# Introduction to Computer Graphics

# – Modeling (1) –

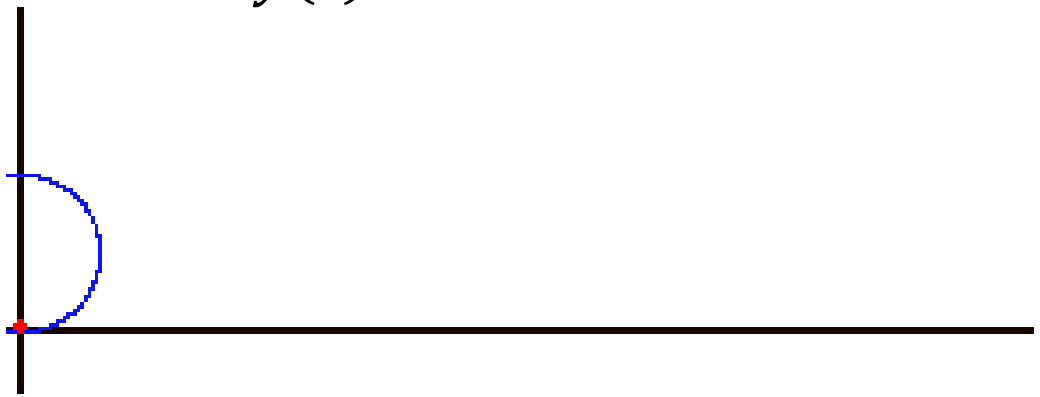
April 13, 2017

Kenshi Takayama

# Parametric curves

- X & Y coordinates defined by parameter t (≅ time)
  - Example: Cycloid

$$x(t) = t - \sin t$$
$$y(t) = 1 - \cos t$$



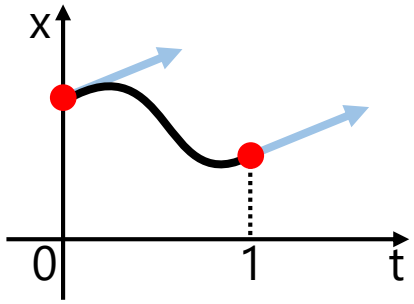
- Tangent (aka. derivative, gradient) vector: $\left(x'(t), y'(t)\right)$

- Polynomial curve: $x(t) = \sum_i a_i t^i$

# Cubic Hermite curves

- Cubic polynomial curve interpolating derivative constraints at both ends (Hermite interpolation)

$$x(0) = x_0$$
$$x(1) = x_1$$
$$x'(0) = x'_0$$
$$x'(1) = x'_1$$




- 4 constraints ➔ 4 DoF needed
  ➔ 4 coefficients ➔ cubic
  - $x(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3$
  - $x'(t) = a_1 + 2a_2 t + 3a_3 t^2$

- Coeffs determined by substituting constrained values & derivatives

$$x(0) = a_0 = x_0$$
$$x(1) = a_0 + a_1 + a_2 + a_3 = x_1$$
$$x'(0) = a_1 = x'_0$$
$$x'(1) = a_1 + 2\,a_2 + 3\,a_3 = x'_1$$

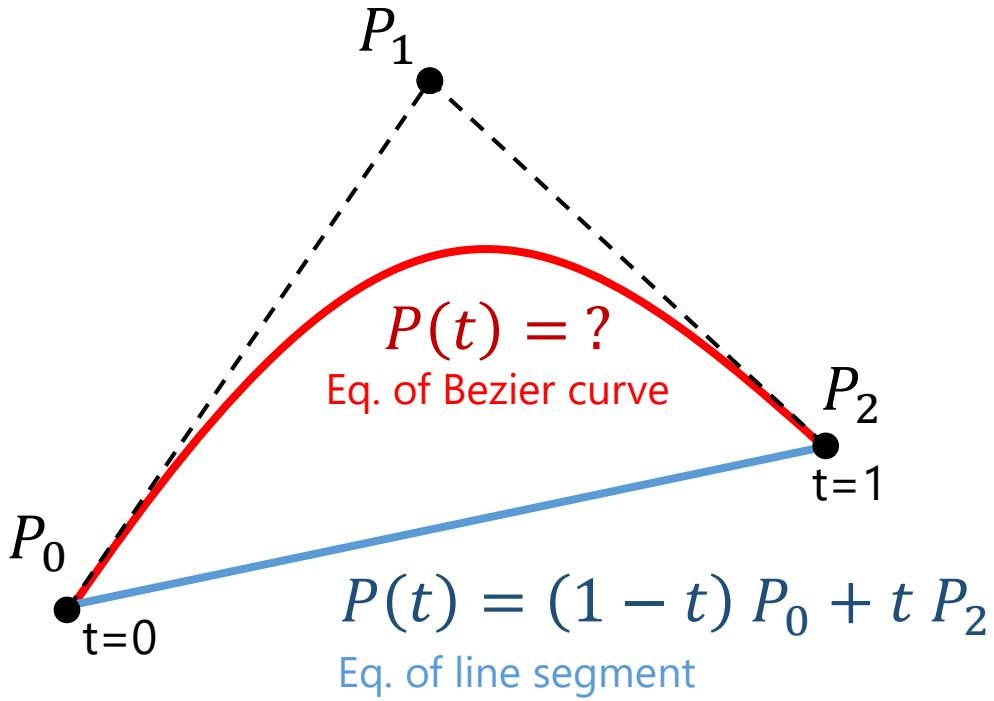
➔

$$a_0 = x_0$$
$$a_1 = x'_0$$
$$a_2 = -3\,x_0 + 3\,x_1 - 2\,x'_0 - x'_1$$
$$a_3 = 2\,x_0 - 2\,x_1 + x'_0 + x'_1$$

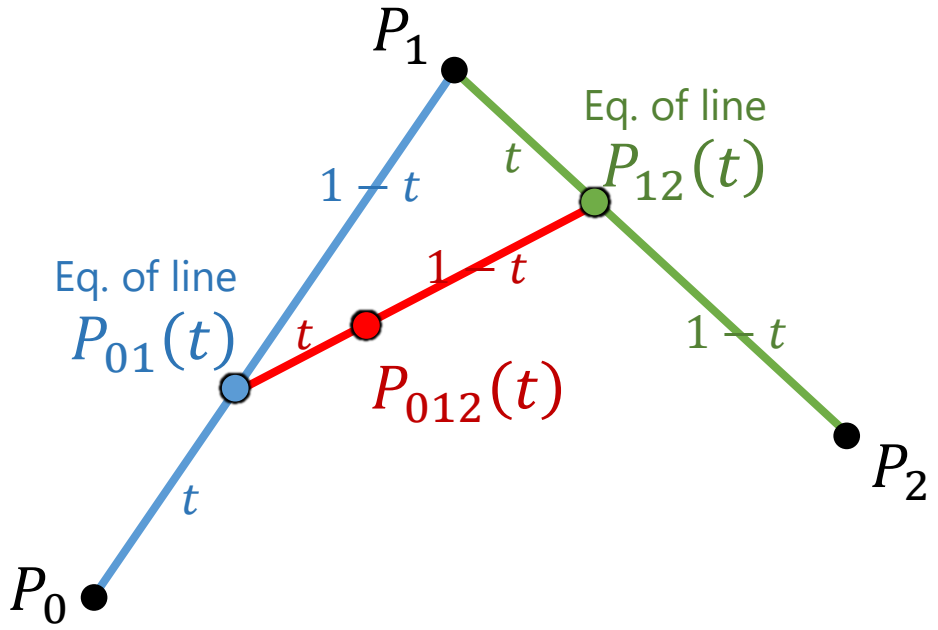
# Bezier curves

- Input: 3 control points $P_0, P_1, P_2$
  - Coordinates of points in arbitrary domain (2D, 3D, …)

- Output: Curve $P(t)$ satisfying
  $P(0) = P_0$
  $P(1) = P_2$
  while being "pulled" by $P_1$



# Bezier curves




- $P_{01}(t) = (1-t)P_0 + t\,P_1$
- $P_{12}(t) = (1-t)P_1 + t\,P_2$
  - $P_{01}(0) = P_0$
  - $P_{12}(1) = P_2$

- Idea: "Interpolate the interpolation"
  As $t$ changes $0 \rightarrow 1$ , smoothly transition from $P_{01}$ to $P_{12}$

