matfyz

Computer Graphics Charles University

Mathematics for 3D graphics

CGG MFF UK Praha

pepca@cgg.mff.cuni.cz

https://cgg.mff.cuni.cz/~pepca/

Content

Homogeneous coordinates, matrix transformations

- coordinate-system conversions

Coordinate systems, projections, frustum

Orientations

- Euler angles, quaternions
- orientation interpolation

Smooth interpolations and approximations

- spline functions, natural spline, B-spline
- Hermite-type interpolations
- KB spline, Catmull-Rom…

Geometric transformations in 3D

Cartesian 3D coordinate vector [x, y, z]

Multiplying by a 3×3 matrix

- **row** vector multiplied **from the right** (DirectX)

$$[x, y, z] \cdot \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = [x', y', z']$$

- **column** vector multiplied **from the left** (OpenGL)

Transform matrices 3×3 have serious drawback – **cannot do translations!**

Homogeneous coordinates

Homogeneous coordinate vector [x, y, z, w]

Transformation: multiplying by a 4×4 matrix

$$[x, y, z, w] \cdot \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} = [x', y', z', w']$$

Homogeneous matrix is able to do **translations** and **perspective projections**

Coordinate conversions

From **homogeneous coordinates** [x, y, z, w] into Cartesian coordinates: by division ($w \neq 0$) **[x/w, y/w, z/w]**

Coordinate vector **[x, y, z, 0]** does not correspond to any real point in space

- can be interpreted as a **directional vector** (point in infinity)

From **Cartesian coordinates** to homogeneous: trivial extension **[x, y, z]** … **[x, y, z, 1]**

Elementary transformations

Affine transformation

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & 0 \\ a_{21} & a_{22} & a_{23} & 0 \\ a_{31} & a_{32} & a_{33} & 0 \\ t_1 & t_2 & t_3 & 1 \end{bmatrix}$$

Upper left submatrix **[a_{11} to a_{33}]** defines scaling, orientation and shear

Vector **[t_1, t_2, t_3]** defines translation

- translation is performed as the last step

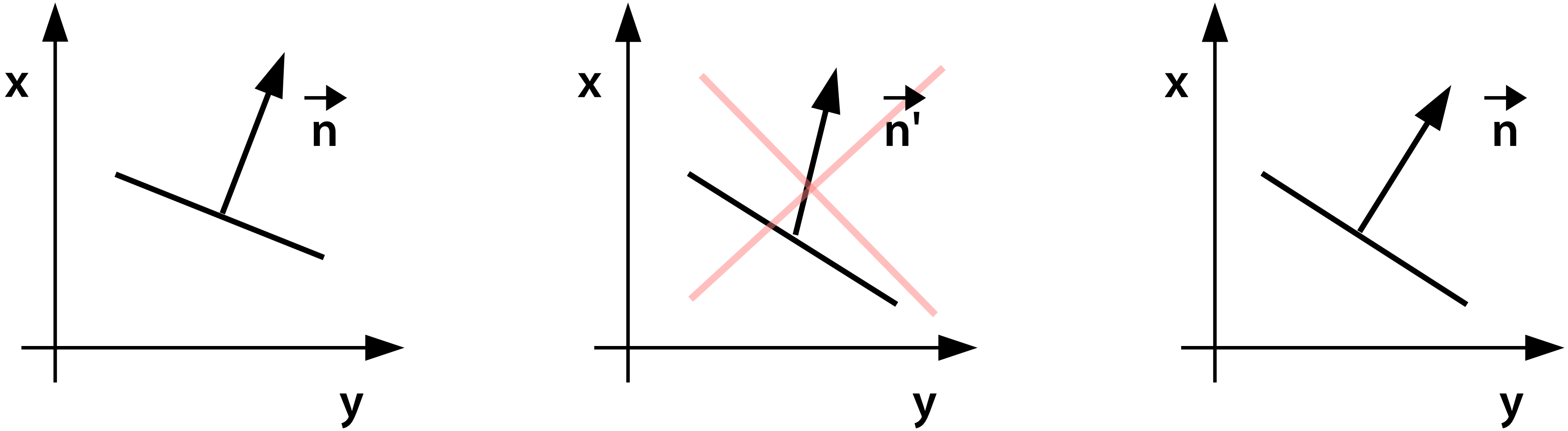
Normal vector transformation

Normal vectors must not be transformed by regular matrices (like point positions are)

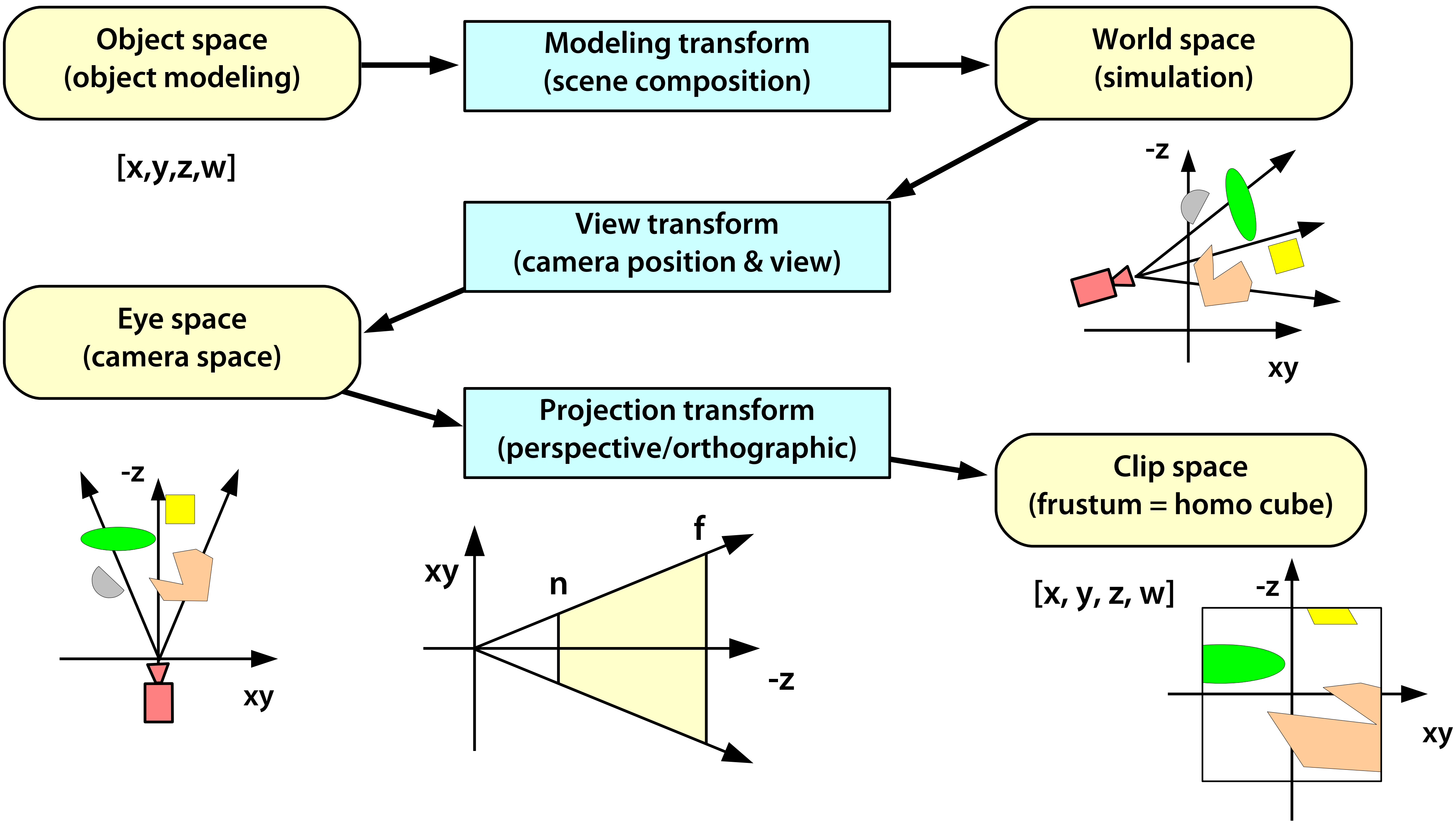
– exception: M is rotational (orthonormal)

Normal-vector transformation matrix **N**:

$$N = (M^{-1})^T$$



Coordinate systems in OpenGL



Coordinate systems in OpenGL

Clip space
(frustum = homo cube)

[x, y, z, w]

→ **Perspective divide**
(hardwired)

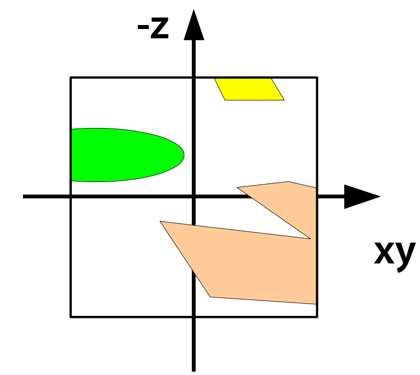
→ **Normalized Device Space**
(frustum = cartesian cube)

[x, y, z]

→ **Viewport transform**
(3D scaling & translation)

→ **Window space**
(actual pixel coords)

[x, y, z]



OpenGL: [-1, -1, -1] to [1, 1, 1]
DirectX: [-1 ,-1, 0] to [1 ,1, 1]

[x, y]	actual screen coordinates (fragments)
z	depth value compatible with actual depth-buffer

Coordinate systems in OpenGL

Object space

- modeling of individual objects, modularity
- 3D modeling software (3DS Max, Blender, Rhino…)

World space

- absolute (real) coordinates in simulated virtual world
- object instantiation, collision detection, AI planning…

Camera space

- the whole virtual world transforms into coordinates relative to a camera
- center of projection: **origin**, view direction: **-z** (or **z**)

Coordinate systems & transformations

Transformation "model $\rightarrow$ camera"

- altogether – "model-view" matrix
- world coordinates are not directly used in rendering pipeline

Projection transformation

- defines visible volume = **frustum** [**l**, **r**, **b**, **t**, **n**, **f**]
- front & back clip distances: **n**, **f**
- result: homogeneous coordinate (before clipping)

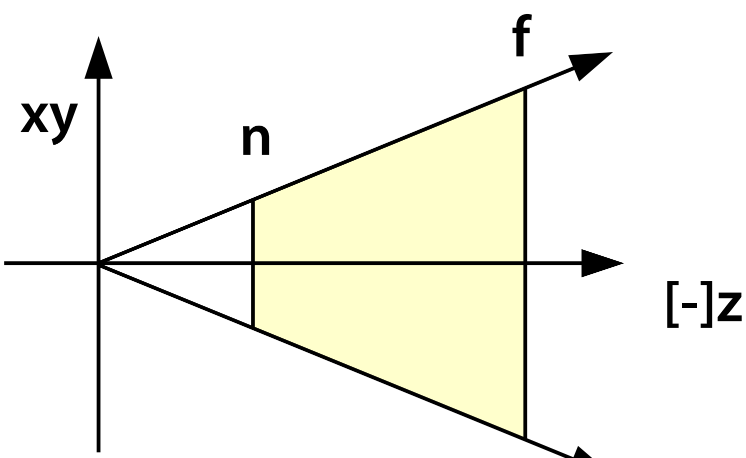
"Clip space"

- **mandatory output coordinate** of vertex shader!

