

# Mathematics for real-time graphics

© 2005-2018 Josef Pelikán  
CGG MFF UK Praha

[pepca@cgg.mff.cuni.cz](mailto:pepca@cgg.mff.cuni.cz)

<http://cgg.mff.cuni.cz/~pepca/>



# Content

- ◆ homogeneous coordinates, matrix transformations
  - ◆ coordinate-system conversions
- ◆ coordinate systems, projections, frustum
- ◆ perspective correct interpolation
- ◆ orientations
  - ◆ Euler angles, quaternions
  - ◆ orientation interpolation
- ◆ smooth interpolations and approximations
  - ◆ spline functions, natural spline, B-spline, Catmull-Rom, ...

# Geometric transformations in 3D



- ◆ Cartesian 3D coordinate vector  $[x, y, z]$
- ◆ multiplying by a  $3 \times 3$  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  $3 \times 3$  have serious drawback – **cannot do translations!**



# Homogeneous coordinates

- ◆ **homogeneous coordinate vector**  $[x, y, z, w]$
- ◆ **transformation: multiplying by a  $4 \times 4$  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 **translate** and to do **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  $[x, y, z] \dots [x, y, z, 1]$



# Elementary transformations

Affine transformation:

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

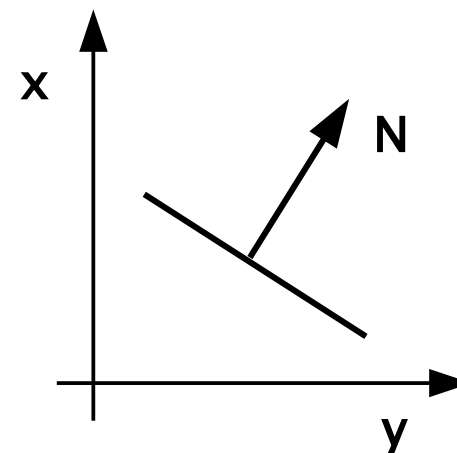
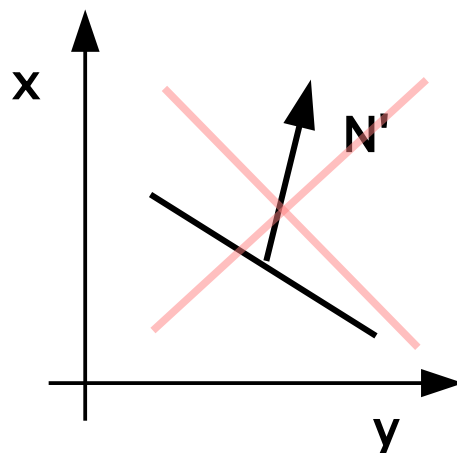
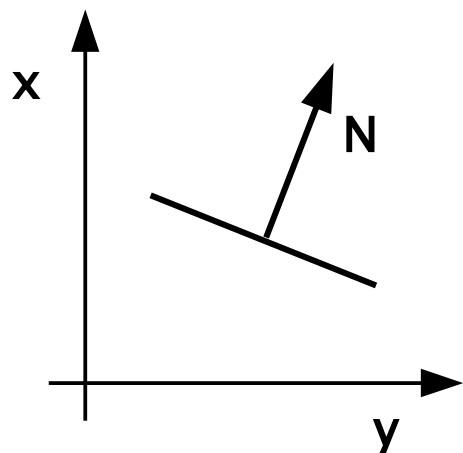
- ◆ upper left submatrix [  $\mathbf{a}_{11}$  to  $\mathbf{a}_{33}$  ] defines scaling, orientation and shear
- ◆ vector [  $\mathbf{t}_1, \mathbf{t}_2, \mathbf{t}_3$  ] defines translation
  - ◆ translation is performed after former transformations



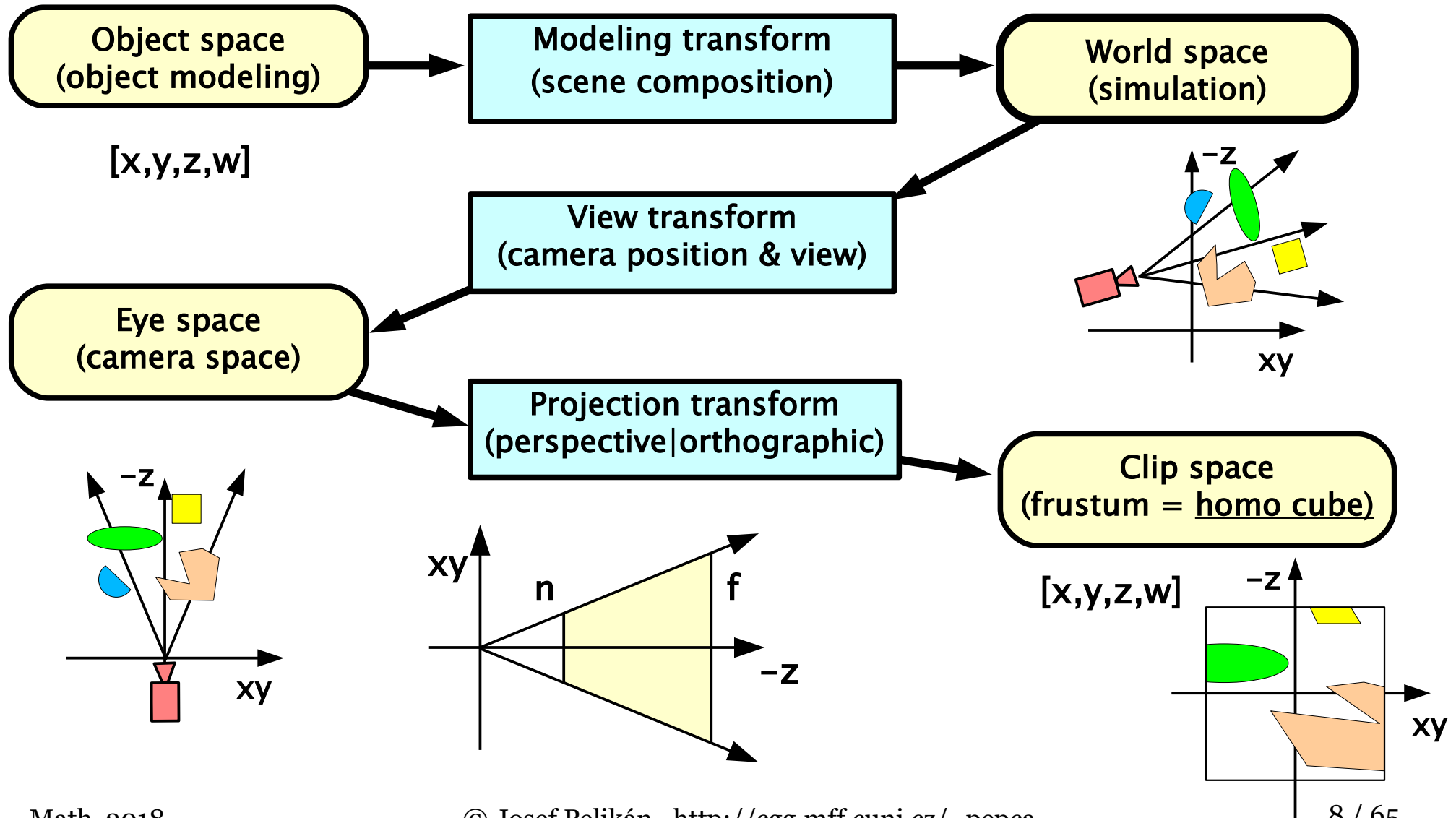
# Normal vector transformation

- normal vectors must not be transformed by regular matrices (like point positions are)
  - exception:  $M$  is rotational (orthonormal)
- normal-vector** transformation matrix  $\mathbf{N}$ :

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

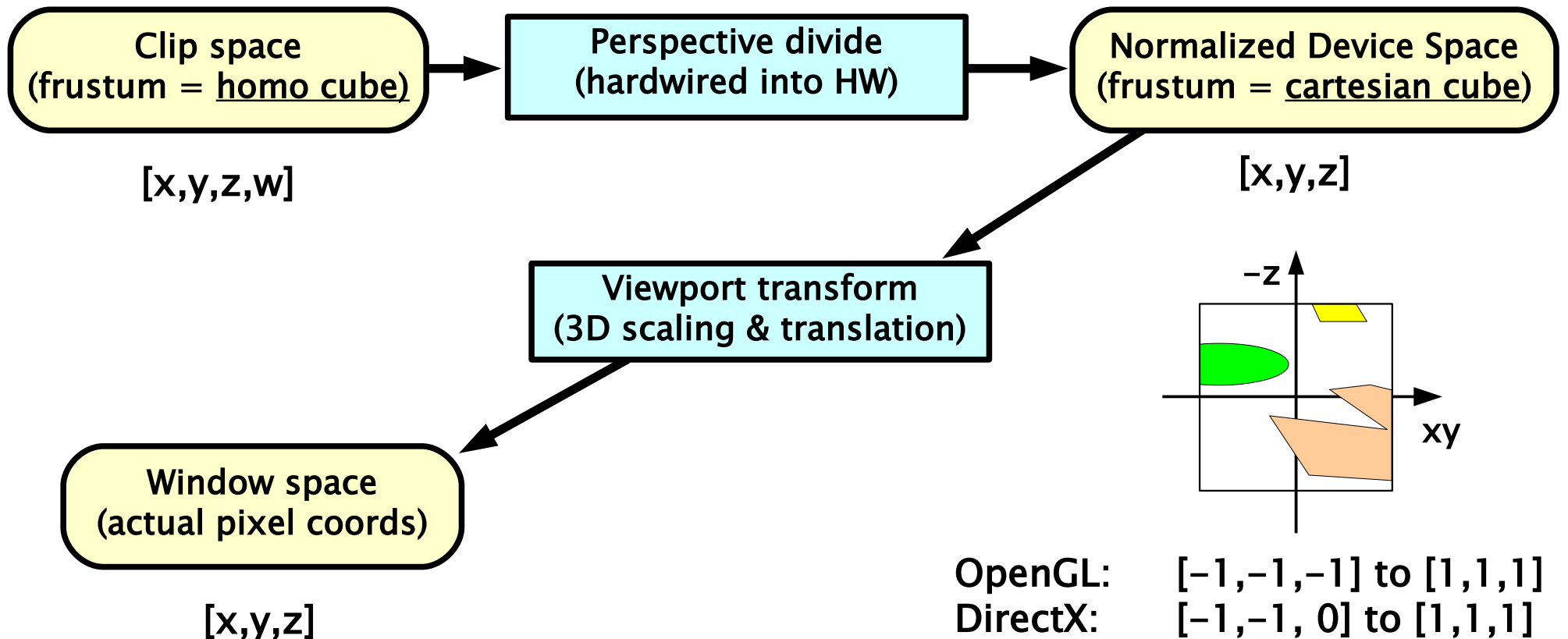


# Coordinate systems in OpenGL





# Coordinate systems in OpenGL



[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 (3DSMax, 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

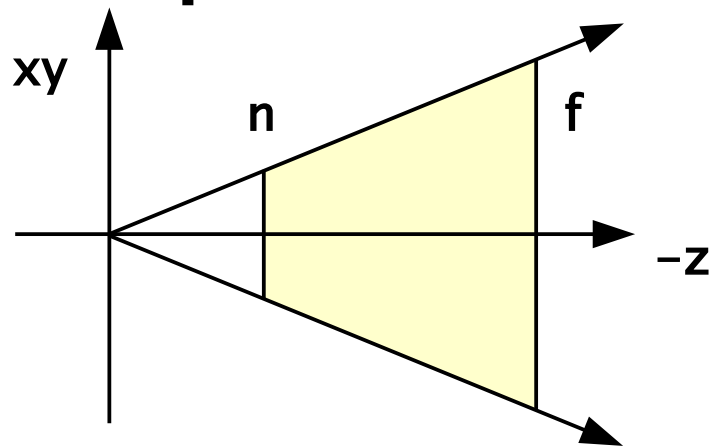
- ◆ **transformation “model → camera”**
  - ◆ altogether – “modelview” matrix
  - ◆ world coordinates are not directly used in render. pipel.
- ◆ **projection transformation**
  - ◆ defines visible volume = **frustum** [ **l**, **r**, **b**, **t**, **n**, **f** ]
  - ◆ front & back clip distances: **n**, **f**
  - ◆ result: homogeneous coordinate (before clipping)
- ◆ **“clip space”**
  - ◆ **mandatory output coordinate** of vertex shader !



