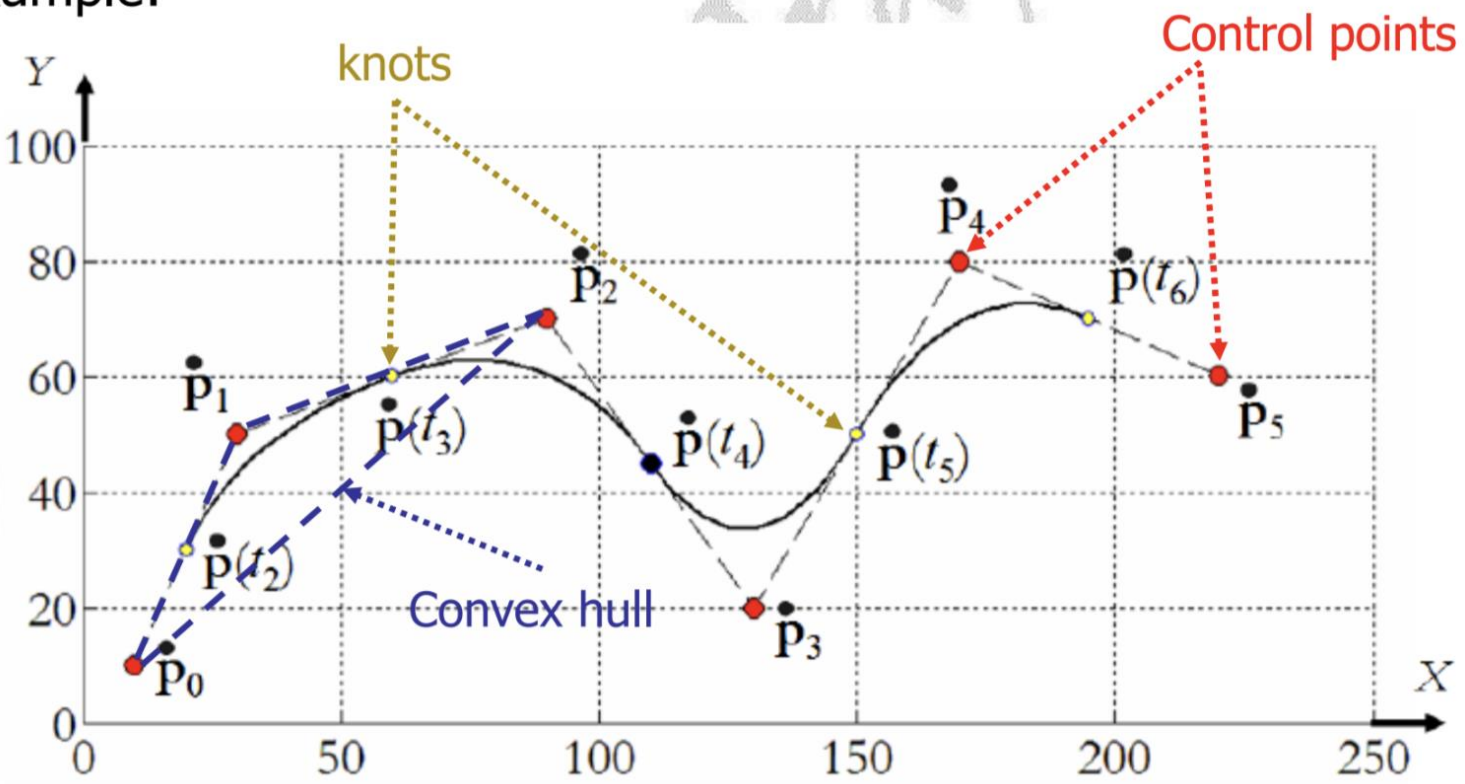
- Those points can approximate a **quadratic** B-spline curve that consists of a sequence of ***n* – 1** (where $n \geq 2$) **quadratic** curve segments.

Segment	Control points	Domain
$\mathcal{S}_2$	$\dot{\mathbf{p}}_0, \dot{\mathbf{p}}_1, \dot{\mathbf{p}}_2$	$t_2 \leq t < t_3$
$\mathcal{S}_3$	$\dot{\mathbf{p}}_1, \dot{\mathbf{p}}_2, \dot{\mathbf{p}}_3$	$t_3 \leq t < t_4$
$\vdots$	$\vdots$	$\vdots$
$\mathcal{S}_n$	$\dot{\mathbf{p}}_{n-2}, \dot{\mathbf{p}}_{n-1}, \dot{\mathbf{p}}_n$	$t_n \leq t < t_{n+1}$

- Points at the start and end of each segment are called **knots**.
- As each quadratic curve segment is controlled by **3** control points, a control point affects and contributes to **3** curve segments.

Quadratic B-spline Curves: An Example

- Example:



Cubic B-spline Curves

- Consider the control points $\dot{\mathbf{p}}_0, \dot{\mathbf{p}}_1, \ldots, \dot{\mathbf{p}}_n$.

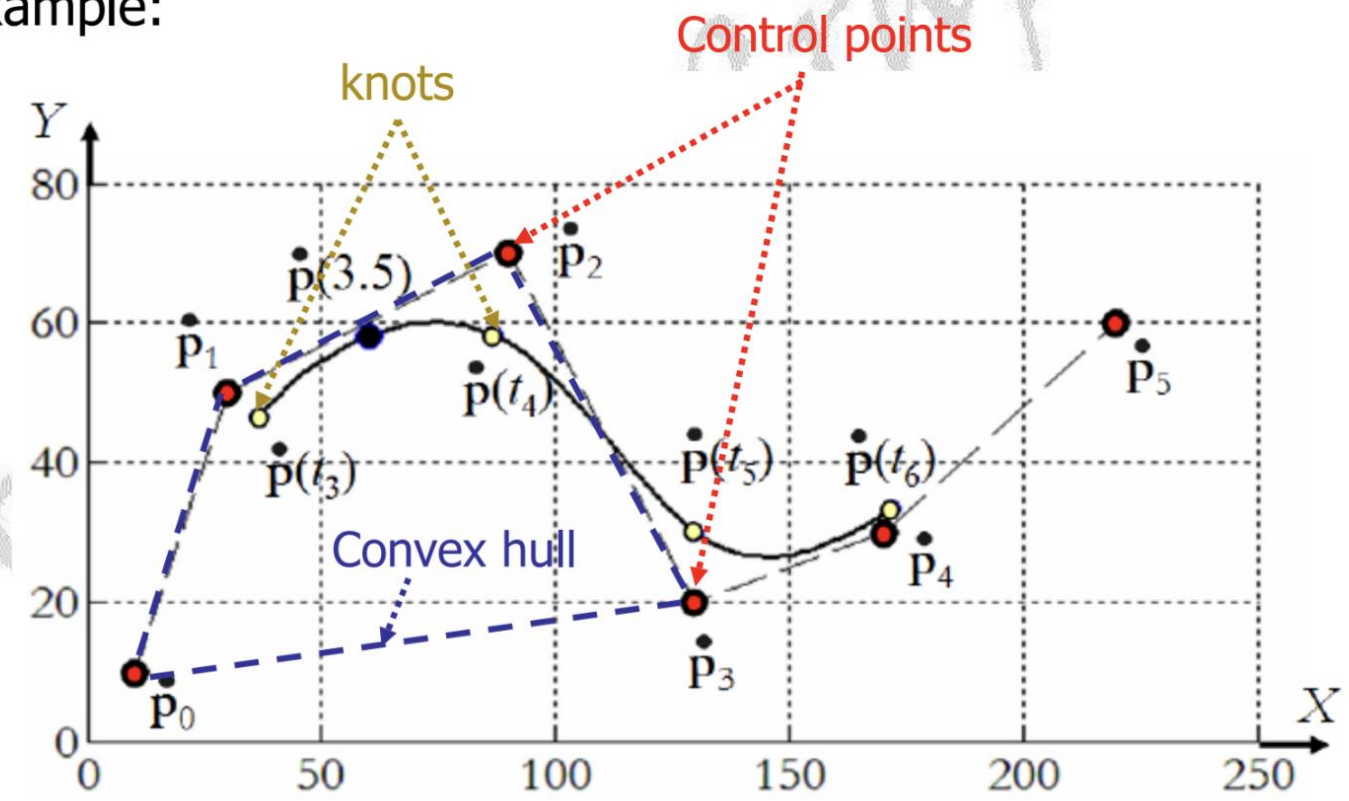
- Those points can approximate a **cubic** B-spline curve that consists of a sequence of ***n* – 2** (where $n \geq 3$) **cubic** curve segments.