Projection transform (perspective)

Far point **f** can be **in infinity**

$$\begin{bmatrix} \frac{2n}{r-l} & 0 & 0 & 0 \\ 0 & \frac{2n}{t-b} & 0 & 0 \\ -\frac{r+l}{r-l} & -\frac{t+b}{t-b} & \frac{f+n}{f-n} & 1 \\ 0 & 0 & -\frac{2fn}{f-n} & 0 \end{bmatrix} \rightarrow \begin{bmatrix} \frac{2n}{r-l} & 0 & 0 & 0 \\ 0 & \frac{2n}{t-b} & 0 & 0 \\ -\frac{r+l}{r-l} & -\frac{t+b}{t-b} & 1 & 1 \\ 0 & 0 & -2n & 0 \end{bmatrix}$$



Coordinate systems & transforms

Perspective division

– just converts **homogeneous** coordinates into **cartesian**

Normalized coordinates ("NDS")

– standard-sized cube/cuboid
– OpenGL: [-1, -1, -1] to [1, 1, 1]
– DirectX: [-1, -1 ,0] to [1, 1, 1]

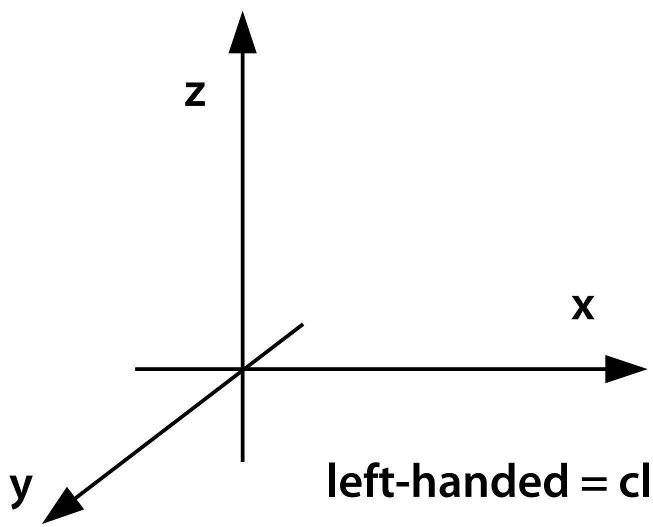
Window coordinates ("window space")

– result of **linear adjustment** to window size in pixels
– used in **rasterizer** and all **fragment processing**

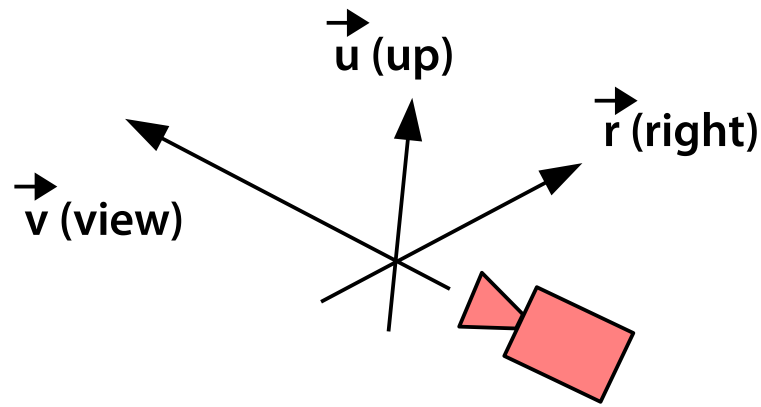
Rigid body transformation

Preserves **shapes**, alters **orientation & position**

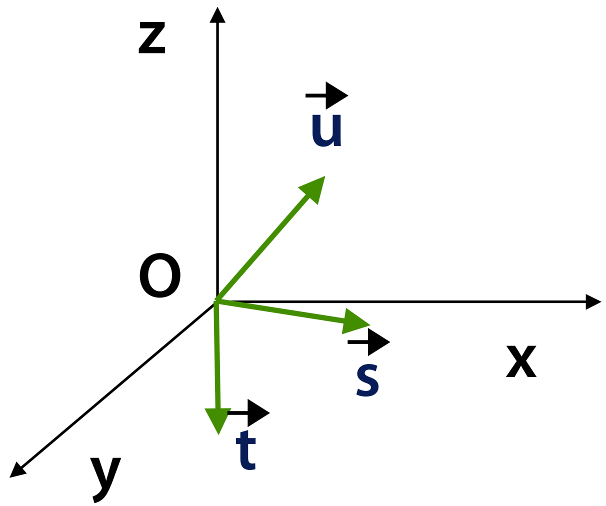
- **translation** and **rotation**
- **conversion between coordinate systems** (e.g. between world-space and camera-space)



left-handed = clockwise ("pravotočivý" in Czech)



Conversion between two orientations



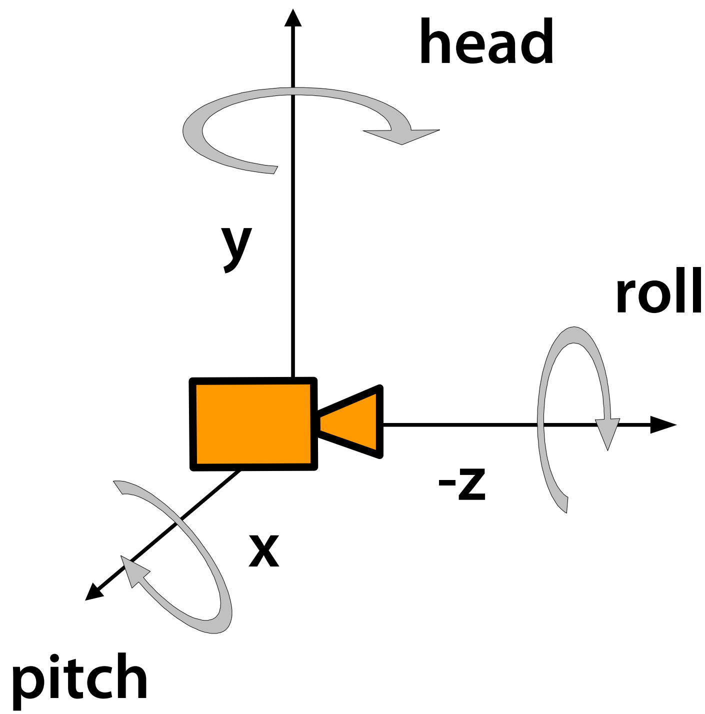
Coordinate system has an origin **O** and is defined by three unit vectors **[s, t, u]**

$$M_{stu \to xyz} = \begin{bmatrix} s_x & s_y & s_z \\ t_x & t_y & t_z \\ u_x & u_y & u_z \end{bmatrix}$$

$$[1, 0, 0] \cdot M_{stu \to xyz} = s$$
$$[0, 1, 0] \cdot M_{stu \to xyz} = t$$
$$[0, 0, 1] \cdot M_{stu \to xyz} = u$$

$$M_{xyz \to stu} = M^T_{stu \to xyz}$$

Euler transformation



Arbitrary rotation decomposed into **three components**

– Leonard Euler (1707-1783)

$$E(h, p, r) = R_y(h) \cdot R_x(p) \cdot R_z(r)$$

h (**head**, yaw): plan view direction

p (**pitch**): forward/backward pitching

r (**roll**): rolling around the view vector

Euler transformation II

Result matrix of rotation

$$E = \begin{pmatrix} c(r)c(h) - s(r)s(p)s(h) & s(r)c(h) + c(r)s(p)s(h) & -c(p)s(h) \\ -s(r)c(p) & c(r)c(p) & s(p) \\ c(r)s(h) + s(r)s(p)c(h) & s(r)s(h) - c(r)s(p)c(h) & c(p)c(h) \end{pmatrix}$$

s(x) … sin(x), c(x) … cos(x)

Backward matrix → angles computation h, p, r

- p … e_{23}
- r … e_{21}/e_{22}
- h … e_{13}/e_{33}

Rotations: different conventions

Main convention

- 1. rotation around **z** by φ
- 2. rotation around **x'** by θ
- 3. rotation around **z''** by ψ

X-convention

- 1. rotation around **z**
- 2. rotation around <u>original</u> **x**
- 3. rotation around <u>original</u> **z**

More systems (24): aeronautics, gyroscopes, physics…

Quaternions

Sir William Rowan **Hamilton**, 16 Oct 1843 (Dublin)

- $i^2 = j^2 = k^2 = ijk = -1$
- usage in graphics since 1985 (Shoemake)
- **generalization of complex numbers** in 4D space

$\mathbf{q} = (\mathbf{v}, w) = i\,x + j\,y + k\,z + w = \mathbf{v} + w$ sometimes $(w, \mathbf{v})$!