# Projection transformation

- ◆ far point  $f$  can be in infinity

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



# Coordinate systems & transformations

## ◆ 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  $[1,1,1]$

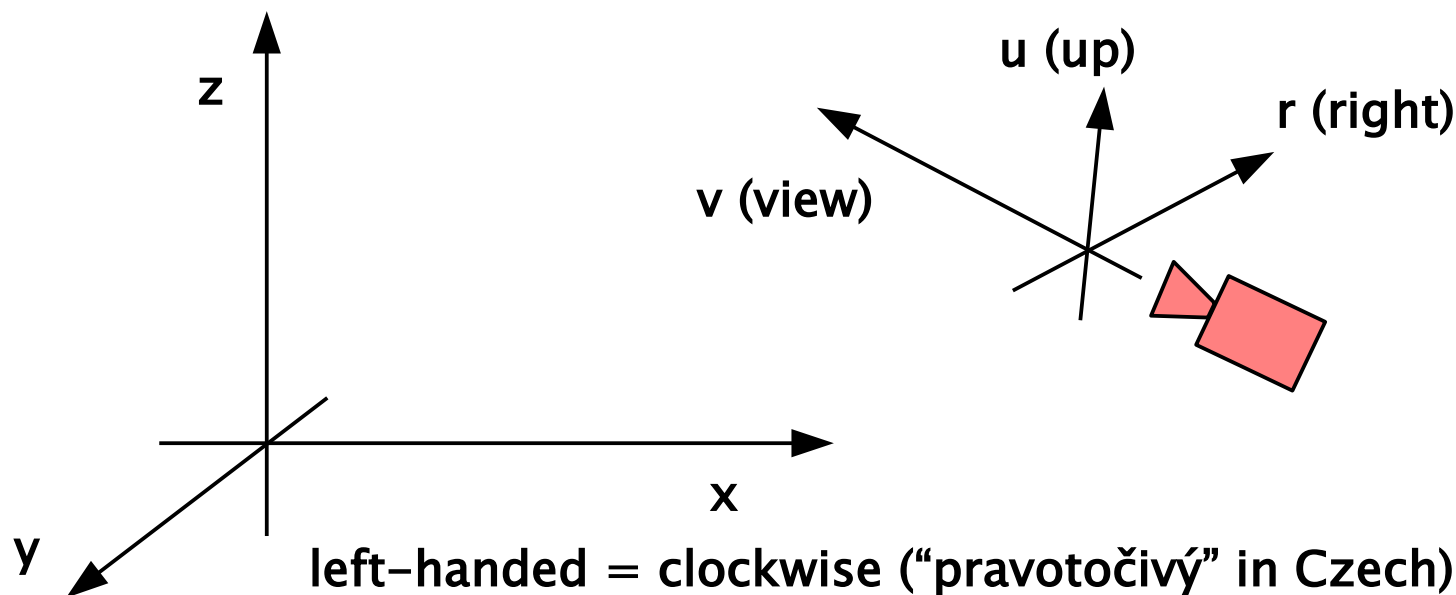
## ◆ window coordinates (“window space”)

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

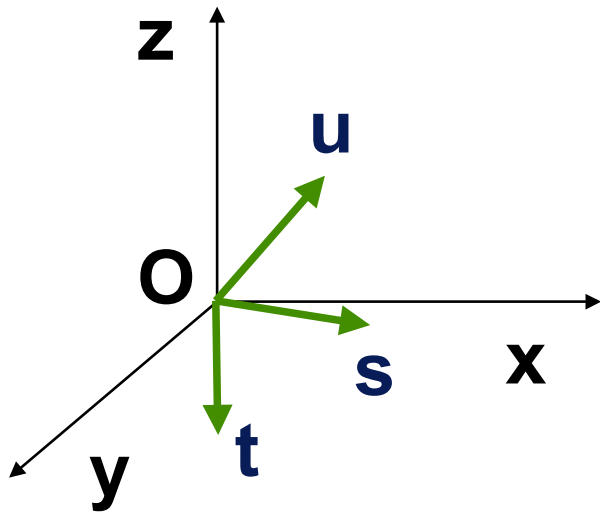


# Rigid body transformation

- ◆ preserves **shapes**, alters **orientation & position**
  - ◆ translation and rotation
  - ◆ **conversion between coordinate systems** (e.g. between world-space and camera-space)



# Conversion between two orientations



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

$$\begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \cdot M_{stu \rightarrow xyz} = s$$

$$\begin{bmatrix} 0 & 1 & 0 \end{bmatrix} \cdot M_{stu \rightarrow xyz} = t$$

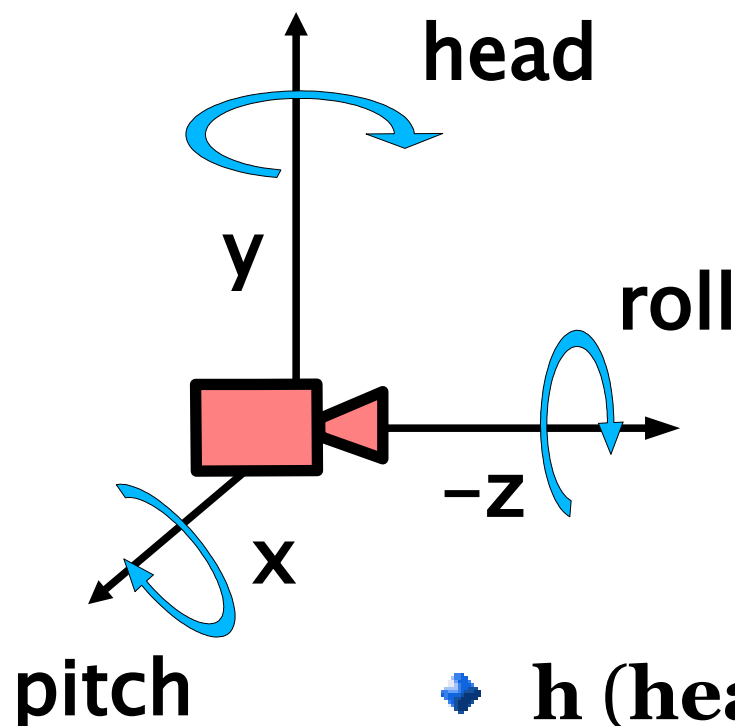
$$\begin{bmatrix} 0 & 0 & 1 \end{bmatrix} \cdot M_{stu \rightarrow xyz} = u$$

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

$$M_{xyz \rightarrow stu} = M_{stu \rightarrow xyz}^T$$



# 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

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

- ◆  $s(x) \dots \sin(x)$ ,  $c(x) \dots \cos(x)$

- ◆ backward matrix  $\rightarrow$  angles computation  $h, p, r$ :

- ◆  $p \dots e_{23}$

- ◆  $r \dots e_{21}/e_{22}$

- ◆  $h \dots e_{13}/e_{33}$

# Rotations: different conventions



## ◆ **main convention:**

- ◆ 1. rotation around  $\mathbf{z}$  by  $\varphi$
- ◆ 2. rotation around  $\mathbf{x}'$  by  $\theta$
- ◆ 3. rotation around  $\mathbf{z}''$  by  $\psi$

## ◆ **x-convention:**

- ◆ 1. rotation around  $\mathbf{z}$
- ◆ 2. rotation around original  $\mathbf{x}$
- ◆ 3. rotation around original  $\mathbf{z}$

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

$$◆ \mathbf{q} \mathbf{r} = (\mathbf{v}_q \times \mathbf{v}_r + w_r \mathbf{v}_q + w_q \mathbf{v}_r, w_q w_r - \mathbf{v}_q \cdot \mathbf{v}_r)$$

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



# 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

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



# Unit quaternions

- ◆ **unit quaternion** can be expressed by goniometry 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  $\mathbf{q}$  and  $-\mathbf{q}$  represent the same rotation!
  - ◆ identity (zero rotation):  $(\mathbf{o}, 1)$  ( $\phi + \pi$ )
- ◆ power, exponential, logarithm:
  - ◆  $\mathbf{q} = \mathbf{u}_q \sin \phi + \cos \phi = \exp(\phi \mathbf{u}_q), \quad \log \mathbf{q} = \phi \mathbf{u}_q$
  - ◆  $\mathbf{q}^t = (\mathbf{u}_q \sin \phi + \cos \phi)^t = \exp(t\phi \mathbf{u}_q) = \mathbf{u}_q \sin t\phi + \cos t\phi$

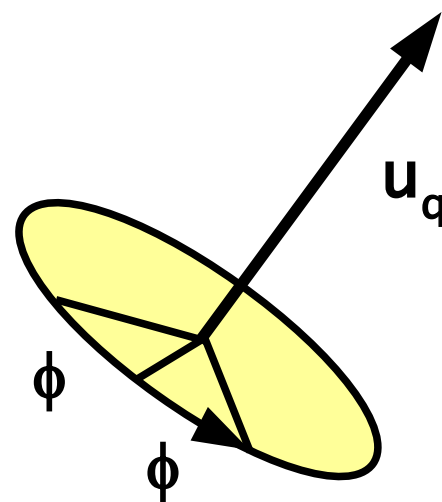


# Rotation using quaternion

## ◆ unit quaternion

◆  $\mathbf{q} = (\mathbf{u}_q \sin \phi, \cos \phi)$

◆  $\mathbf{u}_q$  ... axis of rotation,  $\phi$  ... angle



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

◆ **rotation** of vector (point)  $\mathbf{p}$  around  $\mathbf{u}_q$  by angle  $2\phi$

