

Game Programming

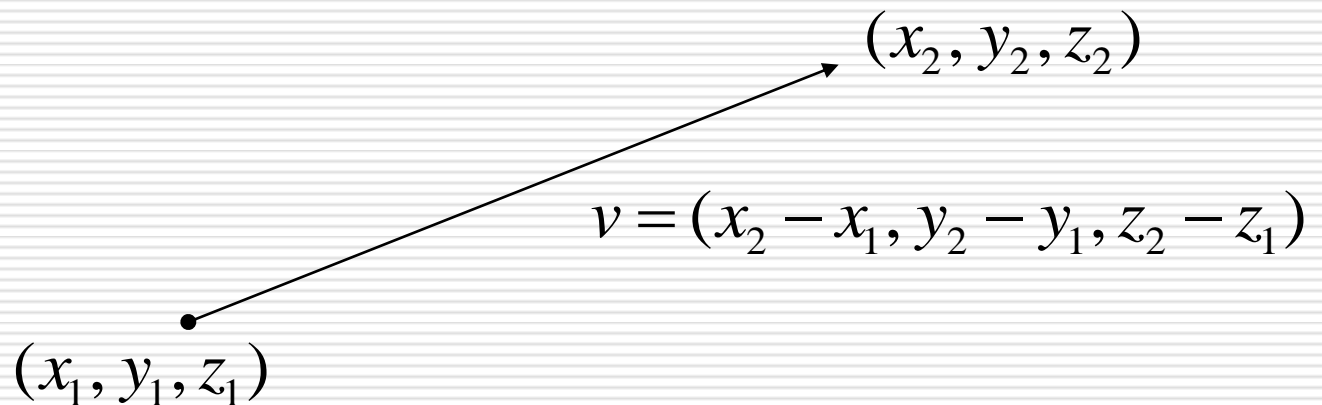
Bing-Yu Chen
National Taiwan University

Game Mathematics

- Vectors
- Matrices
- Transformations
- Homogeneous Coordinates
- 3D Viewing
- Triangle Mathematics
- Intersection Issues
- Fixed-point Real Numbers
- Quaternions
- Parametric Curves

Vectors

- A vector is an entity that possesses *magnitude* and *direction*.
- A ray (directed line segment), that possesses *position*, *magnitude*, and *direction*.