Imaginary part $\mathbf{v} = (x, y, z) = i\,x + j\,y + k\,z$

$i^2 = j^2 = k^2 = -1, \quad jk = -kj = i, \quad ki = -ik = j, \quad ij = -ji = k$

Quaternions: operations I

Addition

- $(\mathbf{v_1}, w_1) + (\mathbf{v_2}, w_2) = (\mathbf{v_1} + \mathbf{v_2}, w_1 + w_2)$

Multiplication

- $\mathbf{q}\,\mathbf{r} = (\mathbf{v_q} \times \mathbf{v_r} + w_r\mathbf{v_q} + w_q\mathbf{v_r},\ w_q w_r - \mathbf{v_q} \cdot \mathbf{v_r})$

$$
\begin{aligned}
& i(q_y r_z - q_z r_y + r_w q_x + q_w r_x), \\
& j(q_z r_x - q_x r_z + r_w q_y + q_w r_y), \\
& k(q_x r_y - q_y r_x + r_w q_z + q_w r_z), \\
& \quad q_w r_w - q_x r_x - q_y r_y - q_z r_z
\end{aligned}
$$

Quaternions: operations II

Conjugation

- $(\mathbf{v}, w)^* = (-\mathbf{v}, w)$

Norm (squared absolute value)

- $||\mathbf{q}||^2 = n(\mathbf{q}) = \mathbf{q}\,\mathbf{q}^* = x^2 + y^2 + z^2 + w^2$

Unit

- $\mathbf{i} = (\mathbf{0}, 1)$

Reciprocal

- $\mathbf{q}^{-1} = \mathbf{q}^* / n(\mathbf{q})$

Multiplication by a scalar

- $s\,\mathbf{q} = (0, s)\,(\mathbf{v}, w) = (s\,\mathbf{v}, s\,w)$

Unit quaternions

Every unit quaternion ($x^2 + y^2 + z^2 + w^2 = 1$) can be expressed as

- $\mathbf{q} = (\mathbf{u_q} \sin\phi, \cos\phi)$
- for some **unit 3D vector** $\mathbf{u_q}$

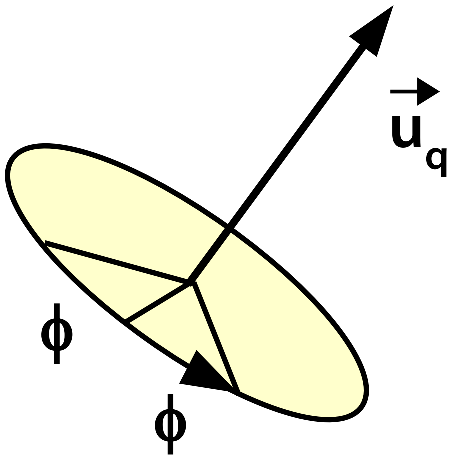
It represents a **rotation (orientation)** in 3D

- **ambiguity**: both **q** and **-q** represent the same rotation! $(\phi + \pi)$
- **identity** (zero rotation): $(\mathbf{0}, 1)$

Power, exponential, logarithm

- $\mathbf{q} = \mathbf{u_q} \sin\phi + \cos\phi = \exp(\phi\, \mathbf{u_q})$, $\log \mathbf{q} = \phi\, \mathbf{u_q}$
- $\mathbf{q}^t = (\mathbf{u_q} \sin\phi + \cos\phi)^t = \exp(t\phi\, \mathbf{u_q}) = \mathbf{u_q} \sin t\phi + \cos t\phi$

Rotation using a quaternion



Unit quaternion

- $\mathbf{q} = (\mathbf{u_q} \sin\phi, \cos\phi)$
- $\mathbf{u_q}$ ... axis of rotation, $\boldsymbol{\phi}$ ... angle

Vector (point) in 3D: $\mathbf{p} = [p_x, p_y, p_z, 0]$

Rotation of vector (point) **p** around $\mathbf{u_q}$ by angle $\mathbf{2\phi}$

$$\mathbf{p' = q\,p\,q^{-1} = q\,p\,q^*}$$

Quaternion ↔ matrix conversions

Quaternion **q** converted to a matrix

$$M=\begin{pmatrix} 1-2(y^2+z^2) & 2(xy+wz) & 2(xz-wy) \\ 2(xy-wz) & 1-2(x^2+z^2) & 2(yz+wx) \\ 2(xz+wy) & 2(yz-wx) & 1-2(x^2+y^2) \end{pmatrix}$$

Reverse conversion is based on equations

$$\begin{aligned} m_{23}-m_{32} &= 4wx \\ m_{31}-m_{13} &= 4wy \\ m_{12}-m_{21} &= 4wz \\ tr\,M+1 &= 4w^2 \end{aligned} \qquad (\$)$$

Matrix → quaternion II

1. "**matrix_trace+1**" has large enough absolute value

$$w=\frac{1}{2}\sqrt{tr\,M+1} \quad x=\frac{m_{23}-m_{32}}{4w}$$

$$y=\frac{m_{31}-m_{13}}{4w} \quad z=\frac{m_{12}-m_{21}}{4w}$$

2. … otherwise compute a component with largest absolute value first and then apply **$**

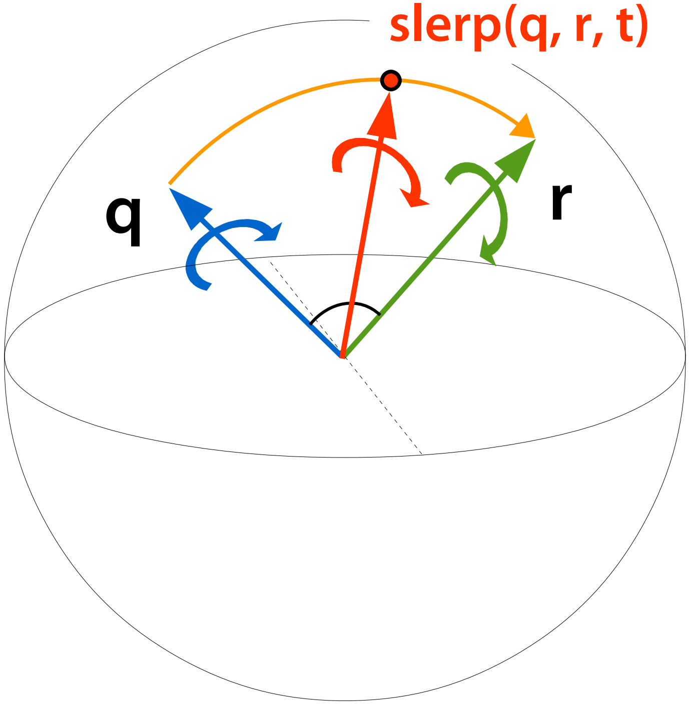
$$4x^2=1+m_{11}-m_{22}-m_{33}$$
$$4y^2=1-m_{11}+m_{22}-m_{33}$$
$$4z^2=1-m_{11}-m_{22}+m_{33}$$

Spherical linear interpolation (slerp)

Two quaternions **q** and **r** (**q** · **r** ≥ 0, else take **-q**)

Real parameter **0 ≤ t ≤ 1**

Interpolated quaternion **slerp(q, r, t) = q (q* r)^t**



$$slerp(q,r,t) = \frac{\sin(\phi(1-t))}{\sin\phi} \cdot q + \frac{\sin(\phi t)}{\sin\phi} \cdot r$$

$$\cos\phi = q_x r_x + q_y r_y + q_z r_z + q_w r_w$$

The shortest spherical arc
between **q** and **r**
(quaternion splines will be explained later)

Rotation between two vectors

Two vectors **s** and **t**

1. normalization of **s**, **t**
2. unit rotation axis $\mathbf{u} = (\mathbf{s} \times \mathbf{t}) \,/\, ||\mathbf{s} \times \mathbf{t}||$
3. angle between **s** and **t** $\mathbf{e} = \mathbf{s} \cdot \mathbf{t} = \cos 2\phi$

   $||\mathbf{s} \times \mathbf{t}|| = \sin 2\phi$

4. final quaternion $\mathbf{q} = (\, \mathbf{u} \cdot \sin \phi, \cos \phi \,)$

$$q = (q_v, q_w) = \left(\frac{1}{\sqrt{2(1+e)}} (s \times t), \frac{\sqrt{2(1+e)}}{2} \right)$$

Slerp of rotational matrices (theory)

Two rotational matrices **Q** and **R**

Real parameter $\mathbf{0 \le t \le 1}$

Interpolated matrix $\mathbf{slerp(Q, R, t) = Q\,(Q^T R)^t}$

Technical problem – how to do power operation on matrices?

Need to compute axis and angle $Q^T R$
(not very efficient)

See “RotationIssues.pdf” for details (D. Eberly)

Rotation representation – summary

Rotational matrix

+ HW support, efficient point/vector transformation

– memory (float[9]), other operations are not so efficient

Rotational axis and angle

+ memory (float[4] or float[6]), similar to quaternion

– inefficient composition and interpolation

Quaternion

+ memory (float[4]), composition, interpolation

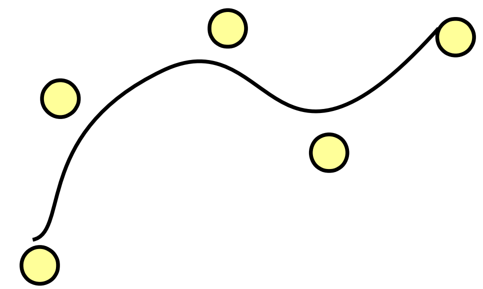
– inefficient point/vector transformation

See "RotationIssues.pdf" for details (D. Eberly)

Approximation and interpolation

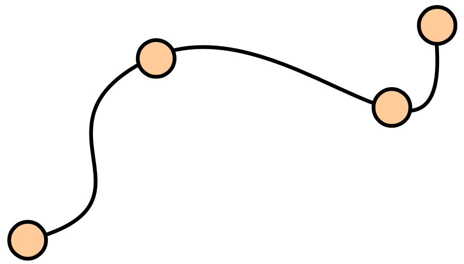
Approximation (e.g. B-spline)

– needs not to pass through control points



Interpolation (e.g. Catmull-Rom)

– curve passes through control points



Curve continuity

– $\mathbf{G}^n$ – geometric continuity of the $\mathbf{n}^{th}$ order ($\mathbf{G}^0$ – simple continuity, $\mathbf{G}^1$ – tangent, $\mathbf{G}^2$ – curvature…)

– $\mathbf{C}^n$ – analytical continuity of the $\mathbf{n}^{th}$ order, $\mathbf{n}^{th}$ derivative continuity ($\mathbf{C}^1$ – speed, $\mathbf{C}^2$ – acceleration), superior to geometric continuity