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

# Quaternion $\leftrightarrow$ matrix conversions

- ◆ quaternion  $\mathbf{q}$  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 (\$):

$$m_{23} - m_{32} = 4wx$$

$$m_{31} - m_{13} = 4wy$$

$$m_{12} - m_{21} = 4wz$$

$$\text{tr } M + 1 = 4w^2$$





# Matrix $\rightarrow$ quaternion II

1. “matrix trace+1” has large enough absolute value:

$$w = \frac{1}{2} \sqrt{\text{tr } M + 1} \quad x = \frac{m_{23} - m_{32}}{4w}$$
$$y = \frac{m_{31} - m_{13}}{4w} \quad z = \frac{m_{12} - m_{21}}{4w}$$

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  $\mathbf{q}$  and  $\mathbf{r}$  ( $\mathbf{q} \cdot \mathbf{r} \geq 0$ , else take  $-\mathbf{q}$ )
- ◆ real parameter  $0 \leq t \leq 1$

**Interpolated quaternion**  $\text{slerp}(\mathbf{q}, \mathbf{r}, t) = \mathbf{q} (\mathbf{q}^* \mathbf{r})^t$

- ◆ different formulation:

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

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

**... the shortest spherical arc** between  $\mathbf{q}$  and  $\mathbf{r}$

(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**:  $e = \mathbf{s} \cdot \mathbf{t} = \cos 2\phi$

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

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

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



# Slerp of rotational matrices

- ▶ two rotational matrices  $\mathbf{Q}$  and  $\mathbf{R}$
- ▶ real parameter  $0 \leq t \leq 1$

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

- ▶ **technical problem:** ? power operation on matrices
- ▶ need to compute axis and angle  $\mathbf{Q}^T \mathbf{R}$ 
  - ▶ not very efficient

(for details see “RotationIssues.pdf”)

# 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

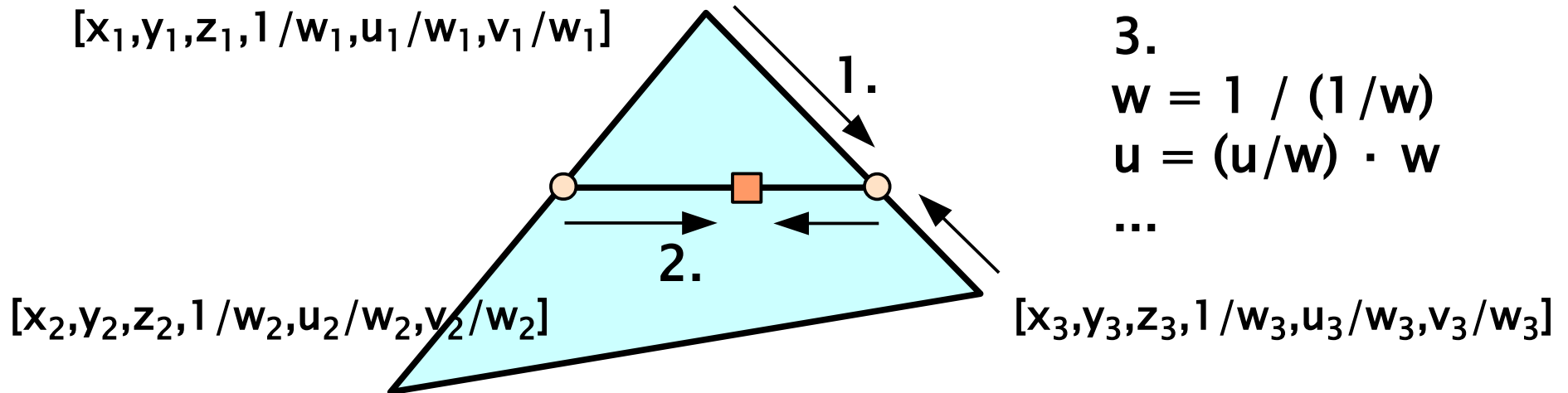
(for details see “RotationIssues.pdf”)

# Interpolation in the screen space



- ◆ concerns **clip-space** and next spaces
  - ◆ clip space:  $[x, y, z, w]$
  - ◆ NDS:  $[x/w, y/w, z/w, w]$  (“w” could be preserved)
  - ◆ window space (fragments):  $[x_i, y_i, z_i, w]$
- ◆ **projective perspective transformation** maps depth **z** to NDS **non-linearly** !
  - **nonuniform accuracy** of z-buffer (distant parts could be less accurate: minimization of **f/n** ratio !)
  - + **interpolation of depth z** can be done in the screen space **linearly**
- ◆ **W-buffer** instead of z-bufferu (not used any more)

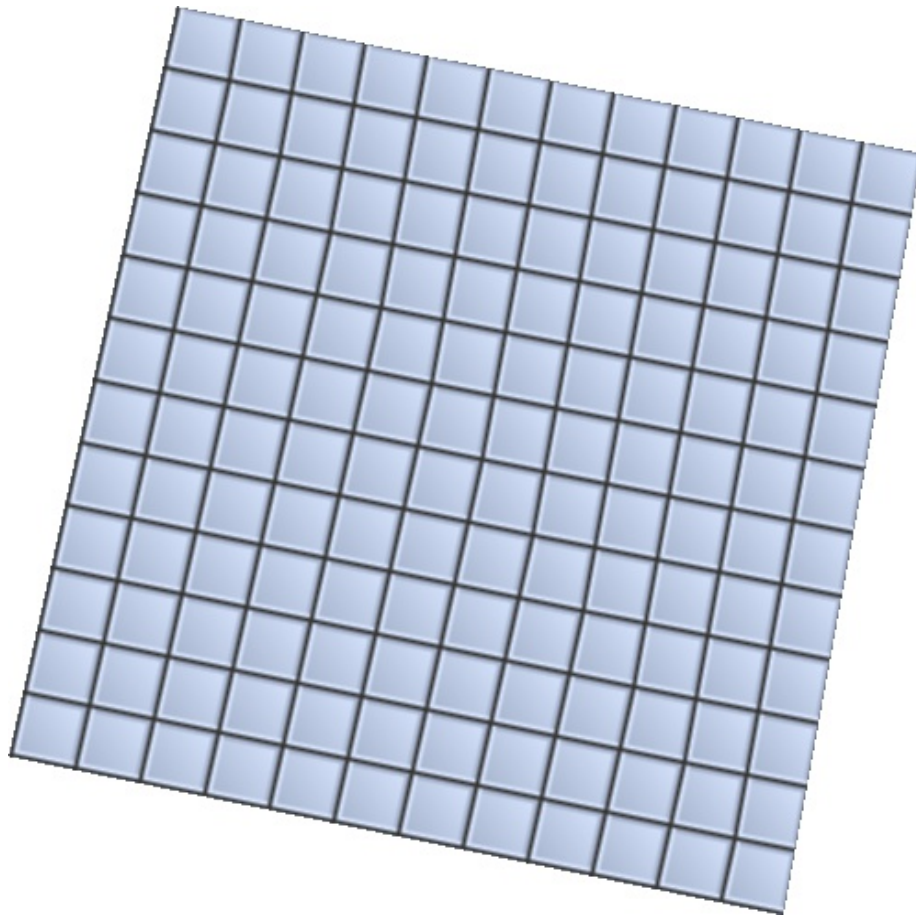
# Perspective-correct interpolation



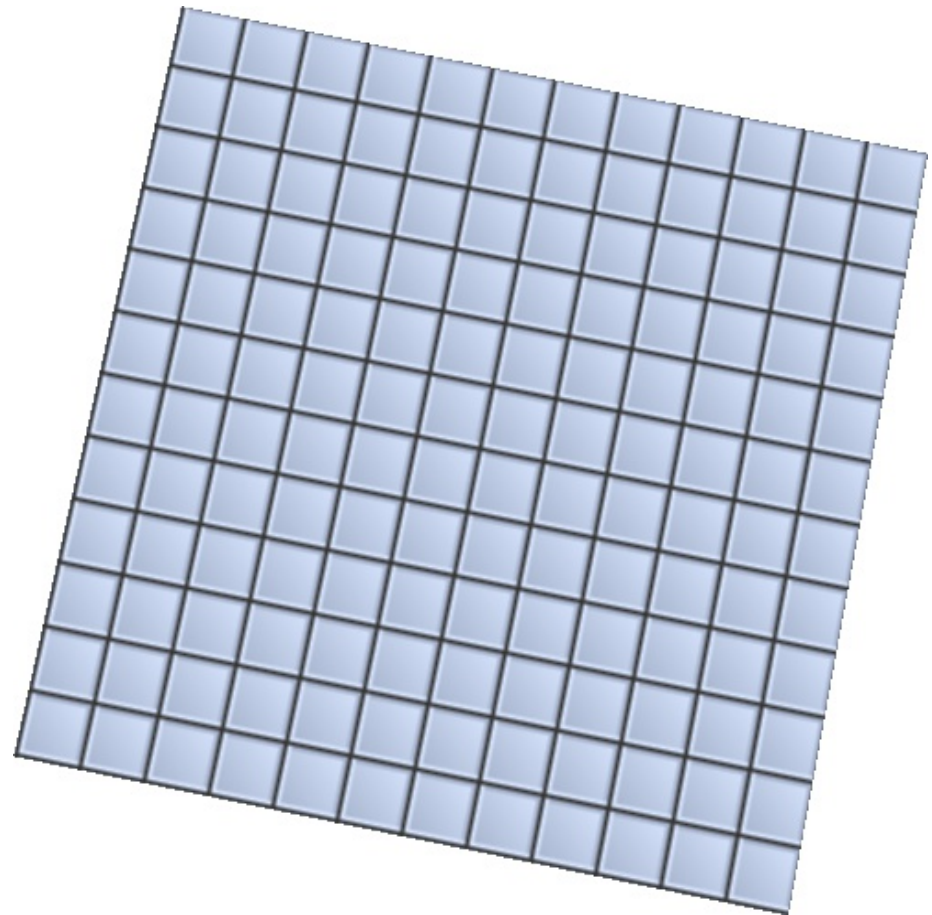
## Linear interpolation + division:

- ◆ depth “z” is linear by itself
- ◆ for **texture coordinates** or “w” perspective-correct interpolation must be used:
- ◆ quantities “ $1/w$ ”, “ $u/w$ ”, “ $v/w$ ”, .. are linearly interpolated, [ **u, v** ] is then computed by division (**hyperbolic interpolation**)