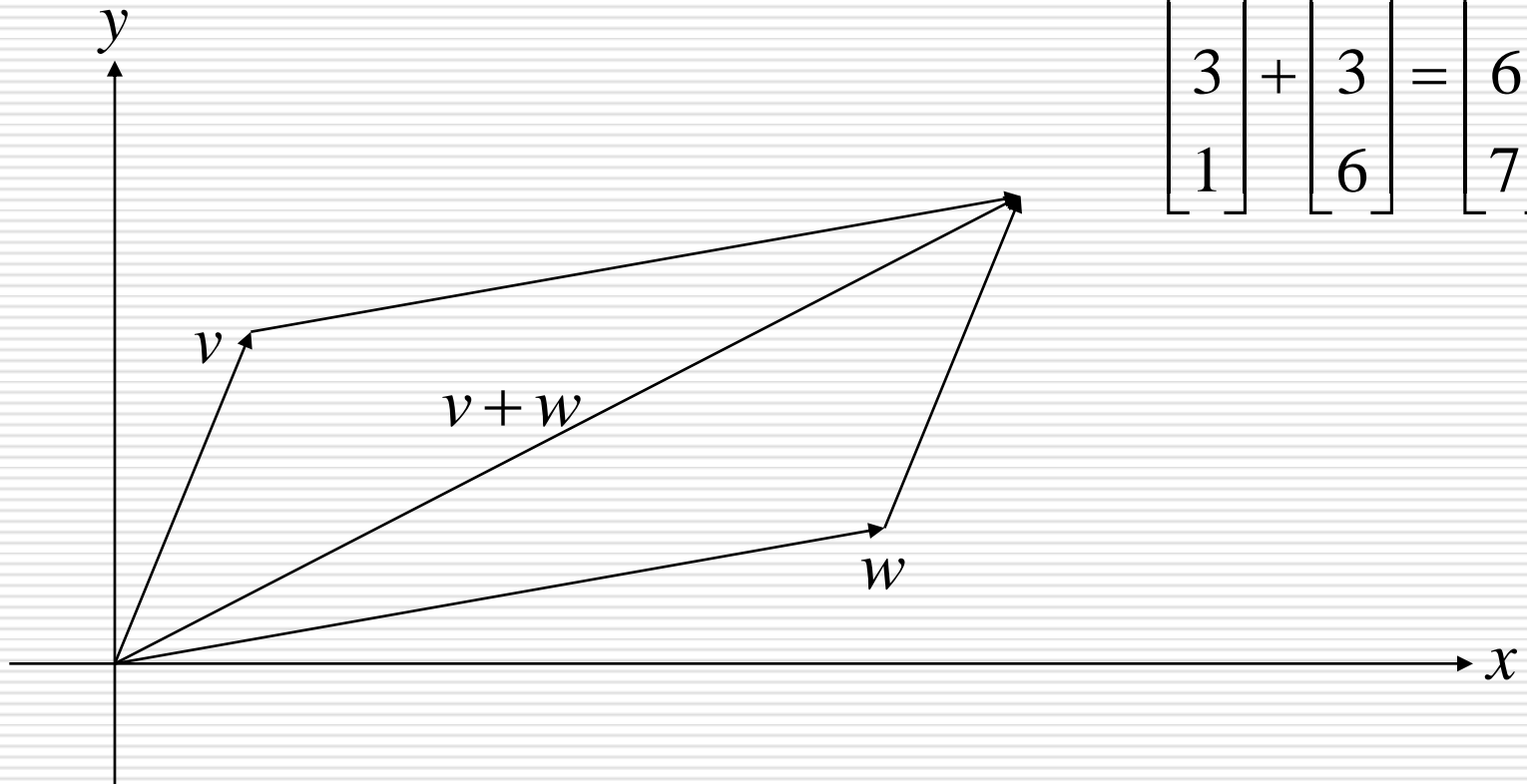


Vectors

- an n-tuple of real numbers (scalars)
- two operations: addition & multiplication
- Commutative Laws
 - $a + b = b + a$
 - $a \cdot b = b \cdot a$
- Identities
 - $a + 0 = a$
 - $a \cdot 1 = a$
- Associative Laws
 - $(a + b) + c = a + (b + c)$
 - $(a \cdot b) \cdot c = a \cdot (b \cdot c)$
- Distributive Laws
 - $a \cdot (b + c) = a \cdot b + a \cdot c$
 - $(a + b) \cdot c = a \cdot c + b \cdot c$
- Inverse
 - $a + b = 0 \rightarrow b = -a$

Addition of Vectors

- parallelogram rule



$$\begin{bmatrix} 1 \\ 3 \\ 1 \end{bmatrix} + \begin{bmatrix} 2 \\ 3 \\ 6 \end{bmatrix} = \begin{bmatrix} 3 \\ 6 \\ 7 \end{bmatrix}$$

The Vector Dot (Inner) Product

$$u = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \quad v = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}$$