Segment	Control points	Domain
$\mathcal{S}_3$	$\dot{\mathbf{p}}_0, \dot{\mathbf{p}}_1, \dot{\mathbf{p}}_2, \dot{\mathbf{p}}_3$	$t_3 \leq t < t_4$
$\mathcal{S}_4$	$\dot{\mathbf{p}}_1, \dot{\mathbf{p}}_2, \dot{\mathbf{p}}_3, \dot{\mathbf{p}}_4$	$t_4 \leq t < t_5$
$\vdots$	$\vdots$	$\vdots$
$\mathcal{S}_n$	$\dot{\mathbf{p}}_{n-3}, \dot{\mathbf{p}}_{n-2}, \dot{\mathbf{p}}_{n-1}, \dot{\mathbf{p}}_n$	$t_n \leq t < t_{n+1}$

- Points at the start and end of each segment are called **knots**.
- As each quadratic curve segment is controlled by **4** control points, a control point affects and contributes to **4** curve segments.

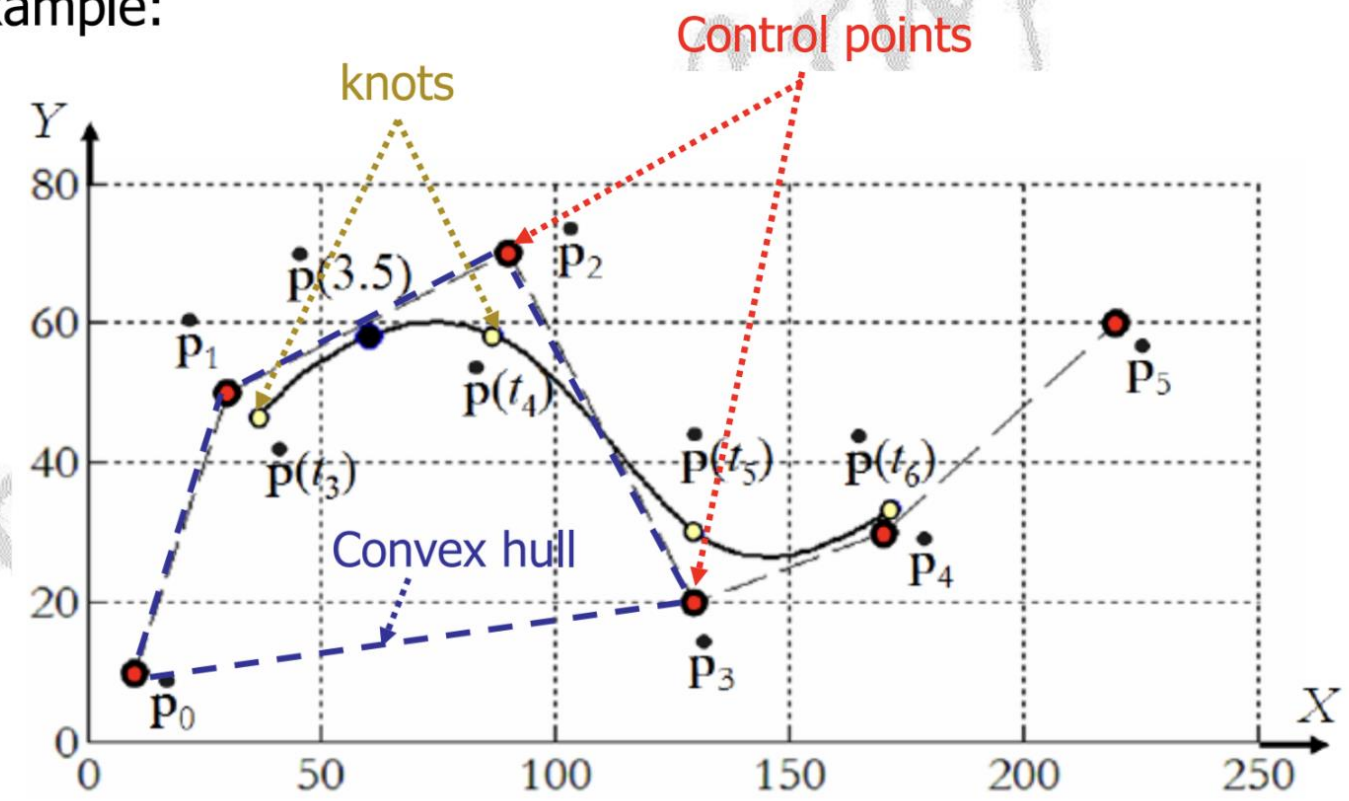
Cubic B-spline Curves: An Example

- Example:



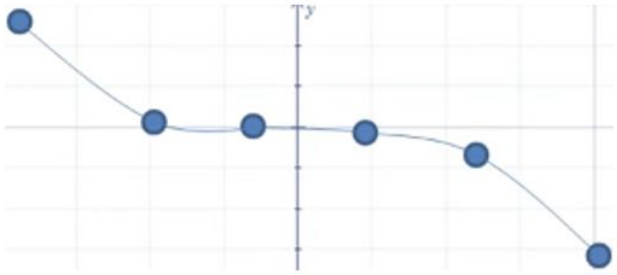
Cubic B-spline Curves: An Example

- Example:



Curves

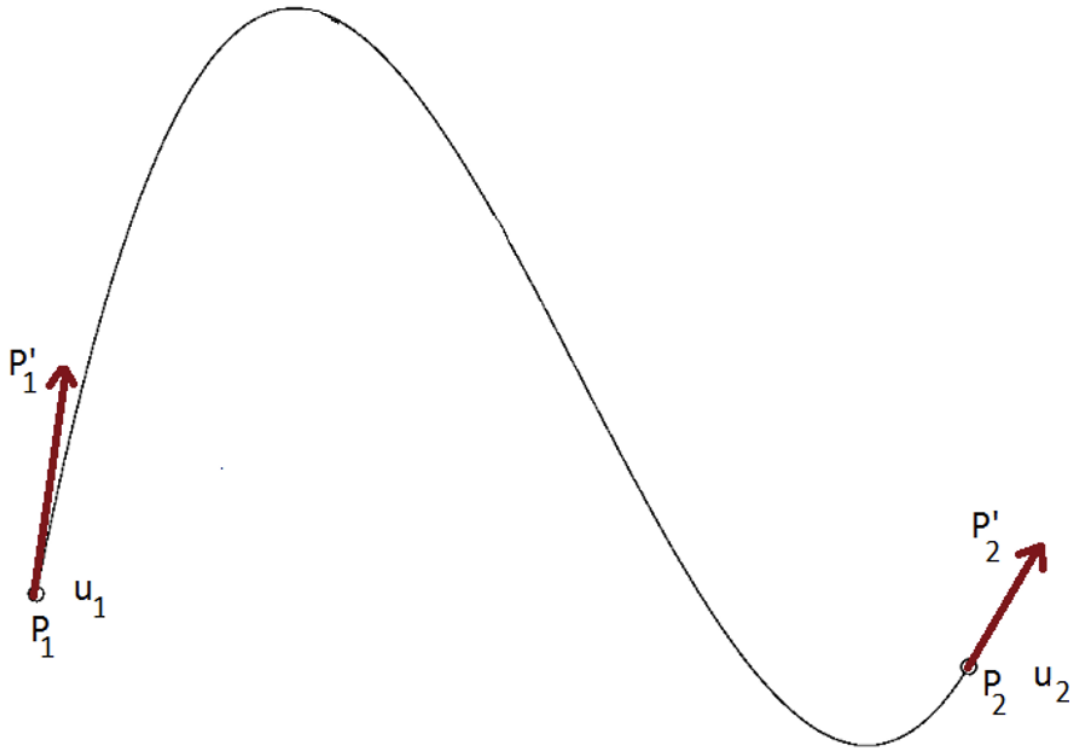
- Piecewise Cubic Spline



- What's special about splines?
  - Function approximations that are continuous at merging points
  - Also have continuous first and second derivatives where they join

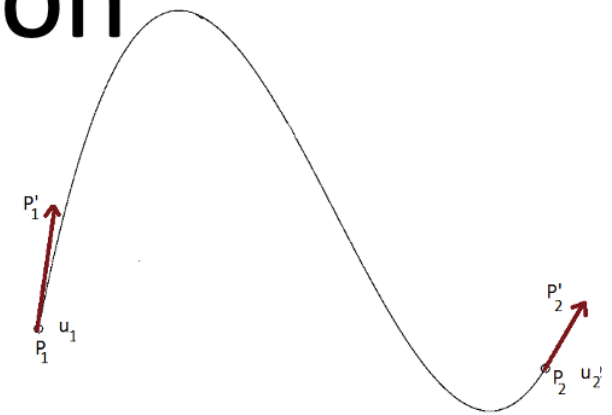
Spline Interpolation

- Calculation of a cubic spline
- Given positions P_1, P_2, and derivatives/tangents P'_1, P'_2



Source: CENG477-METU

Spline Interpolation



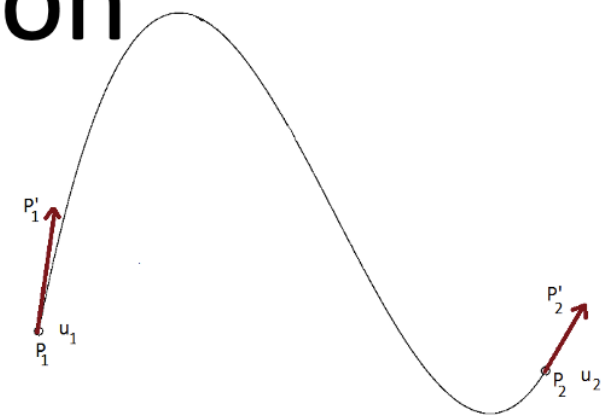
- Calculation of a cubic spline
- Given P_1, P_2, and P'_1, P'_2
- Let $u_1 = 0$
- $P(0) = P_1$ $P'(0) = P'_1$
- $P(u_2) = P_2 P'(u_2) = P'_2$

- Use $P(u) = B_1 + B_2u + B_3u^2 + B_4u^3$ for substitutions
- $P(0) = B_1 = P_1$
- $P(u_2) = B_1 + B_2u_2 + B_3u_2^2 + B_4u_2^3 = P_2$

- Use $P'(u) = B_2 + 2B_3u + 3B_4u^2$ for substitutions
- $P'(0) = B_2 = P'_1$
- $P'(u_2) = B_2 + 2B_3u_2 + 3B_4u_2^2 = P'_2$

Source: CENG477-METU

Spline Interpolation



- Calculation of a cubic spline
- Given P_1, P_2, and P'_1, P'_2
- Let $u_1 = 0$
- $P(0) = P_1$ $P'(0) = P'_1$
- $P(u_2) = P_2 P'(u_2) = P'_2$

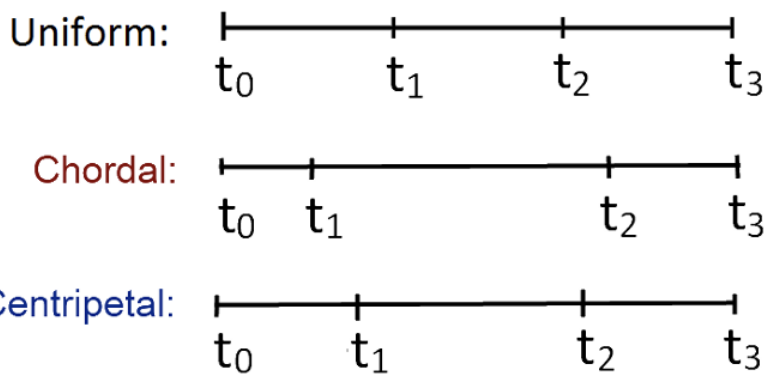
- Use $P(u) = B_1 + B_2u + B_3u^2 + B_4u^3$ for substitutions
- **$P(0) = B_1 = P_1$**
- $B_3 = 3(P_2 - P_1)/u_2^2 - 2P'_1/u_2 - P'_2/u_2$

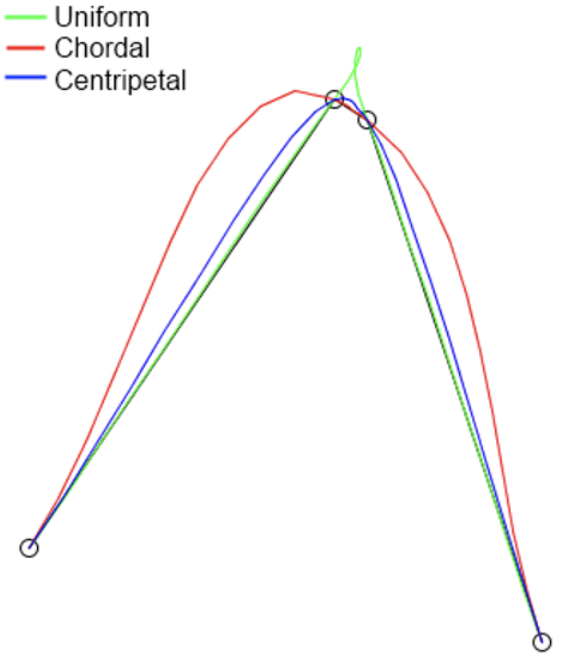
- Use $P'(u) = B_2 + 2B_3u + 3B_4u^2$ for substitutions
- **$P'(0) = B_2 = P'_1$**
- $B_4 = 2(P_1 - P_2)/u_2^3 + P'_1/u_2^2 + P'_2/u_2^2$

