

Introduction to Computer Graphics

– Modeling (1) –

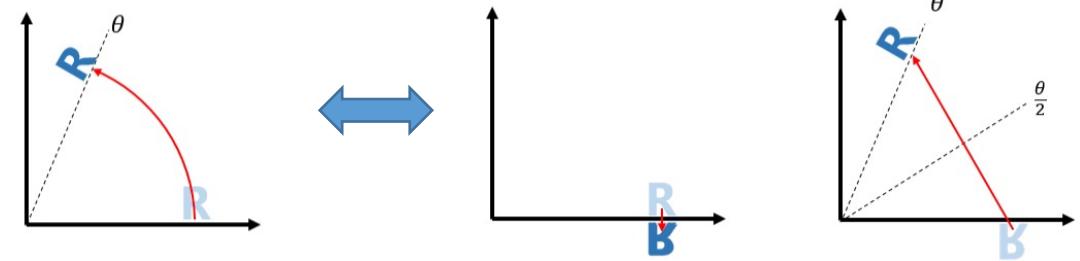
April 16, 2020

Kenshi Takayama

Some additional notes on quaternions

Another explanation for quaternions (overview)

1. Any rotation can be decomposed into even number of reflections



2. Quaternions can concisely describe reflections in 3D

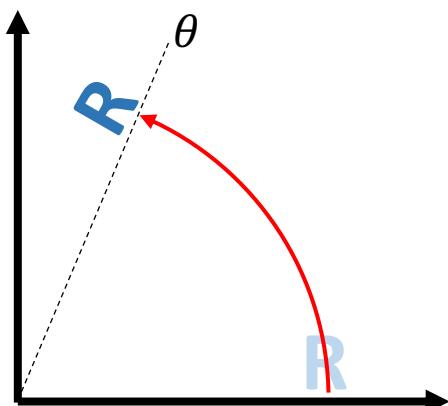
$$R_{\vec{f}}(\vec{x}) = -\vec{f} \vec{x} \vec{f}^{-1}$$

3. Combining two reflections equivalent to the rotation leads to the formula

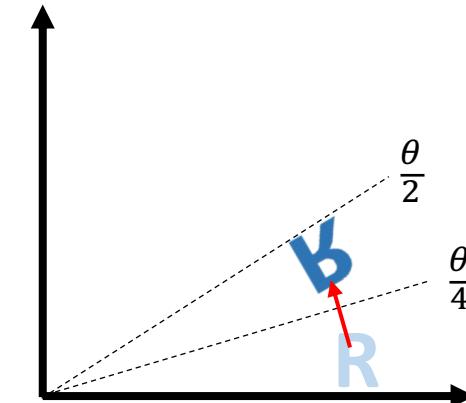
$$R_{\vec{g}}(R_{\vec{f}}(\vec{x})) = \left(\cos \frac{\theta}{2} + \vec{\omega} \sin \frac{\theta}{2} \right) \vec{x} \left(\cos \frac{\theta}{2} - \vec{\omega} \sin \frac{\theta}{2} \right)$$

Any rotation can be decomposed into even number of reflections

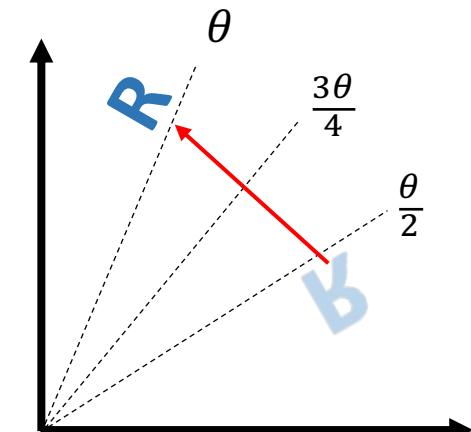
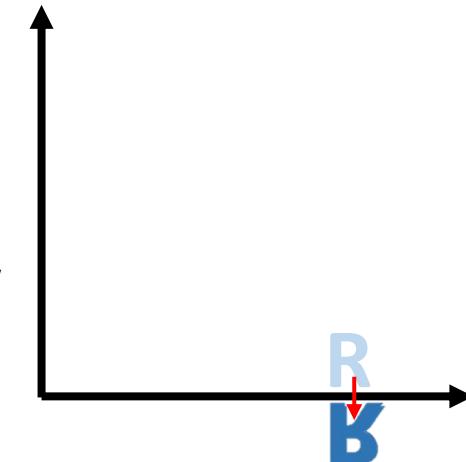
- Mathematically proven
 - Valid for any dimensions
- Not unique (of course!)



One way



Another way



Quaternions recap

- Complex number: real + imaginary

$$a + b \mathbf{i}$$

- Quaternion: scalar + vector

$$a + \vec{v}$$

- Definition of quaternion multiplication:

$$(a_1 + \vec{v}_1)(a_2 + \vec{v}_2) := \underbrace{a_1 a_2 - \vec{v}_1 \cdot \vec{v}_2}_{\text{Scalar part}} + \underbrace{a_1 \vec{v}_2 + a_2 \vec{v}_1 + \vec{v}_1 \times \vec{v}_2}_{\text{Vector part}}$$

- Pure vectors can take multiplication by interpreting them as quaternions:

$$\vec{v}_1 \vec{v}_2 = -\vec{v}_1 \cdot \vec{v}_2 + \vec{v}_1 \times \vec{v}_2$$

- Notable properties:
 - (Relevant later)

$$\vec{v} \vec{v} = -\|\vec{v}\|^2$$

$\vec{v} \times \vec{v}$ is always zero

$$\vec{v}^{-1} = -\frac{\vec{v}}{\|\vec{v}\|^2}$$

Multiplying \vec{v} to rhs produces 1

If $\vec{v} \cdot \vec{w} = 0$, then $\vec{v} \vec{w} = -\vec{w} \vec{v}$

$$\vec{v} \vec{w} = \vec{v} \times \vec{w} = -\vec{w} \times \vec{v} = -\vec{w} \vec{v}$$

Describing reflections using quaternions

- Reflection of a point \vec{x} across a plane orthogonal to \vec{f} :

$$R_{\vec{f}}(\vec{x}) := -\vec{f} \vec{x} \vec{f}^{-1}$$

- Holds essential properties of reflections:

- Linearity:

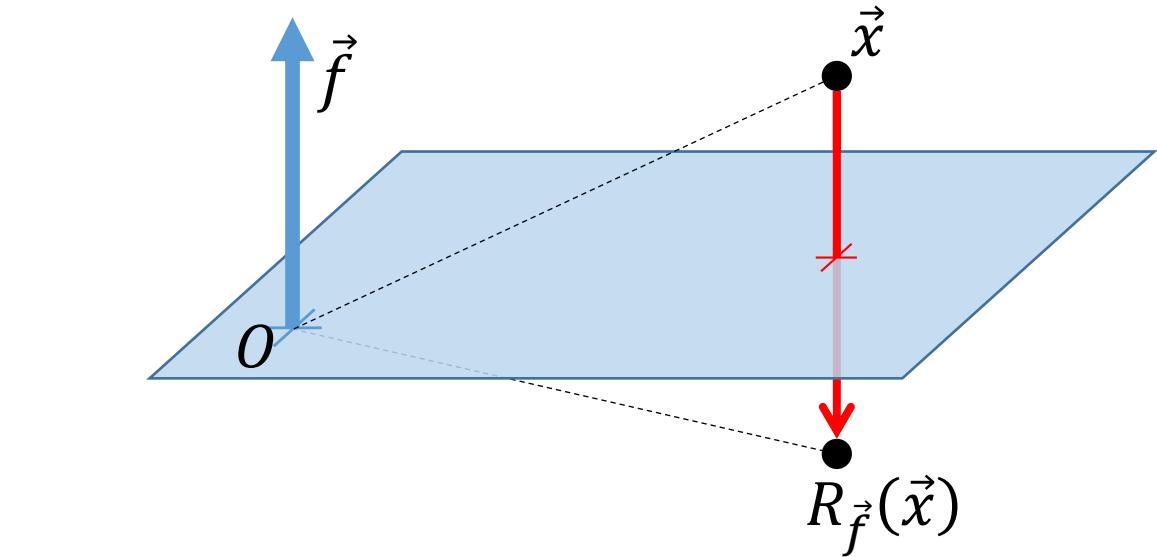
$$R_{\vec{f}}(a \vec{x} + b \vec{y}) = a R_{\vec{f}}(\vec{x}) + b R_{\vec{f}}(\vec{y})$$

- \vec{f} gets mapped to $-\vec{f}$:

$$R_{\vec{f}}(\vec{f}) = -\vec{f} \vec{f} \vec{f}^{-1} = -\vec{f}$$

- If a point \vec{x} satisfies $\vec{x} \cdot \vec{f} = 0$ (i.e. on the plane), \vec{x} doesn't move:

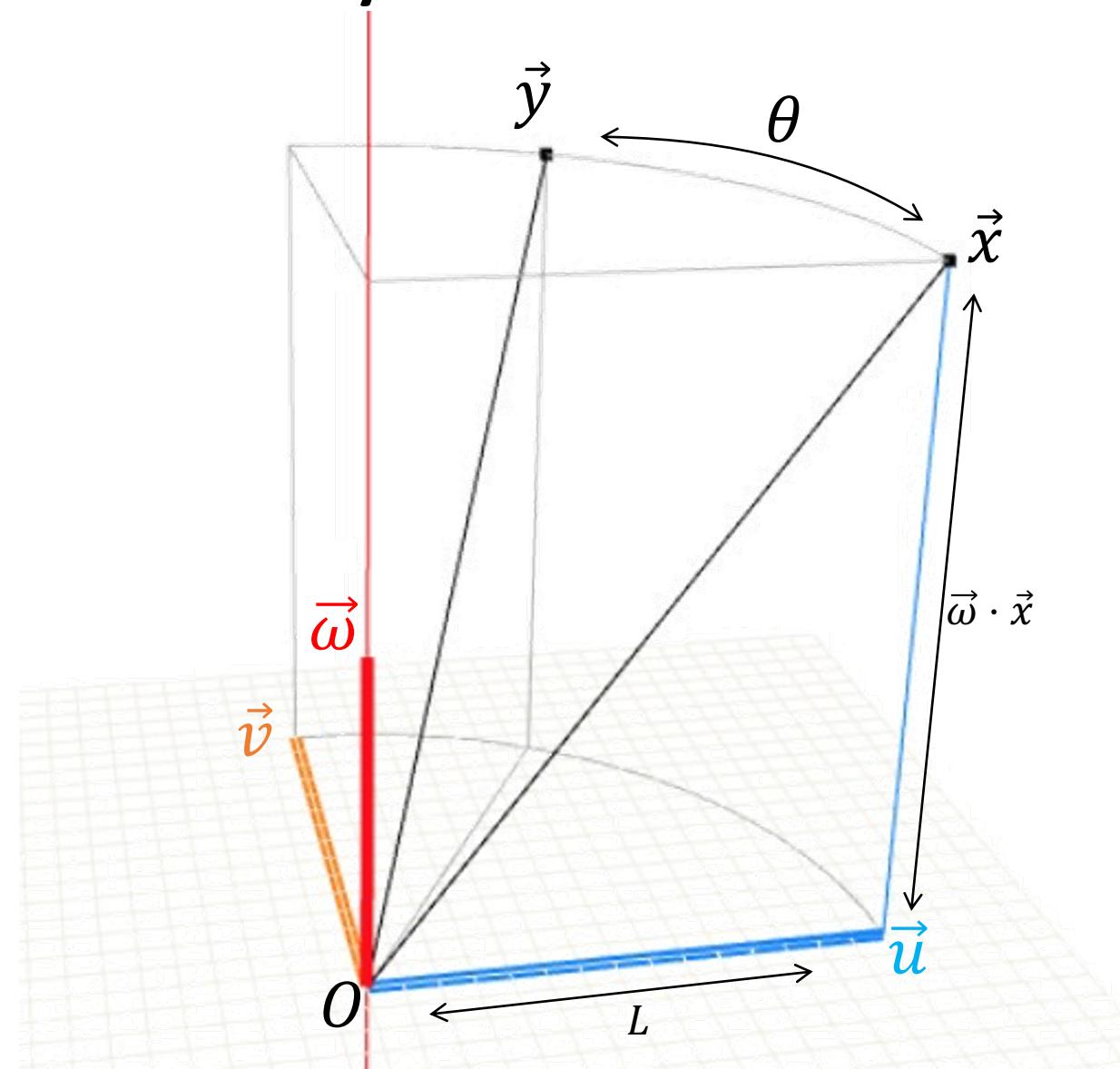
$$R_{\vec{f}}(\vec{x}) = -\vec{f} \vec{x} \vec{f}^{-1} = -(-\vec{x} \vec{f}) \vec{f}^{-1} = \vec{x}$$



Because if $\vec{x} \cdot \vec{f} = 0$, then $\vec{f} \vec{x} = -\vec{x} \vec{f}$