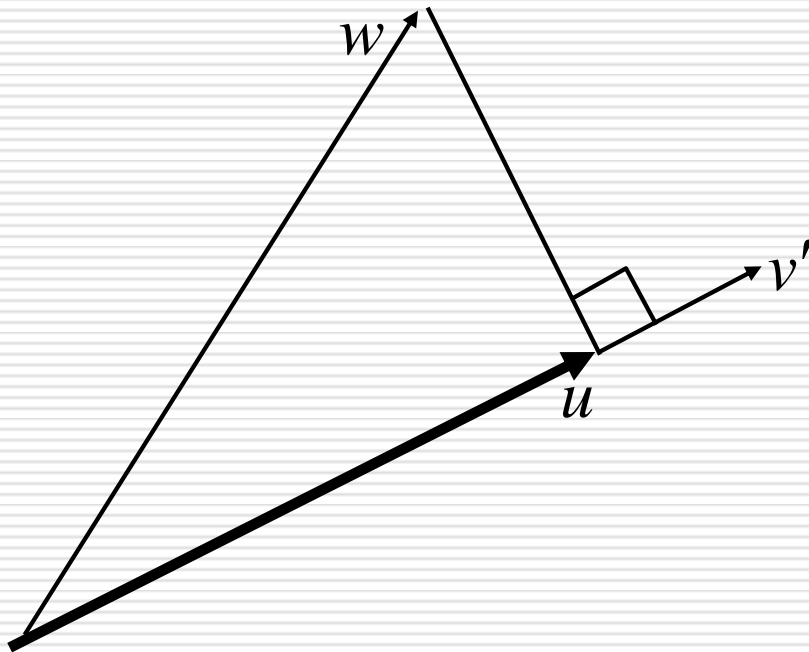
$$\Rightarrow u \bullet v = x_1 y_1 + \dots + x_n y_n$$

□ length = $\sqrt{u \bullet u} = \|u\|$

Properties of the Dot Product

- symmetric
 - $v \bullet w = w \bullet v$
- nondegenerate
 - $v \bullet v = 0$ only when $v = 0$
- bilinear
 - $v \bullet (u + \alpha w) = v \bullet u + \alpha(v \bullet w)$
- unit vector (normalizing)
 - $v' = v / \|v\|$
- angle between the vectors
 - $\cos^{-1}(v \bullet w / \|v\| \|w\|)$

Projection



$$\begin{aligned}\|u\| &= \|w\| \cos \theta \\ &= \|w\| \left(\frac{v' \bullet w}{\|v'\| \|w\|} \right) \\ &= v' \bullet w\end{aligned}$$

Cross Product of Vectors

□ Definition

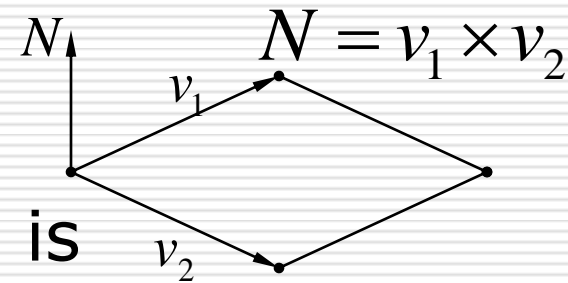
$$x = v \times w$$

$$= (v_2 w_3 - v_3 w_2)i + (v_3 w_1 - v_1 w_3)j + (v_1 w_2 - v_2 w_1)k$$

- where $i = (1, 0, 0)$, $j = (0, 1, 0)$, $k = (0, 0, 1)$ are standard unit vectors

□ Application

- A normal vector to a polygon is calculated from 3 (non-collinear) vertices of the polygon.



Matrix Basics

□ Definition

$$\mathbf{A} = (a_{ij}) = \begin{bmatrix} a_{11} & \dots & a_{1n} \\ \vdots & & \vdots \\ a_{m1} & \dots & a_{mn} \end{bmatrix}$$

□ Transpose

$$\mathbf{C} = \mathbf{A}^T \quad c_{ij} = a_{ji} \Rightarrow \mathbf{C} = \begin{bmatrix} a_{11} & \dots & a_{n1} \\ \vdots & & \vdots \\ a_{1m} & \dots & a_{nm} \end{bmatrix}$$