# Interpolation example I



**Affine**

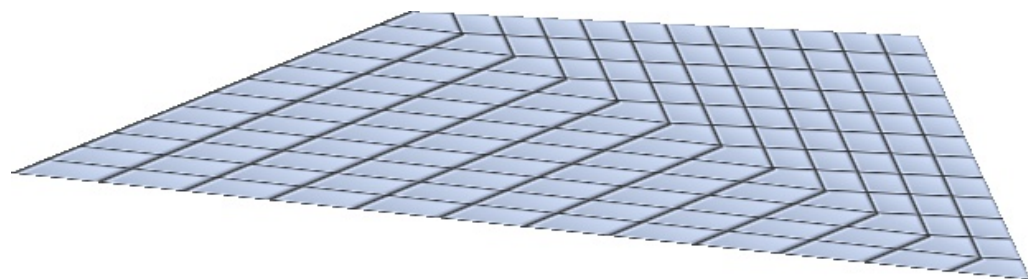


**Correct**

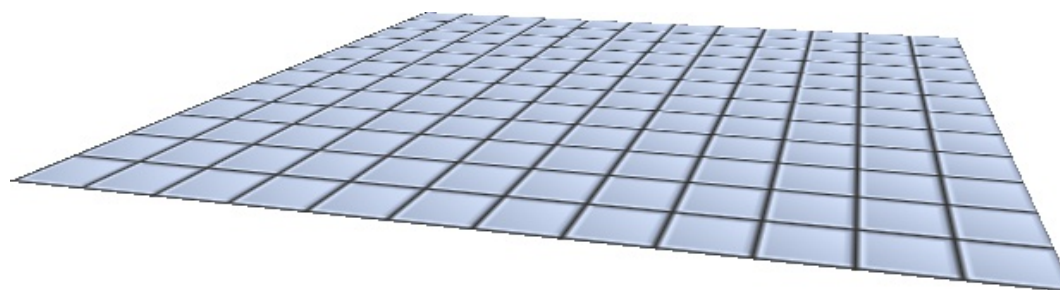




# Interpolation example II



**Affine**



**Correct**

# Approximation and interpolation



- ◆ **approximation** (e.g. B-spline)

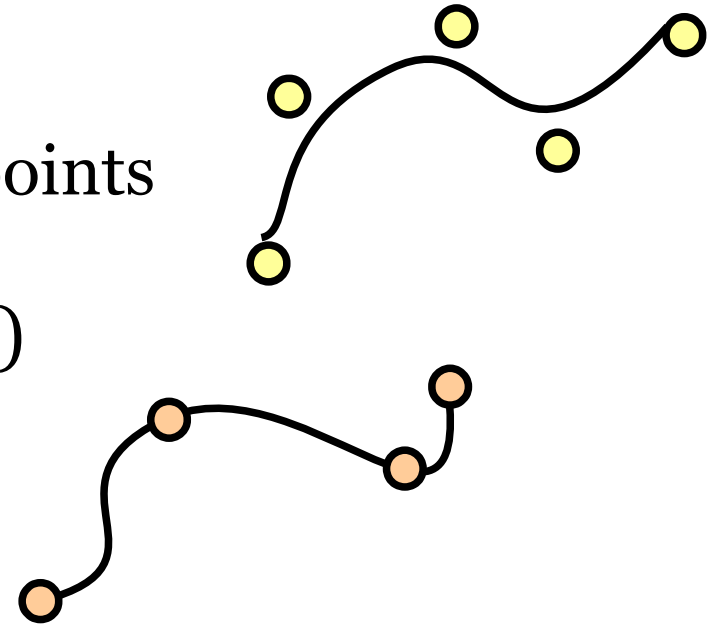
- ◆ needs not to pass through control points

- ◆ **interpolation** (e.g. Catmull-Rom)

- ◆ curve passes through ctrl. points

- ◆ **curve continuity**

- ◆  $\mathbf{G}^n$  – geometric continuity of  $n^{\text{th}}$  order ( $\mathbf{G}^0$  – simple continuity,  $\mathbf{G}^1$  – tangent,  $\mathbf{G}^2$  – curvature, ...)
- ◆  $\mathbf{C}^n$  – analytical continuity of  $n^{\text{th}}$  order,  $n^{\text{th}}$  derivative cont. ( $\mathbf{C}^1$  – speed,  $\mathbf{C}^2$  – acceleration), superior to geometric



# History



## ◆ curves in modeling industry

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

## ◆ spline function theory

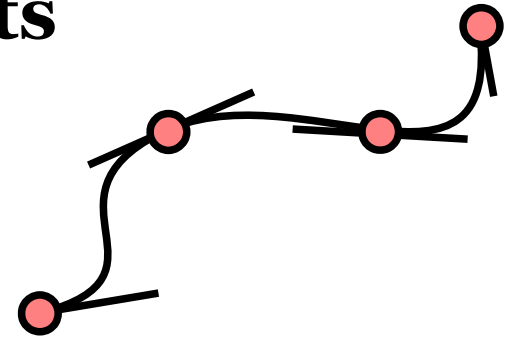
- ◆ B-spline: Isaac Jacob **Schoenberg**, (ballistics, Aberdeen, MD, 1946)
- ◆ theory: Carl **de Boor** (also worked in General Motors)
- ◆ Gordon, Riesenfeld united Bèzier and B-spline curves (1972)



# “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 ctrl. pts. (Hermit)
- ◆ interpolation  $\Rightarrow$  closer control

- ◆ **locality**

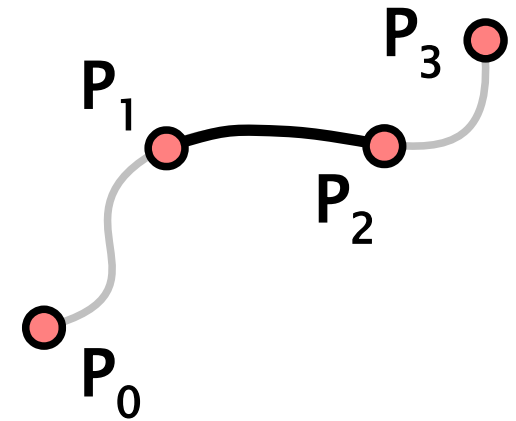
- ◆ change of single control point (one tangent vector) induces change in restricted neighborhood only



# “Free-form“ curves II

- ◆ **parametric expression** ( $0 \leq t \leq 1$ )

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



- ◆ **convex hull** property

- ◆ curve lies in convex hull of its control polygon

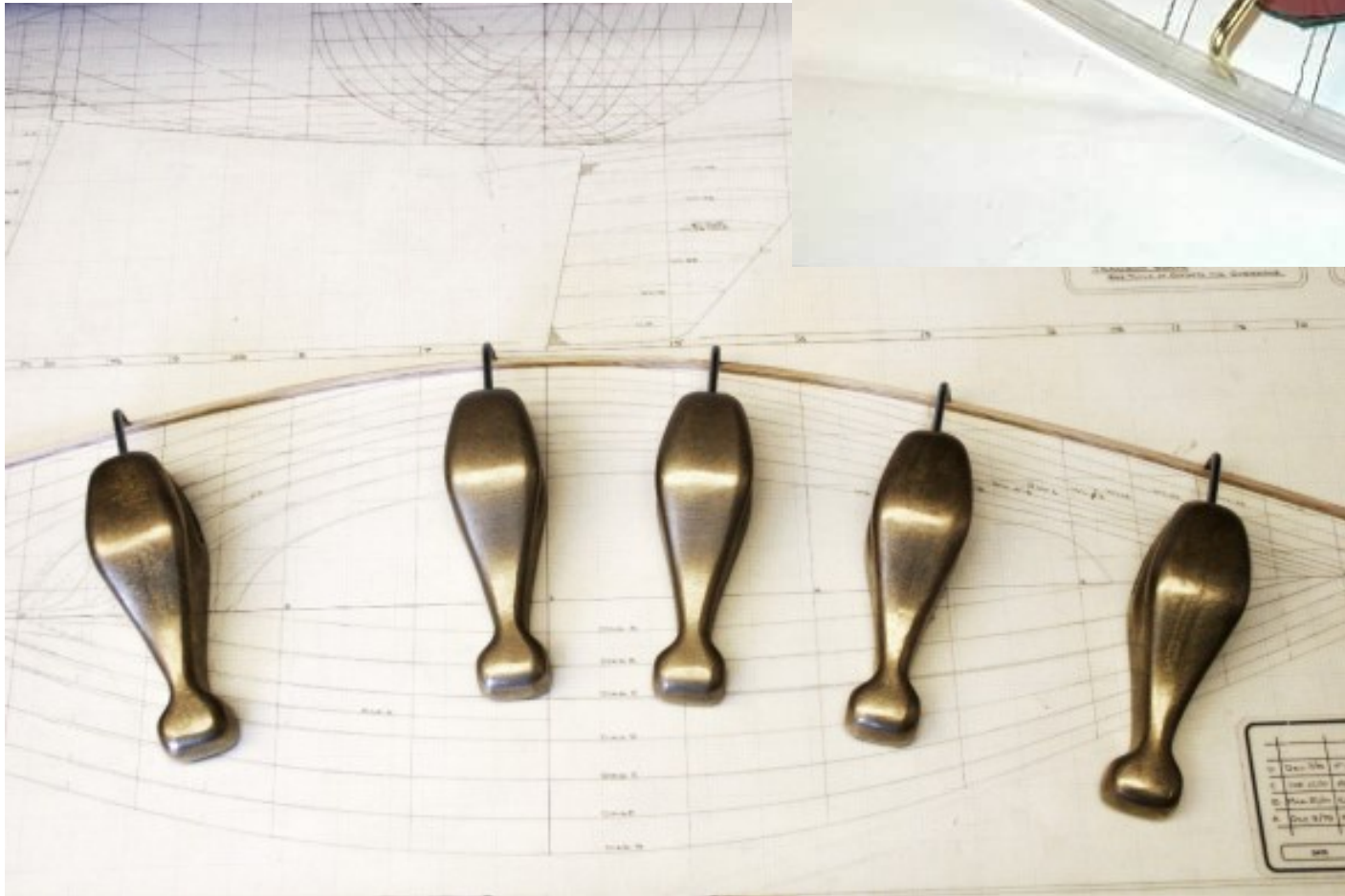
- ◆ **Cauchy condition** for blending functions

- ◆ 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:

$C^{n-1}$  – continuity of  $n-1^{th}$  derivative (polyn. of degree  $n$ )

◆ global parametrization  $\mathbf{u}$ ,  $\mathbf{u}_0 \leq \mathbf{u} \leq \mathbf{u}_N$  [ $u_0, u_1, \dots, u_N$ ]

◆ individual parts are often uniformly parametrized – uniform spline:  $\mathbf{t}_i = (\mathbf{u} - \mathbf{u}_i) / (\mathbf{u}_{i+1} - \mathbf{u}_i)$ ,  $0 \leq \mathbf{t}_i \leq 1$



# Polynomial curve

- matrix notation:

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

- basis matrix  $\mathbf{M}$  and vector of geometric conditions  $\mathbf{G}$ :

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





# Matrix notation of a curve

◆  $\mathbf{P}(t) = \mathbf{T} \mathbf{C} = \mathbf{T} \mathbf{M} \mathbf{G}$

- ◆ separation of a parameter vector ( $\mathbf{T}$ ) from polynomial basis ( $\mathbf{M}$ ) and geometric control conditions/points ( $\mathbf{G}$ )
- ◆ differentiation (tangent, curvature) restricted to  $\mathbf{T}$
- ◆ control polynomial  $\mathbf{T}\mathbf{M}$  times “geometry”  $\mathbf{G}$