Setup for rotation around arbitrary axis

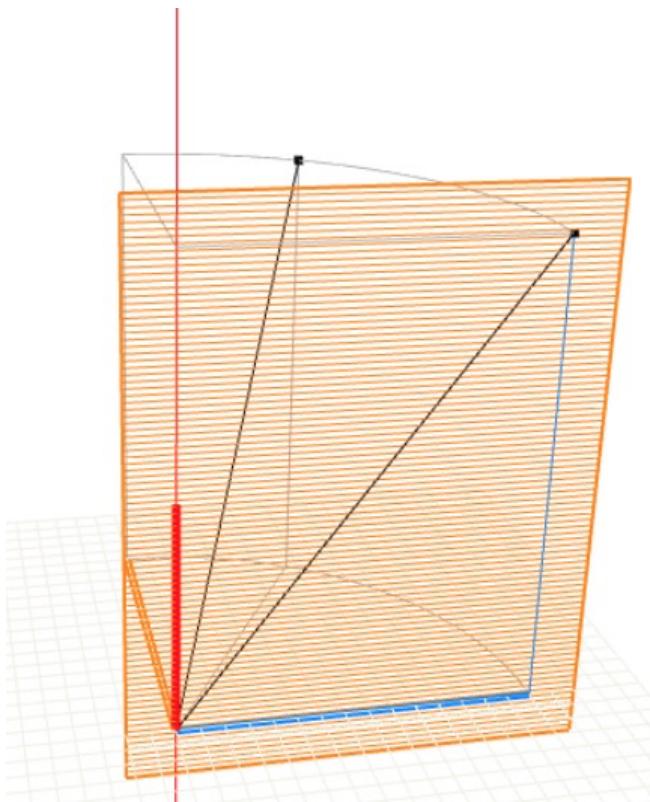
- Rotation axis (unit vector) : $\vec{\omega}$
- Rotation angle : θ
- Point before rotation : \vec{x}
- Point after rotation : $\vec{y} := R_{\vec{\omega}, \theta}(\vec{x})$
- Think of local 2D coordinate system :
 - “Right” vector : $\vec{u} := \vec{x} - (\vec{\omega} \cdot \vec{x})\vec{\omega}$
 - “Up” vector : $\vec{v} := \vec{\omega} \times \vec{x}$
 - Note that $\|\vec{u}\| = \|\vec{v}\|$
 - (Let’s call it L)



Decompose rotation into two reflections

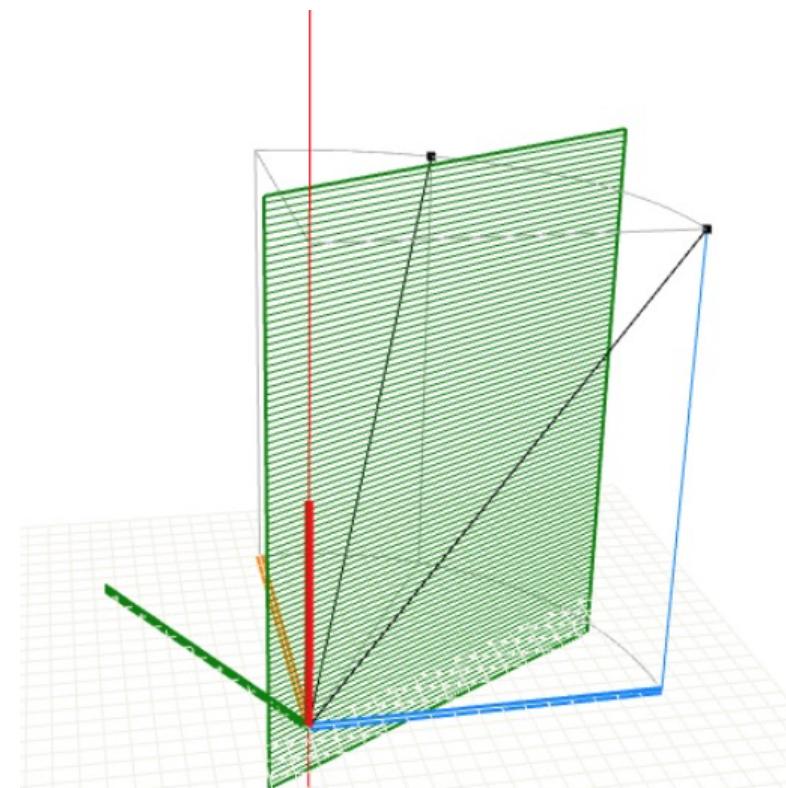
1st reflection :

$$\vec{f} := \vec{v}$$

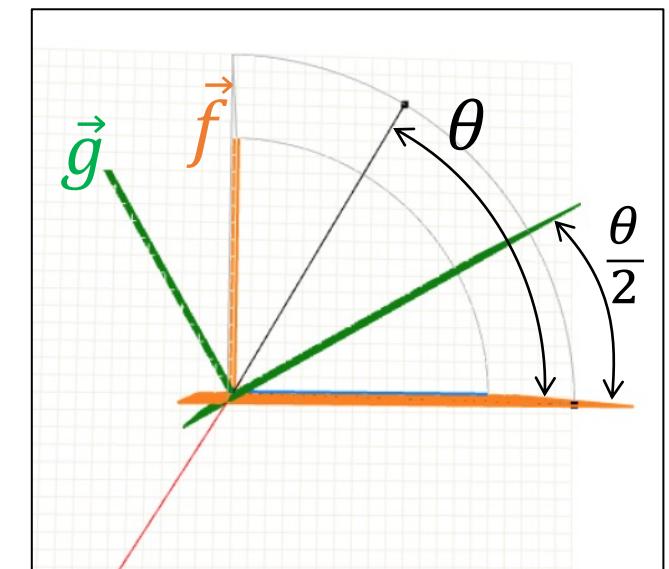


2nd reflection :

$$\vec{g} := -\sin \frac{\theta}{2} \vec{u} + \cos \frac{\theta}{2} \vec{v}$$



Top view



Combining two reflections

- Formula : $R_{\vec{g}}(R_{\vec{f}}(\vec{x})) = R_{\vec{g}}(-\vec{f} \vec{x} \vec{f}^{-1}) = -\vec{g}(-\vec{f} \vec{x} \vec{f}^{-1})\vec{g}^{-1} = (\vec{g} \vec{f}) \vec{x} (\vec{f}^{-1} \vec{g}^{-1})$
- Substitute $\vec{f} := \vec{v}$, $\vec{g} := -\sin \frac{\theta}{2} \vec{u} + \cos \frac{\theta}{2} \vec{v}$ to the above

- For the left part $\vec{g} \vec{f}$:

$$\begin{aligned}\vec{g} \cdot \vec{f} &= (-\sin \frac{\theta}{2} \vec{u} + \cos \frac{\theta}{2} \vec{v}) \cdot \vec{v} &= L^2 \cos \frac{\theta}{2} & \text{(because } \vec{u} \cdot \vec{v} = 0 \text{)} \\ \vec{g} \times \vec{f} &= (-\sin \frac{\theta}{2} \vec{u} + \cos \frac{\theta}{2} \vec{v}) \times \vec{v} &= -L^2 \sin \frac{\theta}{2} \vec{\omega} & \text{(because } \vec{u} \times \vec{v} = L^2 \vec{\omega} \text{)}\end{aligned}$$

Therefore,

$$\vec{g} \vec{f} = -\vec{g} \cdot \vec{f} + \vec{g} \times \vec{f} = -L^2 \left(\cos \frac{\theta}{2} + \vec{\omega} \sin \frac{\theta}{2} \right)$$

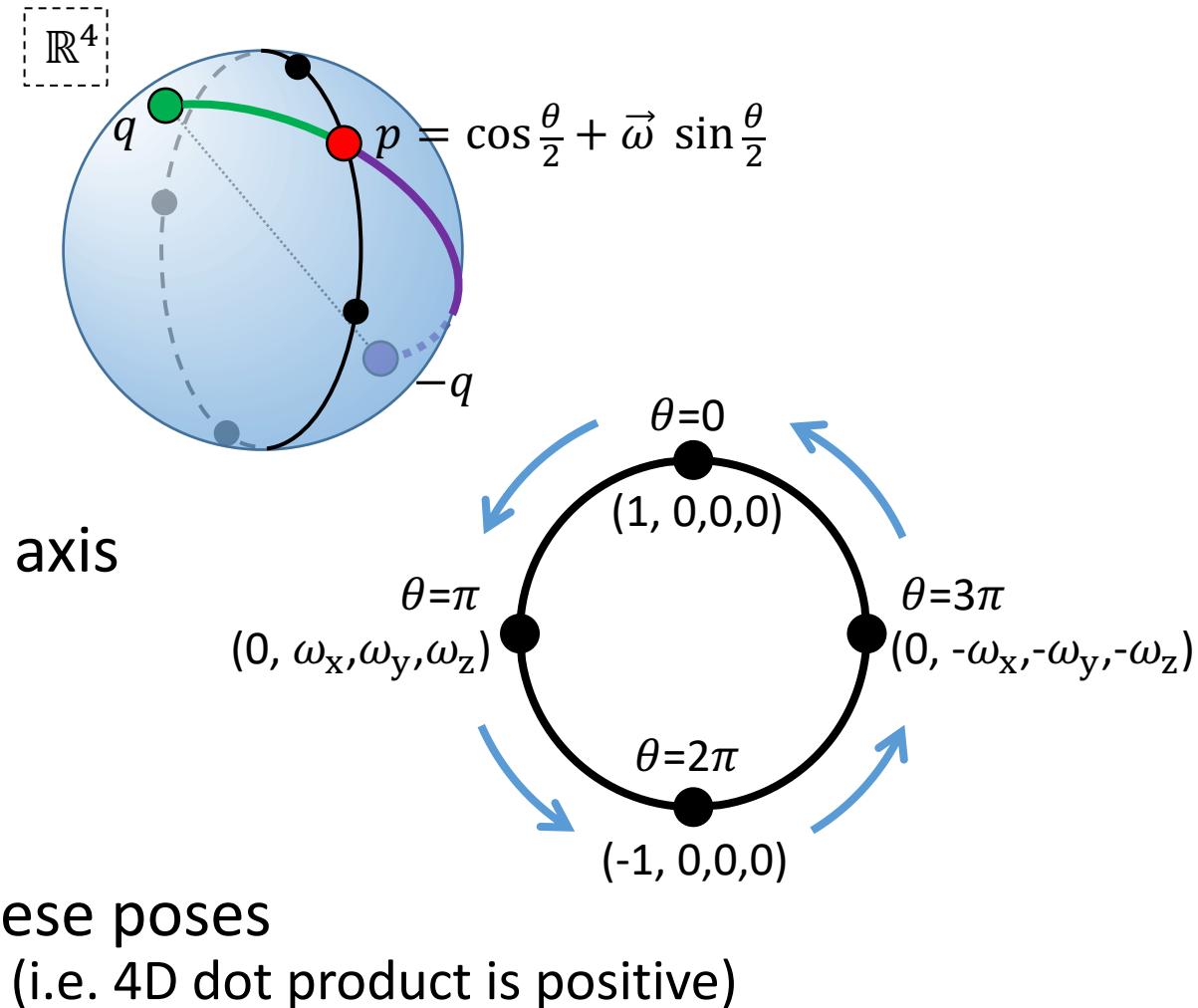
- The right part $\vec{f}^{-1} \vec{g}^{-1} = \frac{\vec{f} \vec{g}}{L^4}$ is analogous : $\vec{f}^{-1} \vec{g}^{-1} = -L^{-2} \left(\cos \frac{\theta}{2} - \vec{\omega} \sin \frac{\theta}{2} \right)$

- Finally, we get the formula :

$$\begin{aligned}R_{\vec{\omega}, \theta}(\vec{x}) &= R_{\vec{g}}(R_{\vec{f}}(\vec{x})) = \left(-L^2 \left(\cos \frac{\theta}{2} + \vec{\omega} \sin \frac{\theta}{2} \right) \right) \vec{x} \left(-L^{-2} \left(\cos \frac{\theta}{2} - \vec{\omega} \sin \frac{\theta}{2} \right) \right) \\ &= \boxed{\left(\cos \frac{\theta}{2} + \vec{\omega} \sin \frac{\theta}{2} \right) \vec{x} \left(\cos \frac{\theta}{2} - \vec{\omega} \sin \frac{\theta}{2} \right)}\end{aligned}$$

Representing and blending poses using quaternions

- Any rotations (poses) can be represented as unit quaternions
 - Points on hypersphere of 4D space
- Fix $\vec{\omega}$ and vary θ
→ unit circle in 4D space
- A pose after rotating 360° about a certain axis is represented as another quaternion
 - One pose corresponds to two quaternions (double cover)
- A geodesic between two points p, q on the hypersphere represents interpolation of these poses
 - Should pick either q or $-q$ which is closer to p (i.e. 4D dot product is positive)



Normalize quaternions or not?

- Any quaternions can be written as scaling of unit quaternions

$$q = r \left(\cos \frac{\theta}{2} + \vec{\omega} \sin \frac{\theta}{2} \right), \quad q^{-1} = r^{-1} \left(\cos \frac{\theta}{2} - \vec{\omega} \sin \frac{\theta}{2} \right)$$

- In the rotation formula, the scaling part is cancelled

$$q \vec{x} q^{-1} = \cancel{r} \left(\cos \frac{\theta}{2} + \vec{\omega} \sin \frac{\theta}{2} \right) \vec{x} \cancel{r^{-1}} \left(\cos \frac{\theta}{2} - \vec{\omega} \sin \frac{\theta}{2} \right) = \left(\cos \frac{\theta}{2} + \vec{\omega} \sin \frac{\theta}{2} \right) \vec{x} \left(\cos \frac{\theta}{2} - \vec{\omega} \sin \frac{\theta}{2} \right)$$

→ so, normalization isn't needed?