□ Addition

$$\mathbf{C} = \mathbf{A} + \mathbf{B} \quad c_{ij} = a_{ij} + b_{ij}$$

Matrix Basics

- Scalar-matrix multiplication

$$\mathbf{C} = \alpha \mathbf{A} \quad c_{ij} = \alpha a_{ij}$$

- Matrix-matrix multiplication

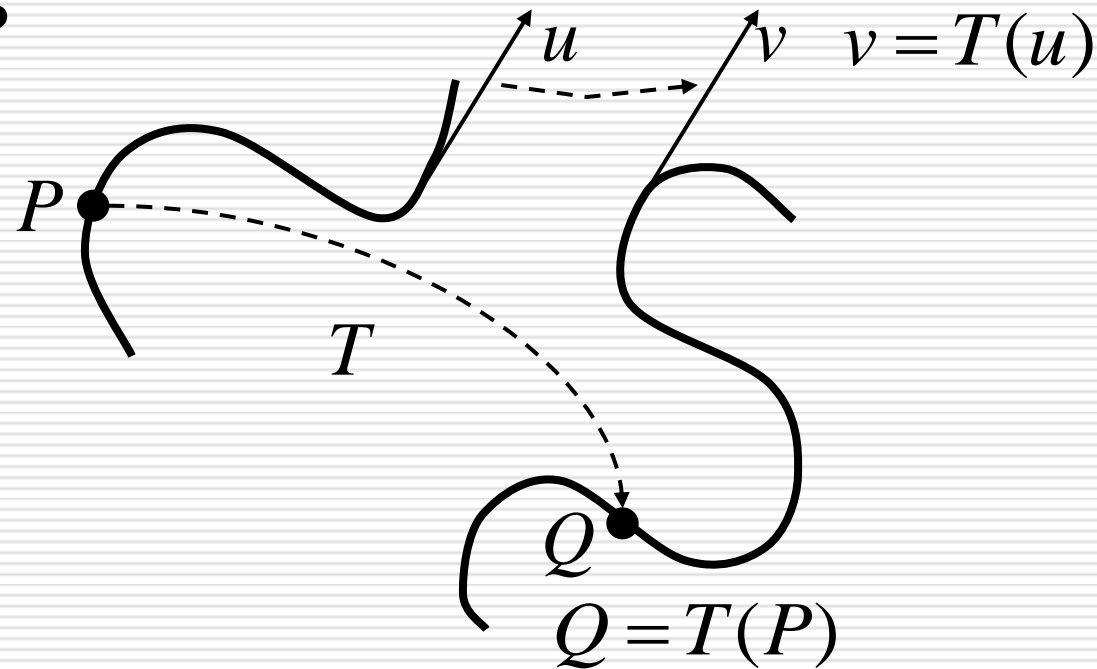
$$\mathbf{C} = \mathbf{AB} \quad c_{ij} = \sum_{k=1}^n a_{ik} b_{kj}$$