Source: CENG477-METU

Spline Parametrization

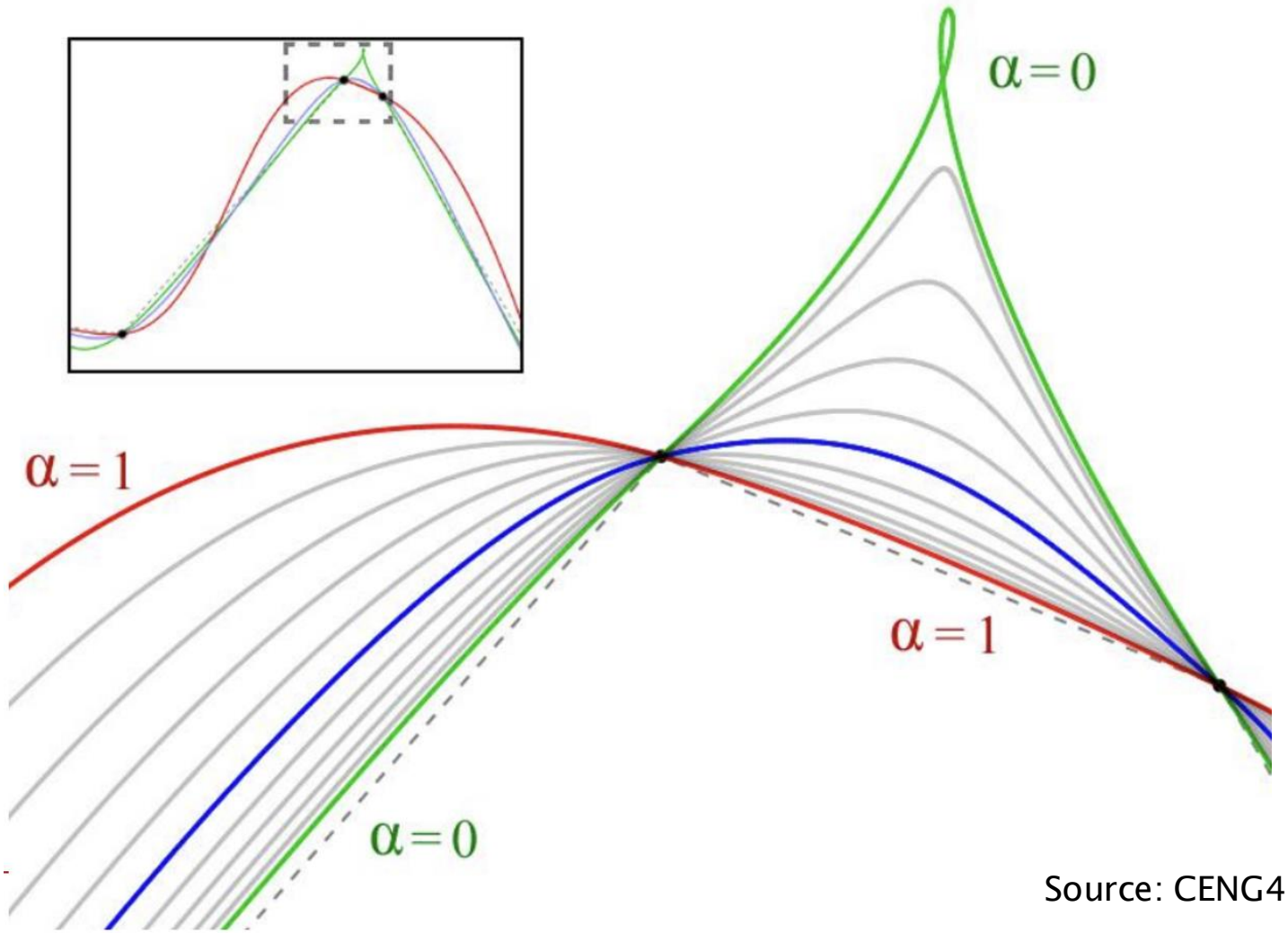
- All from the same family: $||P_{k-1} - P_k||^{\alpha}$
- Uniform: $u_k = ||P_{k-1} - P_k||^0$
- Chordal: $u_k \sim ||P_{k-1} - P_k||^1$
- Centripetal: $u_k \sim ||P_{k-1} - P_k||^{0.5}$ (best)
  - Will not form loop or self-intersection within a curve segment
  - Cusp will never occur within a curve segment
  - Spline follows the control points more tightly





Spline Parametrization

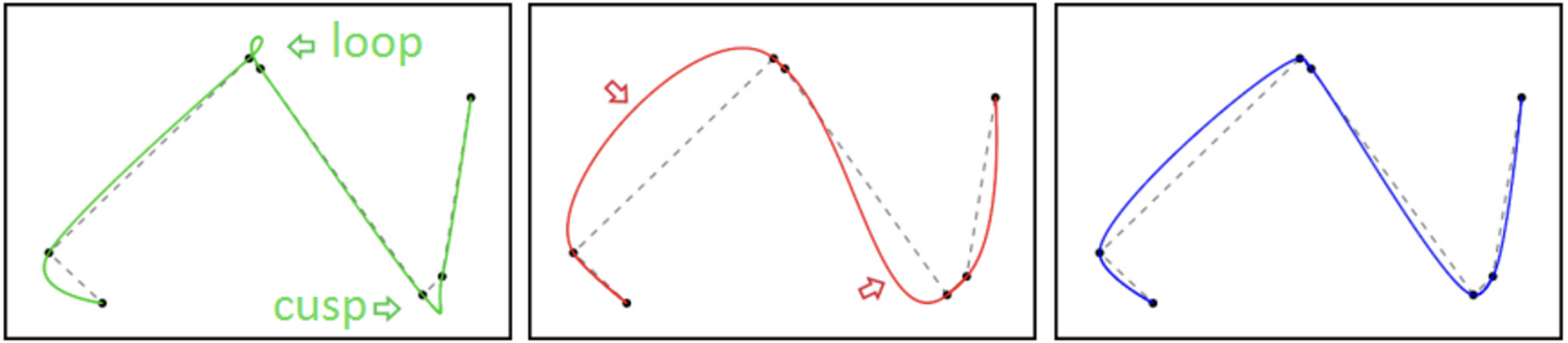
All from the same family: $||P_{k-1} - P_k||^{\alpha}$