History

Curves in modeling industry

- Paul de Faget **de Casteljau**, Citroën (1959)
- Pierre **Bèzier** (Renault 1933-1975, UNISURF)
  - late start, but his results were more popular
- application of spline function theory – mostly in USA (James **Ferguson**, 1964, Boeing, C^2 spline curves)

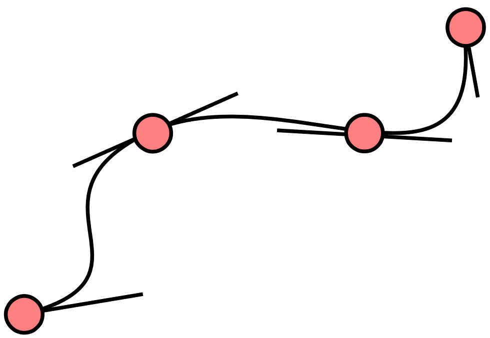
Spline function theory

- B-spline: Isaac Jacob **Schoenberg**, (ballistics, Aberdeen, MD, 1946)
- theory: Carl **de Boor** (also worked for General Motors)
- Gordon, Riesenfeld **united** Bèzier and B-spline curves (1972)

“Free-form“ curves I

Defined by a sequence of **control points**

- “control polygon”
- approximation or interpolation
- boundary conditions can be different



Controllability

- sometimes **tangent vectors** added in control points (Hermit)
- interpolation $\Rightarrow$ closer control

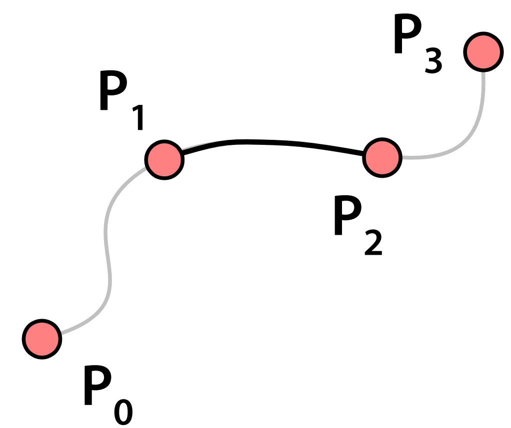
Locality

- change of single control point (one tangent vector) induces change in a **restricted neighborhood** only

“Free-form“ curves II

Parametric expression (**0 ≤ t ≤ 1**)

$$P(t) = \sum_{i=0}^{N-1} w_i(t)\, P_i$$



Convex hull property

– curve lies in convex hull of its control polygon

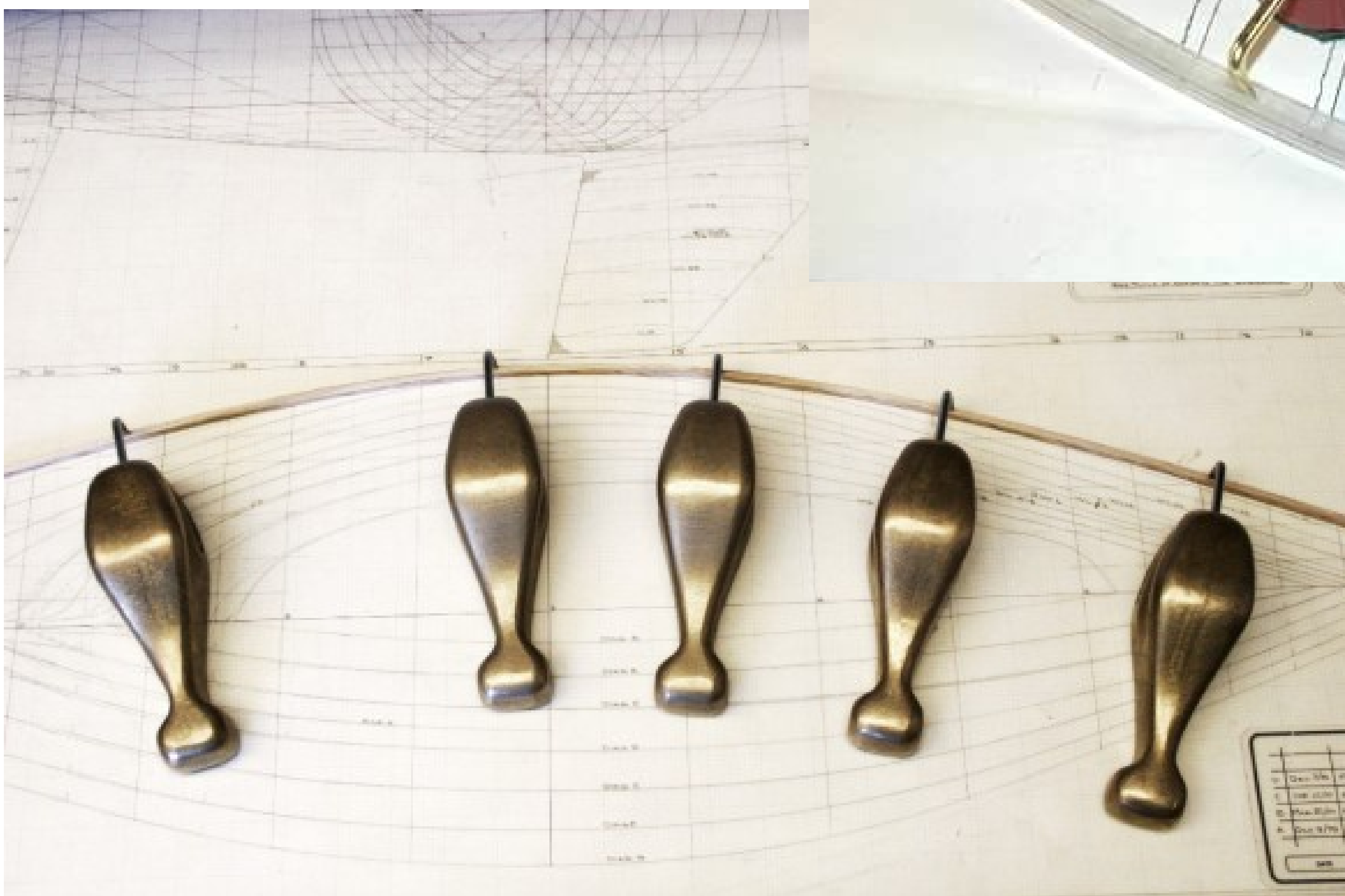
Cauchy condition for blending functions

– sufficient for convex hull property

– ensures **affine transformation invariancy**

$$\sum_{i=0}^{N-1} w_i(t) = 1$$

Splines



© Jay Greer

© Edson International

Spline functions

Named after elastic ruler used in ship design (pinned in several points by “ducks”)

Definition: **spline function of degree n**

- piece-wise **polynomial** (of degree **n**)
- **maximum-smoothness connection**:

$\mathbf{C^{n-1}}$ – continuity of **n-1**th derivative (polynomial of degree **n**)

- **global parametrization u**, $\mathbf{u_0 \le u \le u_N}$ $[u_0, u_1, \dots u_N]$
- individual parts are often uniformly parametrized – **uniform spline** $\mathbf{t_i = (u - u_i) / (u_{i+1} - u_i)}$, $\mathbf{0 \le t_i \le 1}$

Polynomial curve

Matrix notation

$$P(t) = \boldsymbol{TC} = [t^n, t^{n-1}, \ldots t, 1] \cdot \begin{bmatrix} x_n & y_n & z_n \\ x_{n-1} & y_{n-1} & z_{n-1} \\ .. & .. & .. \\ x_1 & y_1 & z_1 \\ x_0 & y_0 & z_0 \end{bmatrix}$$

Basis matrix **M** and vector of geometric conditions **G**

$$\boldsymbol{C} = \boldsymbol{MG} = [m_{ij}]_{i=n,\, j=1}^{0,\, k} \cdot \begin{bmatrix} G_1 \\ .. \\ G_k \end{bmatrix} \qquad P(t) = \boldsymbol{TMG}$$

Matrix notation of a curve

P(t) = T C = T M G

- separation of a parameter vector (**T**) from polynomial basis (**M**) and geometric control conditions/points (**G**)
- differentiation (tangent, curvature) restricted to **T**
- control polynomial **TM** times "geometry" **G**

Cubic: n = 3, k = 4

$$Q(t) = [t^3, t^2, t, 1] \cdot \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ m_{41} & m_{42} & m_{43} & m_{44} \end{bmatrix} \cdot \begin{bmatrix} G_1 \\ G_2 \\ G_3 \\ G_4 \end{bmatrix}$$