- Matrix multiplication are ***not commutative***

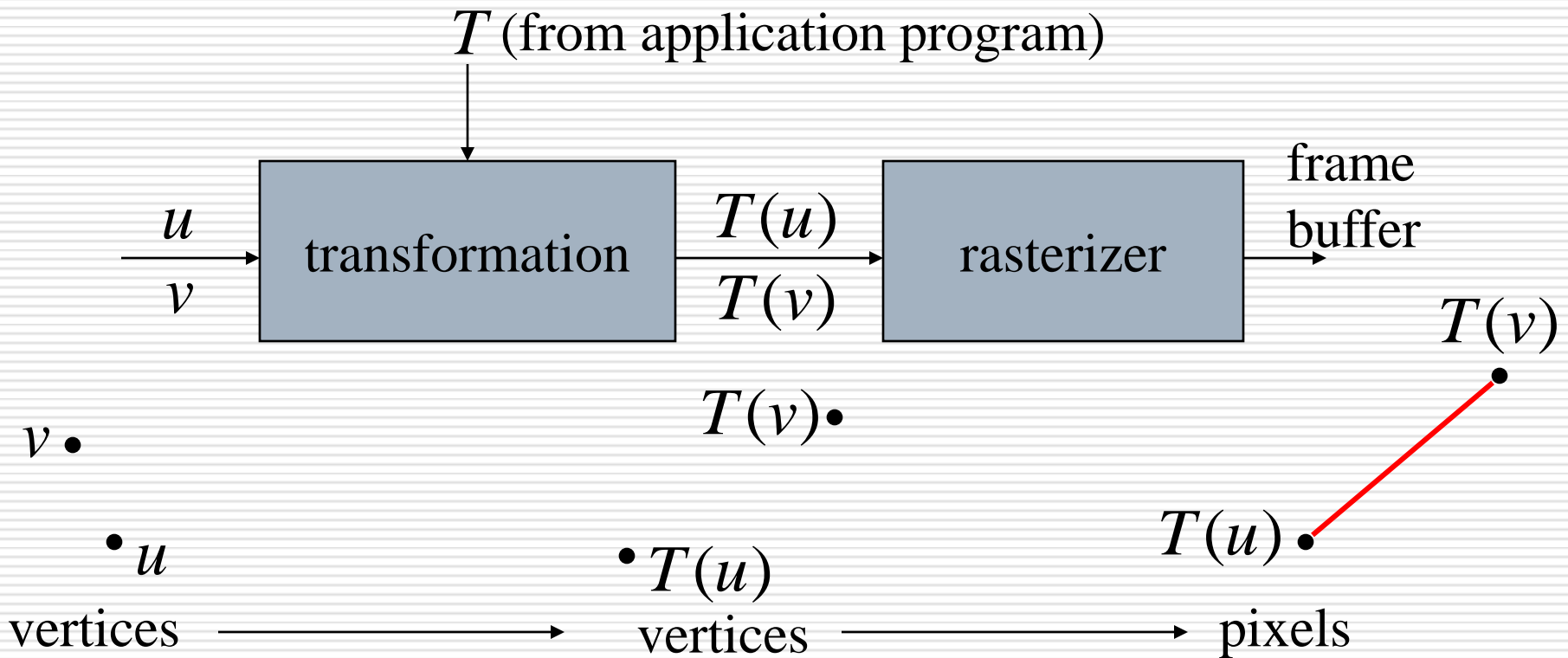
$$\mathbf{AB} \neq \mathbf{BA}$$

General Transformations

- A transformation maps points to other points and/or vectors to other vectors



Pipeline Implementation



Representation

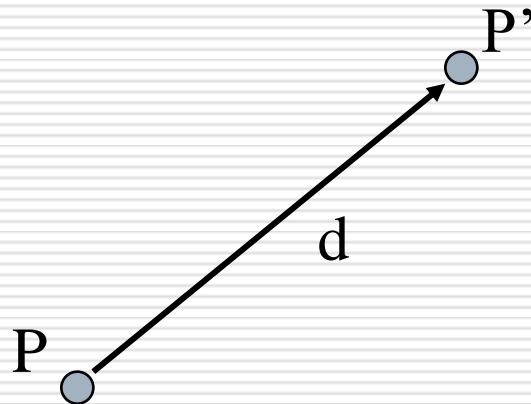
- We can represent a **point**, $\mathbf{p} = (x, y)$ in the plane
 - as a column vector $\begin{bmatrix} x \\ y \end{bmatrix}$
 - as a row vector $[x \ y]$