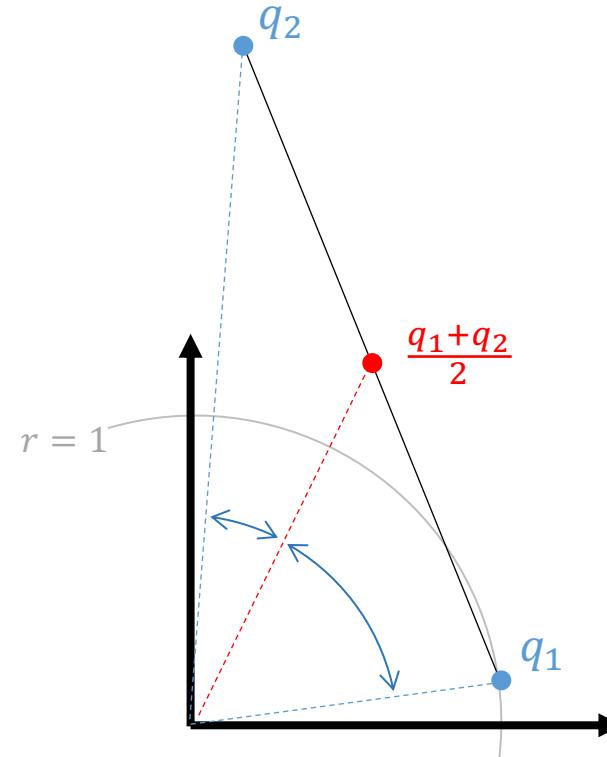
- In practice, don't use quaternion mults for computing coordinate transformation (because inefficient)

- Just do explicit vector calc using axis & angle

$$(\vec{x} - (\vec{\omega} \cdot \vec{x})\vec{\omega}) \cos \theta + (\vec{\omega} \times \vec{x}) \sin \theta + (\vec{\omega} \cdot \vec{x})\vec{\omega}$$

- Can get axis & angle only after normalization

- Un-normalized can cause artifact when interpolated



Modeling curves

Parametric curves

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

$$\begin{aligned}x(t) &= t - \sin t \\y(t) &= 1 - \cos t\end{aligned}$$

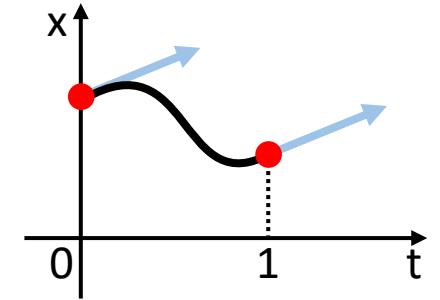


- Tangent (aka. derivative, gradient) vector: $(x'(t), y'(t))$
- Polynomial curve: $x(t) = \sum_i a_i t^i$

Cubic Hermite curves

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

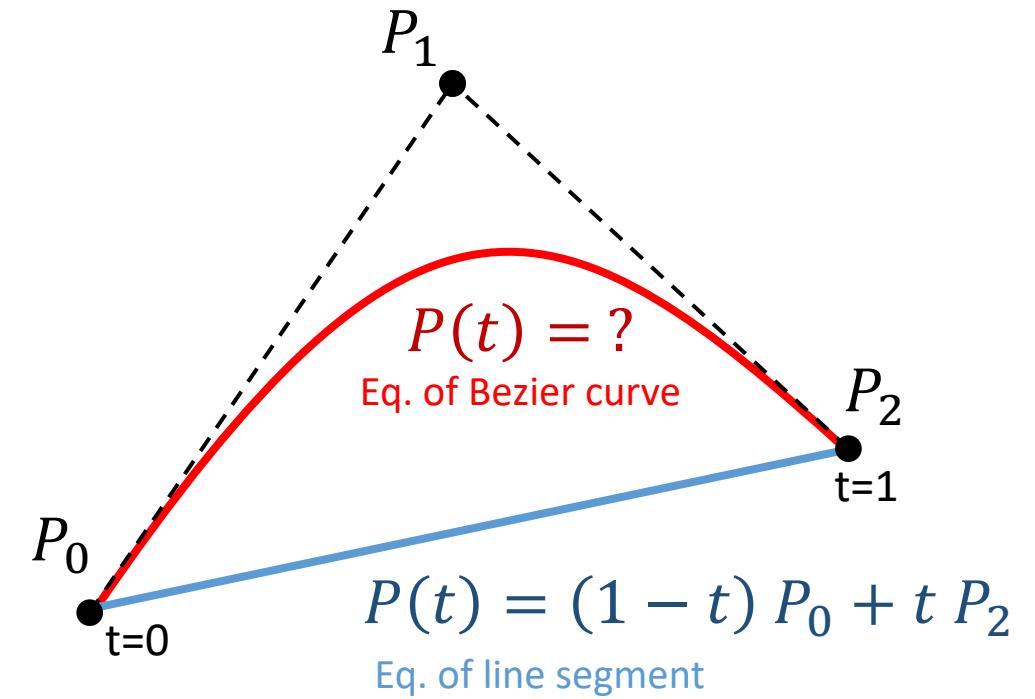
$$\begin{aligned}x(0) &= x_0 \\x(1) &= x_1 \\x'(0) &= x'_0 \\x'(1) &= x'_1\end{aligned}$$



$$\begin{aligned}x(0) &= a_0 &= x_0 \\x(1) &= a_0 + a_1 + a_2 + a_3 &= x_1 \\x'(0) &= a_1 &= x'_0 \\x'(1) &= a_1 + 2a_2 + 3a_3 &= x'_1 \\ \rightarrow \\ a_0 &= x_0 \\a_1 &= x'_0 \\a_2 &= -3x_0 + 3x_1 - 2x'_0 - x'_1 \\a_3 &= 2x_0 - 2x_1 + x'_0 + x'_1\end{aligned}$$

Bezier curves

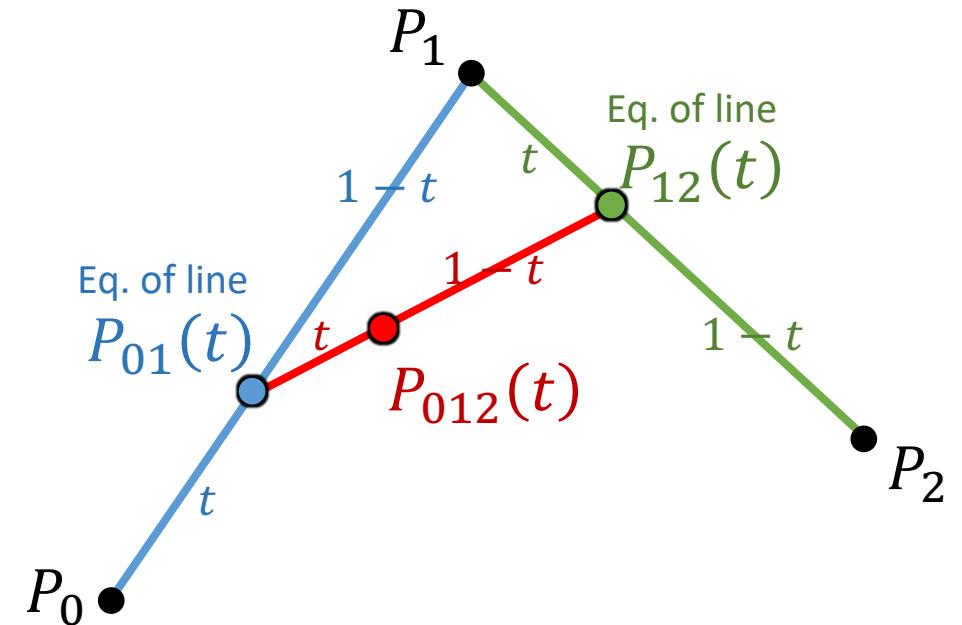
- Input: 3 **control points** (CPs) P_0, P_1, P_2
 - Coordinates of points in arbitrary domain (2D, 3D, ...)
- Output: Curve $P(t)$ satisfying
$$P(0) = P_0$$
$$P(1) = P_2$$
while being “pulled” by P_1



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}
- $$\begin{aligned} P_{012}(t) &= (1 - t)P_{01}(t) + t P_{12}(t) \\ &= (1 - t)\{(1 - t)P_0 + t P_1\} + t \{(1 - t)P_1 + t P_2\} \\ &= (1 - t)^2 P_0 + 2t(1 - t)P_1 + t^2 P_2 \end{aligned}$$

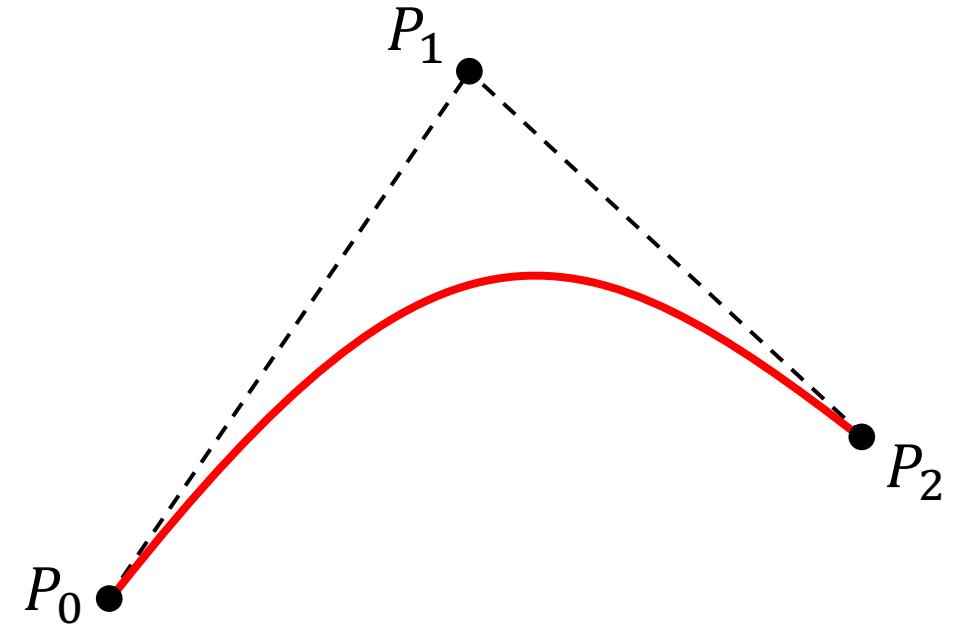
Quadratic Bezier curve



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}
- $$\begin{aligned} P_{012}(t) &= (1 - t)P_{01}(t) + t P_{12}(t) \\ &= (1 - t)\{(1 - t)P_0 + t P_1\} + t \{(1 - t)P_1 + t P_2\} \\ &= (1 - t)^2 P_0 + 2t(1 - t)P_1 + t^2 P_2 \end{aligned}$$

Quadratic Bezier curve



Cubic Bezier curve

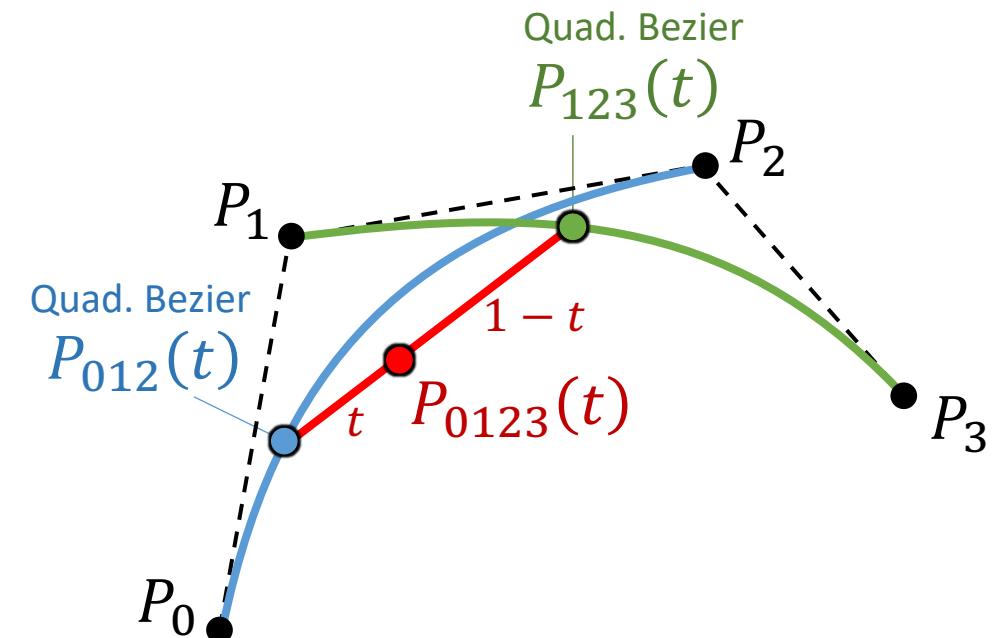
- Exact same idea applied to 4 points P_0, P_1, P_2, P_3 :
 - As t changes $0 \rightarrow 1$, transition from P_{012} to P_{123}

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

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

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

Cubic Bezier curve



Cubic Bezier curve

- Exact same idea applied to 4 points P_0, P_1, P_2, P_3 :