Hermite cubic curve

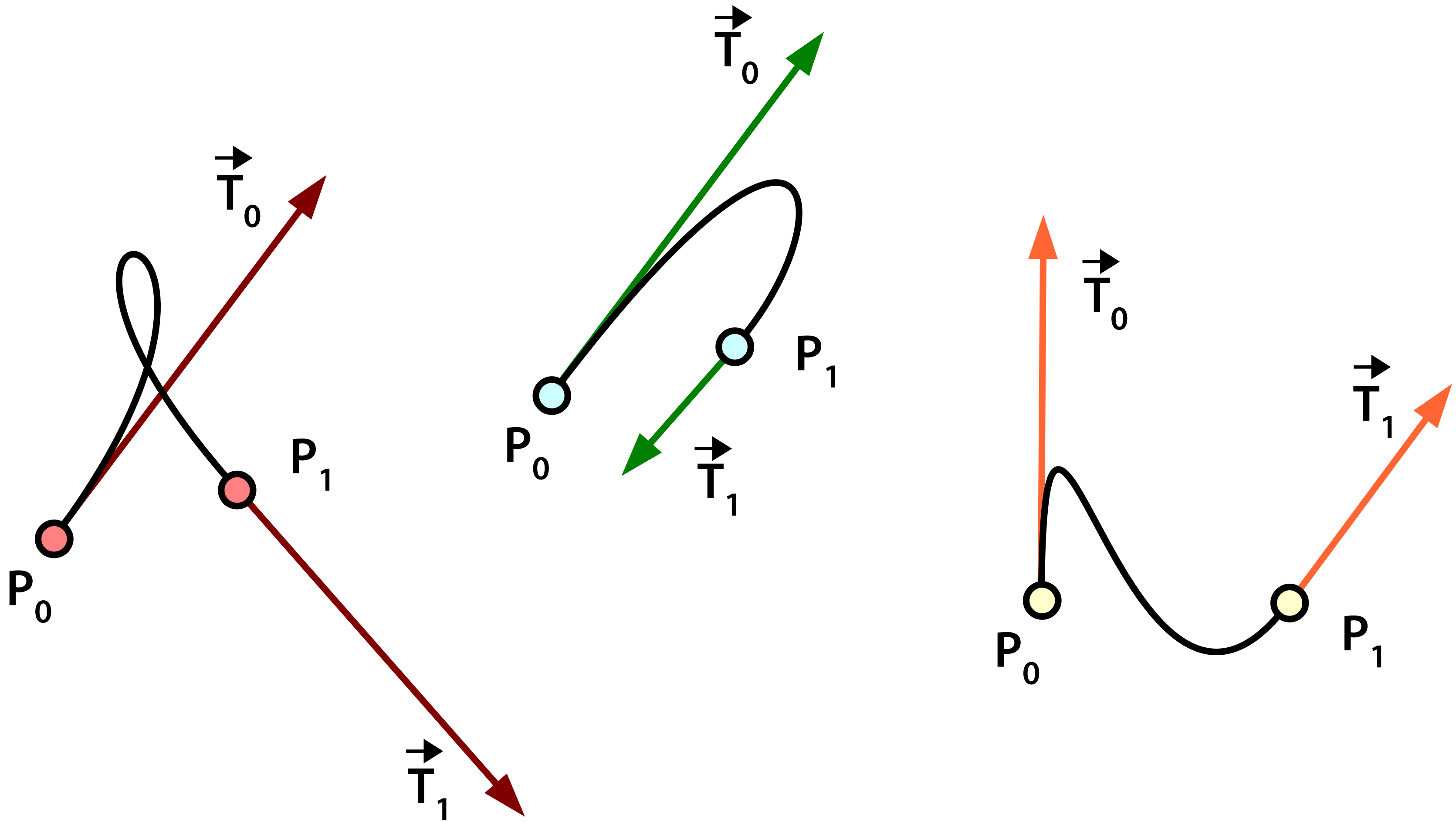
Ferguson curve (cubic)

Geometry: **endpoints** and **tangent vectors**

- beginning (**P_0**) and end (**P_1**) of a curve
- tangents in beginning (**T_0**) and ending (**T_1**) points

$$F(t) = [t^3, t^2, t, 1] \cdot \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} P_0 \\ P_1 \\ T_0 \\ T_1 \end{bmatrix}$$

Hermite cubic – examples



More curves

Interpolating cubics derived from Hermite

- general: **Kochanek-Bartels** (KB-spline, TCB cubic)
- special: **cardinal** spline, **Catmull-Rom** spline
- **Akima** interpolation ("Akima spline", not C^2)
- **D-spline** cubic

Another popular curves

- **Bèzier** curves
- **B-spline** curve, **Coons** spline (approximation)
- **natural** spline (interpolation)

Kochanek-Bartels cubic (KB-spline, TCB)

Derived from **Hermite** cubic (3DS Max, Lightwave)

- **tangent vectors** are derived from **control points**
- three additional scalar parameters (**zero** by default)
  - “**tension**” **t**: sharpness of a curve passing control point (absolute value of a tangent vector)
  - “**continuity**” **c**: in control points
  - “**bias**” **b**: tangent direction in control point

Left and **right** tangent (**T_0** and **T_1** in local sense):

$$L_i = \frac{(1-t)(1-c)(1+b)}{2} \cdot (P_i - P_{i-1}) + \frac{(1-t)(1+c)(1-b)}{2} \cdot (P_{i+1} - P_i)$$

$$R_i = \frac{(1-t)(1+c)(1+b)}{2} \cdot (P_i - P_{i-1}) + \frac{(1-t)(1-c)(1-b)}{2} \cdot (P_{i+1} - P_i)$$

Cardinal spline, Catmull-Rom spline

Special cases of KB-spline

cardinal spline

– parameter **a** only (in fact relates to “**t**”, **c = b = 0**)

$$T_i = a \cdot (P_{i+1} - P_{i-1}) \qquad 0 \leqslant a \leqslant 1$$

Catmull-Rom spline

$$T_i = \frac{1}{2} \cdot (P_{i+1} - P_{i-1})$$

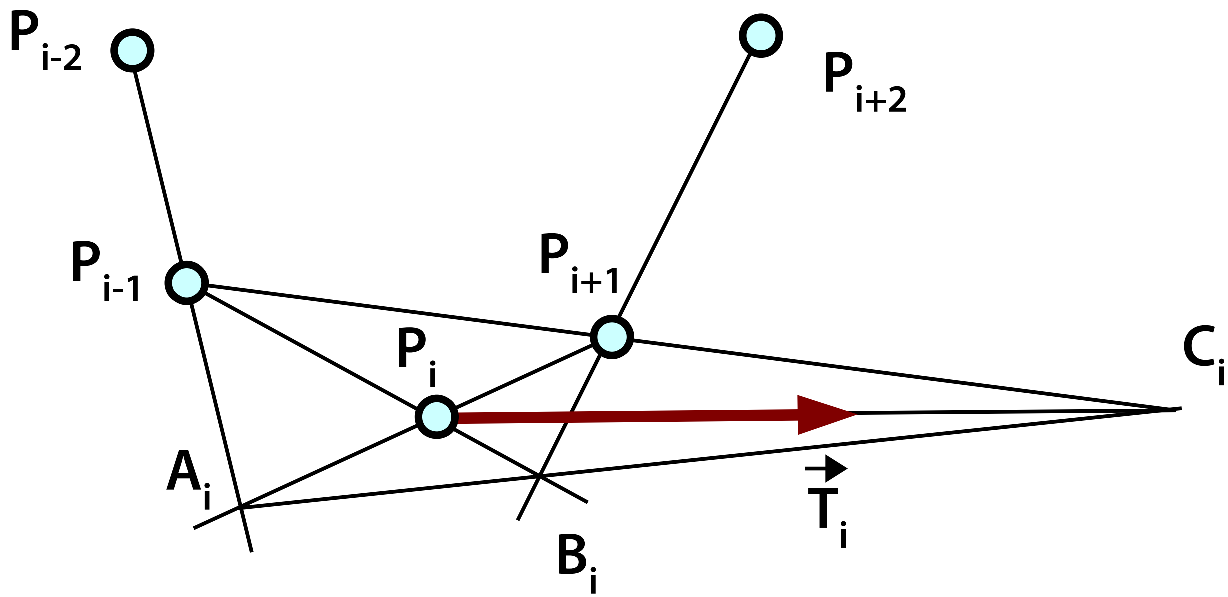
– a = t = 1/2

$$\boldsymbol{MG} = \frac{1}{2}\begin{bmatrix} -1 & 3 & -3 & 1 \\ 2 & -5 & 4 & -1 \\ -1 & 0 & 1 & 0 \\ 0 & 2 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} P_{i-1} \\ P_i \\ P_{i+1} \\ P_{i+2} \end{bmatrix}$$

Akima interpolation

Alternative definition of **tangent vectors** for Hermite cubic:

– **non-C^2 !**

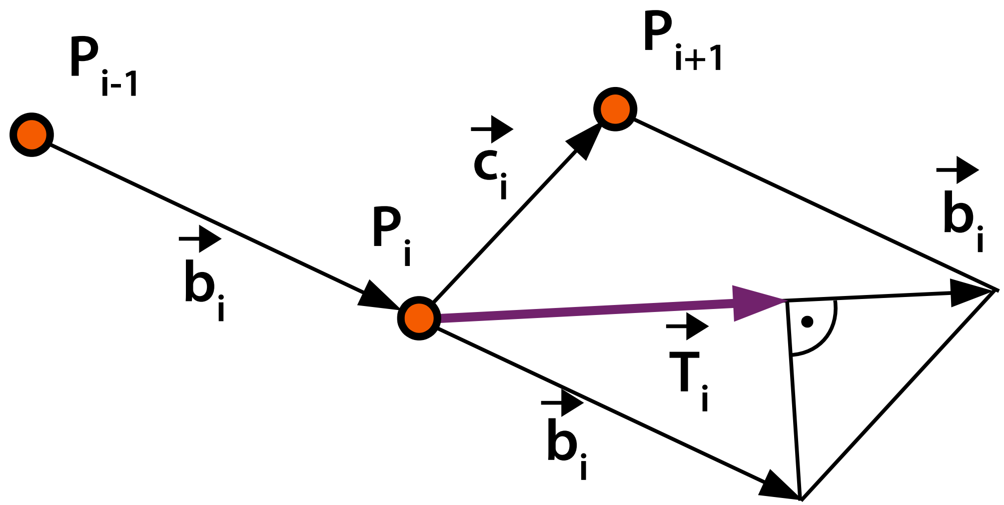


$$|\vec{T}_i| = |\overrightarrow{P_{i+1} - P_{i-1}}|$$

D-spline cubic

One more variant of Hermite cubic

– tangent vector computed by the **“D-interpolation”**



$$G = \begin{bmatrix} P_i \\ P_{i+1} \\ T_i \\ T_{i+1} \end{bmatrix}$$

Bèzier curves I

Polynomial curve of degree N

- **N+1** control points
  - **boundary control points** define **endpoints** of a curve
  - boundary control-point pairs define **tangent vectors**
- parametric expression using **Bernstein polynomials**
- easy $\mathbf{G}^1$ or $\mathbf{C}^1$ connection
- spline-join is also possible, but much more complicated

Bernstein polynomials:

$$B_i^n(t) = \binom{n}{i} t^i (1-t)^{n-i} \quad 0 \leqslant i \leqslant n, \quad 0 \leqslant t \leqslant 1$$