2D Transformations

- 2D Translation
- 2D Scaling
- 2D Reflection
- 2D Shearing
- 2D Rotation

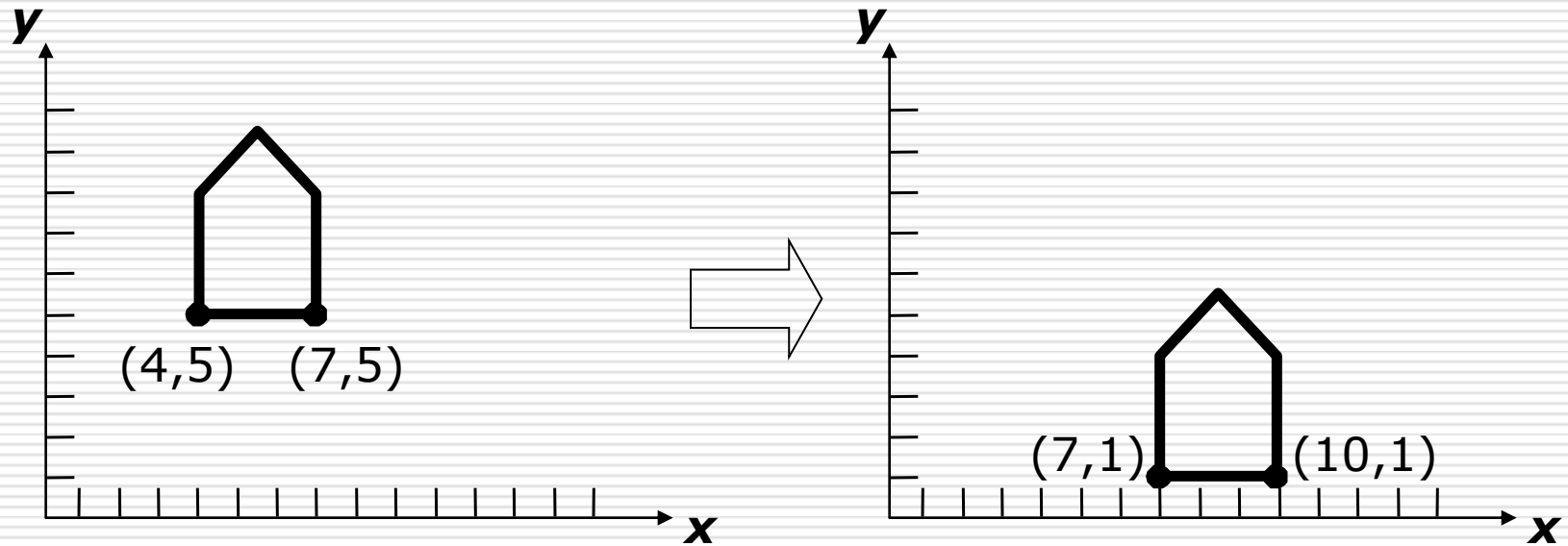
Translation

- Move (translate, displace) a point to a new location



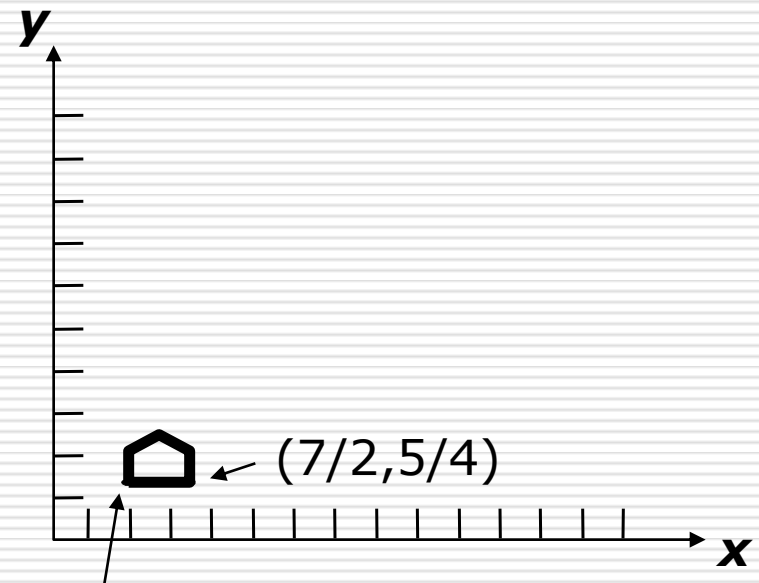
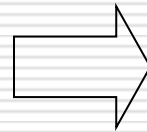
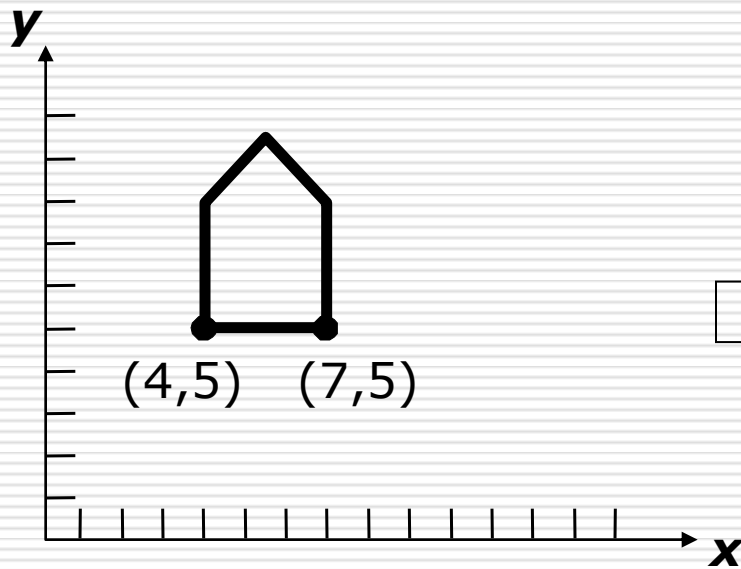
- Displacement determined by a vector d
 - Three degrees of freedom
 - $P' = P + d$