Source: CENG477-METU

Spline Parametrization

- All from the same family: $||P_{k-1} - P_k||^{\alpha}$



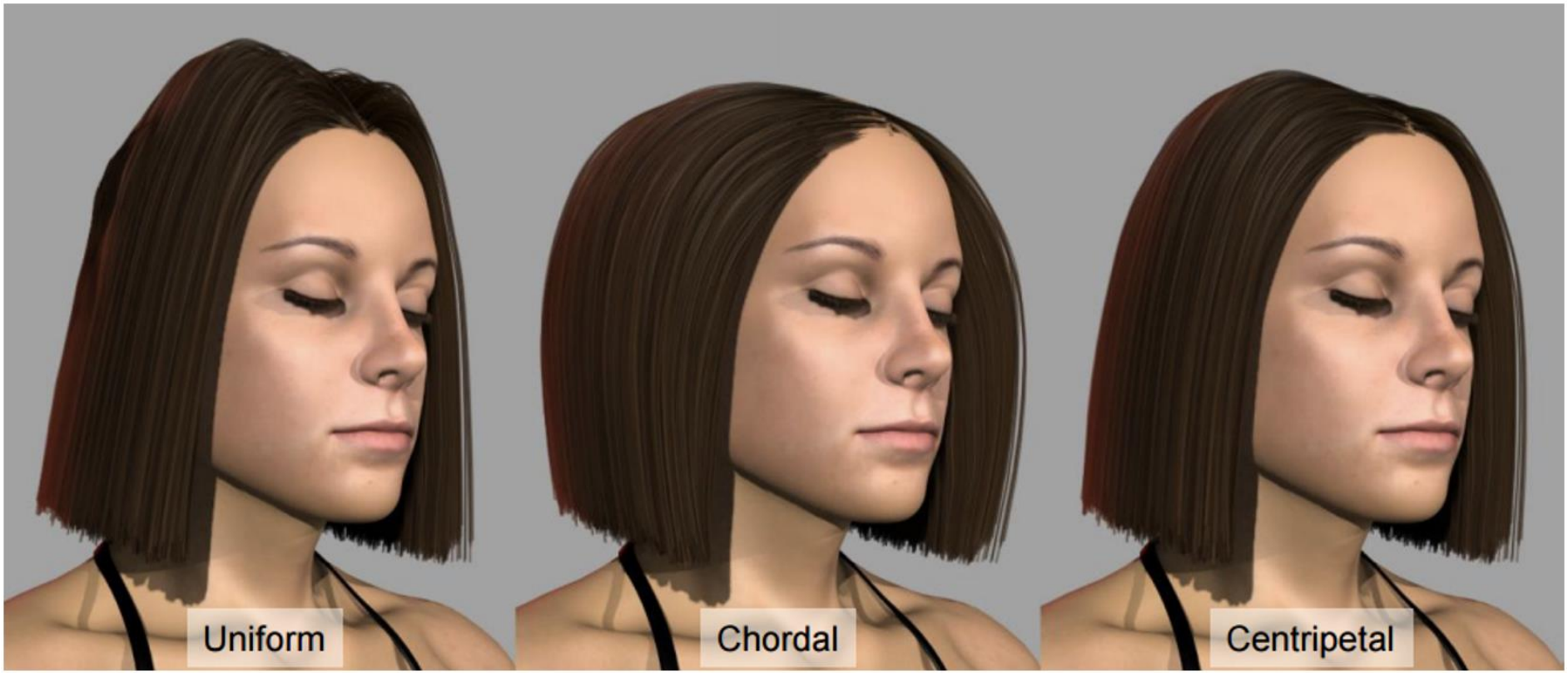
Uniform | Chordal | Centripetal

- Uniform parameterization overshoots and often generates cusps and intersections within short curve segments
- Chordal has the same problems for longer segments
- Centripetal is guaranteed to be loop/intersection-free and cusp-free and does not suffer from overshooting (tight)

Source: CENG477-METU

Spline Parametrization

- All from the same family: $||P_{k-1} - P_k||^{\alpha}$
- A hair design application of splines (from Hair Meshes paper, 2009)



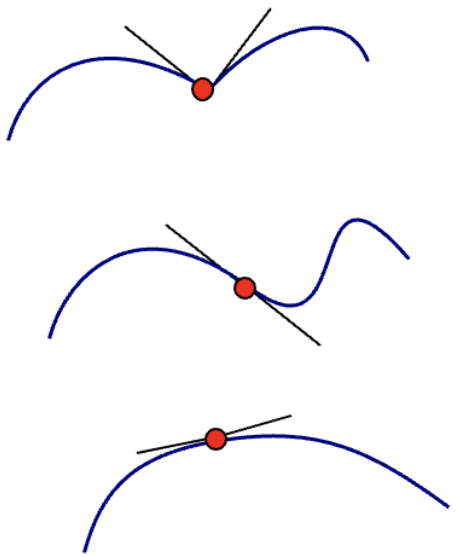
Source: CENG477-METU

Parametric Continuity

- Parametric equations:

  $x = x(u), y = y(u), z = z(u)$

- Parametric continuity: Continuity properties of curve segments.
  - Zero order: Curves intersects at one end-point: C^0
  - First order: C^0 *and* curves has same tangent at intersection: C^1
  - Second order: C^0, C^1 *and* curves has same second order derivative: C^2
    - Change in rate of change
    - E.g., position'' w.r.t. time is acceleration

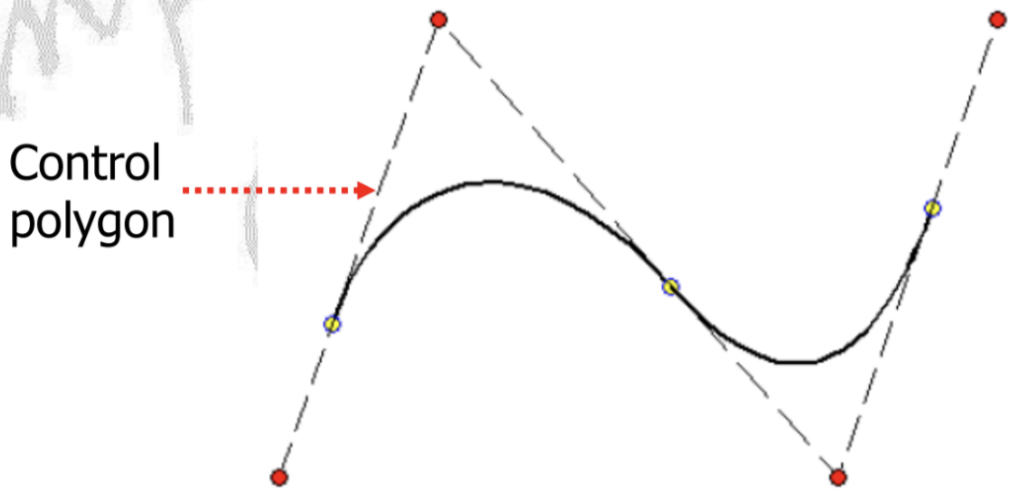


Rational Curves

- Both Bézier curves and B-splines are polynomial parametric curves.
- Polynomial parametric forms cannot exactly represent some simple curves such as circles.
- Introducing homogeneous coordinates (adding one dimension) makes them rational.
- Bézier curves and B-splines are generalized to rational Bézier curves and NURBS.
- Rational curves are more powerful as they can represent circles and ellipses.

What are "NURBS?"

- **NURBS** is an abbreviation for *non-uniform rational B-splines*.
  - **NU** or *non-uniform* implies that the knots may be placed at non-equal intervals.
  - **R** or *rational* refers to the underlying mathematical representation.
  - **BS** or *B-splines* refer to piecewise polynomial curves.



NURBS Curves

- NURBS curve is defined as:

$$\dot{\mathbf{p}}(t) = \frac{\sum_{i=0}^{n} \dot{\mathbf{p}}_i N_{i,p}(t) w_i}{\sum_{i=0}^{n} N_{i,p}(t) w_i}$$

- $\dot{\mathbf{p}}(t)$ is the curve point;
- $\dot{\mathbf{p}}_i$ is the control point i such that $i \in \{0, 1, 2, \ldots, n\}$;
- $n + 1$ is the number of control points;
- p is the curve degree;
- $N_{i,p}(t)$ is the B-spline basis function; and
- w_i is the weight of $\dot{\mathbf{p}}_i$ (the last entry in the homogeneous $\mathbf{p}_i$).