- $P_{012}(t) = (1-t)P_{01}(t) + t\,P_{12}(t)$

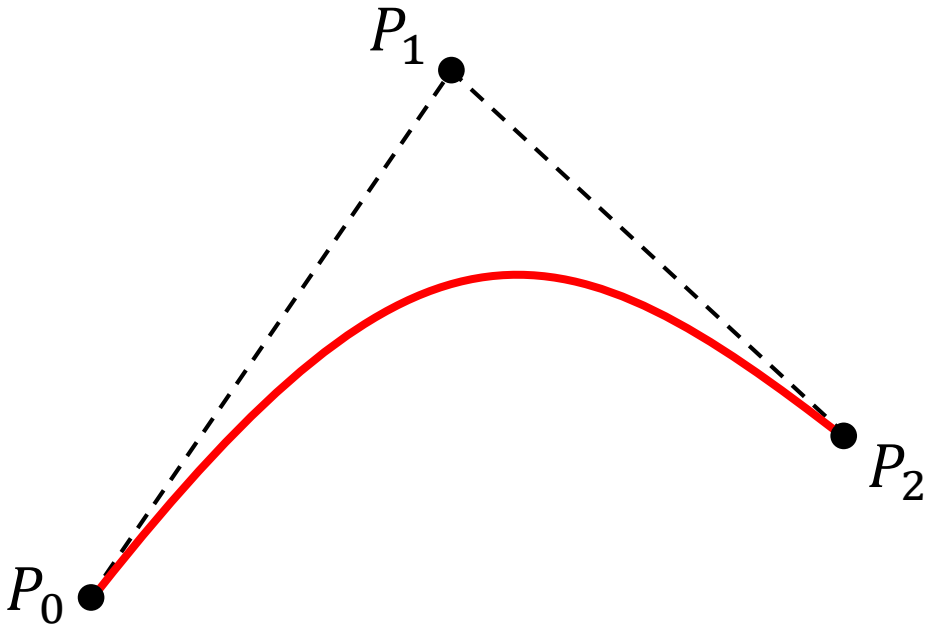
$$= (1-t)\{(1-t)P_0 + t\,P_1\} + t\,\{(1-t)P_1 + t\,P_2\}$$

$$= \underline{(1-t)^2P_0 + 2t(1-t)P_1 + t^2P_2}$$

Quadratic Bezier curve

# Bezier curves

- $P_{01}(t) = (1-t)P_0 + t\,P_1$
- $P_{12}(t) = (1-t)P_1 + t\,P_2$
  - $P_{01}(0) = P_0$
  - $P_{12}(1) = P_2$




- Idea: "Interpolate the interpolation"
  As $t$ changes $0 \rightarrow 1$ , smoothly transition from $P_{01}$ to $P_{12}$

- $P_{012}(t) = (1-t)P_{01}(t) + t\,P_{12}(t)$

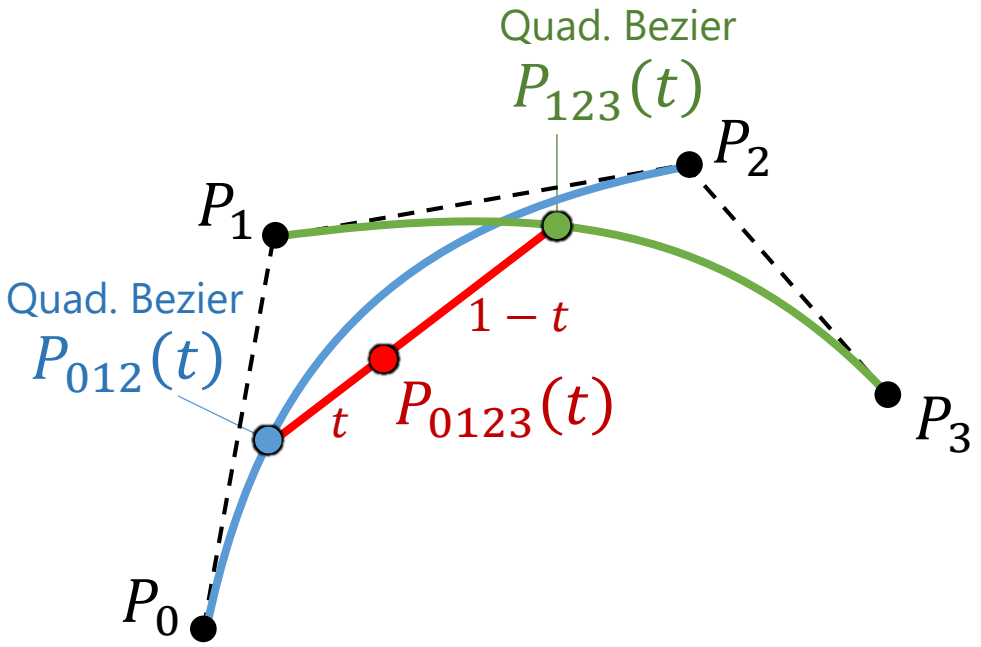
$$= (1-t)\{(1-t)P_0 + t\,P_1\} + t\,\{(1-t)P_1 + t\,P_2\}$$

$$= \underline{(1-t)^2P_0 + 2t(1-t)P_1 + t^2P_2}$$

Quadratic Bezier curve

# Cubic Bezier curve

- Exact same idea applied to 4 points $P_0, P_1, P_2\ P_3$:
  - As $t$ changes $0 \rightarrow 1$, transition from $P_{012}$ to $P_{123}$




- $P_{0123}(t) = (1-t)P_{012}(t) + t\ P_{123}(t)$

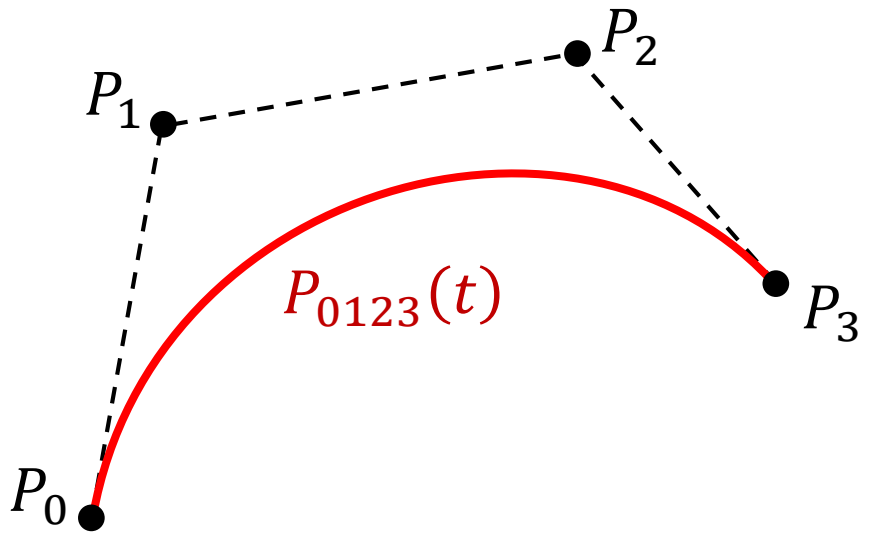
$$= (1-t)\{(1-t)^2P_0 + 2t(1-t)P_1 + t^2P_2\} + t\ \{(1-t)^2P_1 + 2t(1-t)P_2 + t^2P_3\}$$

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

Cubic Bezier curve

# Cubic Bezier curve

- Exact same idea applied to 4 points $P_0, P_1, P_2\ P_3$:
  - As $t$ changes $0 \rightarrow 1$, transition from $P_{012}$ to $P_{123}$




- $P_{0123}(t) = (1-t)P_{012}(t) + t\,P_{123}(t)$

$$= (1-t)\{(1-t)^2P_0 + 2t(1-t)P_1 + t^2P_2\} + t\,\{(1-t)^2P_1 + 2t(1-t)P_2 + t^2P_3\}$$

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

Cubic Bezier curve

- Can easily control tangent at endpoints ➔ ubiquitously used in CG

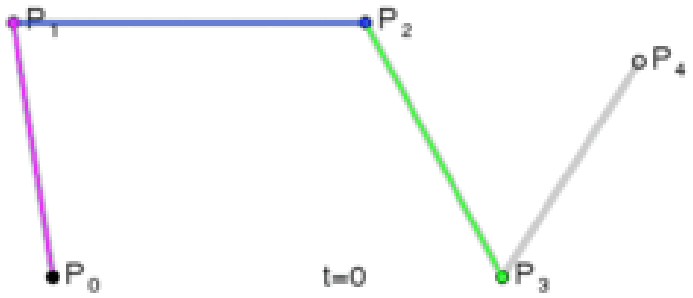
# n-th order Bezier curve

- Input: n+1 control points $P_0, \cdots, P_n$

$$P(t) = \sum_{i=0}^{n} \underbrace{{}_n\mathrm{C}_i \, t^i \, (1-t)^{n-i}}_{b_i^n(t)} \, P_i$$