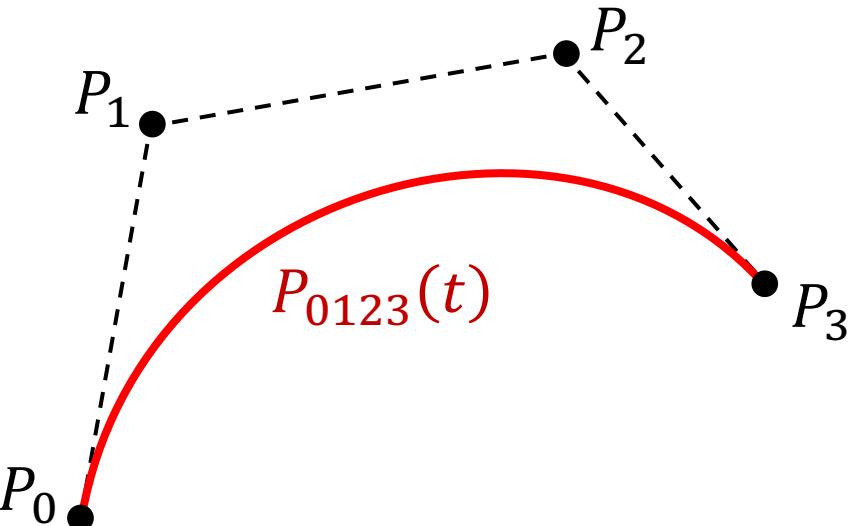
- As t changes $0 \rightarrow 1$, transition from P_{012} to P_{123}

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

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

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

Cubic Bezier curve



- Can easily control tangent at endpoints \rightarrow ubiquitously used in CG

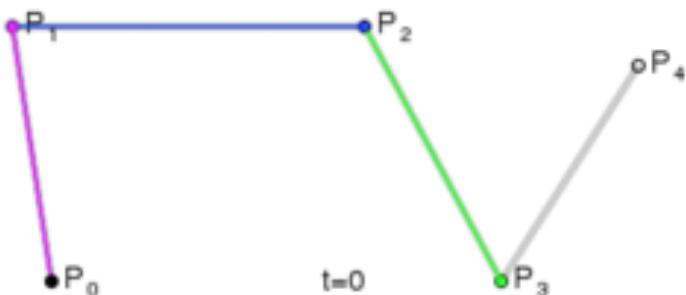
n-th order Bezier curve

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

$$P(t) = \sum_{i=0}^n {}_n C_i t^i (1-t)^{n-i} P_i$$

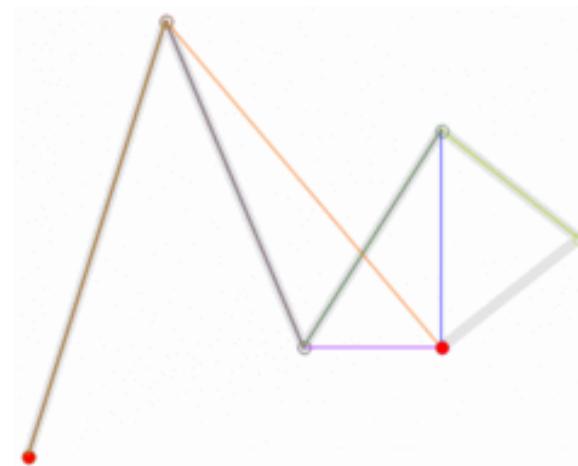
b_iⁿ(t)
Bernstein basis function

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

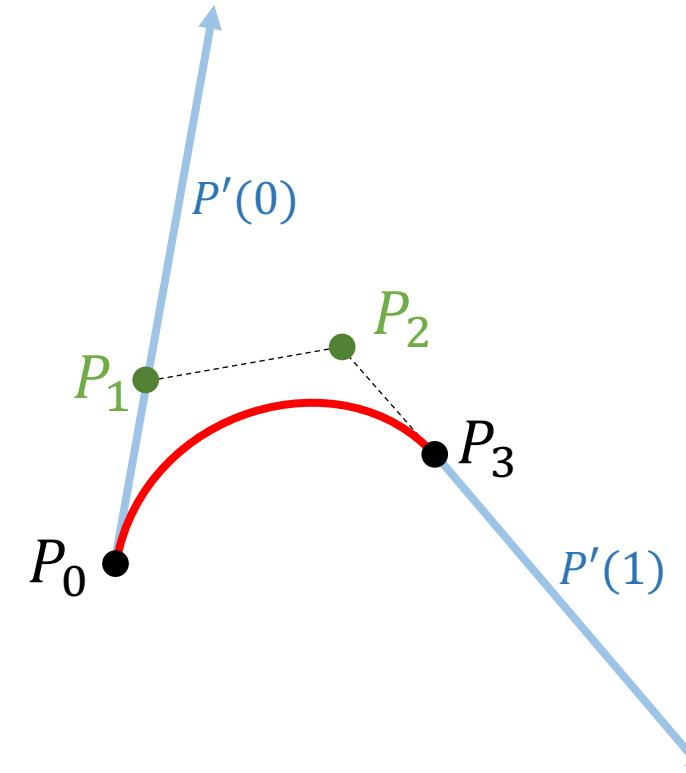
- Cubic Bezier curve & its derivative:

- $P(t) = (1 - t)^3 P_0 + 3t(1 - t)^2 P_1 + 3t^2(1 - t)P_2 + t^3 P_3$
- $P'(t) = -3(1 - t)^2 P_0 + 3\{(1 - t)^2 - 2t(1 - t)\}P_1 + 3\{2t(1 - t) - t^2\}P_2 + 3t^2 P_3$

- Derivatives at endpoints:

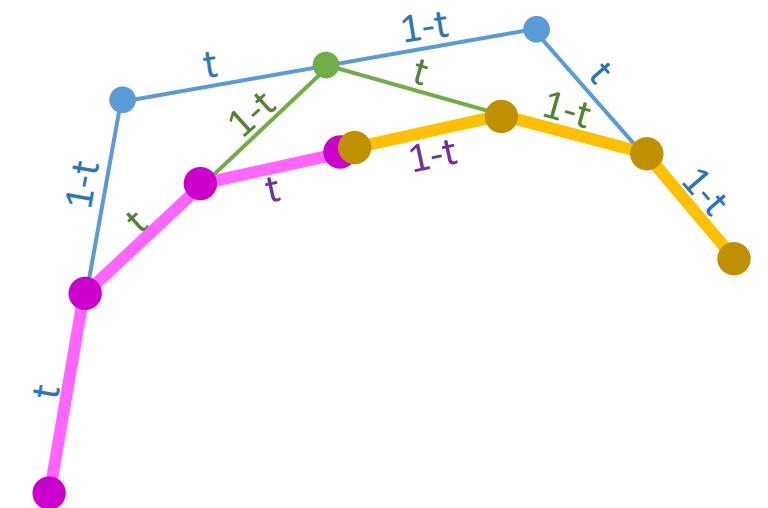
- $P'(0) = -3P_0 + 3P_1 \rightarrow P_1 = P_0 + \frac{1}{3}P'(0)$
- $P'(1) = -3P_2 + 3P_3 \rightarrow P_2 = P_3 - \frac{1}{3}P'(1)$

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



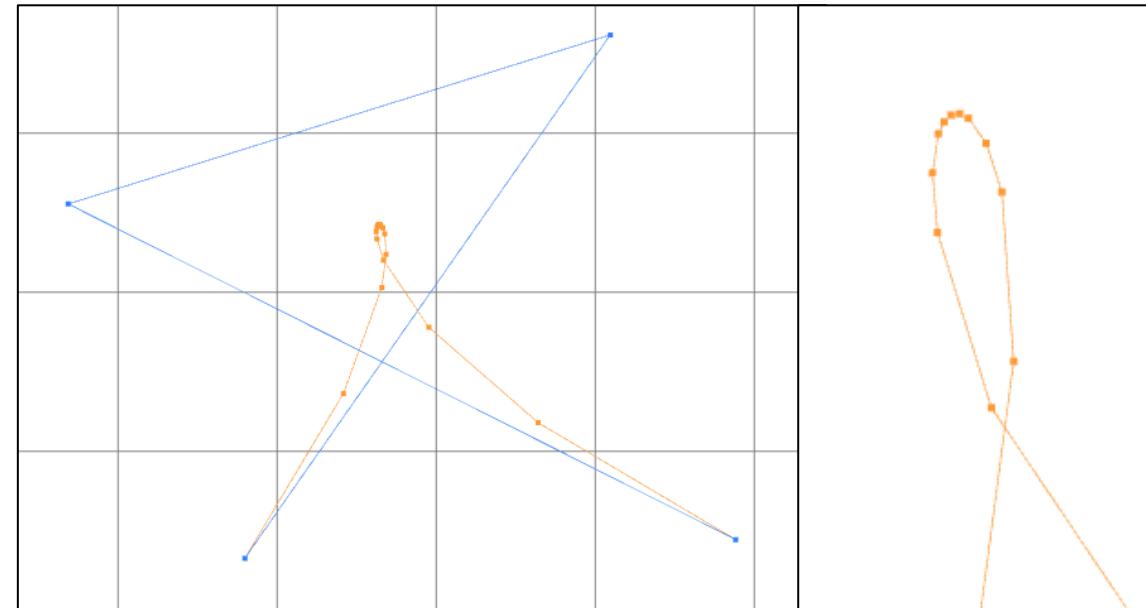
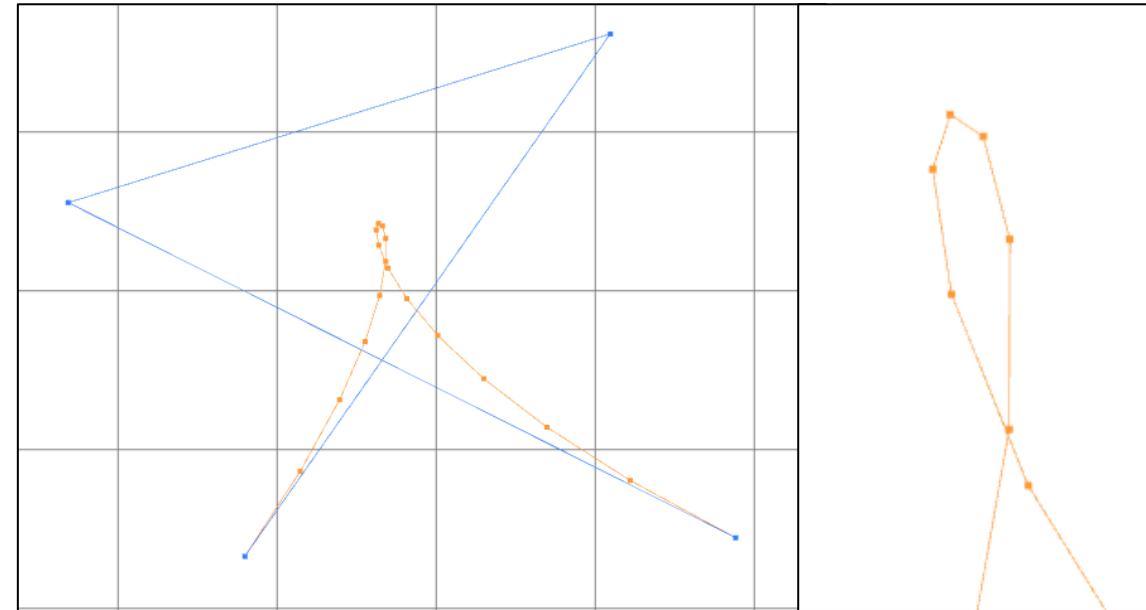
Evaluating Bezier curves

- 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 parameter 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

- $P_{012}(t) = (1-t)^2 P_0 + 2t(1-t)P_1 + t^2 P_2$
 $= \lambda_0(t) P_0 + \lambda_1(t) P_1 + \lambda_2(t) P_2$