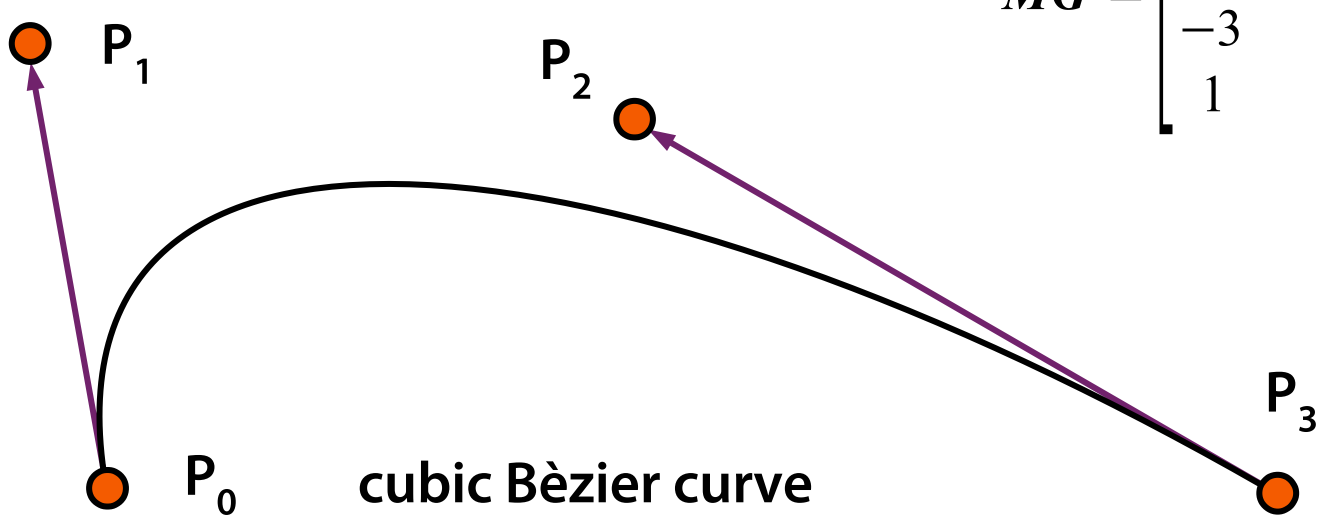
Bèzier curves II

Cauchy condition

$\Rightarrow$ convex combination of control points

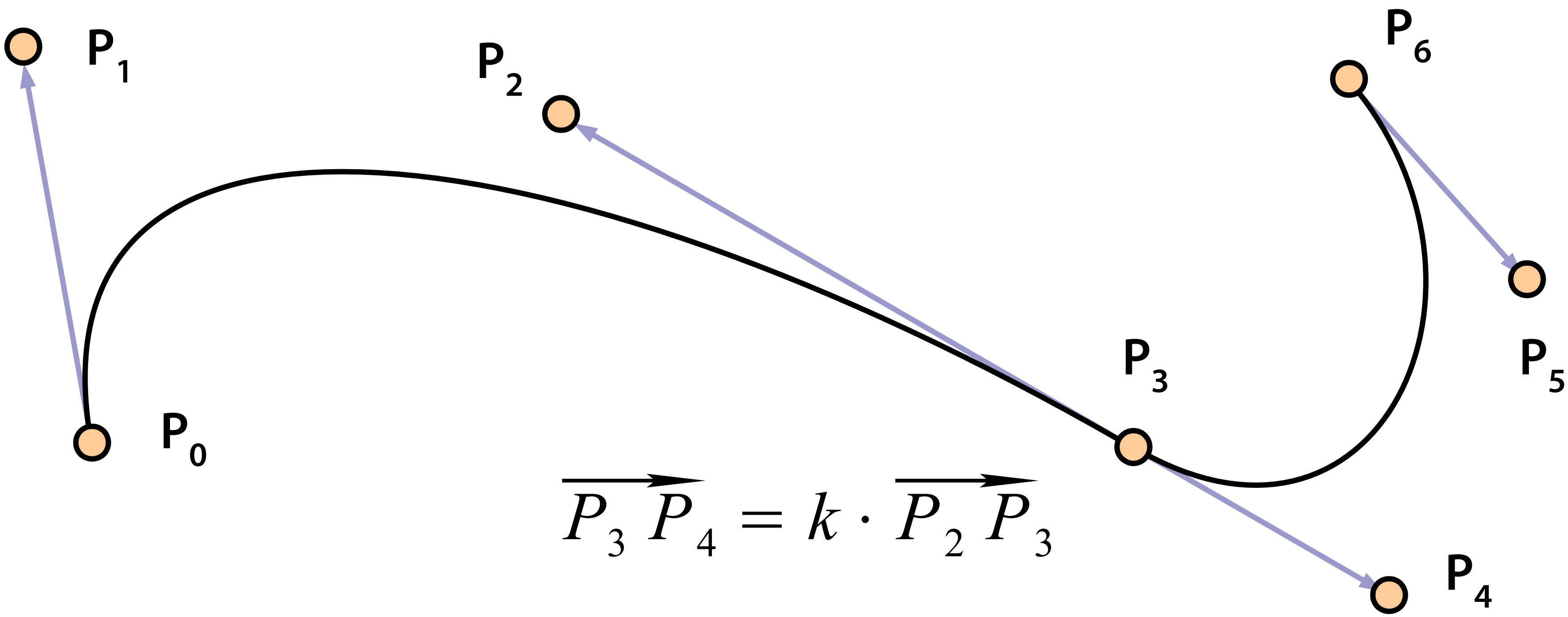
$$\sum_{i=0}^{n} B_i^n(t) = 1 \qquad \textit{for} \quad 0 \leqslant t \leqslant 1$$

$$\boldsymbol{MG} = \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{bmatrix}$$



Joining Bèzier curves I

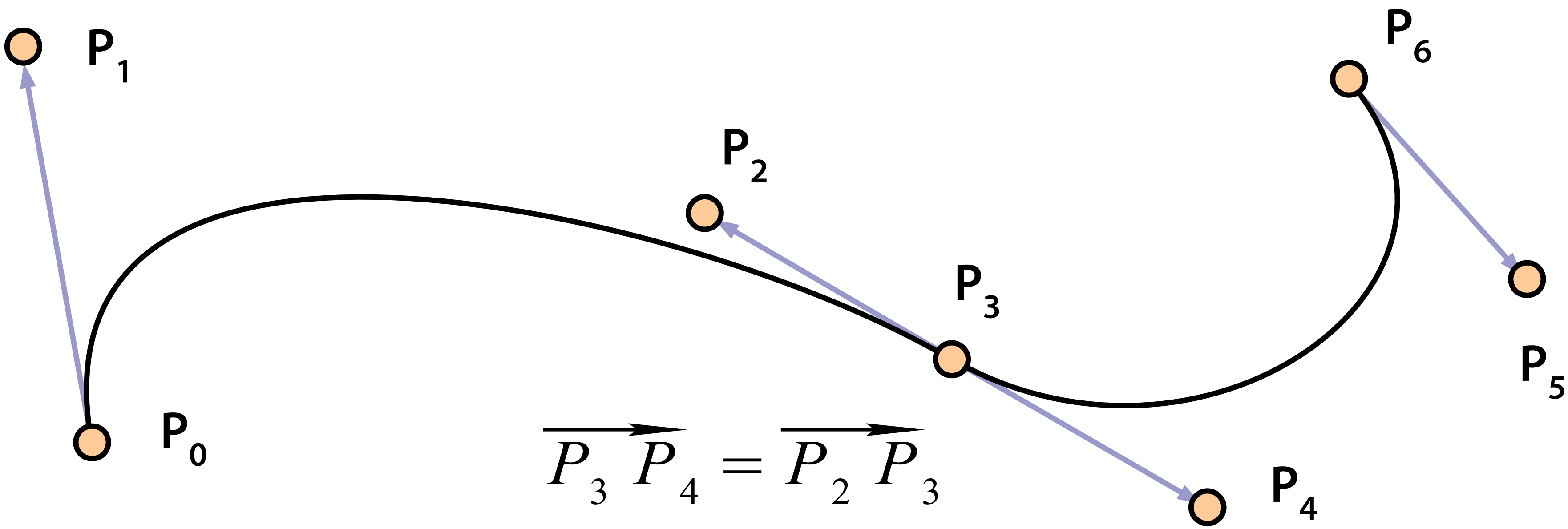
G^1 connection (co-linear tangents)



$$\overrightarrow{P_3 P_4} = k \cdot \overrightarrow{P_2 P_3}$$

Joining Bèzier curves II

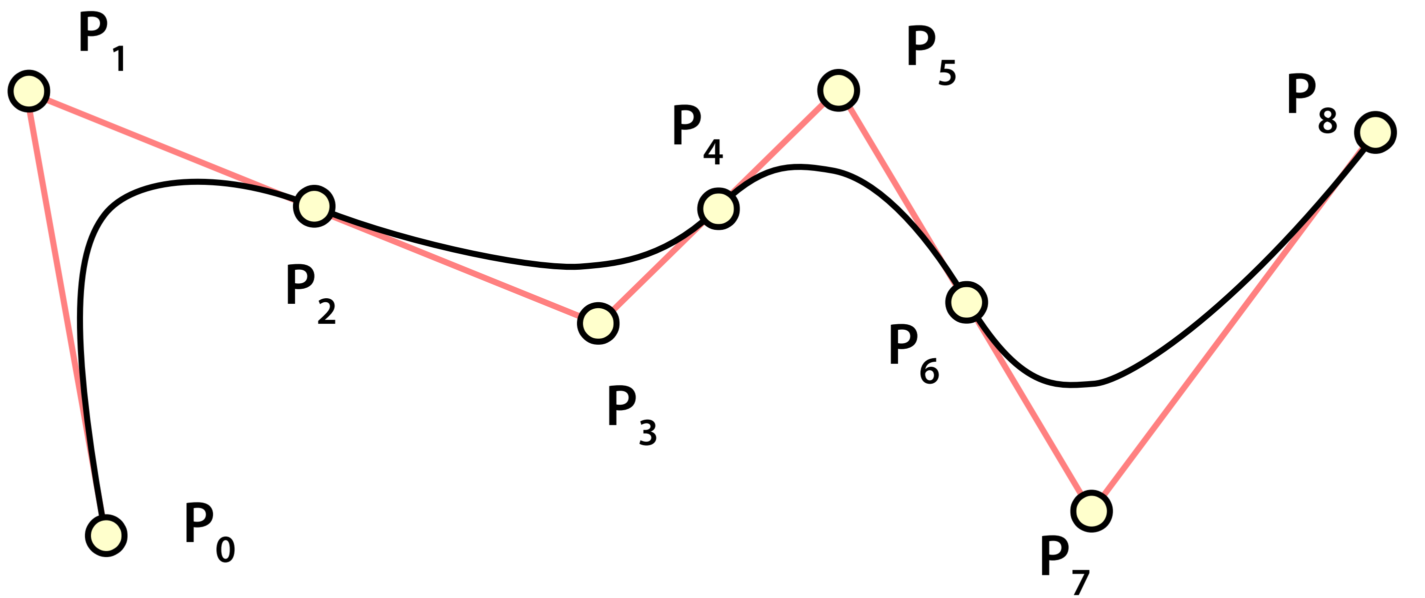
C^1 connection (equal tangent vectors)