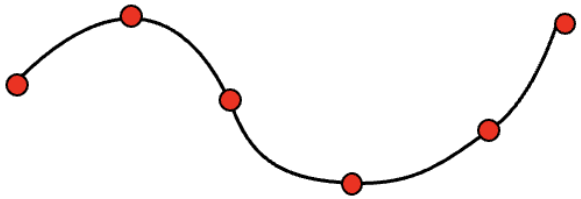
Summary

- Curves:
  - Polynomial equations
  - Representations of curves
  - Degree
  - Interpolating versus approximating curves
  - Bézier curves
  - B-spline curves
  - NURBS curves

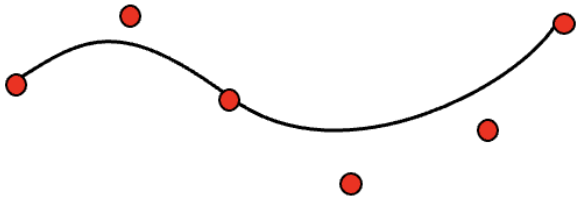
Bézier and Spline Curves

Bezier Curves

- So far our splines pass through the control points: interpolation
- Now they will pass close to the control points: approximation



Interpolated



Approximated

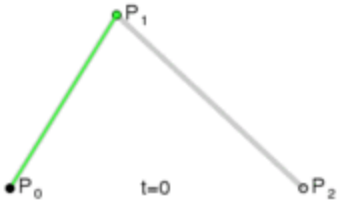
- Bezier curves is the most famous approximation curve
- Not a spline as there is no curve segments here; in fact locally changing one point affects the whole curve globally
- Introduced by Pierre Bezier from Renault Automobiles

Source: CENG477-METU

Bezier Curves

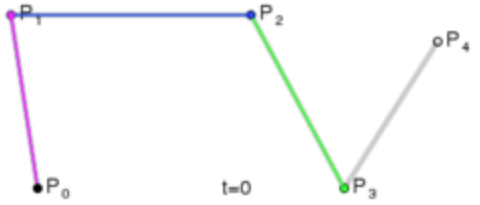
- Essentially it replaces the tangent vectors of cubic splines, that exist virtually but not quite visually, with supporting points from which curve does not pass

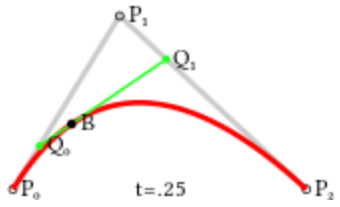
A quadratic Bezier curve

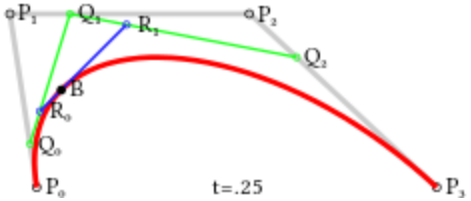


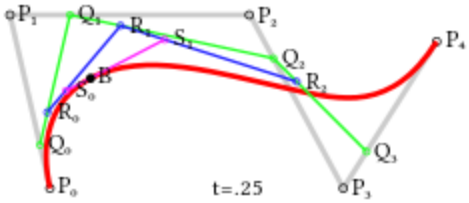
A cubic Bezier curve

A quartic Bezier curve





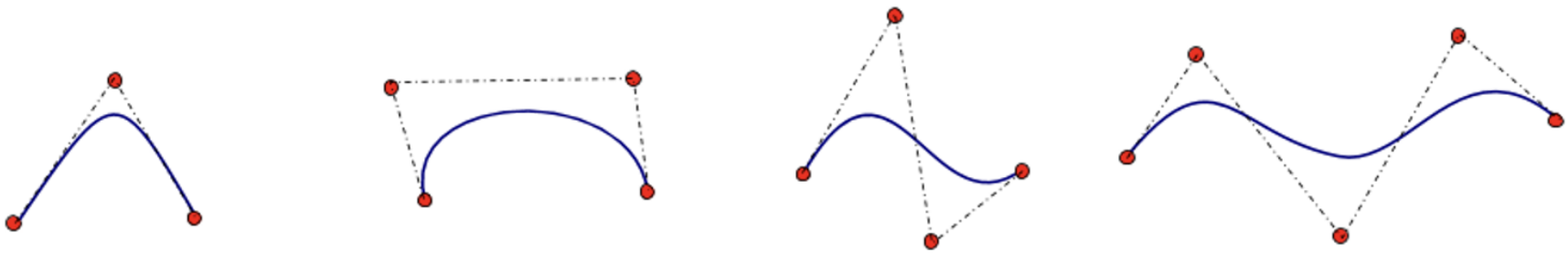




Source: CENG477-METU

Bezier Curves

- Hence the motivation is to design curves that are easy to specify
- Bezier curve captures the shape specified by control points
- Quadratic and cubic Bezier curves are the most common

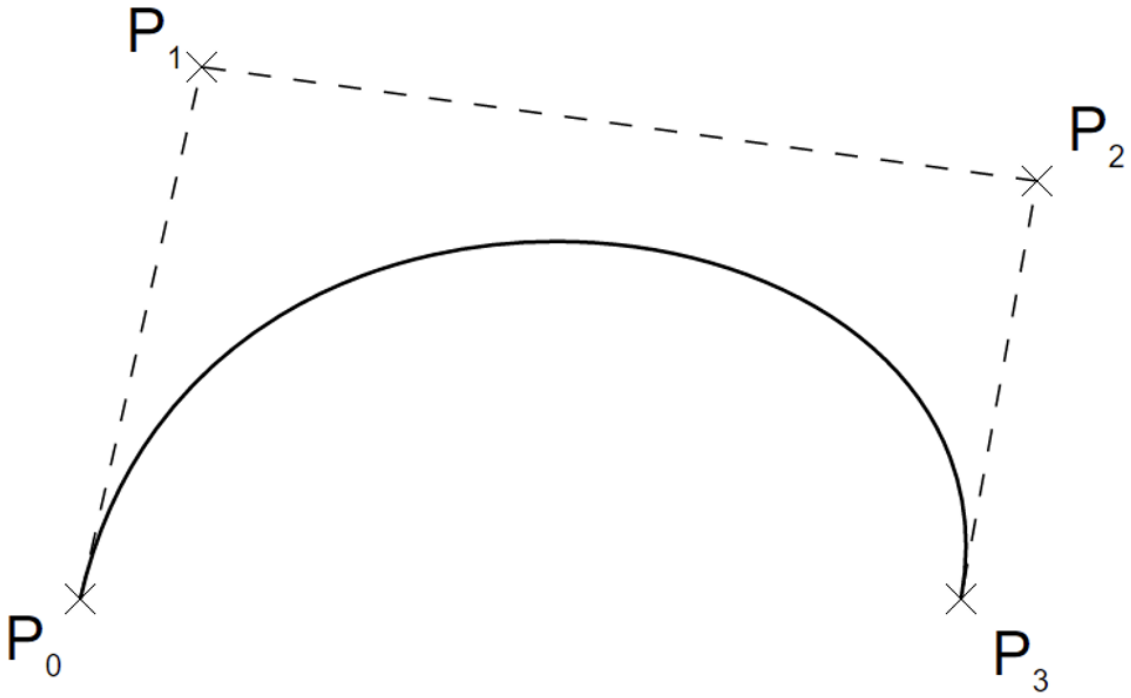


- Higher degree curvs are more computationally expensive to evaluate
- When more complex shapes are needed, low order Bezier curves are merged together, producing a composite Bezier curve or Bezier spline