- 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:

$$\lambda'_i(t) = \frac{\xi_i(t)}{\sum_j \xi_j(t)}$$

$$w_0 = w_2 = 1$$

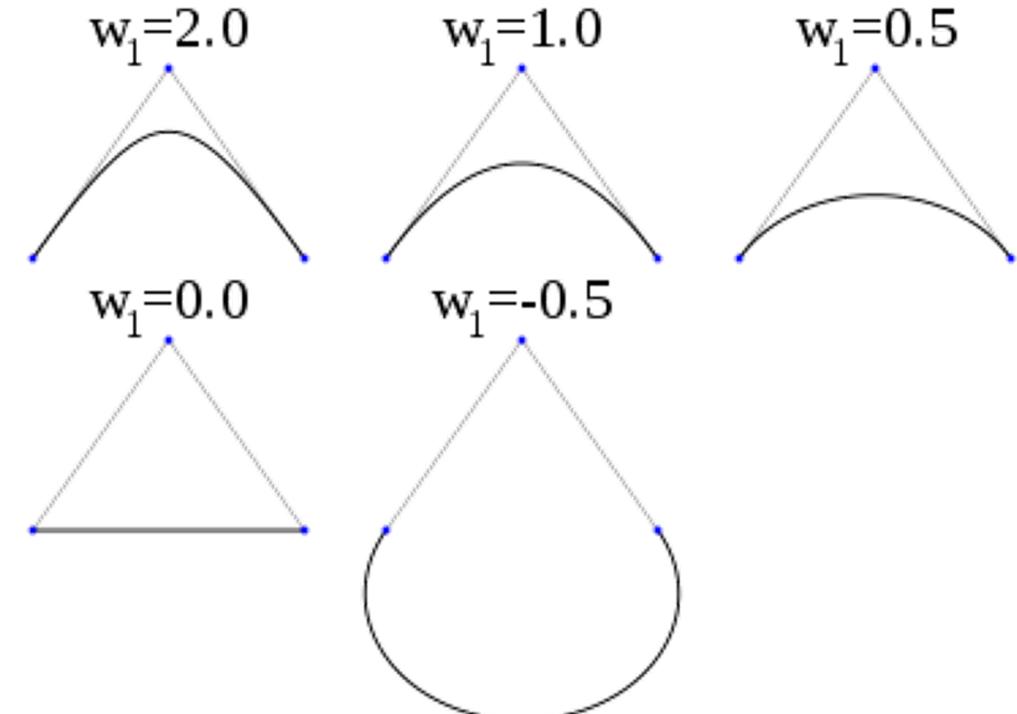
$$w_1=2.0$$

$$w_1=0.0$$

$$w_1=1.0$$

$$w_1=-0.5$$

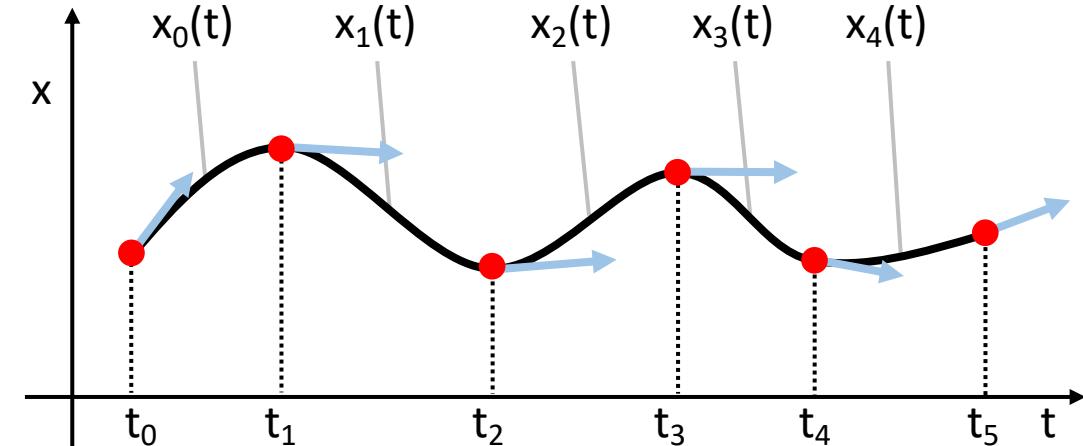
$$w_1=0.5$$



Non-polynomial curve → can represent arcs etc.

Cubic splines

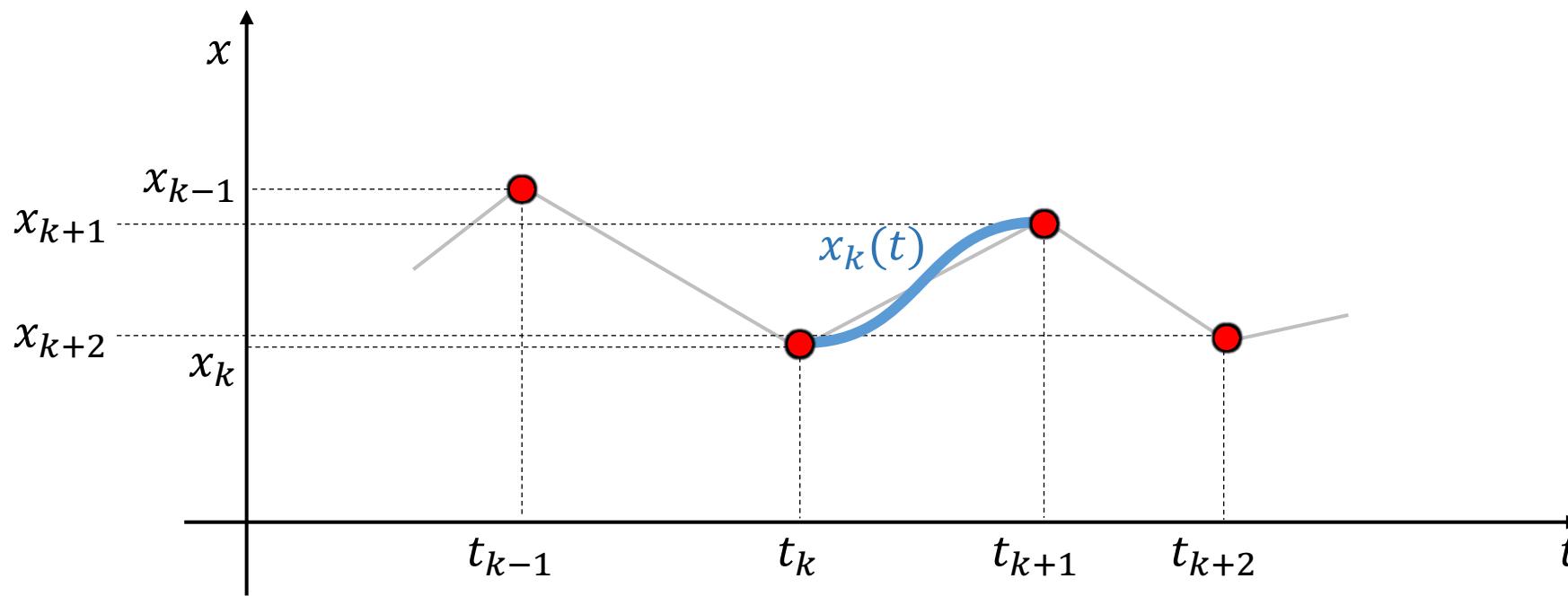
- Series of connected cubic curves
 - Piecewise polynomial
 - Share value & derivative at every transition of intervals (C^1 continuity)
- Parameter range can be other than $[0, 1]$
 - Assumption: $t_k < t_{k+1}$
- Given values as only input,
we want to automatically set derivatives



Curve tool in PowerPoint

Cubic Catmull-Rom spline

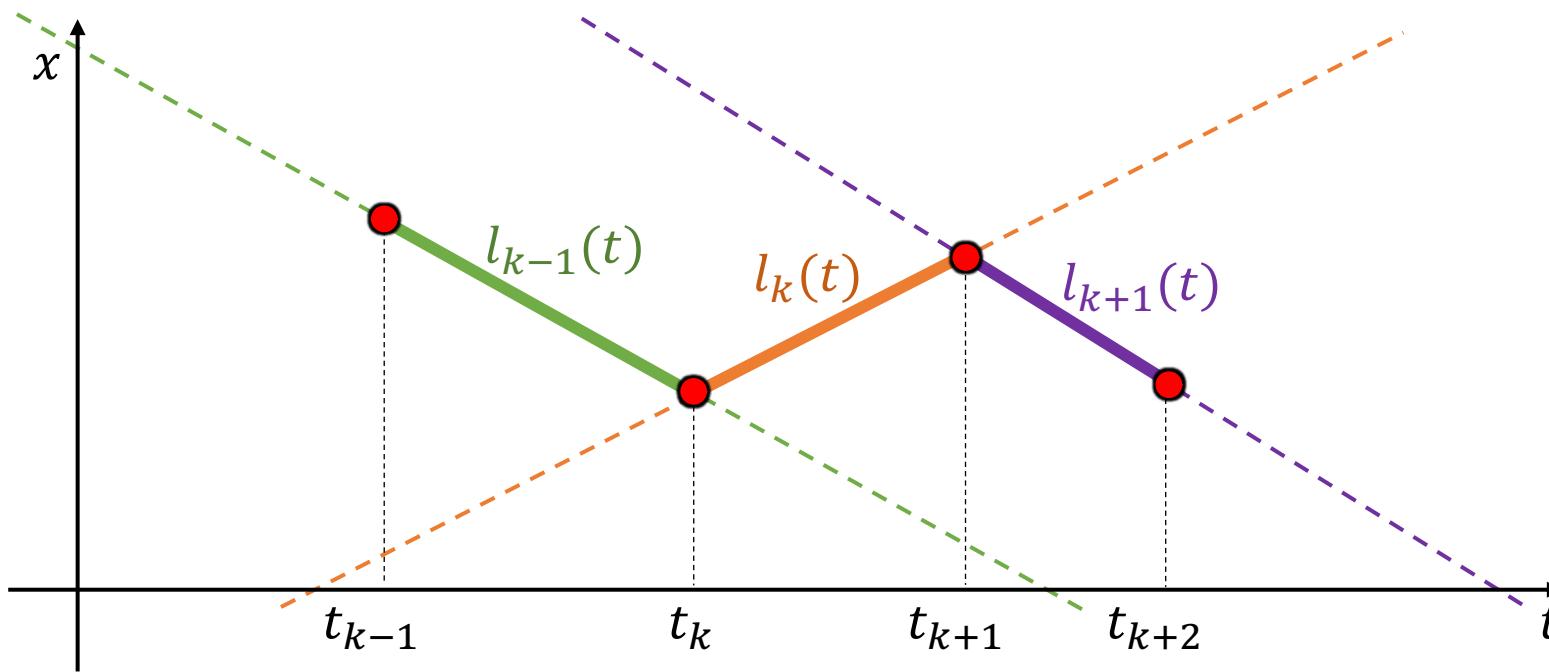
- Cubic function $x_k(t)$ for range $t_k \leq t \leq t_{k+1}$ is defined by adjacent constrained values $x_{k-1}, x_k, x_{k+1}, x_{k+2}$