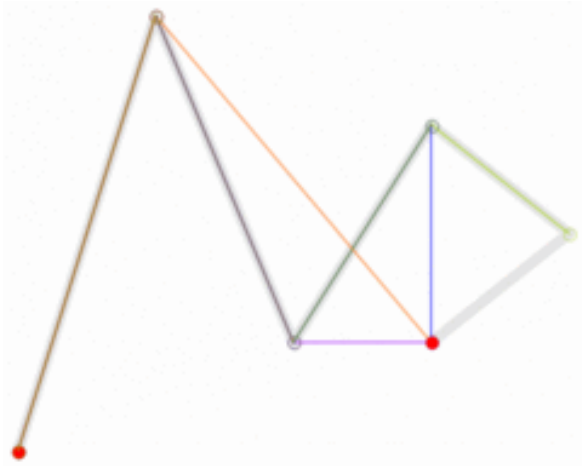
Bernstein basis function

Quartic (4th)




$$(1-t)^4 P_0 + 4t(1-t)^3 P_1 + 6t^2(1-t)^2 P_2 + 4t^3(1-t) P_3 + t^4 P_4$$

Quintic (5th)



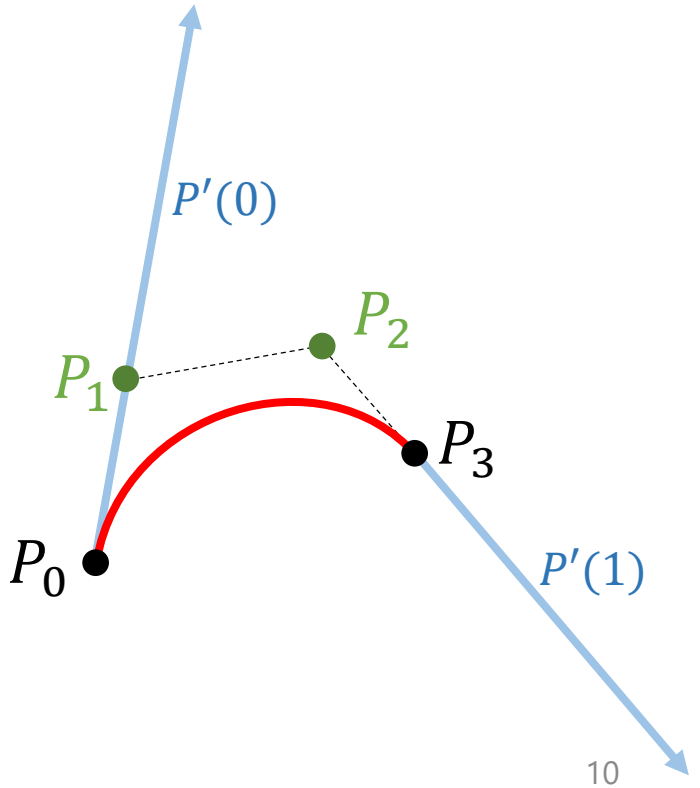
$$(1-t)^5 P_0 + 5t(1-t)^4 P_1 + 10t^2(1-t)^3 P_2 + 10t^3(1-t)^2 P_3 + 5t^4(1-t) P_4 + t^5 P_5$$

# Cubic Bezier curves & cubic Hermite curves

- Cubic Bezier curve & its derivative:
  - $P(t) = (1-t)^3 P_0 + 3t(1-t)^2 P_1 + 3t^2(1-t)P_2 + t^3 P_3$
  - $P'(t) = -3(1-t)^2 P_0 + 3\{(1-t)^2 - 2t(1-t)\}P_1 + 3\{2t(1-t) - t^2\}P_2 + 3t^2 P_3$

- Derivatives at endpoints:
  - $P'(0) = -3P_0 + 3P_1$ ➔ $P_1 = P_0 + \frac{1}{3}P'(0)$
  - $P'(1) = -3P_2 + 3P_3$ ➔ $P_2 = P_3 - \frac{1}{3}P'(1)$

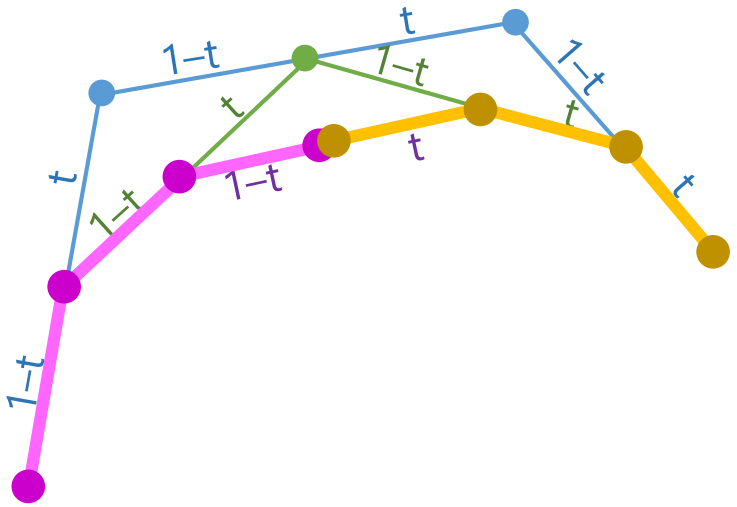
- Different ways of looking at cubic curves, essentially the same



# Evaluating Bezier curves

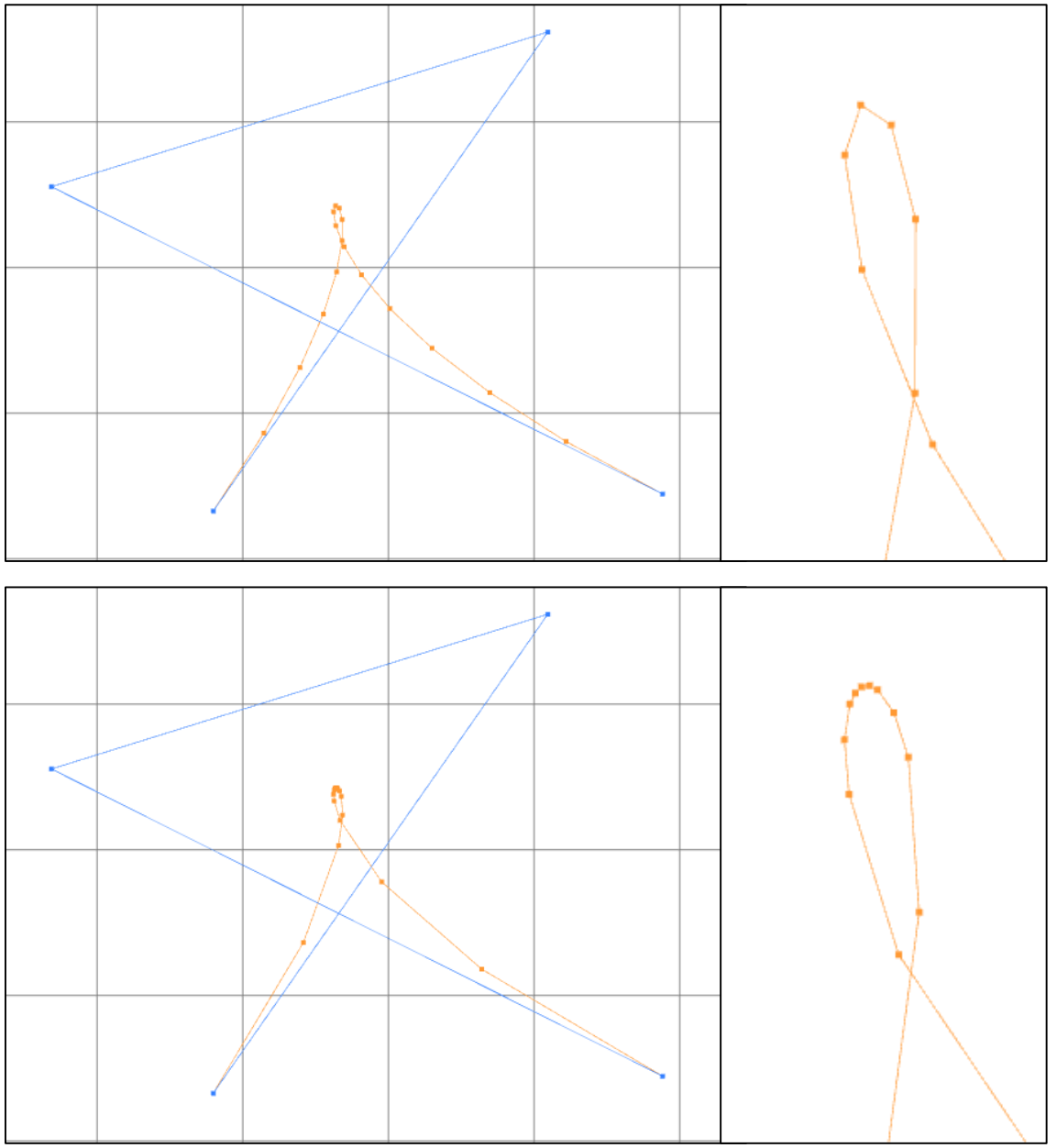
- Method 1: Direct evaluation of polynomials
  - Simple & fast☺, could be numerically unstable☹