$$\overrightarrow{P_3 P_4} = \overrightarrow{P_2 P_3}$$

Joining Bèzier curves III

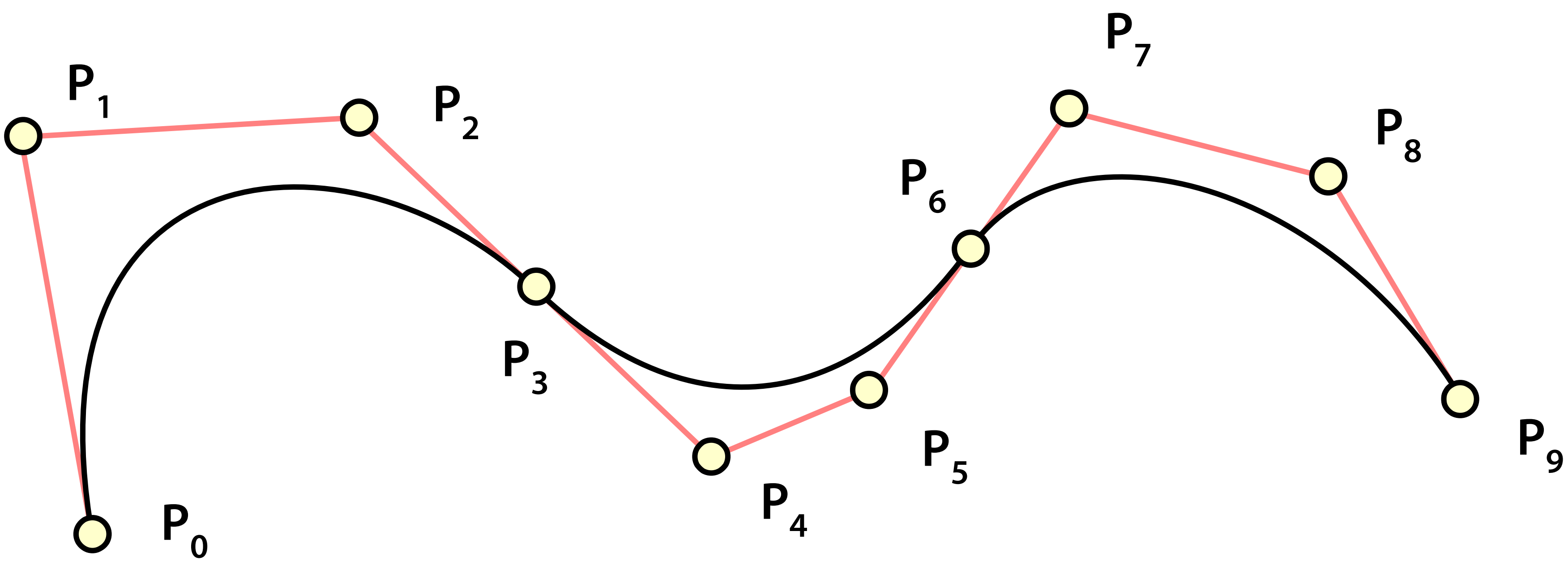
Quadratic spline from Bèzier segments



$$\overrightarrow{P_1 P_2} = \overrightarrow{P_2 P_3} \quad \overrightarrow{P_3 P_4} = \overrightarrow{P_4 P_5} \quad \ldots \quad \overrightarrow{P_{2k-1} P_{2k}} = \overrightarrow{P_{2k} P_{2k+1}}$$

Joining Bèzier curves IV

Cubic spline from Bèzier segments

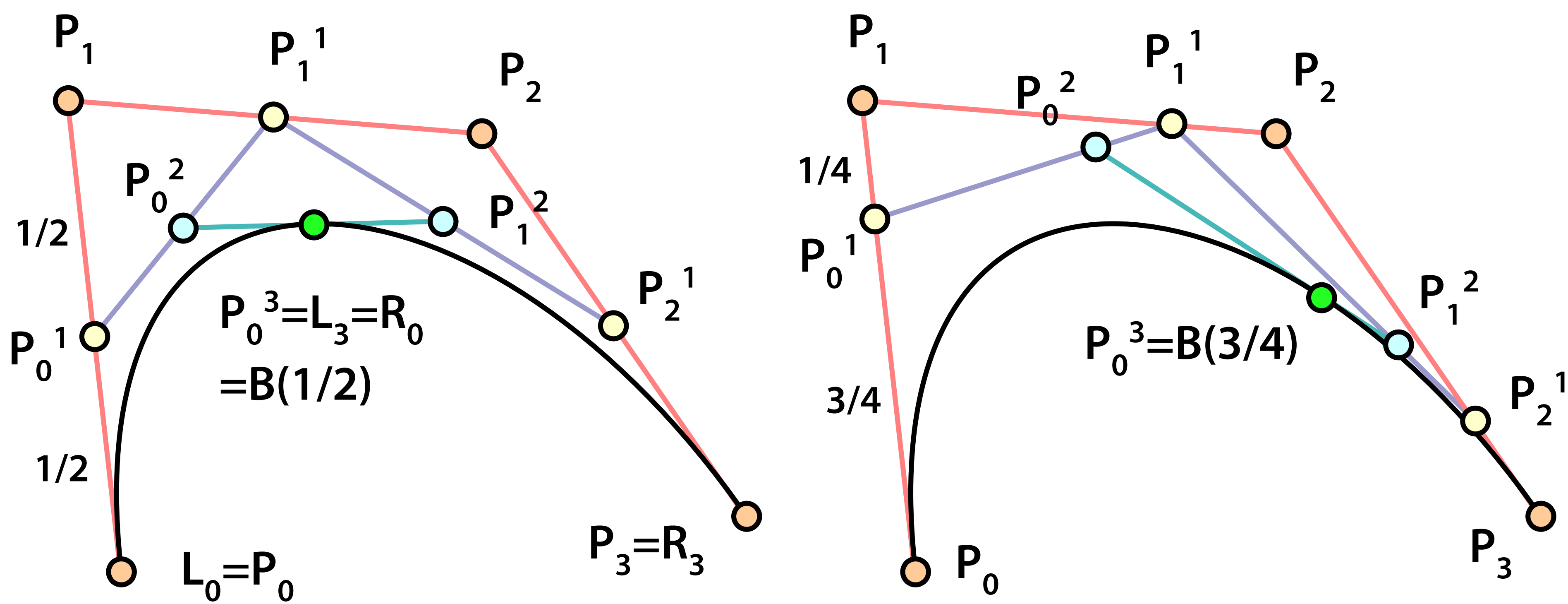


$$\overrightarrow{P_2 P_3} = \overrightarrow{P_3 P_4} \quad \overrightarrow{P_5 P_6} = \overrightarrow{P_6 P_7} \quad \ldots \quad \overrightarrow{P_{3k-1} P_{3k}} = \overrightarrow{P_{3k} P_{3k+1}}$$

De Casteljau (de Boor) algorithm

Geometric construction of Bèzier curve

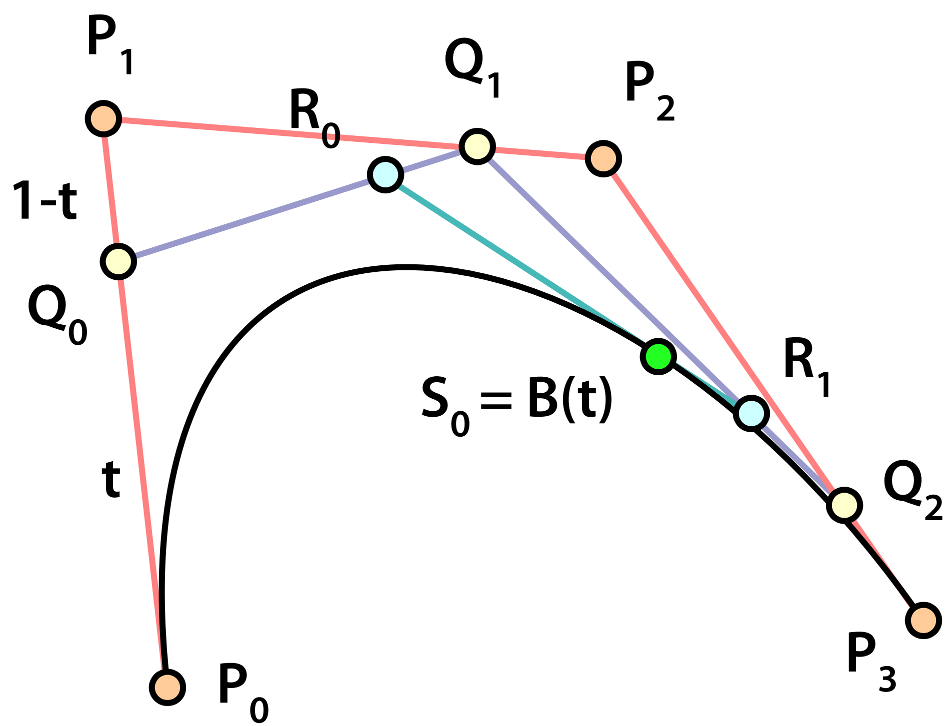
– used as "**subdivision**" scheme or for computation of a specific point…