Source: CENG477-METU

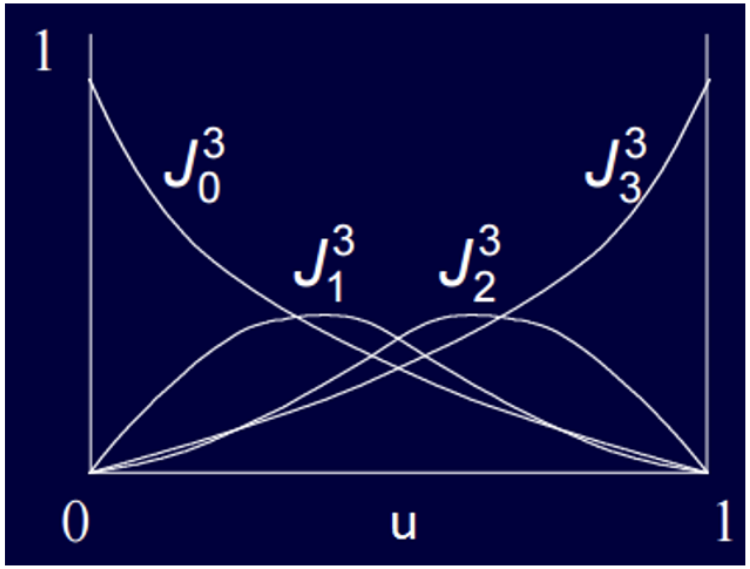
Bezier Curves

- Hence the motivation is to design curves that are easy to specify
- Bezier curve captures the shape specified by control points P_k
- $P(u) = \sum_0^n P_i J_i^n(u)$ where $0 \leq u \leq 1$ and $J_i^n(u)$ are called Bernstein basis/blending functions



Bernstein Polynomials

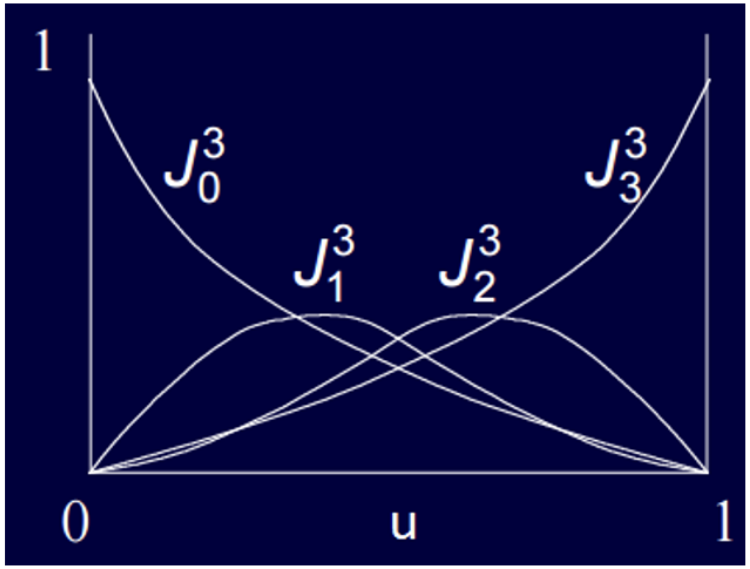
- $J_i^n(u)$ are the Bernstein basis/blending functions or polynomials
- $J_i^n(u) = C(n, i)u^i(1-u)^{n-i}$ where $C(n, i) = n! / (i!(n-i)!)$
- $J_0^0(u) = 1$
- $J_i^n(u) = 0$ for $i < 0$ or $i > n$
- $\Sigma_0^n J_i^n(u) = 1$
- $J_i^n(u) \geq 0$ for $0 \leq u \leq 1$
- For cubic Bezier n = 3 and we have:



Bernstein Polynomials

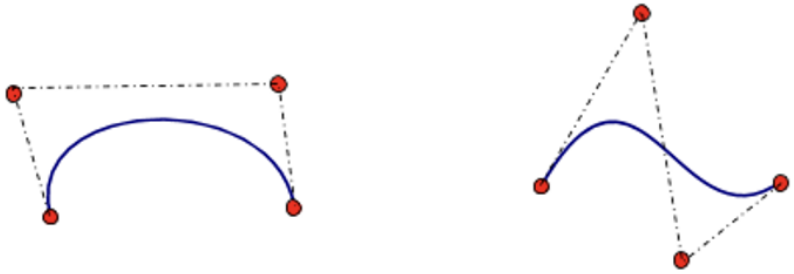
- $J_i^n(u)$ are the Bernstein basis/blending functions or polynomials
- $J_i^n(u) = C(n, i)u^i(1-u)^{n-i}$ where $C(n, i) = n! / (i!(n-i)!)$

- For cubic Bezier n = 3 and we have:
- $J_0^3(u) = u^0(1-u)^3 = (1-u)^3$
- $J_1^3(u) = 3u(1-u)^3$
- $J_2^3(u) = 3u^2(1-u)$
- $J_3^3(u) = u^3$
- $P(u) = P_0J_0^3 + P_1J_1^3 + P_2J_2^3 + P_3J_3^3$

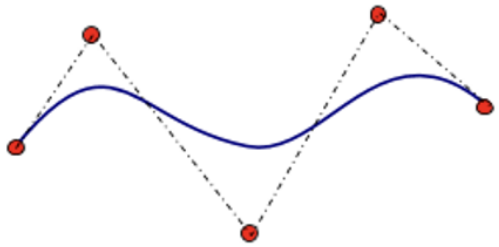


Bezier Curve

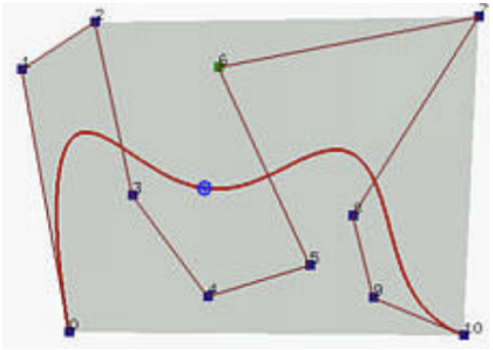
- Degree/order of the Bezier curve is # of control points minus 1
- Degree 3 (cubic)



- Degree 4 (quartic)



- Degree 10 (11 control points)