- Method 2: de Casteljau's algorithm
  - Directly after the recursive definition of Bezier curves
  - More computation steps☹, numerically stable☺
  - Also useful for splitting Bezier curves



# Drawing Bezier curves

- In the end, everything is drawn as polyline
  - Main question: How to sample paramter t?
- Method 1: Uniform sampling
  - Simple
  - Potentially insufficient sampling density
- Method 2: Adaptive sampling
  - If control points deviate too much from straight line, split by de Casteljau's algorithm

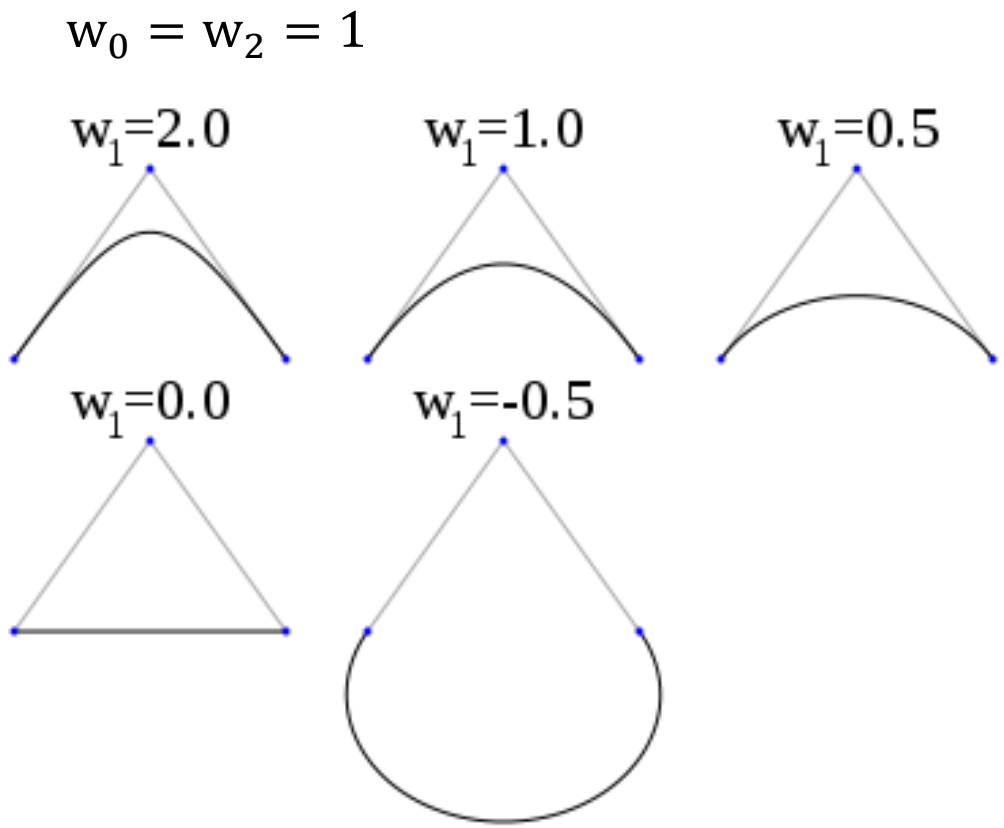


# Further control: Rational Bezier curve

- Another view on Bezier curve:
  "Weighted average" of control points
  - $P_{012}(t) = (1-t)^2 P_0 + 2t(1-t)P_1 + t^2 \quad P_2$
    $= \lambda_0(t) \quad P_0 + \lambda_1(t) \quad P_1 + \lambda_2(t) P_2$
  - Important property: **partition of unity**
    $\lambda_0(t) + \lambda_1(t) + \lambda_2(t) = 1 \quad \forall t$

- Multiply each $\lambda_i(t)$ by arbitrary coeff $w_i$:
  $\xi_i(t) = w_i \, \lambda_i(t)$

- Normalize to obtain new weights:
  $\lambda_i'(t) = \frac{\xi_i(t)}{\sum_j \xi_j(t)}$



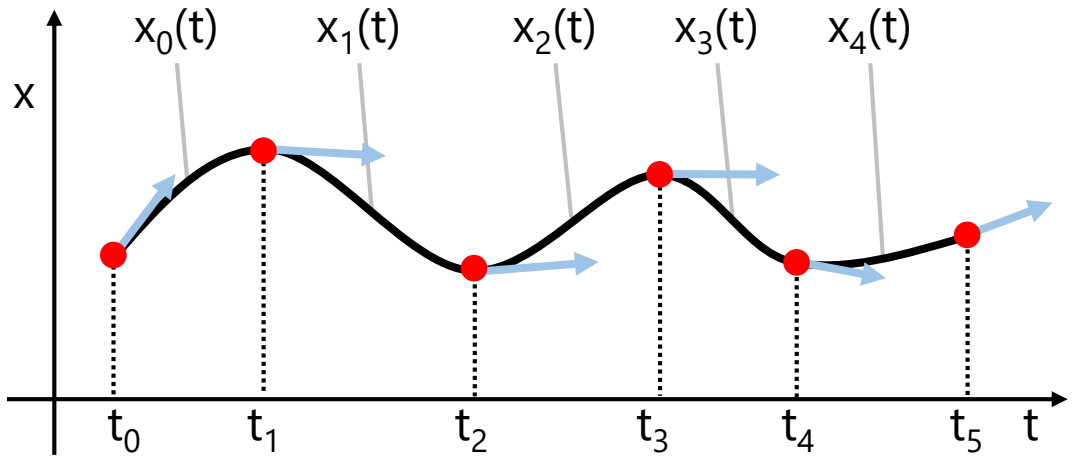

Non-polynomial curve ➔ can represent arcs etc.

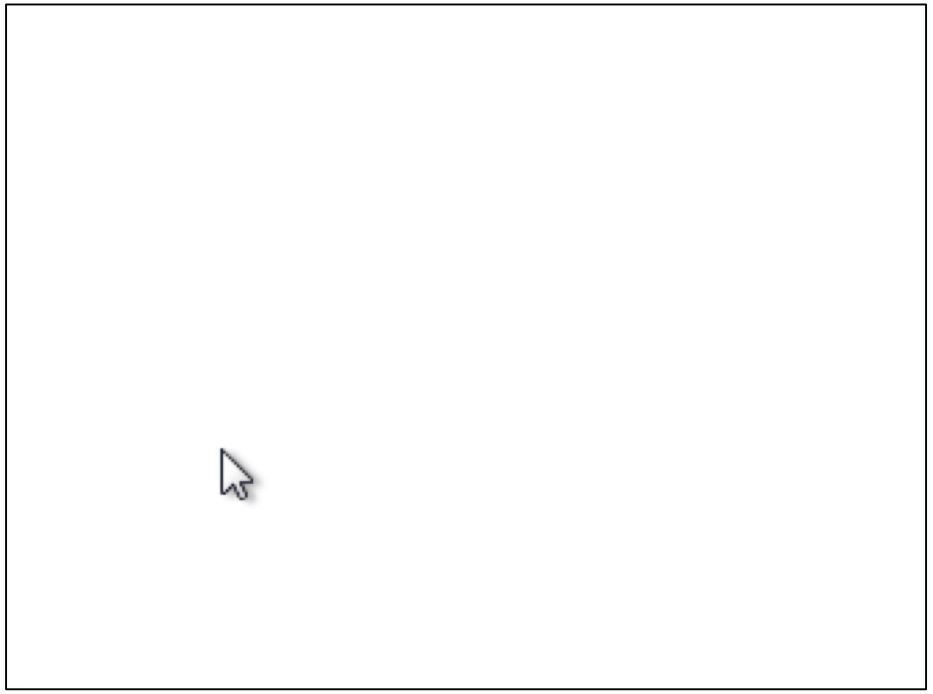
# Cubic splines

- Series of connected cubic curves
  - Piecewise-polynomial
  - Share value & derivative at every transition of intervals ($C^1$ continuity)