Using [S]LERP operation

Linear interpolation LERP (SLERP for quaternions)

$$LERP(A, B, t) = A \cdot (1 - t) + B \cdot t$$



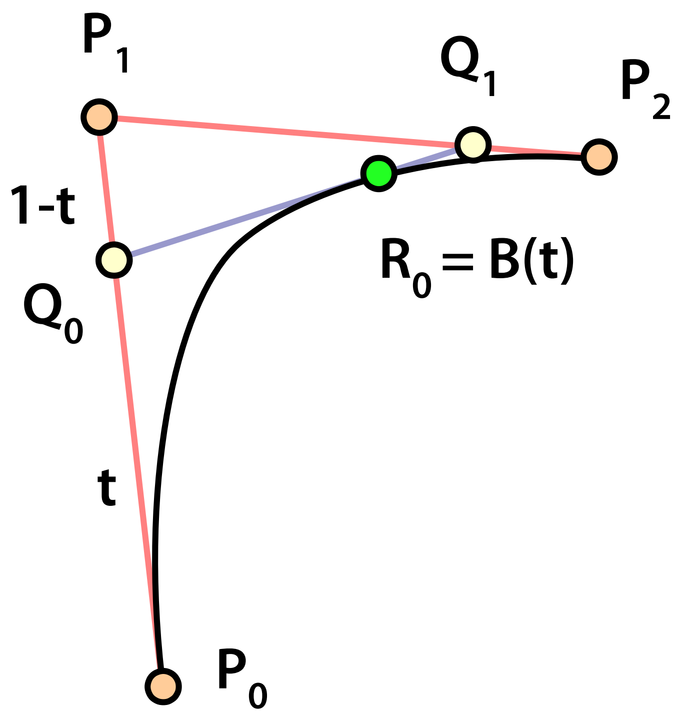
Cubic Bèzier

$$Q_i = LERP(P_i, P_{i+1}, t)$$

$$R_i = LERP(Q_i, Q_{i+1}, t)$$

$$S_i = LERP(R_i, R_{i+1}, t)$$

[S]LERP for quadratic interpolation



Quadratic Bèzier

$$Q_i = \mathrm{LERP}(P_i, P_{i+1}, t)$$

$$R_i = \mathrm{LERP}(Q_i, Q_{i+1}, t)$$

Cubic spline

Function assembled from **cubic polynomials**

- neighbor polynomials have C^2 joint
- elastic "spline-ruler" (see construction)

Interpolating cubic spline

- in knot points $\mathbf{x_0, x_1, \ldots x_n}$ function values $\mathbf{y_0, y_1, \ldots y_n}$ are prescribed

$$S(x) = S_k(x) = s_{k,0} + s_{k,1}(x - x_k) + s_{k,2}(x - x_k)^2 + s_{k,3}(x - x_k)^3$$
$$x \in [x_k, x_{k+1}], \quad k = 0, 1, \ldots, n-1$$

Condition **A**: $\quad S(x_k) = y_k \qquad k = 0, 1, \ldots, n$

Interpolating cubic spline

Condition **B** (C^0 continuity):

$$S_k(x_{k+1}) = S_{k+1}(x_{k+1}) \qquad k=0,1,\dots,n-2$$

Condition **C** (C^1 continuity):

$$S'_k(x_{k+1}) = S'_{k+1}(x_{k+1}) \qquad k=0,1,\dots,n-2$$

Condition **D** (C^2 continuity):

$$S''_k(x_{k+1}) = S''_{k+1}(x_{k+1}) \qquad k=0,1,\dots,n-2$$

Natural cubic spline has an additional condition **E**:

$$S''(x_0) = S''(x_n) = 0$$

Natural cubic spline

Interpolating spline

- **uniquely determined** by the conditions (solution of linear system of equations $\mathbf{s}_{k,l}$)
- **has no local property** (the whole curve changes after altering one control point)

Open spline

- conditions **A, B, C, D** are not sufficient, two more DoF
- additional condition **E** (second derivatives at endpoints)

Closed (cyclic) **spline**: $\mathbf{x}_0 = \mathbf{x}_n$

- **C** and **D** give us missing conditions for $\mathbf{x}_0$

B-spline (basis spline)

"**Free-form**" curve

- shape is defined by a sequence of **control points**
- parametric form using **basis/blending functions** (dependency of a curve point on control polygon)
- **local property** (only local change after altering one CP)

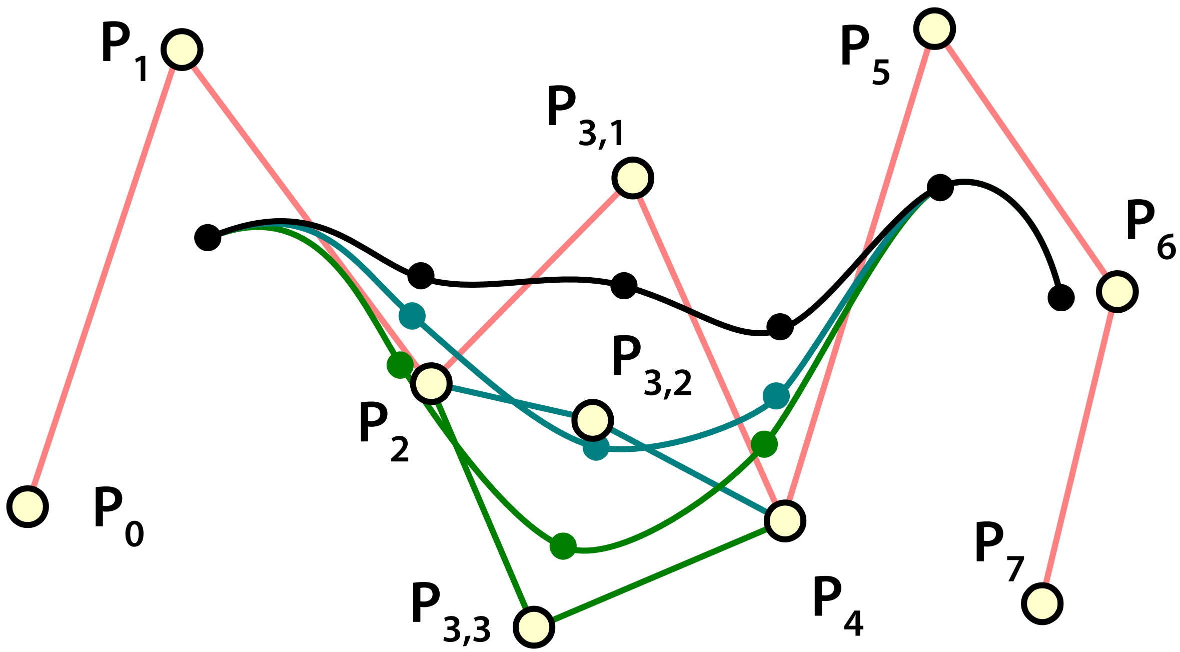
Uniform cubic B-spline (Coons curve)

- unified set of basis functions (cubic polynomials)

Nonuniform B-spline

- more complicated definition using knot vector $[\, \mathbf{t_i} \,]_i \quad \mathbf{0 \le t_i \le 1}$

Coons B-spline



- continuity $\mathbf{C^2}$
- **sharing** 3 CP between neighbours
- altering one CP induces change in closest **4 segments**

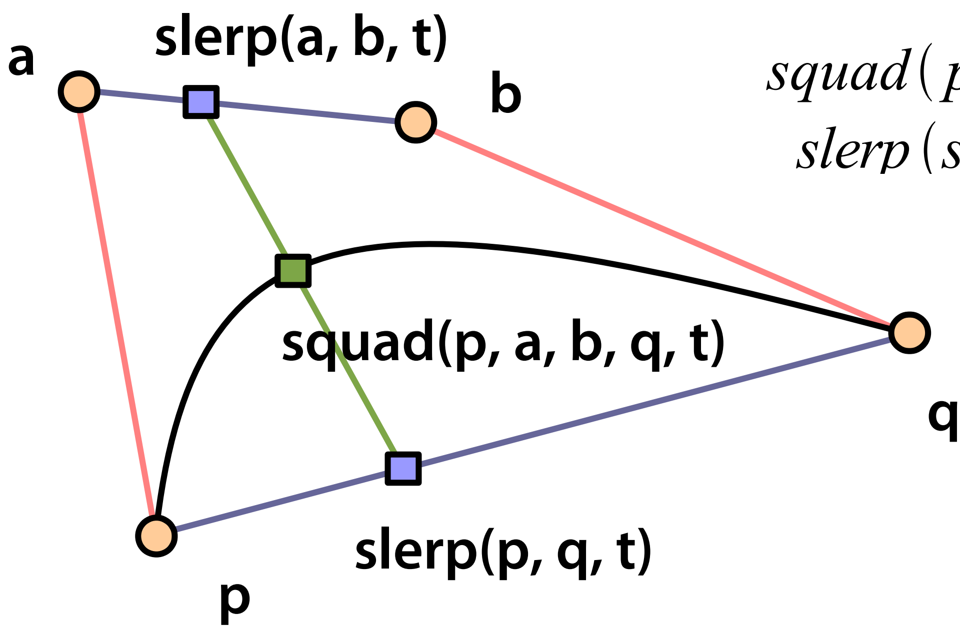
$$MG = \frac{1}{6}\begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} P_{i-1} \\ P_i \\ P_{i+1} \\ P_{i+2} \end{bmatrix}$$

Spline interpolation of quaternions

Subsequent interpolation by a sequence of orientations

$q_0, q_1, \ldots q_n$

– **slerp(q_i, q_{i+1}, t)** has not sufficient continuity (C^0 only)



$$squad(p, a, b, q, t) = slerp(slerp(p, q, t), slerp(a, b, t), 2t(1-t))$$

$$s_i(t) = squad(q_i, a_i, b_i, q_{i+1}, t)$$

$$a_i = b_{i-1} = q_i \exp\left[-\frac{\log(q_i^{-1} q_{i-1}) + \log(q_i^{-1} q_{i+1})}{4}\right]$$

Literature

Tomas Akenine-Möller, Eric Haines et al.: ***Real-time rendering, 4th edition***, A K Peters, 2018, ISBN: 9781138627000

J. Žára, B. Beneš, J. Sochor, P. Felkel: ***Moderní počítačová grafika***, 2. vydání, Computer Press, 2005, ISBN: 8025104540

http://www.geometrictools.com/ (Dave Eberly)