Cubic Catmull-Rom spline: Step 1

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

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

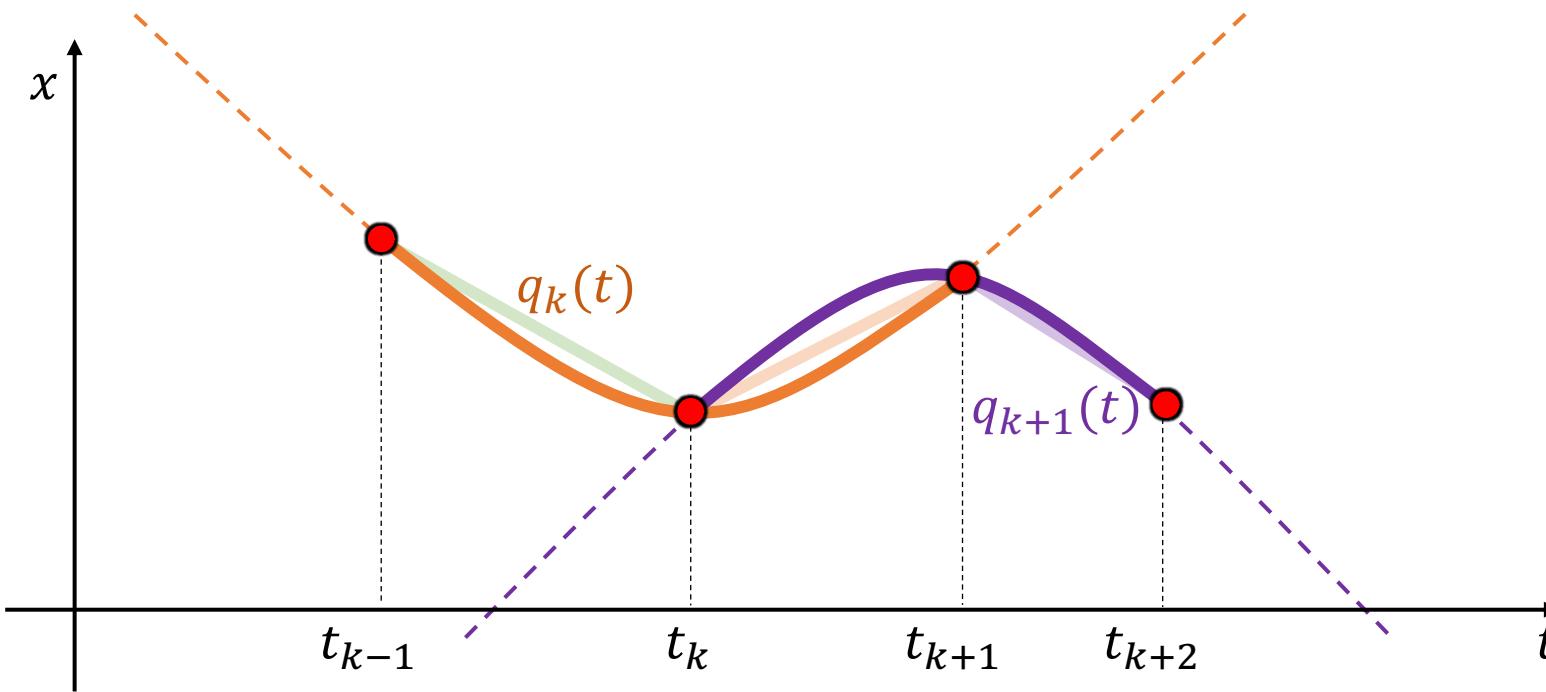


Cubic Catmull-Rom spline: Step 2

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

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

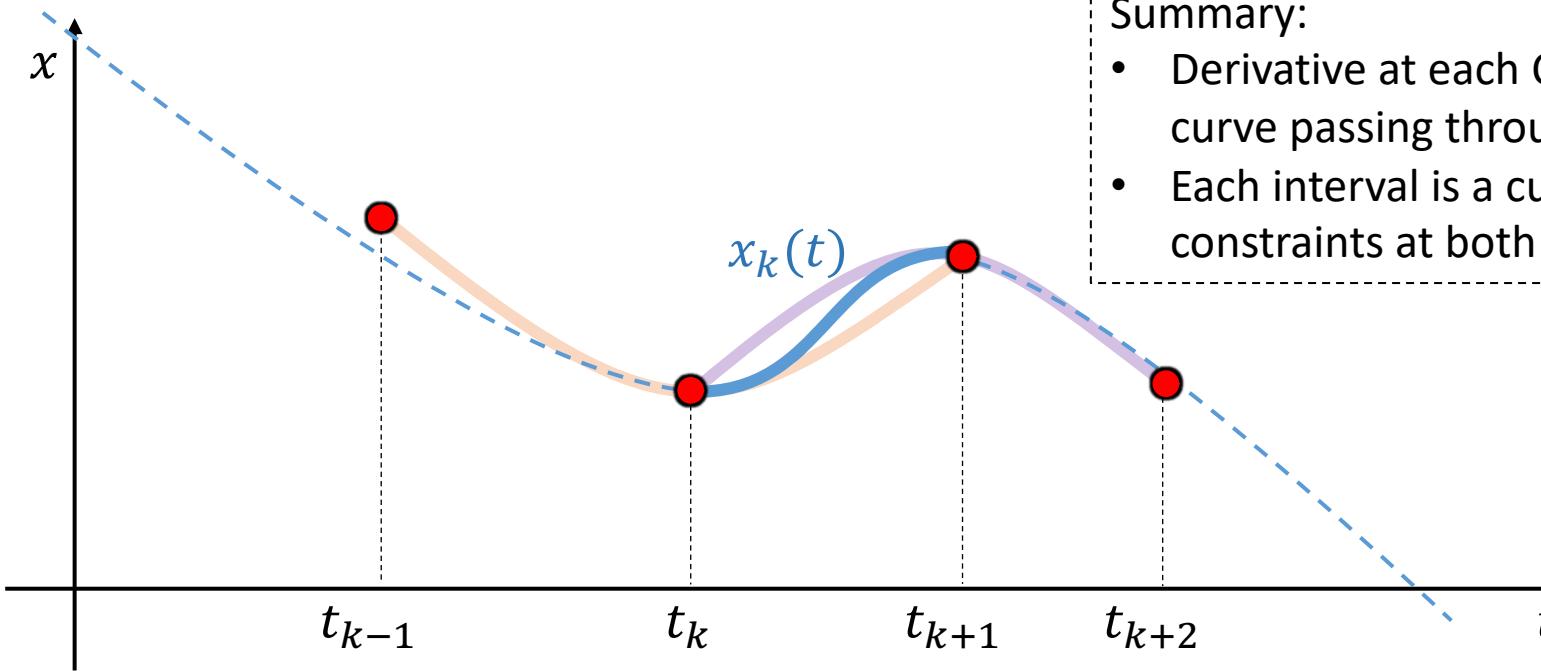
- Passes through 3 points $(t_{k-1}, x_{k-1}), (t_k, x_k), (t_{k+1}, x_{k+1})$



Cubic Catmull-Rom spline: Step 3

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

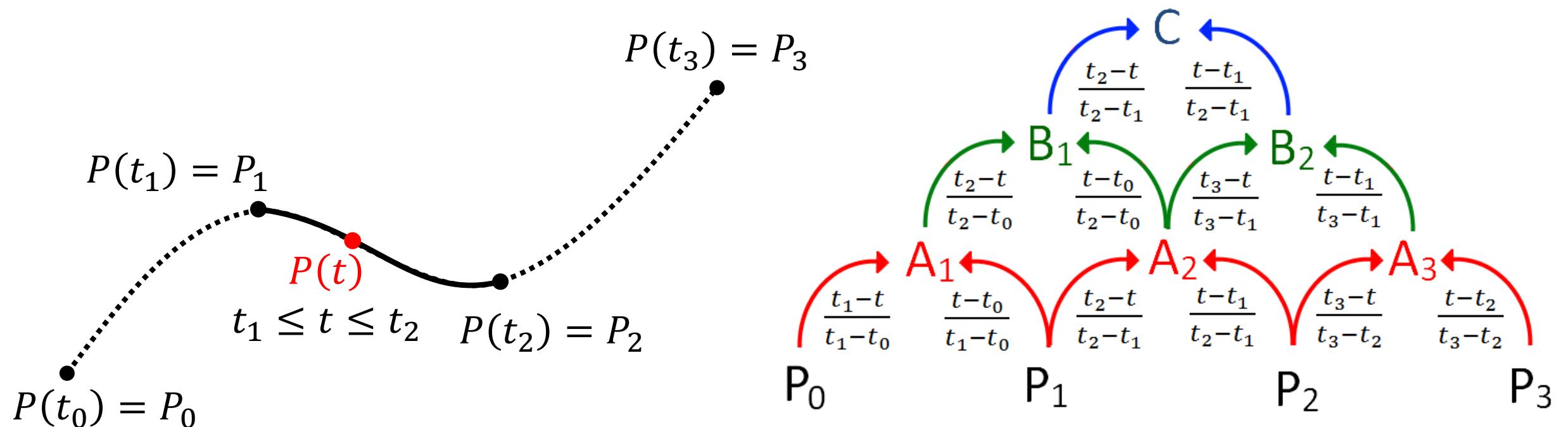
$$x_k(t) = \left(1 - \frac{t - t_k}{t_{k+1} - t_k}\right) q_k(t) + \frac{t - t_k}{t_{k+1} - t_k} q_{k+1}(t)$$