◆ **cubic:**  $n = 3, k = 4$

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



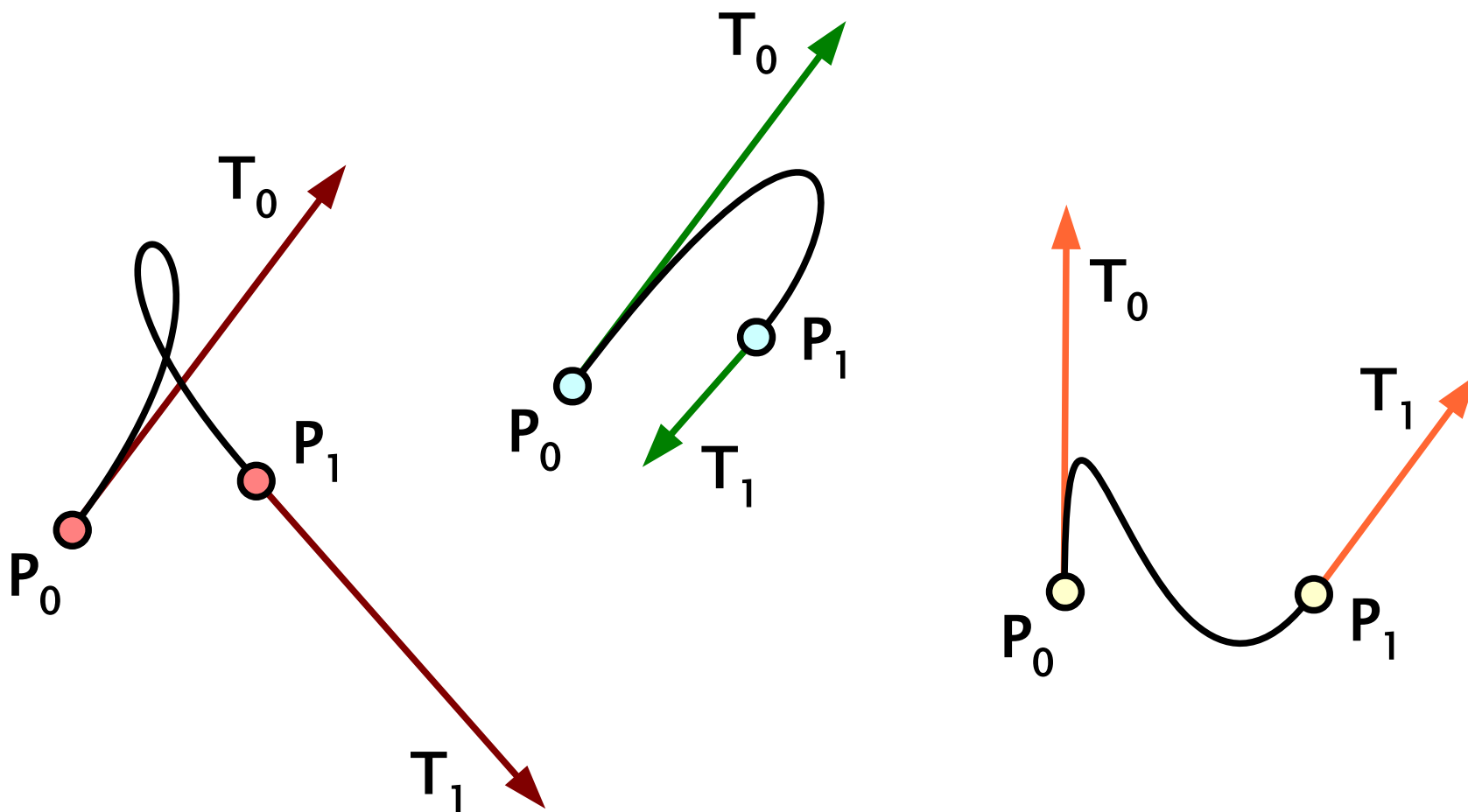
# Hermite cubic curve

- ◆ **Ferguson curve (cubic)**
- ◆ geometry: **endpoints** and **tangent vectors**
  - ◆ beginning ( $\mathbf{P}_0$ ) and end ( $\mathbf{P}_1$ ) of a curve
  - ◆ tangents in beginning ( $\mathbf{T}_0$ ) and ending ( $\mathbf{T}_1$ ) 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 ( $\mathbf{T}_0$  and  $\mathbf{T}_1$  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 = o**)

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

## ◆ Catmull-Rom spline

- ◆ **a = t = 1/2**

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

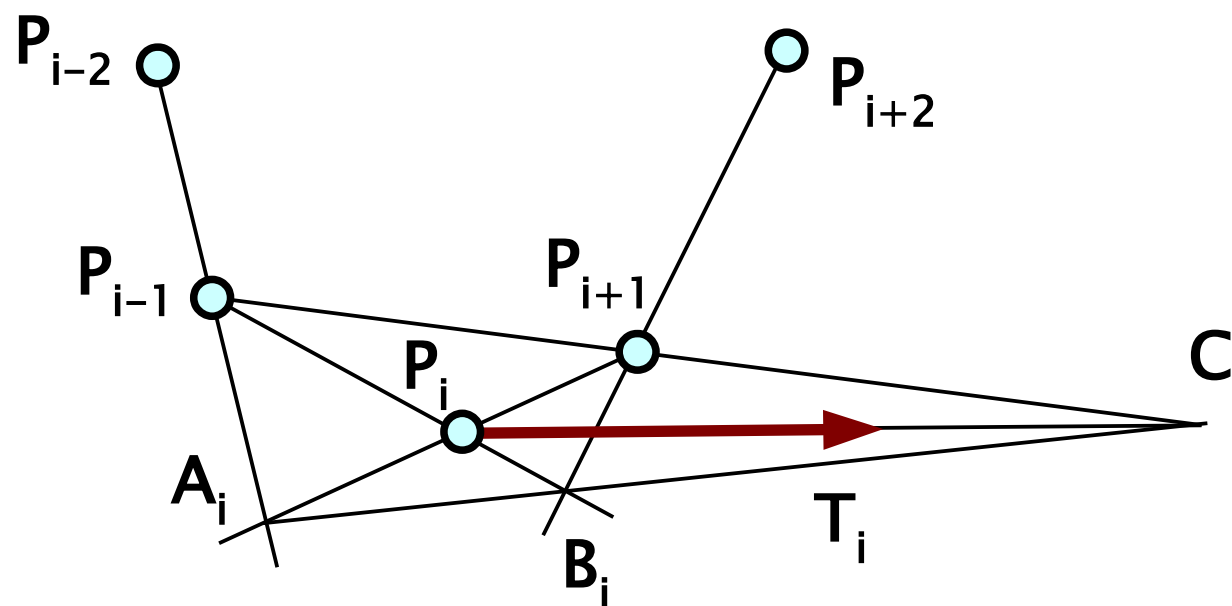
$$MG = \frac{1}{2} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 2 & -5 & 4 & -1 \\ -1 & 0 & 1 & 0 \\ 0 & 2 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} P_{i-1} \\ P_i \\ P_{i+1} \\ P_{i+2} \end{bmatrix}$$



# Akima interpolation

Alternative definition of tangent vectors for Hermite cubic:

♦ non- $C^2$  !

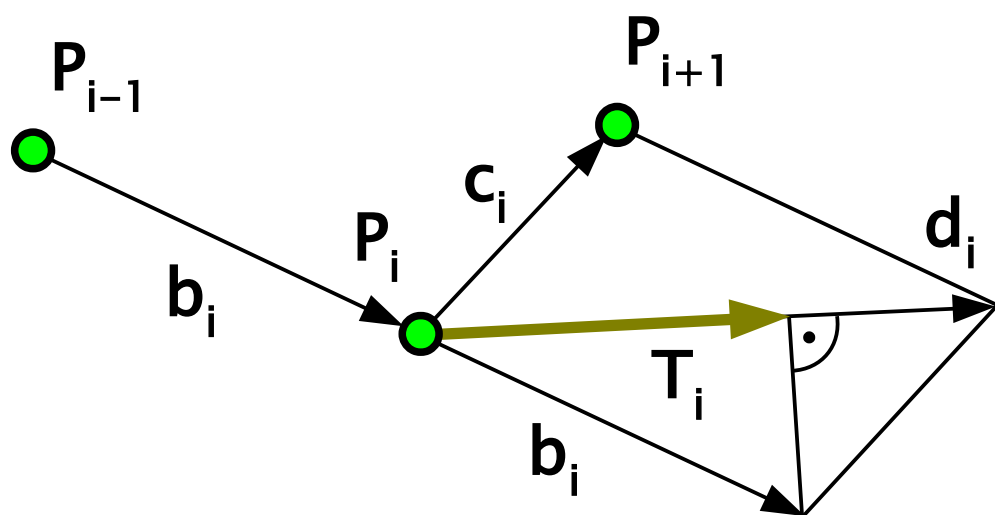


$$|T_i| = |P_{i+1} - P_{i-1}|$$



# 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 **G<sup>1</sup>** or **C<sup>1</sup>** connection
  - ◆ spline-join is also possible, but much more complicated