- Parameter range can be other than $[0, 1]$
  - Assumption: $t_k < t_{k+1}$

- Given values as only input,
  we want to automatically set derivatives

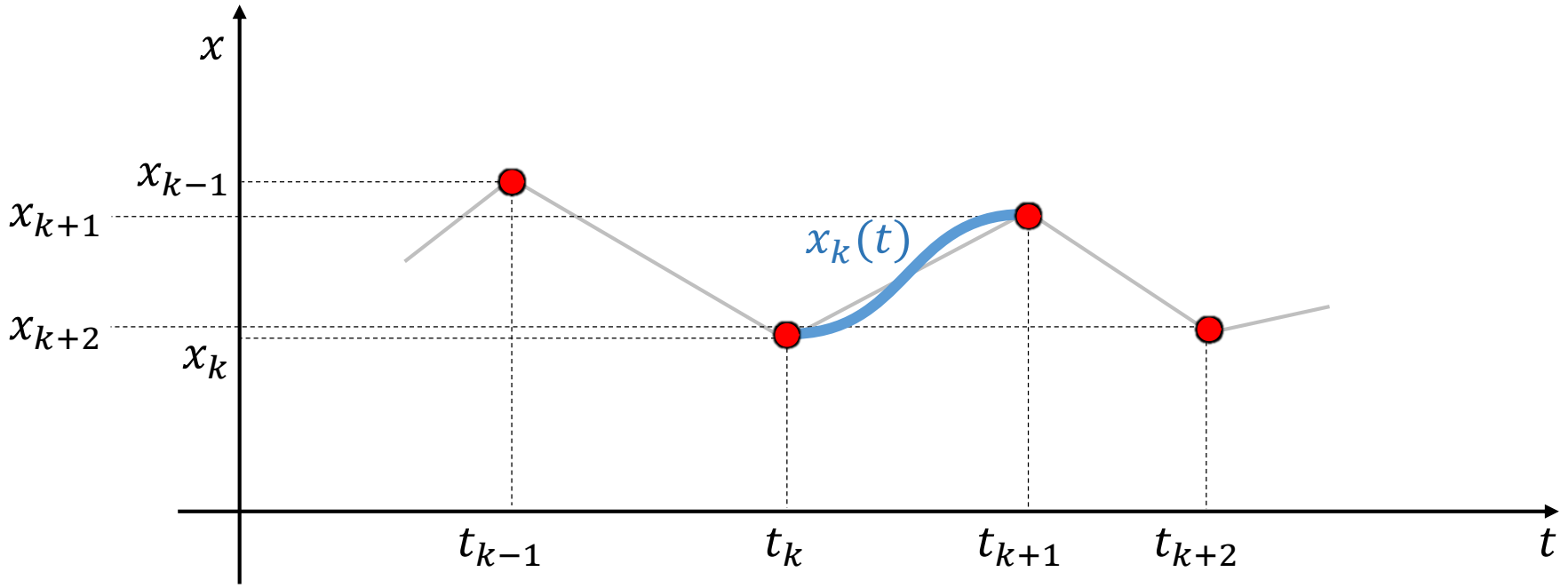


Curve tool in PowerPoint

# Cubic Catmull-Rom spline

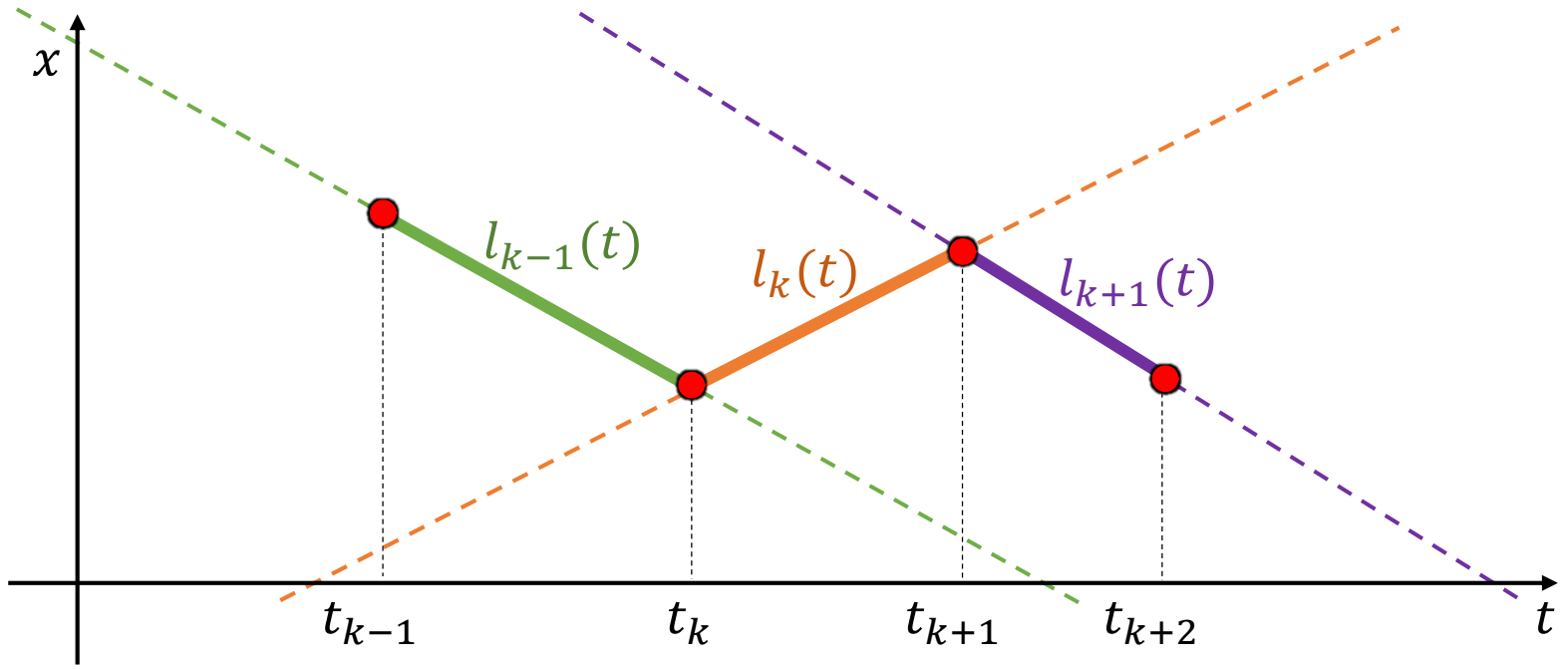
- Cubic function $x_k(t)$ for range $t_k \leq t \leq t_{k+1}$ is defined by adjacent constrained values $x_{k-1}$, $x_k$, $x_{k+1}$, $x_{k+2}$



# Cubic Catmull-Rom spline: Step 1

- As $t_k \to t_{k+1}$, interpolate such that $x_k \to x_{k+1}$ ➔ Line

$$l_k(t) = \left(1 - \frac{t - t_k}{t_{k+1} - t_k}\right) x_k + \frac{t - t_k}{t_{k+1} - t_k} x_{k+1}$$

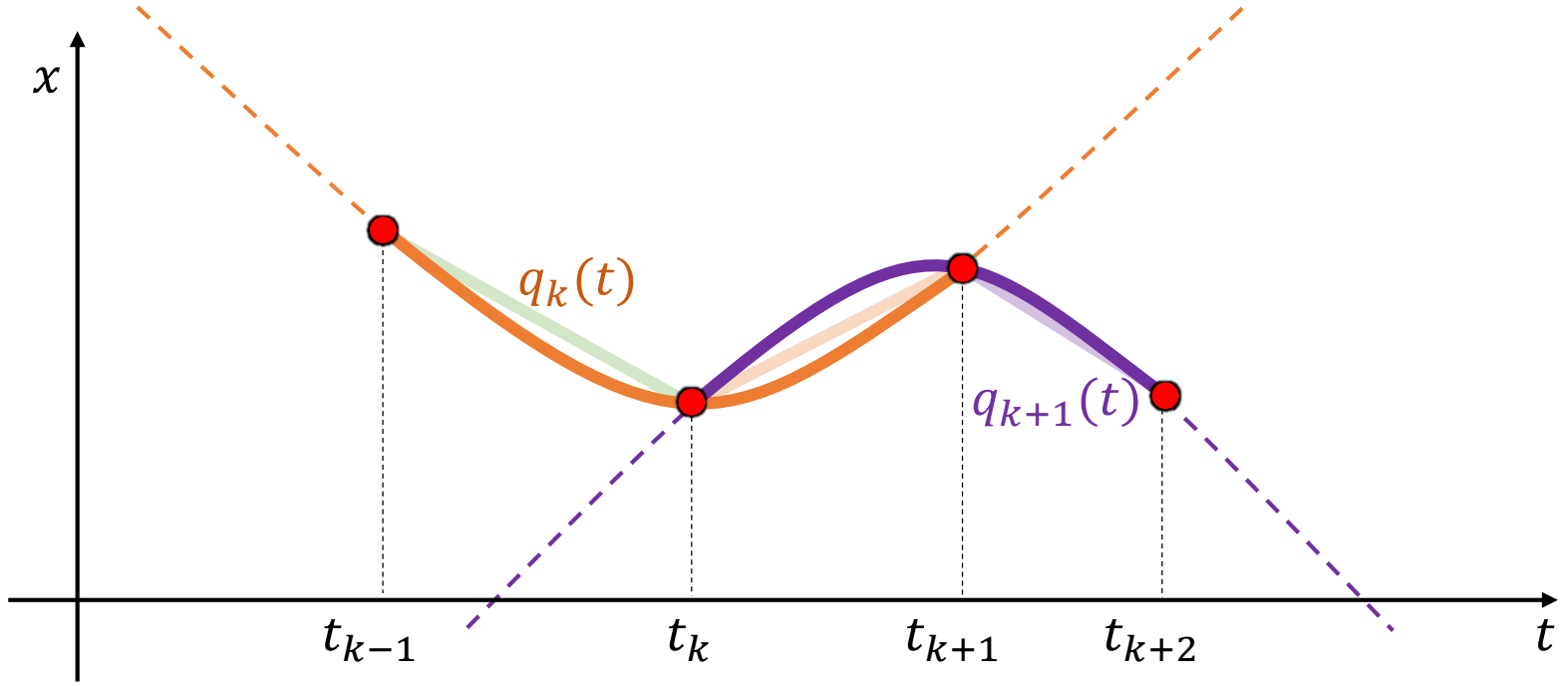


# Cubic Catmull-Rom spline: Step 2

- As $t_{k-1} \to t_{k+1}$, interpolate such that $l_{k-1} \to l_k$ ➔ Quadratic curve

$$q_k(t) = \left(1 - \frac{t - t_{k-1}}{t_{k+1} - t_{k-1}}\right) l_{k-1}(t) + \frac{t - t_{k-1}}{t_{k+1} - t_{k-1}} l_k(t)$$