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



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

- Assume: $t_0 = 0$

- Uniform

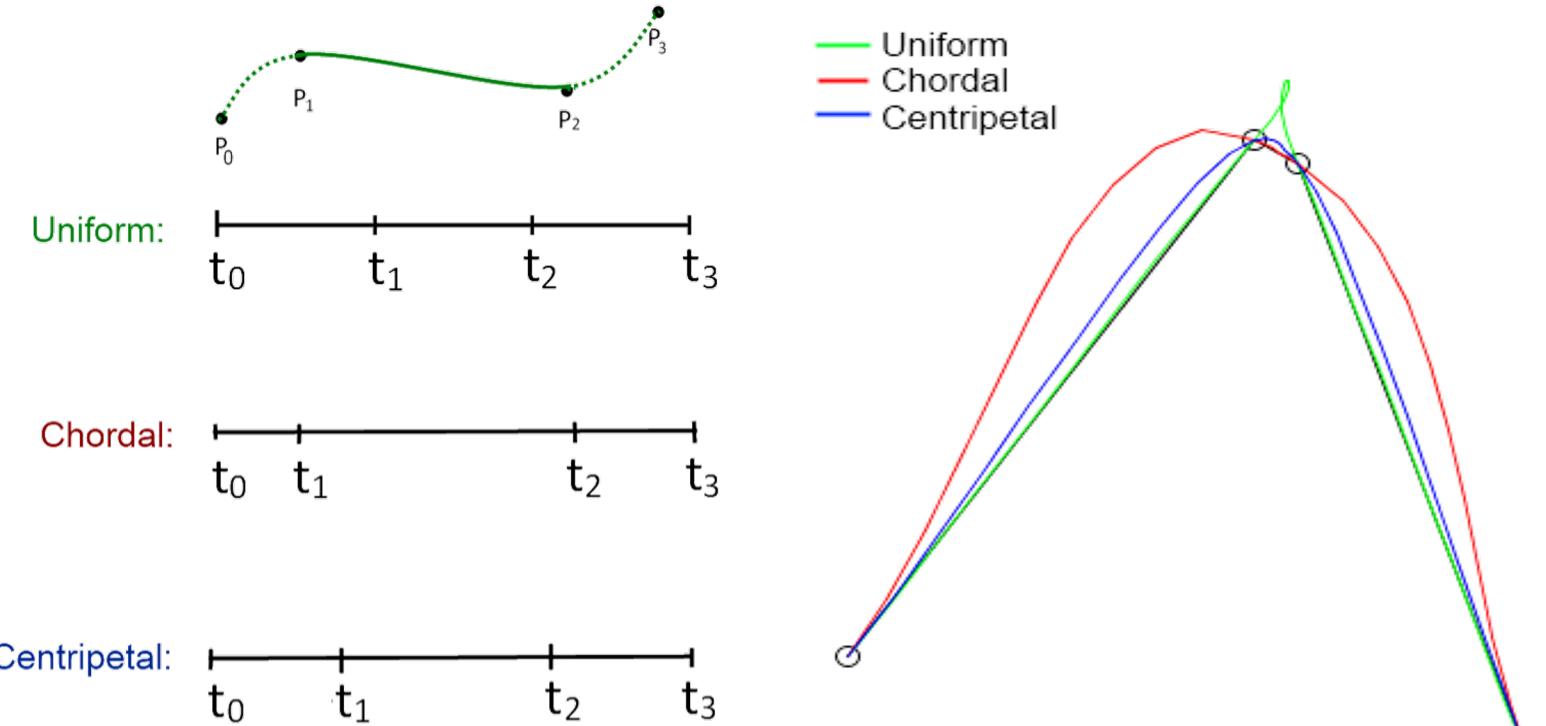
$$t_k = t_{k-1} + 1$$

- Chordal

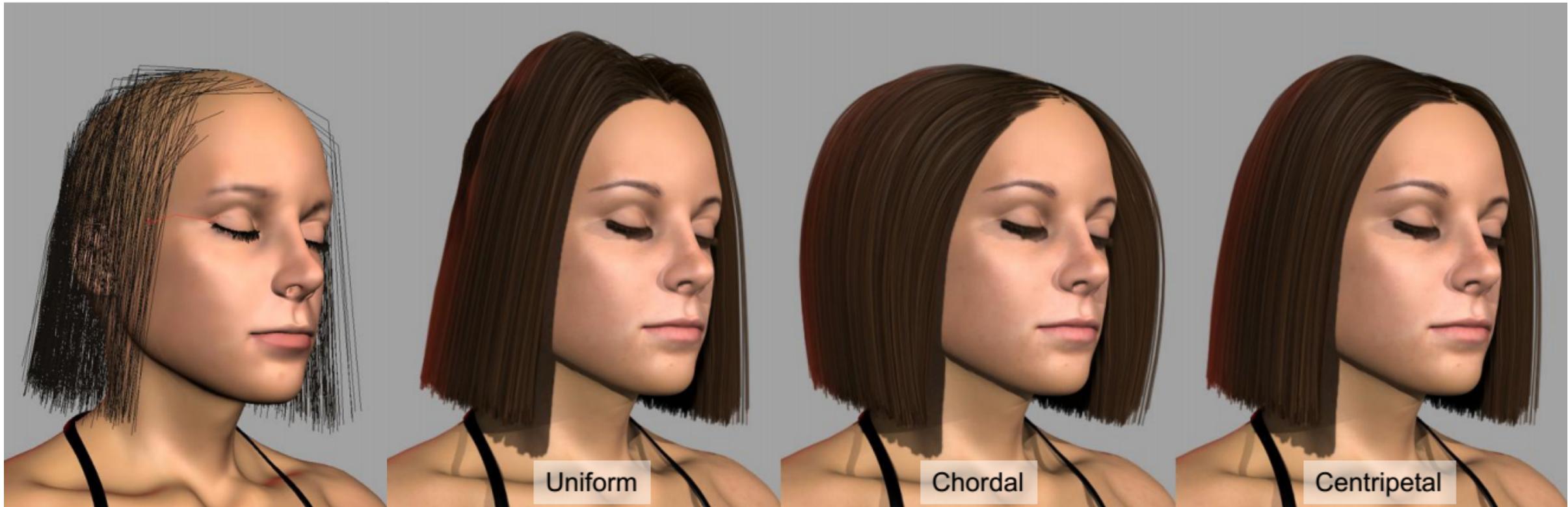
$$t_k = t_{k-1} + |P_{k-1} - P_k|$$

- Centripetal

$$t_k = t_{k-1} + \sqrt{|P_{k-1} - P_k|}$$

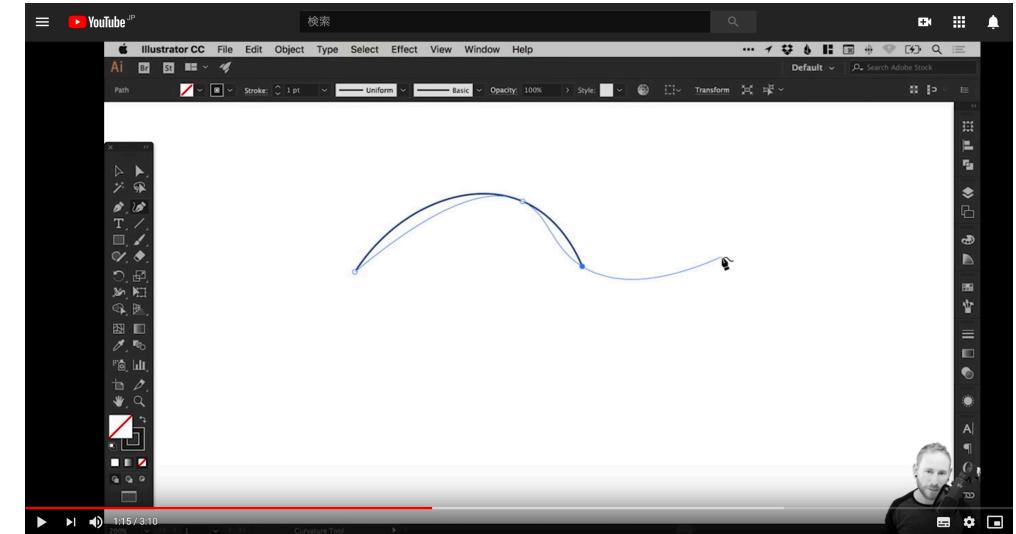
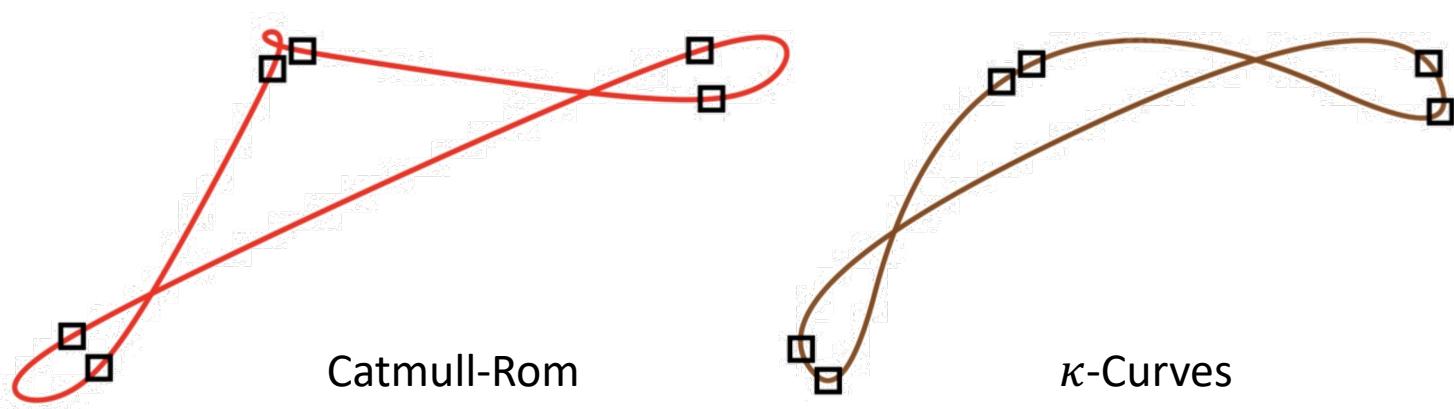
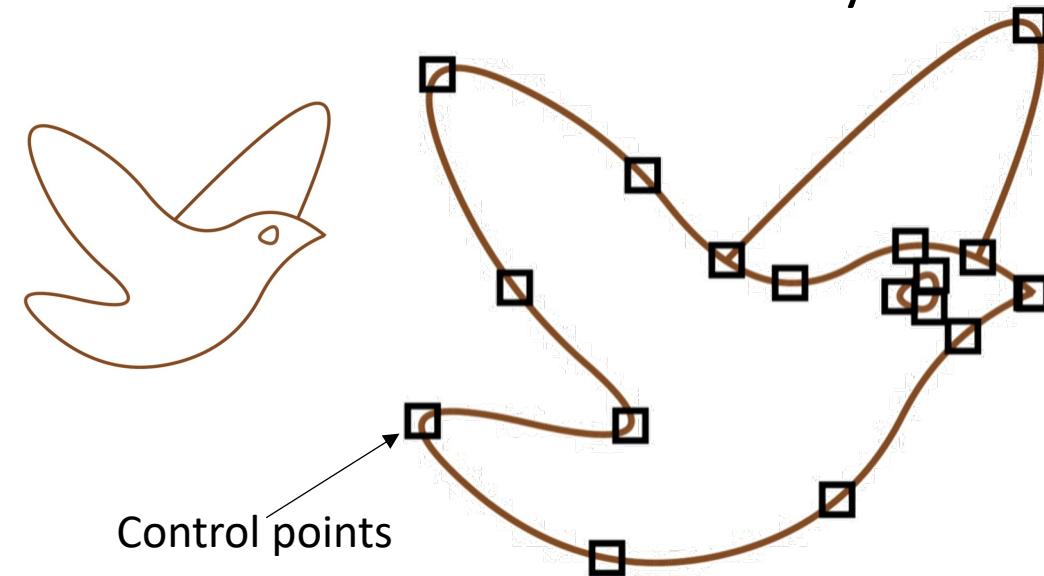


Application of cubic Catmull-Rom spline: Hair modeling



Recent exciting development: κ -Curves

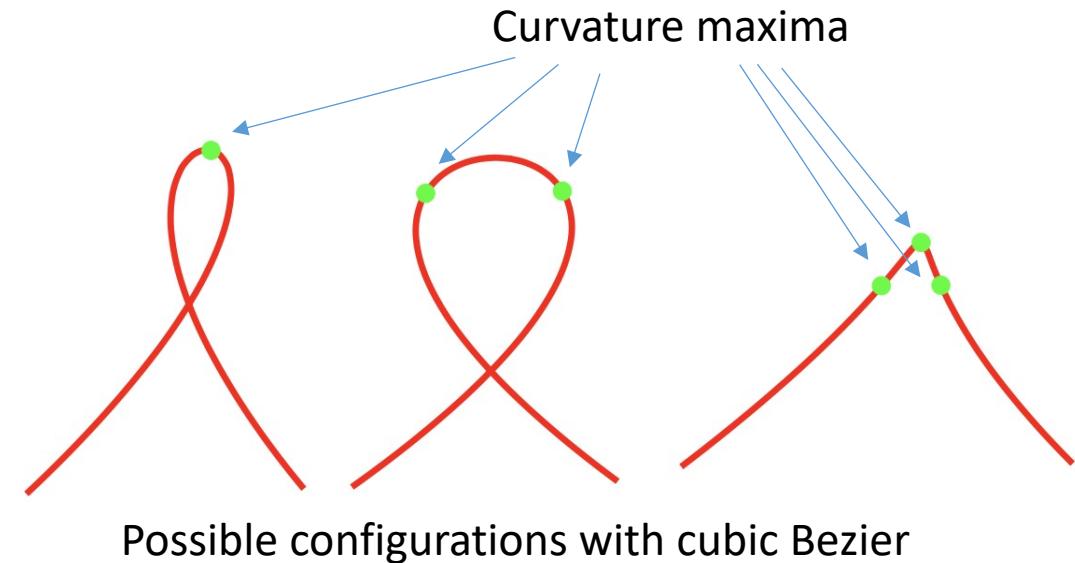
- Collaboration between university & company (Adobe)
- Features:
 - C^2 continuous (smoother)
 - Curvature maxima always on control points



<https://www.youtube.com/watch?v=MLg8xQgElfk>

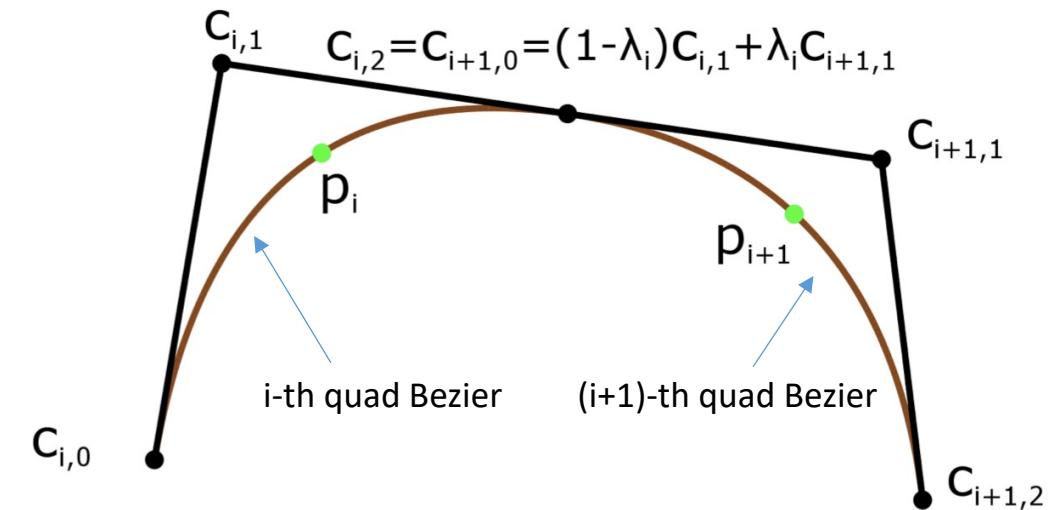
Key ideas of κ -Curves

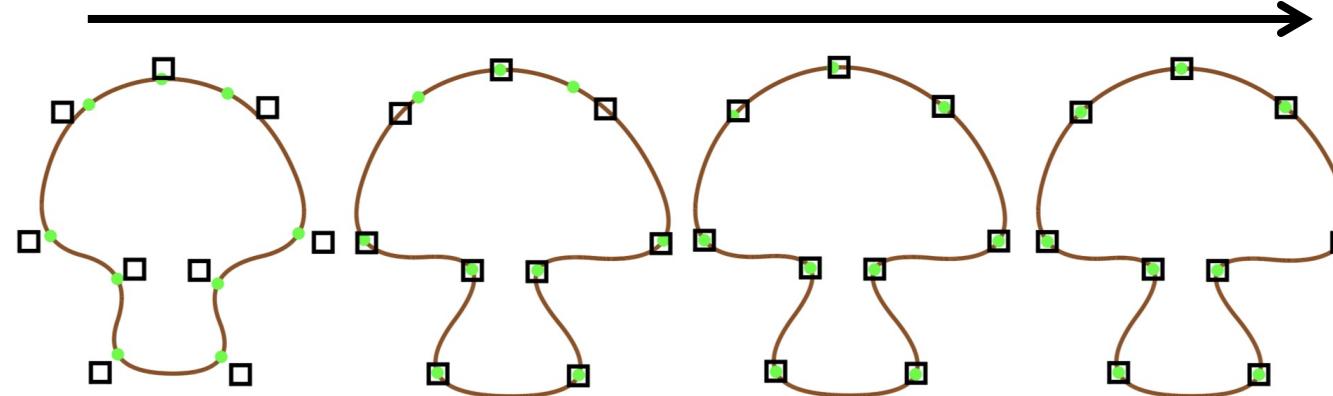
- Cubic Bezier is difficult to control



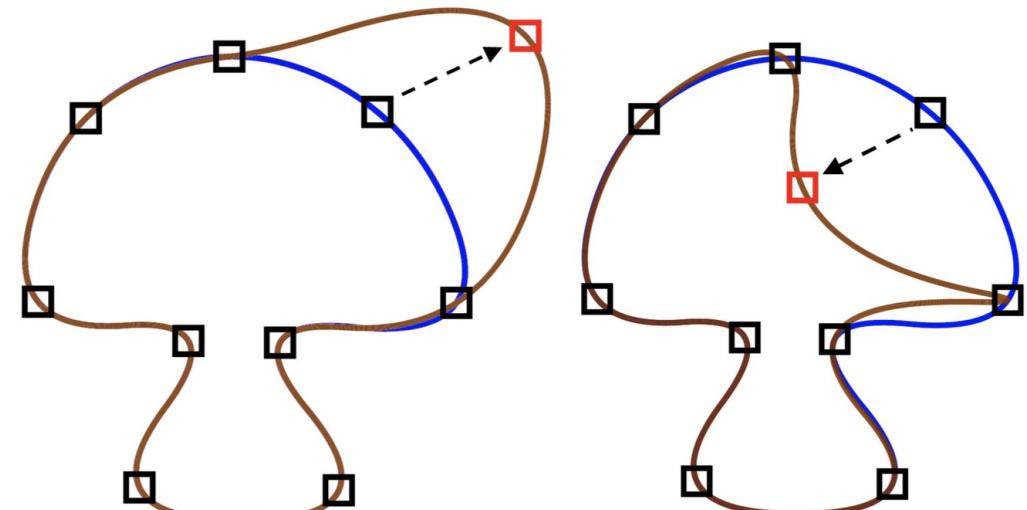
- Actually, *quadratic* Bezier is easier to use!