## Bernstein polynomials:

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

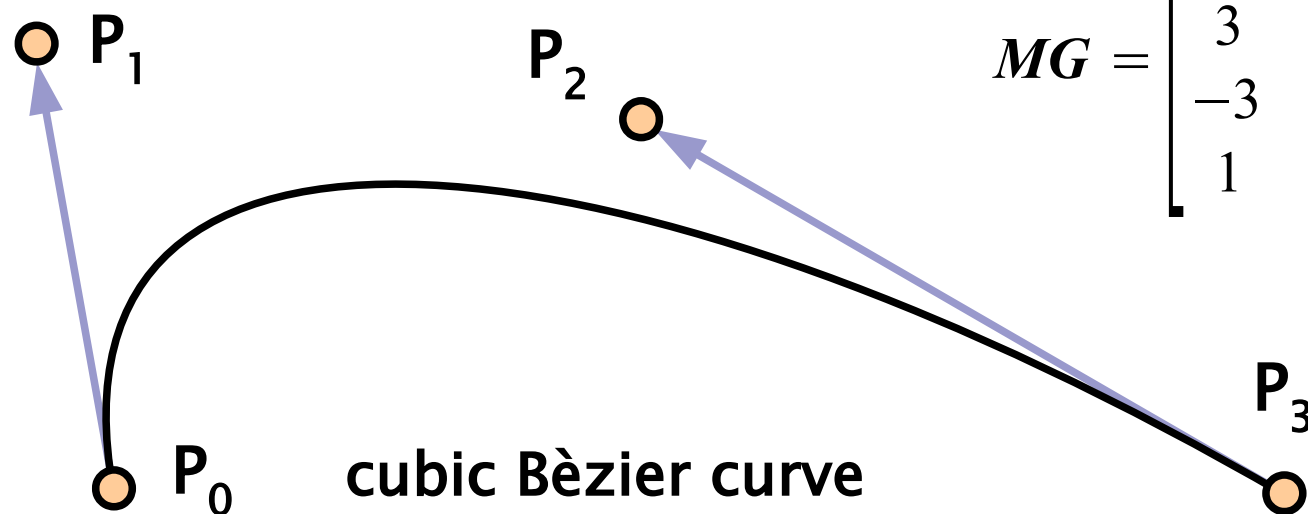


# Bèzier curves II

## Cauchy condition

⇒ convex combination of control points

$$\sum_{i=0}^n B_i^n(t) = 1 \quad \text{for } 0 \leq t \leq 1$$

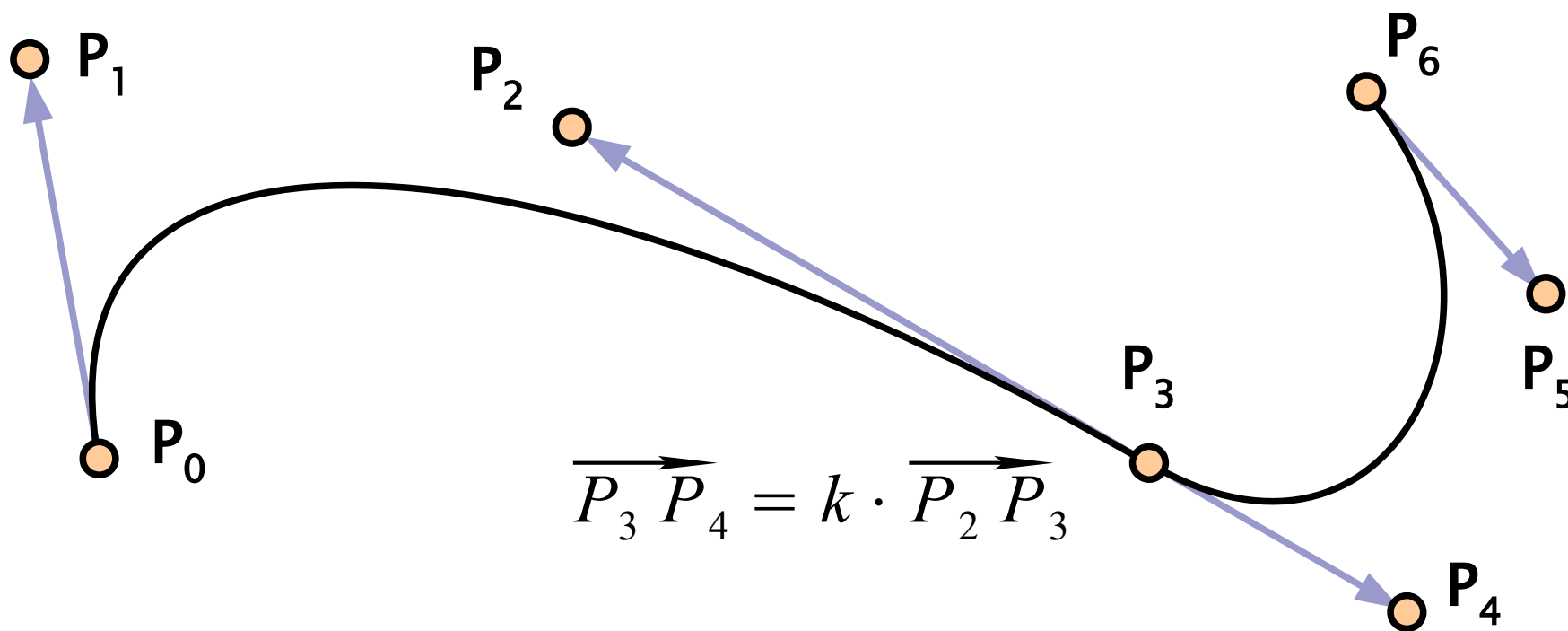


$$MG = \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{bmatrix}$$



# Joining Bézier curves I

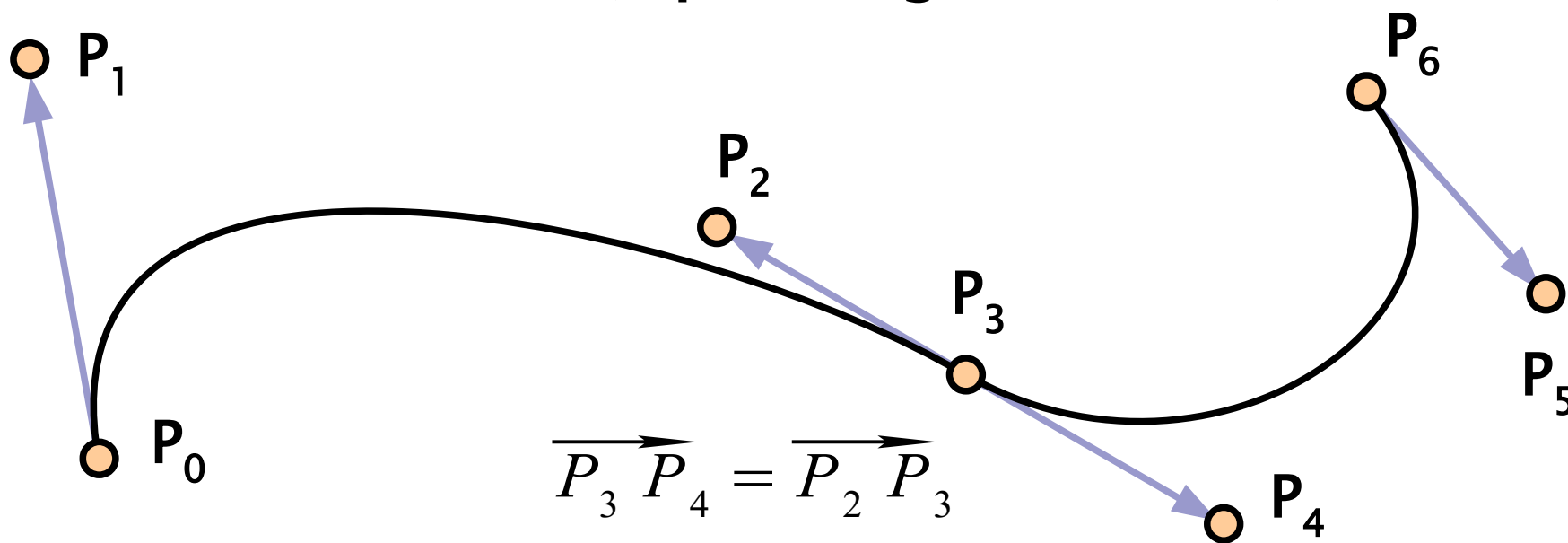
$G^1$  connection (co-linear tangents)





# Joining Bèzier curves II

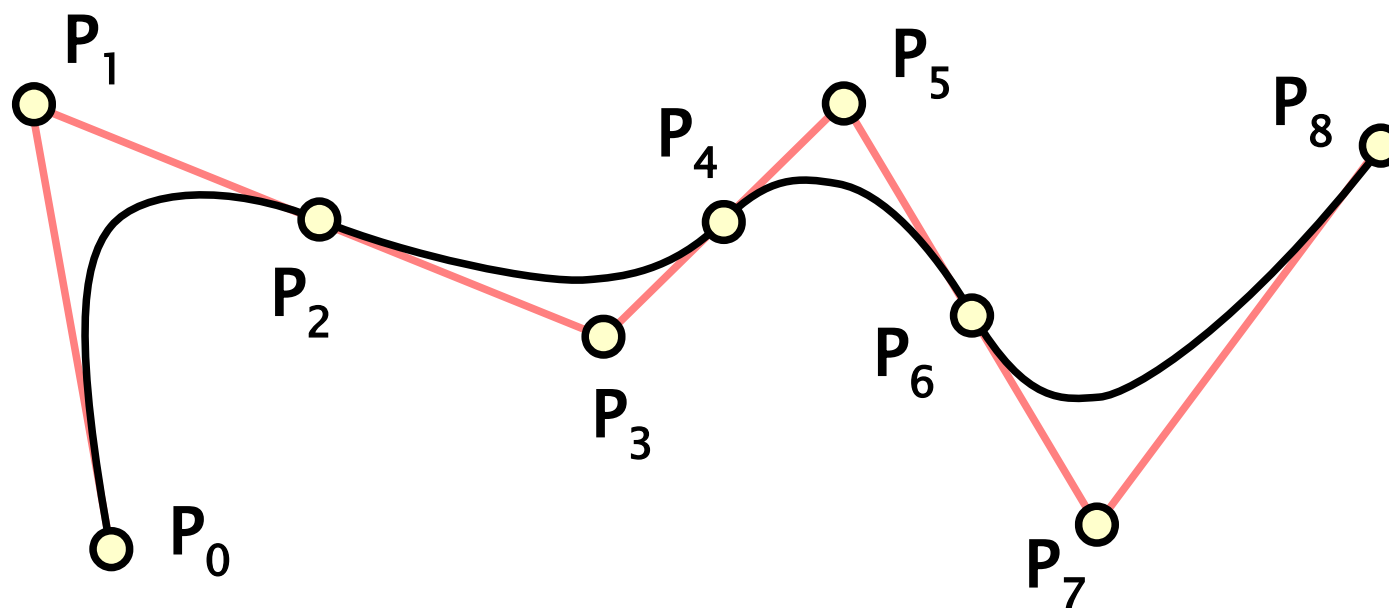
$C^1$  connection (equal tangent vectors)