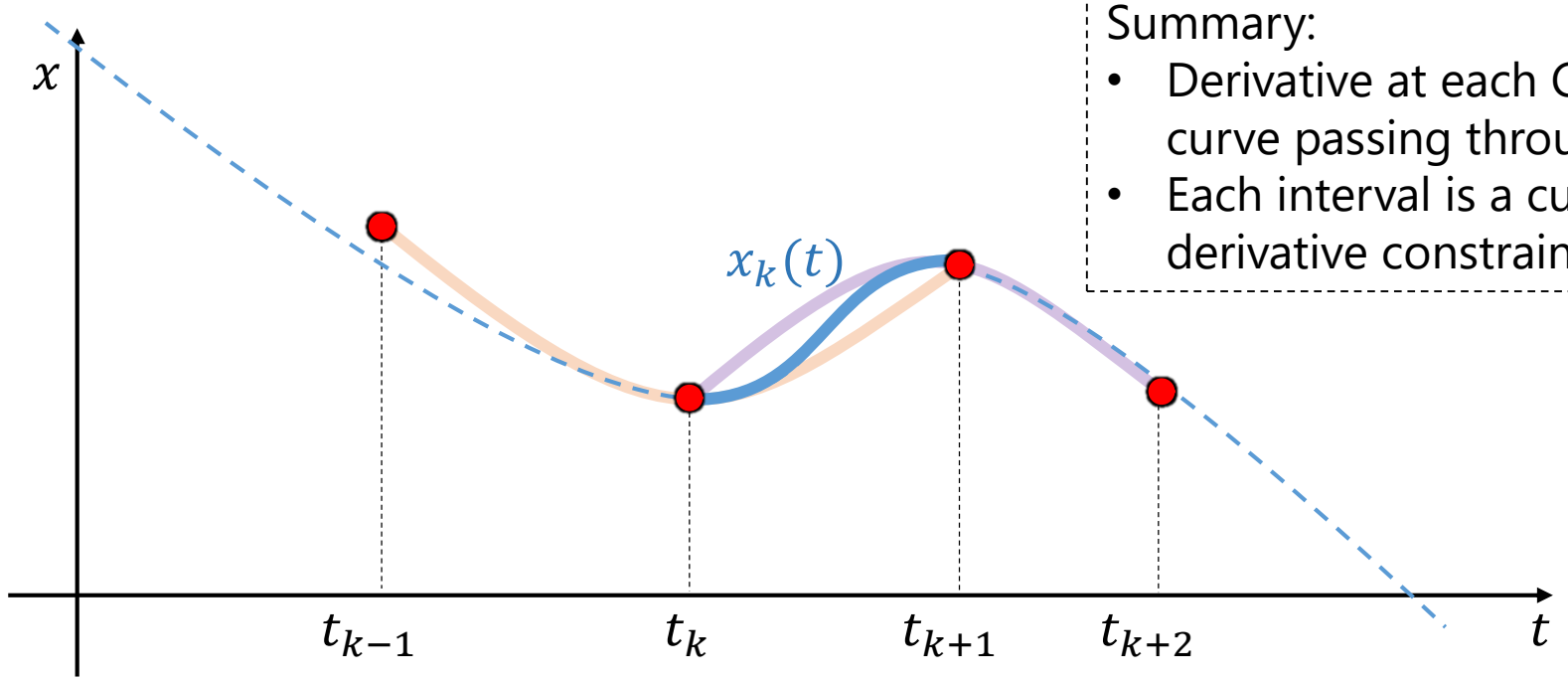
  - Passes through 3 points $(t_{k-1}, x_{k-1}), (t_k, x_k), (t_{k+1}, x_{k+1})$



# Cubic Catmull-Rom spline: Step 3

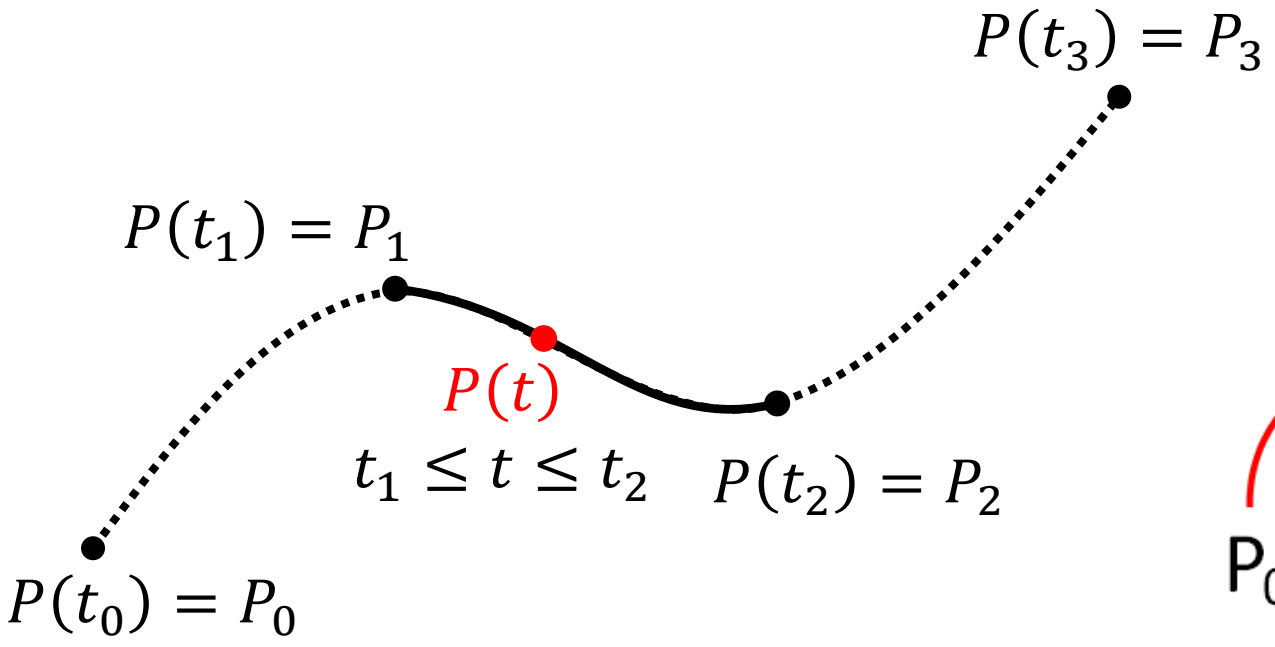
- As $t_k \to t_{k+1}$, interpolate such that $q_k \to q_{k+1}$ ➔ Cubic curve