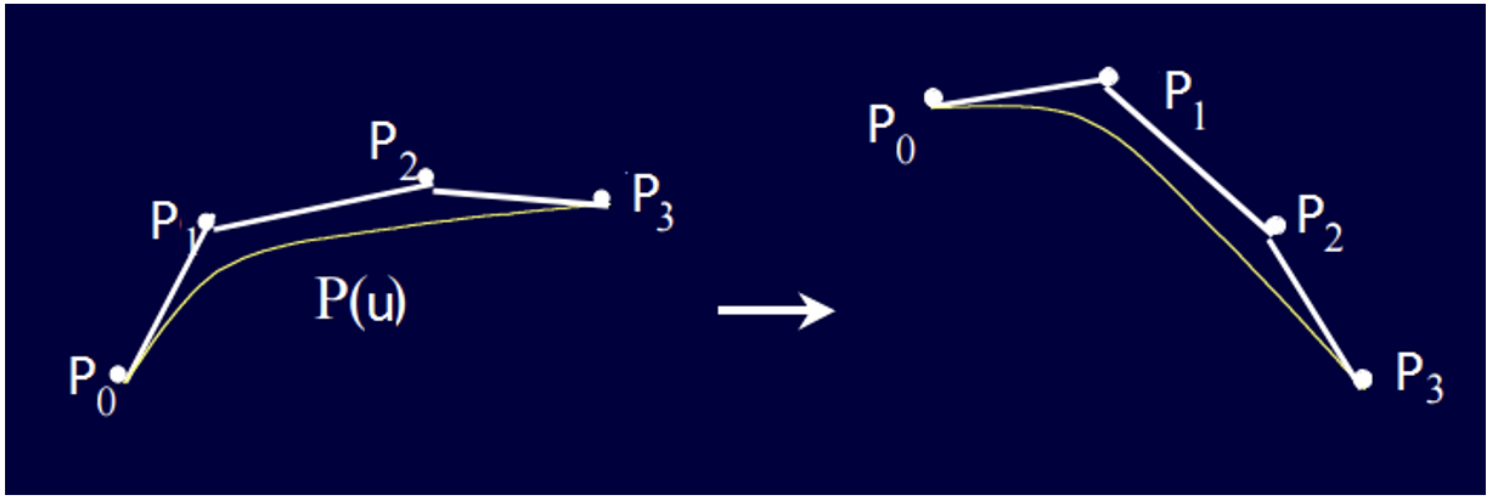


Bezier Curve

- End point interpolation property
  - Bezier curve (of any degree) interpolates the 2 endpoints (passes through)
  - Recall $J_i^n(u) = C(n, i)u^i(1-u)^{n-i}$
  - At u=0 (first endpoint)
  - $i = 0, J_0^n(0) = C(n, 0)u^0(1-u)^{n-0} = 1$
  - $i \neq 0, J_i^n(0) = C(n, i)0^i(1-0)^{n-i} = 0$
  - $P(0) = P_0 J_0^n(0) = P_0$
  - At u=1 (second endpoint)
  - $i = n, J_n^n(1) = C(n, n)1^n(1-u)^0 = 1$
  - $i \neq n, J_i^n(1) = C(n, i)1^i(1-1)^{n-i} = 0$
  - $P(1) = P_n J_n^n(1) = P_n$

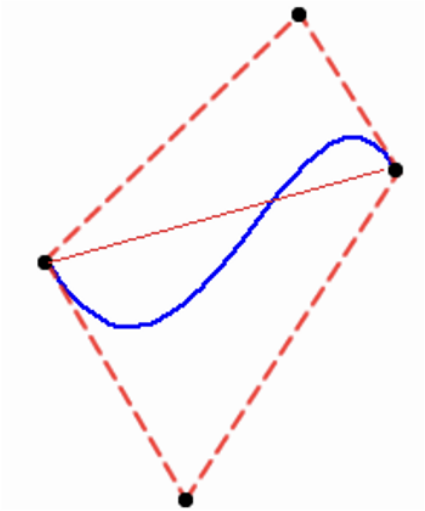
Bezier Curve

- Affine invariance property
  - Applying an affine transformation M to the curve is equivalent to applying the transformation to the control points, and vice versa
  - Useful property because it says it is enough to transform the control points



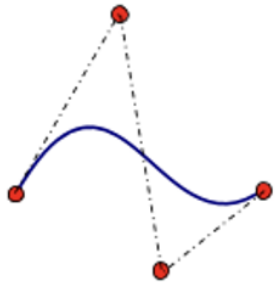
Bezier Curve

- Convex hull property
  - Bezier curve lies in the convex hull of the control points P_k
  - Useful property because intersection/collision computation of one curve with another (or anything designed with this curve) is accelerated by pruning



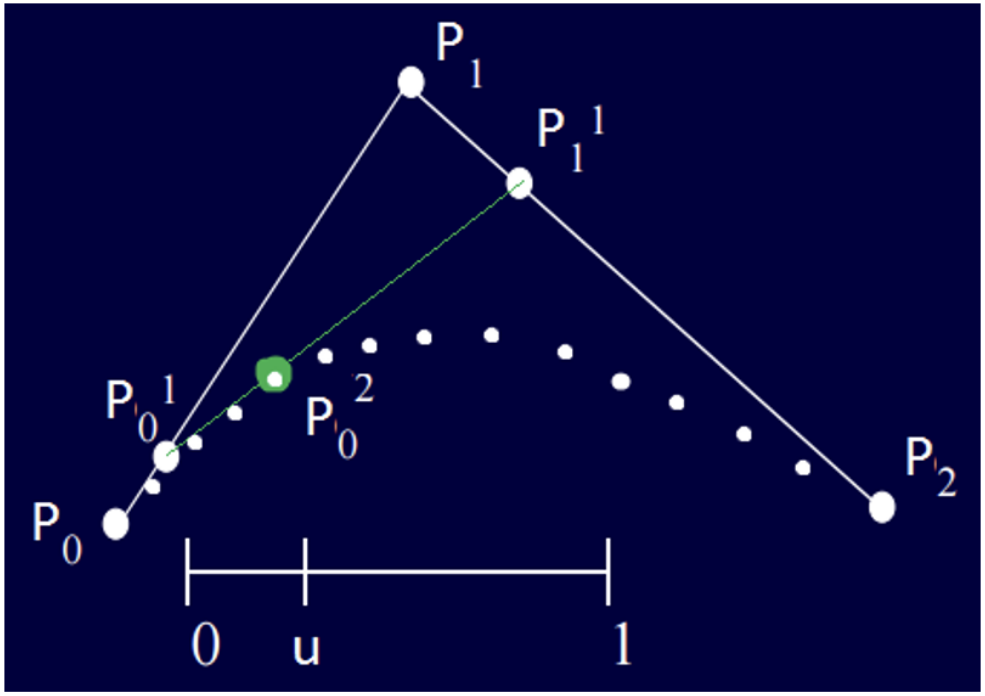
Bezier Curve

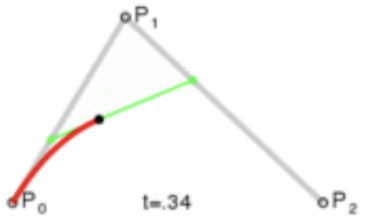
- Symmetry property
  - P(u) defined by P_0, P_1, .., P_n is equal to P(1-u) defined by P_n, P_{n-1}, .., P_0
  - $\Sigma_0^n P_i J_i^n(u) = \Sigma_0^n P_{n-i} J_i^n(1-u)$

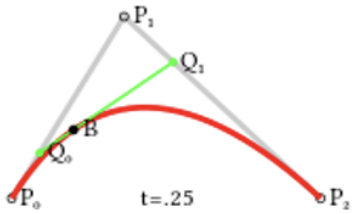


Bezier Curve

- Construction by de Casteljau Algorithm
- Idea: repetitive linear interpolation







- $P_0^1(u) = (1-u)P_0 + uP_1 \quad P_1^1(u) = (1-u)P_1 + uP_2$
- $P_0^2(u) = (1-u)P_0^1(u) + uP_1^1(u) = (1-u)^2P_0 + 2u(1-u)P_1 + u^2P_2$
- Bernstein polynomials of degree 2; so what I get is a Bezier curve ☺

Bezier Surface

- Construction by Bernstein polynomials' Cartesian product
- A surface compute this way is also known as a bicubic Bezier patch
- 32 such bicubic Bezier patches generate this teapot