# Joining Bèzier curves III

## Quadratic spline from Bèzier segments

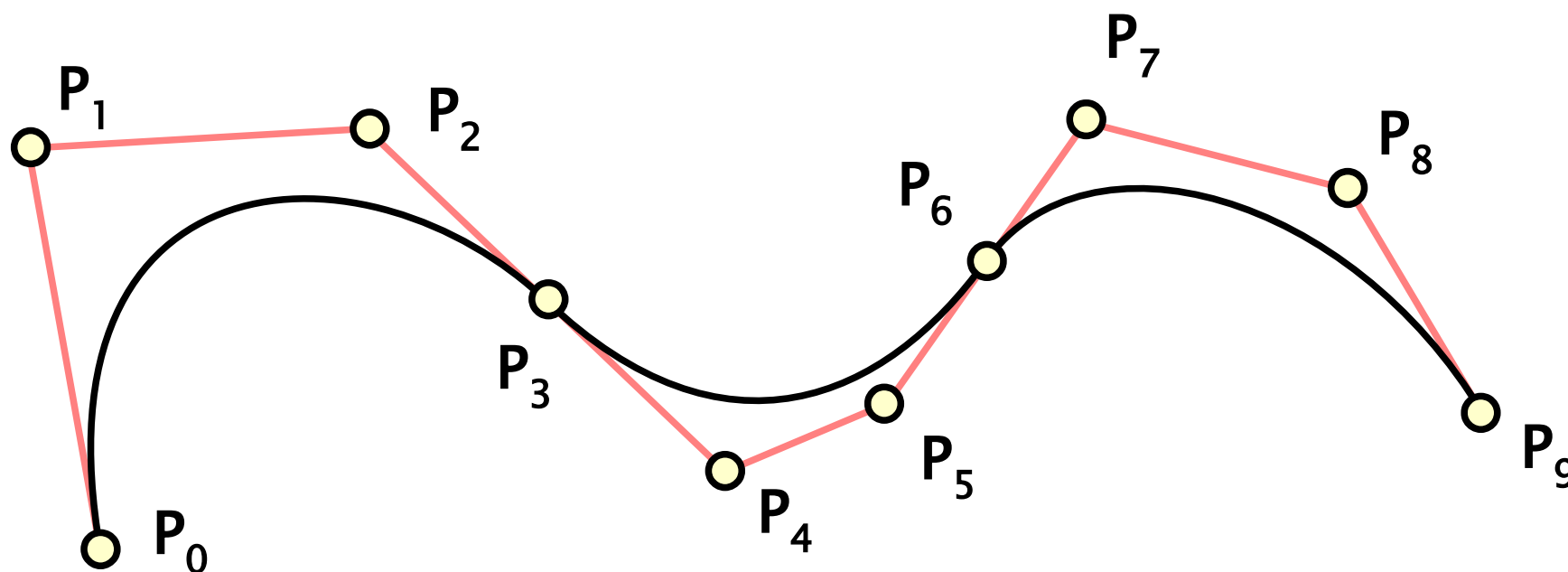


$$\overrightarrow{P_1 P_2} = \overrightarrow{P_2 P_3} \quad \overrightarrow{P_3 P_4} = \overrightarrow{P_4 P_5} \quad \dots \quad \overrightarrow{P_{2k-1} P_{2k}} = \overrightarrow{P_{2k} P_{2k+1}}$$



# Joining Bèzier curves IV

## Cubic spline from Bèzier segments

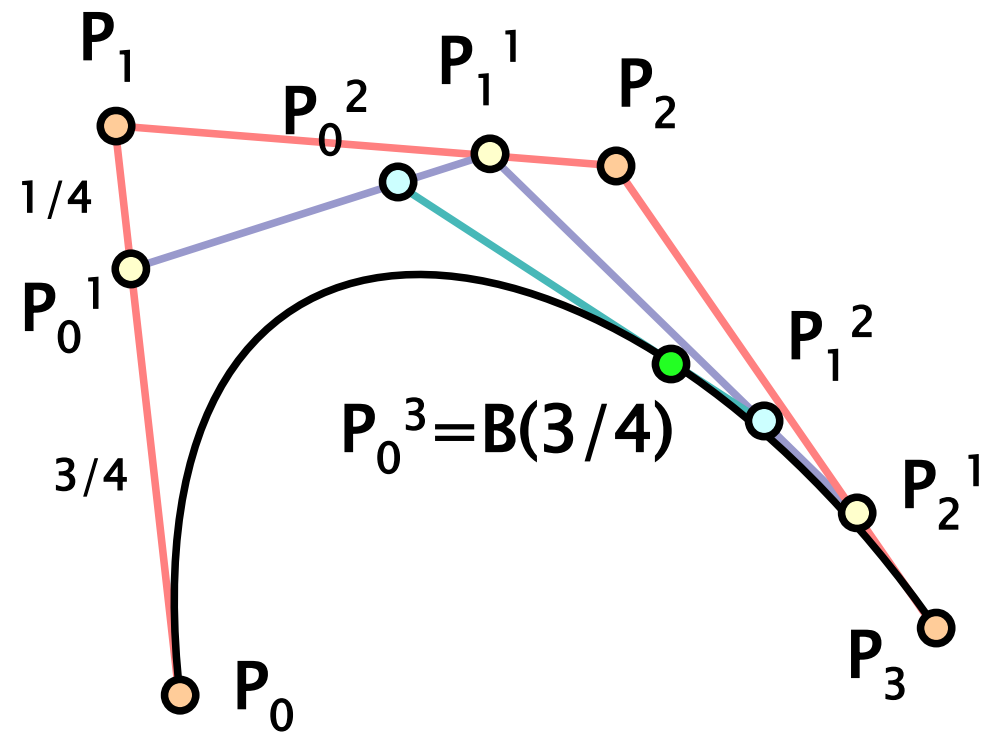
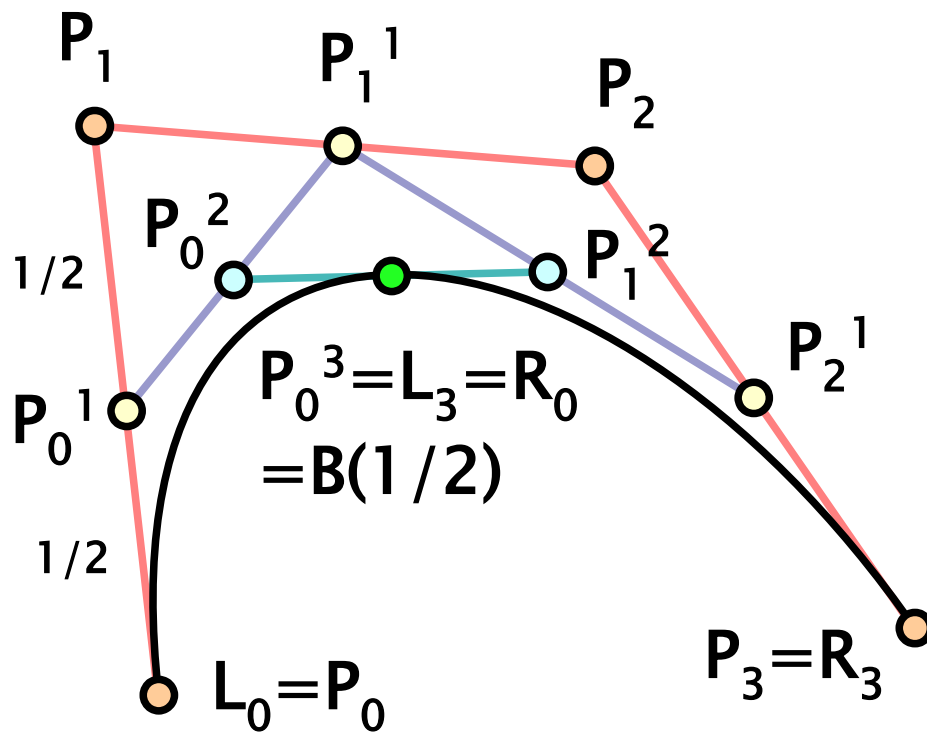


$$\overrightarrow{P_2 P_3} = \overrightarrow{P_3 P_4} \quad \overrightarrow{P_5 P_6} = \overrightarrow{P_6 P_7} \quad \dots \quad \overrightarrow{P_{3k-1} P_{3k}} = \overrightarrow{P_{3k} P_{3k+1}}$$

# De Casteljau (de Boor) algorithm



- ◆ **geometric construction** of Bèzier curve
  - ◆ used as “**subdivision**” scheme or for computation of a specific point..

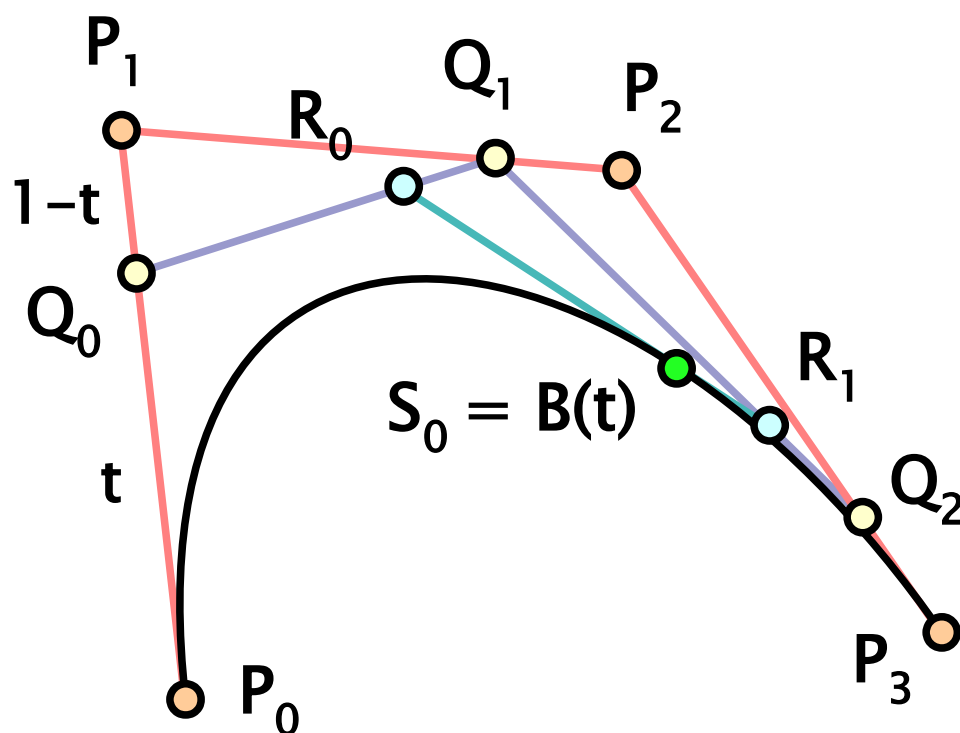




# Using [S]LERP operation

## ◆ linear interpolation LERP (SLERP for quaternions)

◆  $\text{LERP}(A, B, t) = A \cdot (1 - t) + B \cdot t$



## Cubic Bézier:

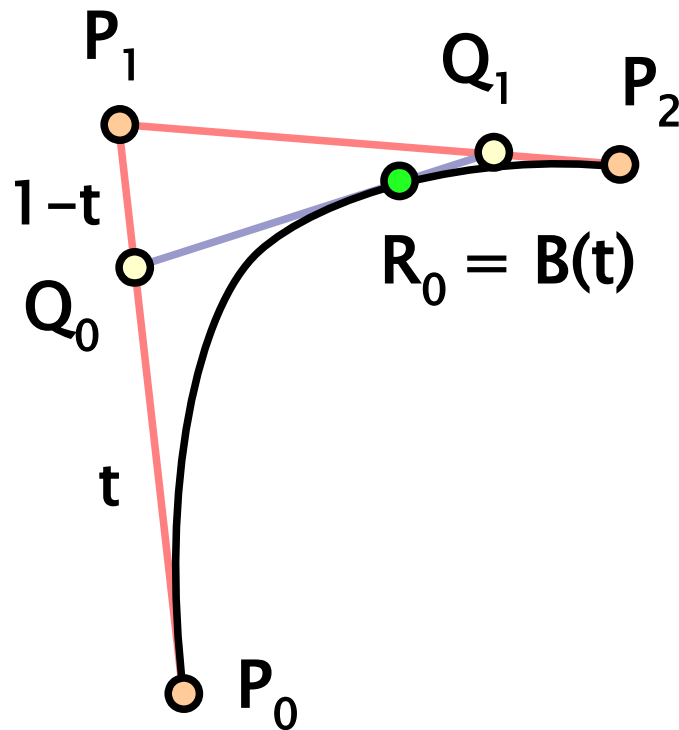
$$Q_i = \text{LERP}(P_i, P_{i+1}, t)$$

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

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



# [S]LERP for quadratic interpolation



## Quadratic Bézier:

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

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



# Cubic spline

- ▶ function assembled from **cubic polynomials**

- ◆ neighbor polynomials have  $C^2$  joint
- ◆ elastic “spline-ruler” (see construction)