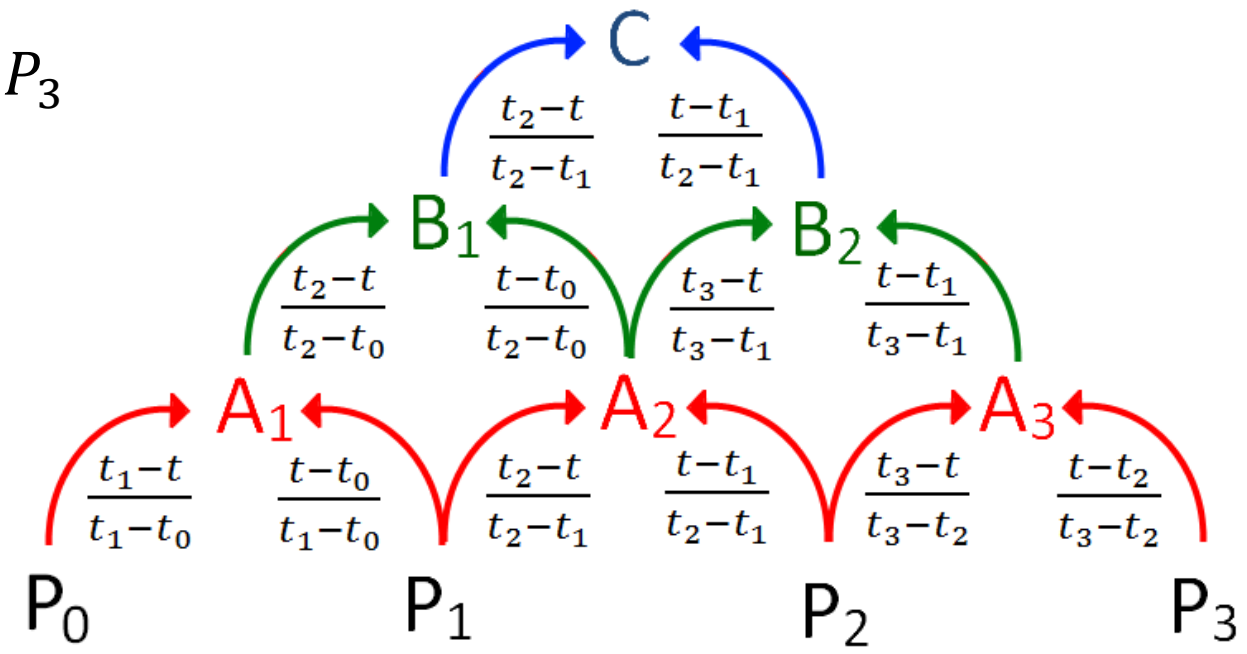
$$x_k(t) = \left(1 - \frac{t - t_k}{t_{k+1} - t_k}\right) q_k(t) + \frac{t - t_k}{t_{k+1} - t_k} q_{k+1}(t)$$




Summary:
- Derivative at each CP is defined by a quadratic curve passing through its adjacent CPs
- Each interval is a cubic curve satisfying derivative constraints at both ends

# Evaluating cubic Catmull-Rom spline

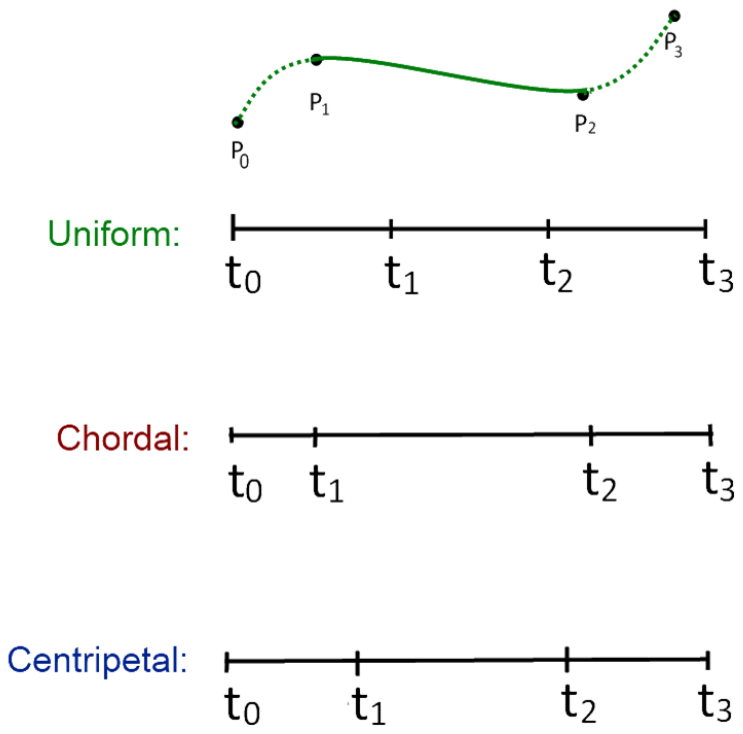


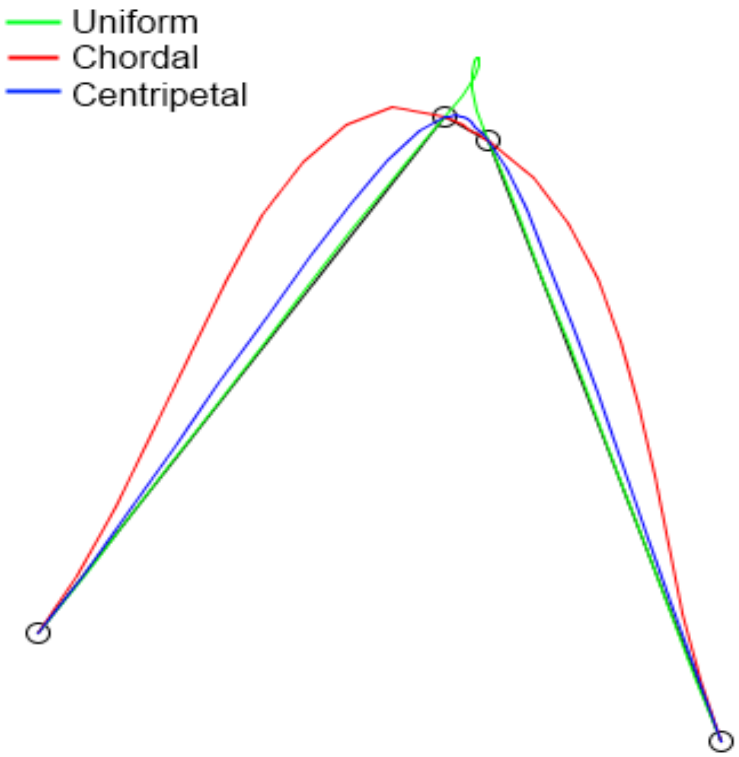




A recursive evaluation algorithm for a class of Catmull-Rom splines [Barry,Colgman,SIGGRAPH88]

# Ways of setting parameter values $t_k$ (aka. knot sequence)

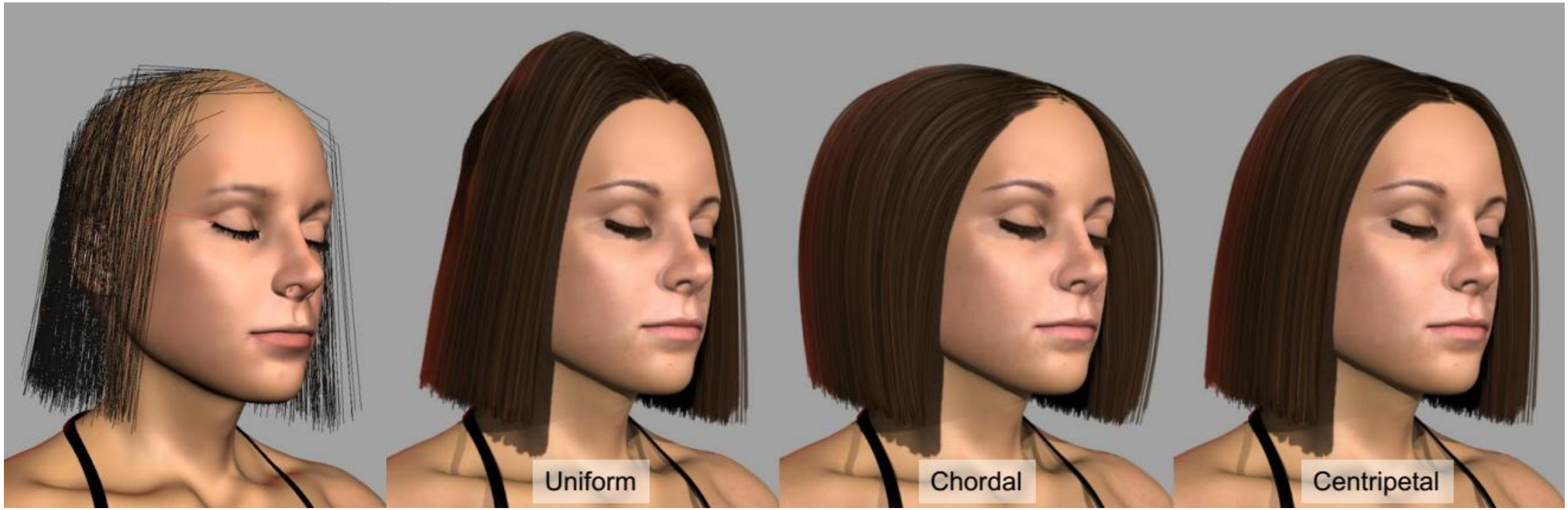
- Assume: $t_0 = 0$
- Uniform
  $t_k = t_{k-1} + 1$
- Chordal
  $t_k = t_{k-1} + |P_{k-1} - P_k|$
- Centripetal
  $t_k = t_{k-1} + \sqrt{|P_{k-1} - P_k|}$




Parameterization of Catmull-Rom Curves [Yuksel,Schaefer,Keyser,CAD11]