- At most one curvature maximum can exist
- User specifies curvature maxima
→ reverse compute control points of quadratic Bezier

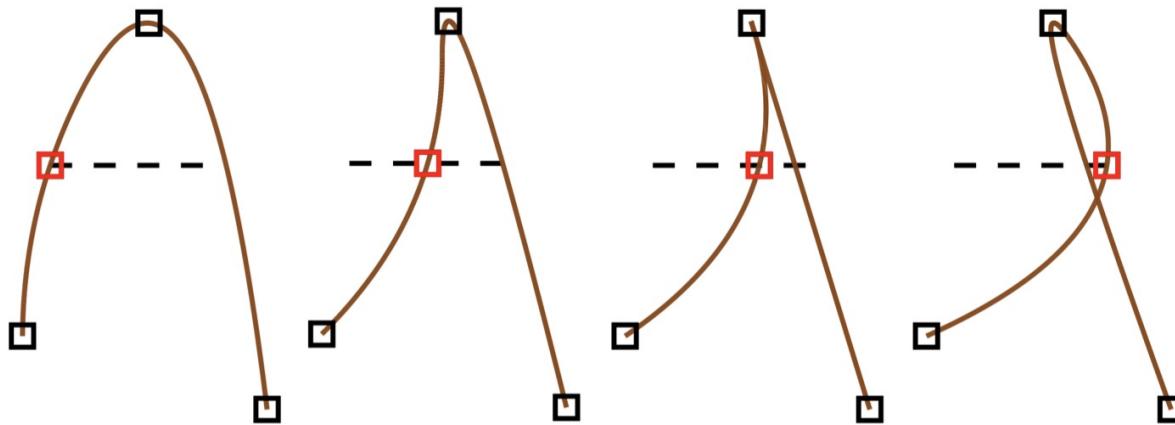




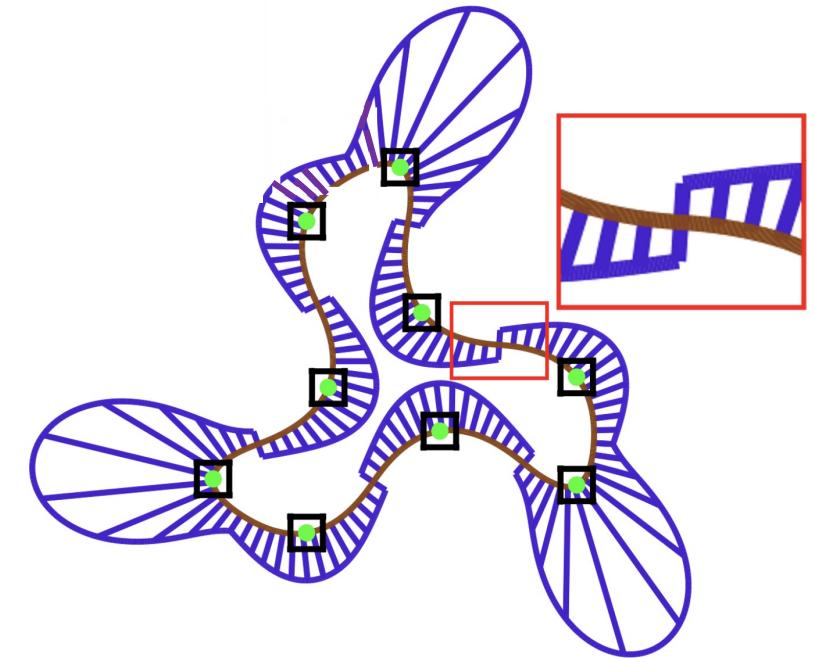
Global/nonlinear formulation = iterative computation



Change of one CP = change of entire shape



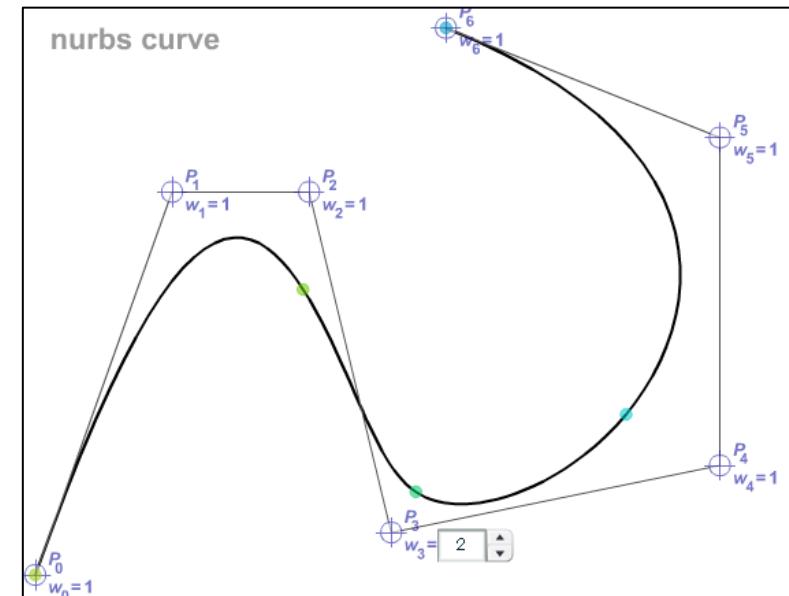
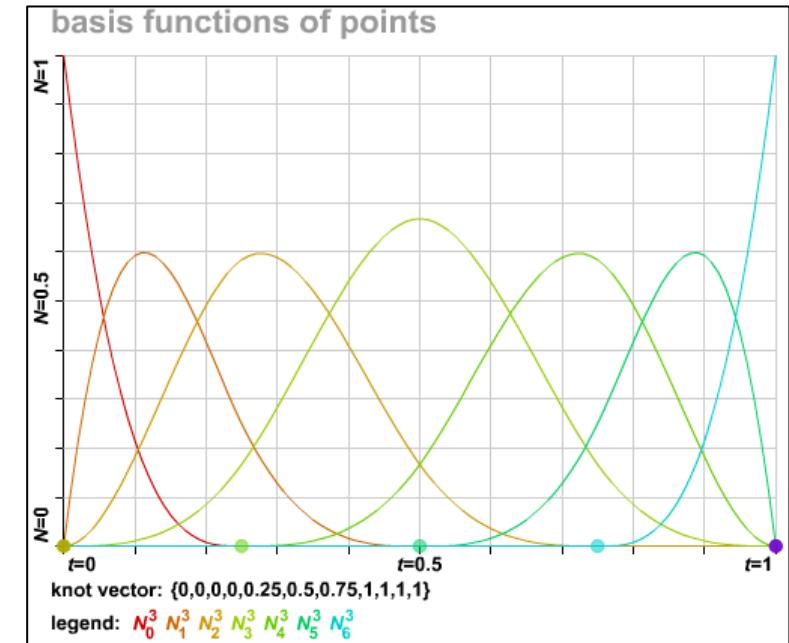
"Buckling" always occurs on CPs



Curvature discontinuity at convex/concave boundary

B-spline

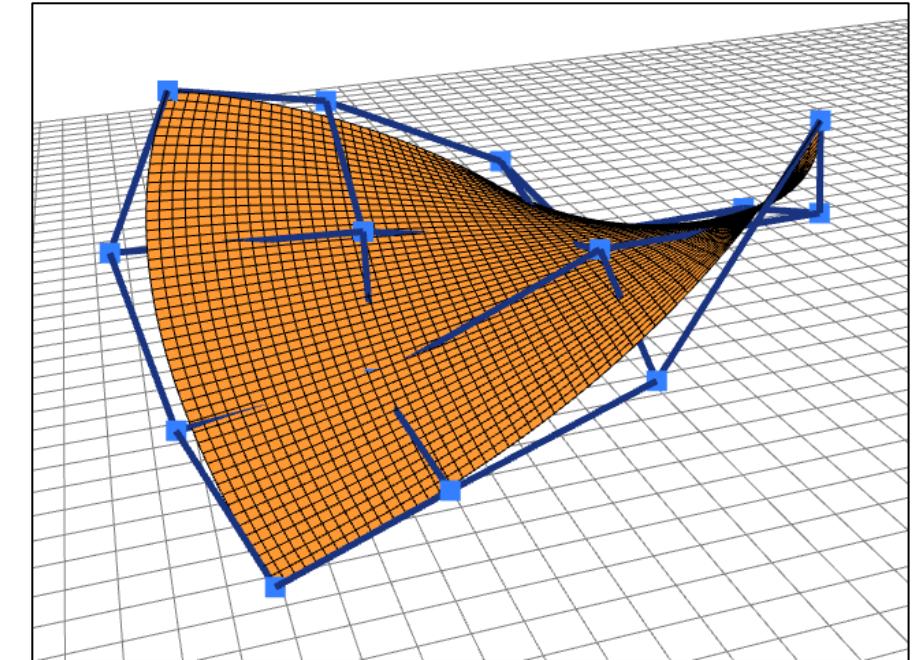
- Another way of defining polynomial spline
 - Represent curve as sum of **basis functions**
 - Cubic basis is the most commonly used
 - Deeply related to subdivision surfaces
→ Next lecture
- Non-Uniform Rational B-Spline
 - Non-Uniform = varying spacing of knots (t_k)
 - Rational = arbitrary weights 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: $4 \times 4 = 16$ 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

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

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

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

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

Coons patch

- Given four curves joining at endpoints, evaluate pairs of opposing curves, then interpolate them → a surface patch

$$L_c(s, t) = (1 - t)c_0(s) + tc_1(s)$$

$$L_d(s, t) = (1 - s)d_0(t) + sd_1(t)$$

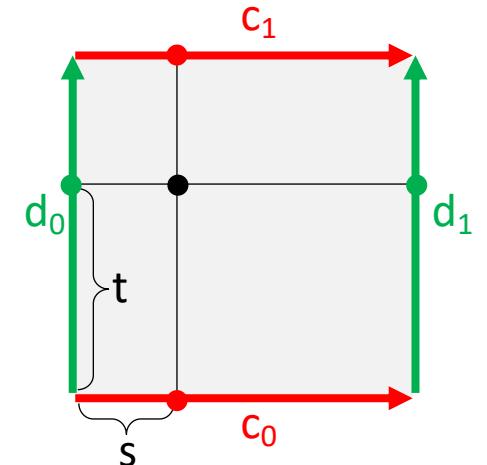
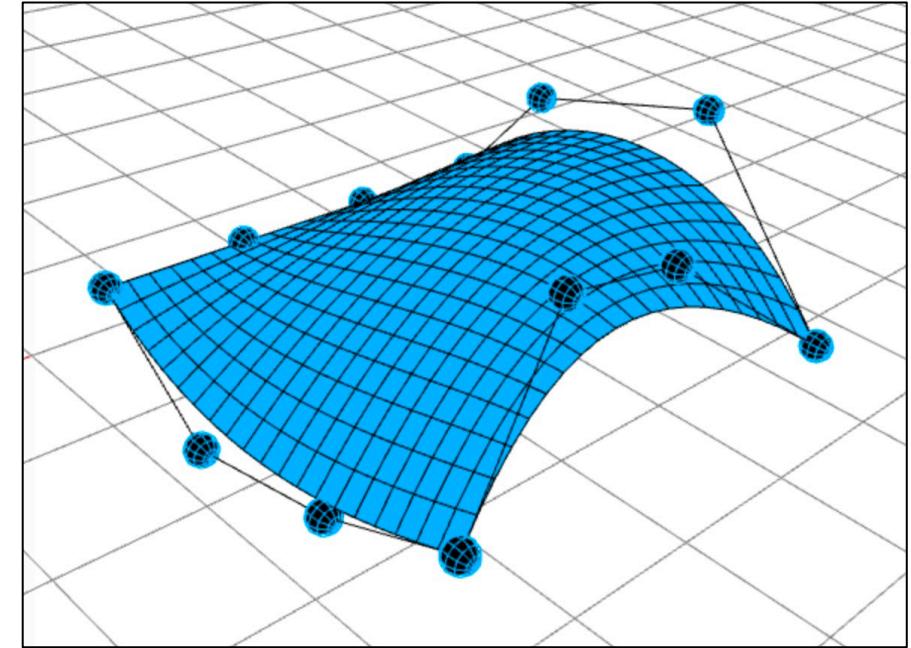
Patch function:

$$C(s, t) = L_c(s, t) + L_d(s, t) - B(s, t)$$

where

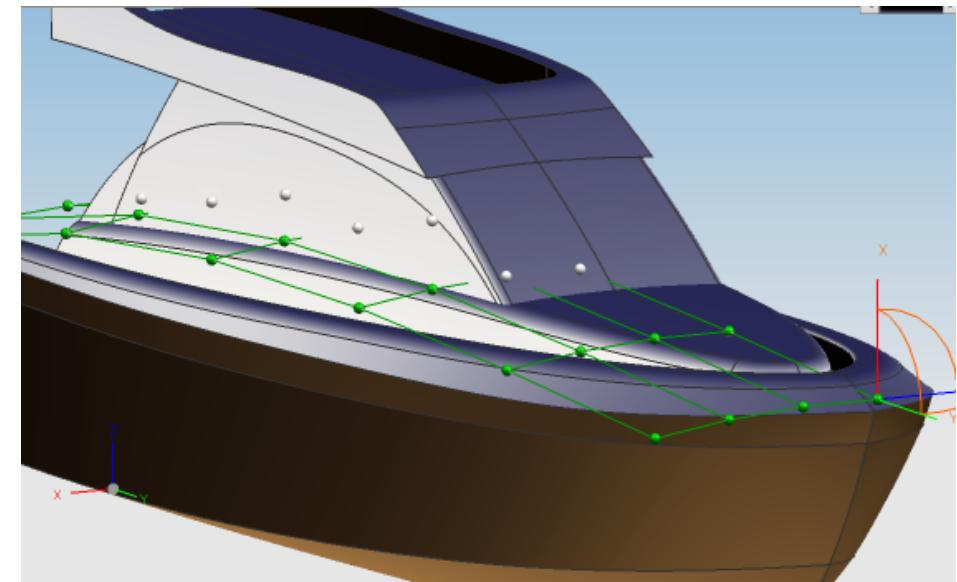
$$B(s, t) = c_0(0)(1 - s)(1 - t) + c_0(1)s(1 - t) + c_1(0)(1 - s)t + c_1(1)st$$

is the bilinear interpolation of four corners



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