- ▶ **interpolating cubic spline**

- ◆ in knot points  $\mathbf{x}_0, \mathbf{x}_1, \dots, \mathbf{x}_n$  function values  $\mathbf{y}_0, \mathbf{y}_1, \dots, \mathbf{y}_n$  are prescribed

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

- ▶ **condition A:**  $S(x_k) = y_k \quad k = 0, 1, \dots, n$



# Interpolating cubic spline

- condition **B** ( $C^0$  continuity):

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

- condition **C** ( $C^1$  continuity):

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

- condition **D** ( $C^2$  continuity):

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

- natural cubic spline** has 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, 2 more DoF
- ◆ additional condition **E** (second derivatives in endpoints)

## ◆ closed (cyclic) spline: $\mathbf{x}_0 = \mathbf{x}_n$

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

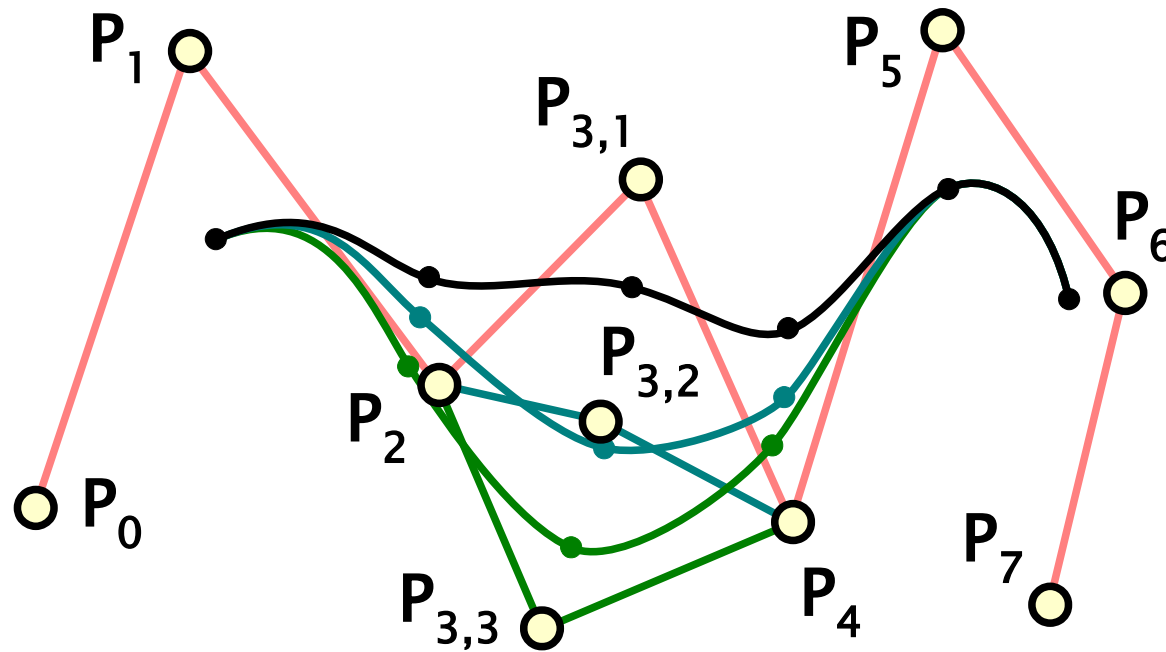


# B-spline (basis spline)

- ◆ “**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  $\mathbf{0} \leq \mathbf{t}_i \leq \mathbf{1}$



# Coons B-spline

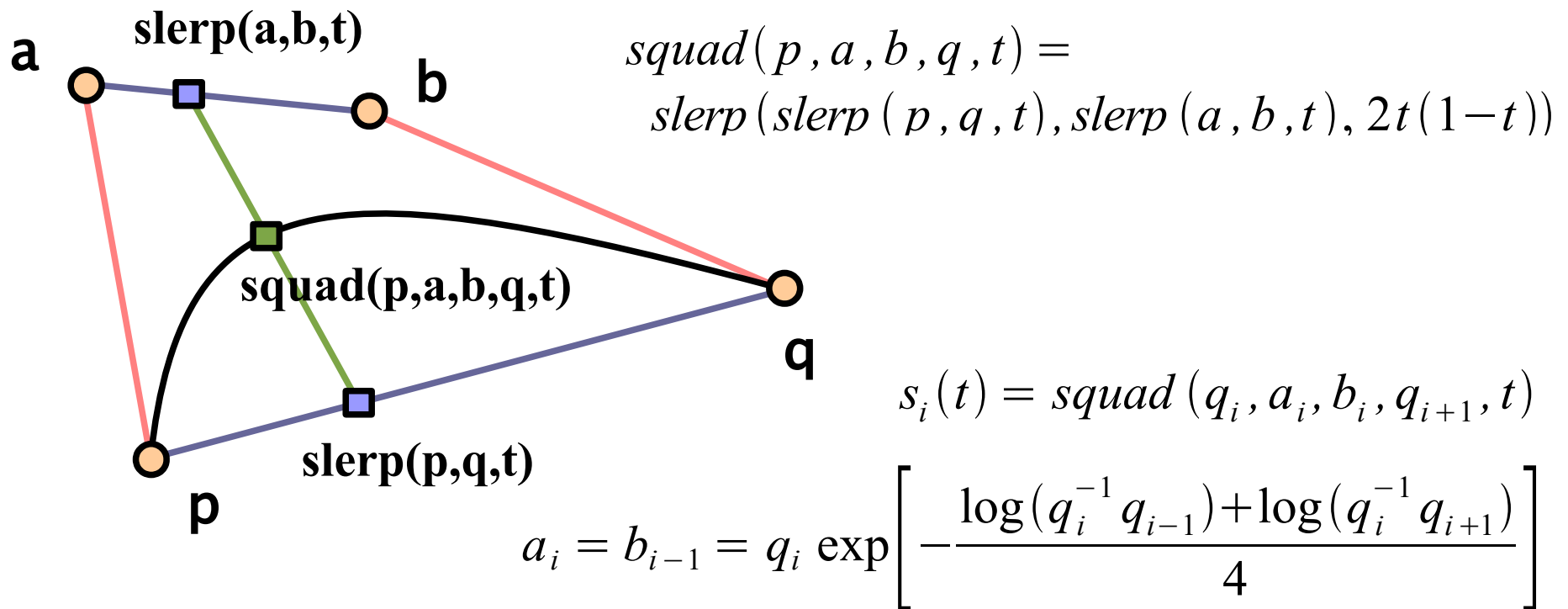


- ◆ continuity  $C^2$
- ◆ sharing 3 CP between neighb.
- ◆ altering one CP induces change in 4 segments

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

# Spline interpolation of quaternions

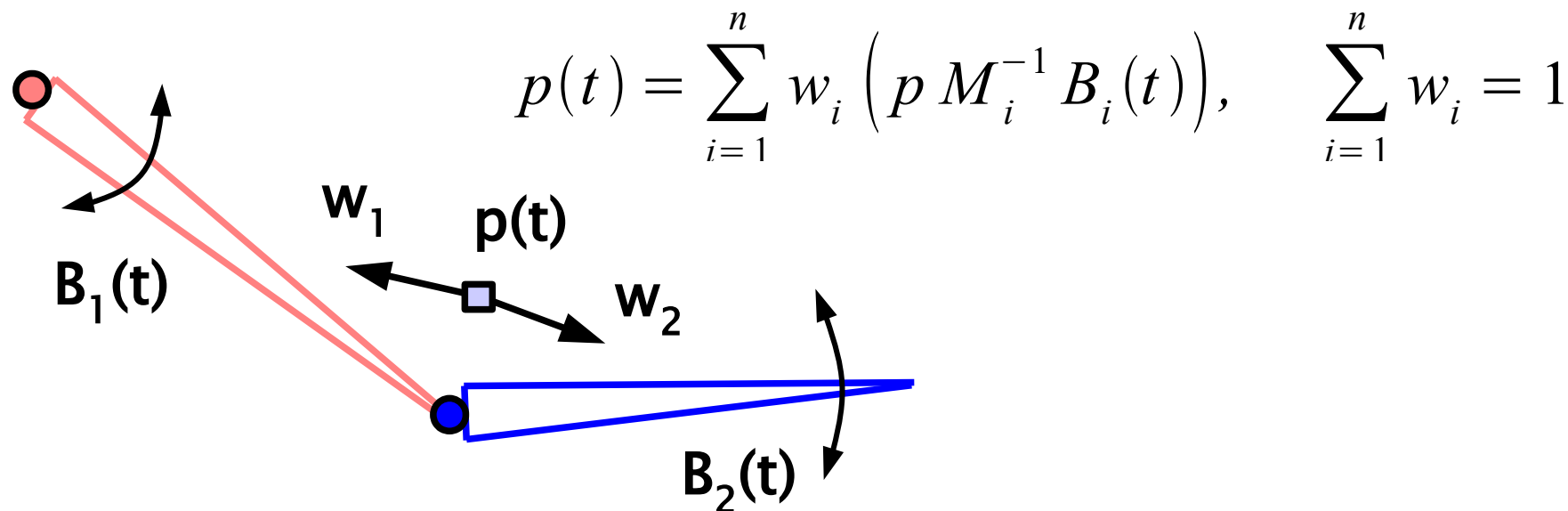
- subsequent interpolation by a sequence of orientations  $\mathbf{q}_0, \mathbf{q}_1, \dots, \mathbf{q}_n$
- $\text{slerp}(\mathbf{q}_i, \mathbf{q}_{i+1}, t)$  not sufficient continuity ( $C^0$  only)





# Vertex blending (“skinning”)

- **vertex  $p$**  of each triangle is connected to some neighbor “**bones**” (coordinate systems)
  - quiescent position:  $t = 0$
  - bones are **animated** by matrices  $\mathbf{B}_i(\mathbf{t})$  (object  $\rightarrow$  world)
  - initial transformation  $\mathbf{M}_i = \mathbf{B}_i(\mathbf{0})$







# Sources

- ◆ Tomas Akenine-Möller, Eric Haines: ***Real-time rendering, 2<sup>nd</sup> edition***, A K Peters, 2002, ISBN: 1568811829
- ◆ J. Žára, B. Beneš, J. Sochor, P. Felkel: ***Moderní počítačová grafika***, 2. vydání, Computer Press, 2005, ISBN: 8025104540
- ◆ **<http://www.geometrictools.com/>** (Dave Eberly)