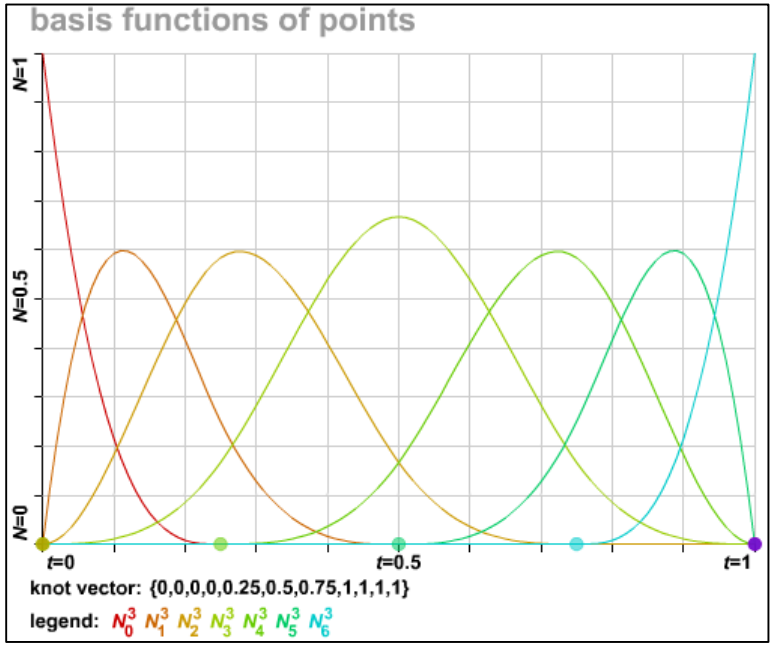
# Application of cubic Catmull-Rom spline: Hair modeling

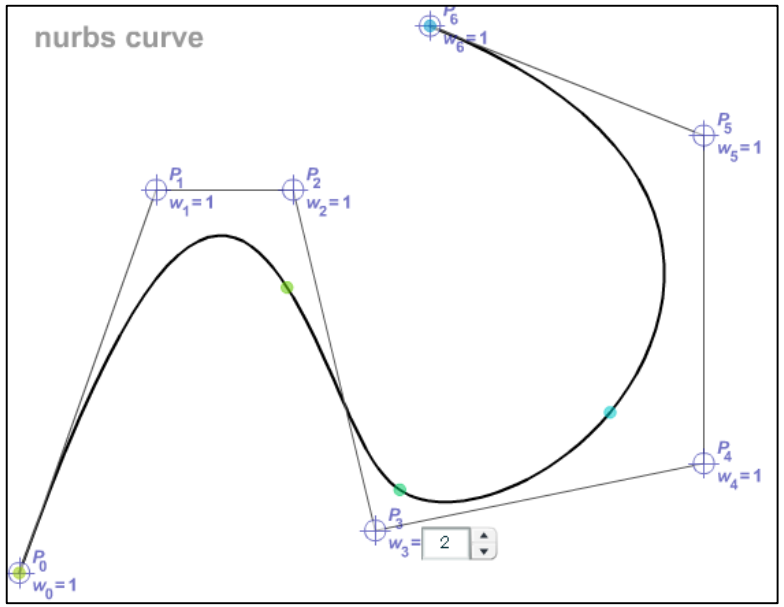



Parameterization of Catmull-Rom Curves [Yuksel,Schaefer,Keyser,CAD11]

# B-spline

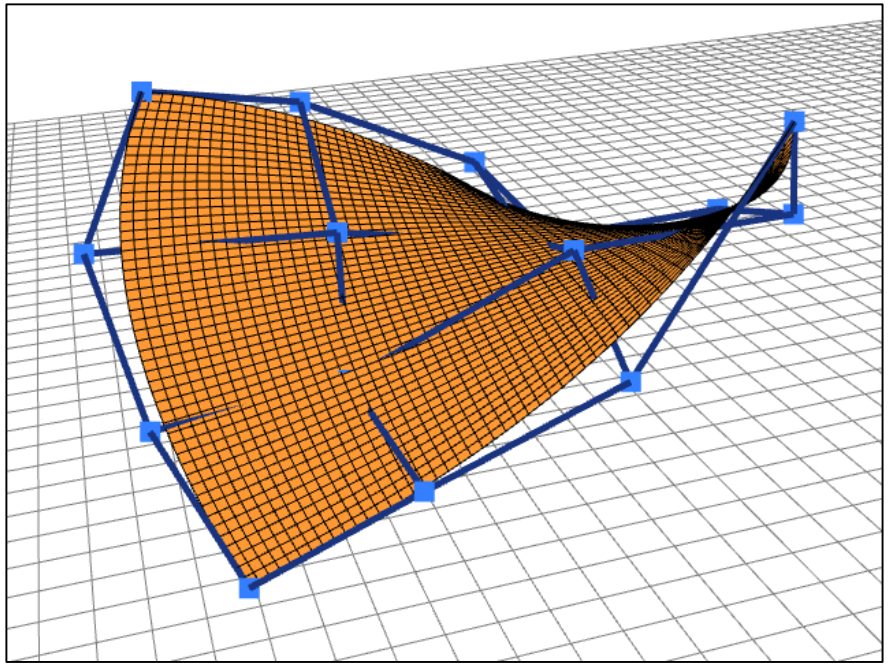
- Another way of defining polynomial spline
  - Represent curve as sum of **b**asis functions
  - Cubic basis is the most commonly used
  - Deeply related to subdivision surfaces
    ➔ Next lecture
- **N**on-**U**niform **R**ational **B**-**S**pline
  - Non-Uniform = varying spacing of knots ($t_k$)
  - Rational = arbitrary weights for CPs
  - (Complex stuff, not covered)
- Cool Flash demo:
  http://geometrie.foretnik.net/files/NURBS-en.swf



# Parametric surfaces

- One parameter ➔ Curve $P(t)$
- Two parameters ➔ Surface $P(s,t)$



- Cubic Bezier surface:
  - Input: 4×4=16 control points $P_{ij}$

$$P(s,t) = \sum_{i=0}^{3}\sum_{j=0}^{3} b_i^3(s)b_j^3(t)P_{ij}$$

Bernstein basis functions

$b_0^3(t) = (1-t)^3$

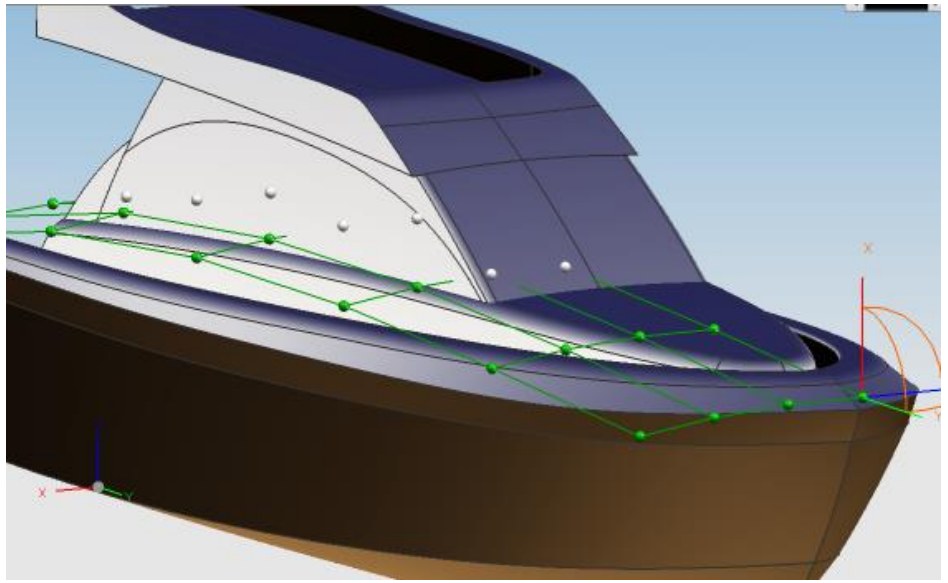
$b_1^3(t) = 3t(1-t)^2$

$b_2^3(t) = 3t^2(1-t)$

$b_3^3(t) = t^3$

# 3D modeling using parametric surface patches

- Pros
  - Can compactly represent smooth surfaces
  - Can accurately represent spheres, cones, etc
- Cons
  - Hard to design nice layout of patches
  - Hard to maintain continuity across patches
- Often used for designing man-made objects consisting of simple parts



# Pointers

- http://en.wikipedia.org/wiki/Bezier_curve
- http://antigrain.com/research/adaptive_bezier/
- https://groups.google.com/forum/#!topic/comp.graphics.algorithms/2FypAv29dG4
- http://en.wikipedia.org/wiki/Cubic_Hermite_spline
- http://en.wikipedia.org/wiki/Centripetal_Catmull%E2%80%93Rom_spline