2D Translation



$$P' = P + T$$
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} d_x \\ d_y \end{bmatrix}$$

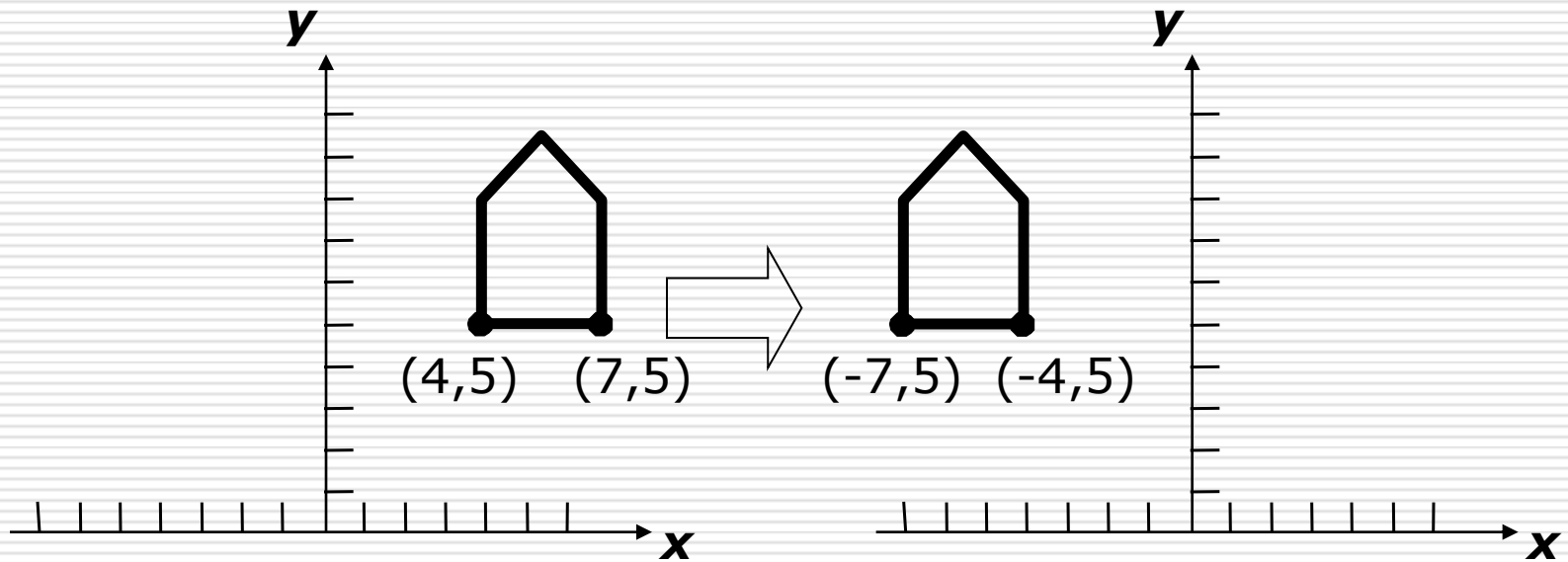
2D Scaling



(2,5/4)

$$P' = S \bullet P$$
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

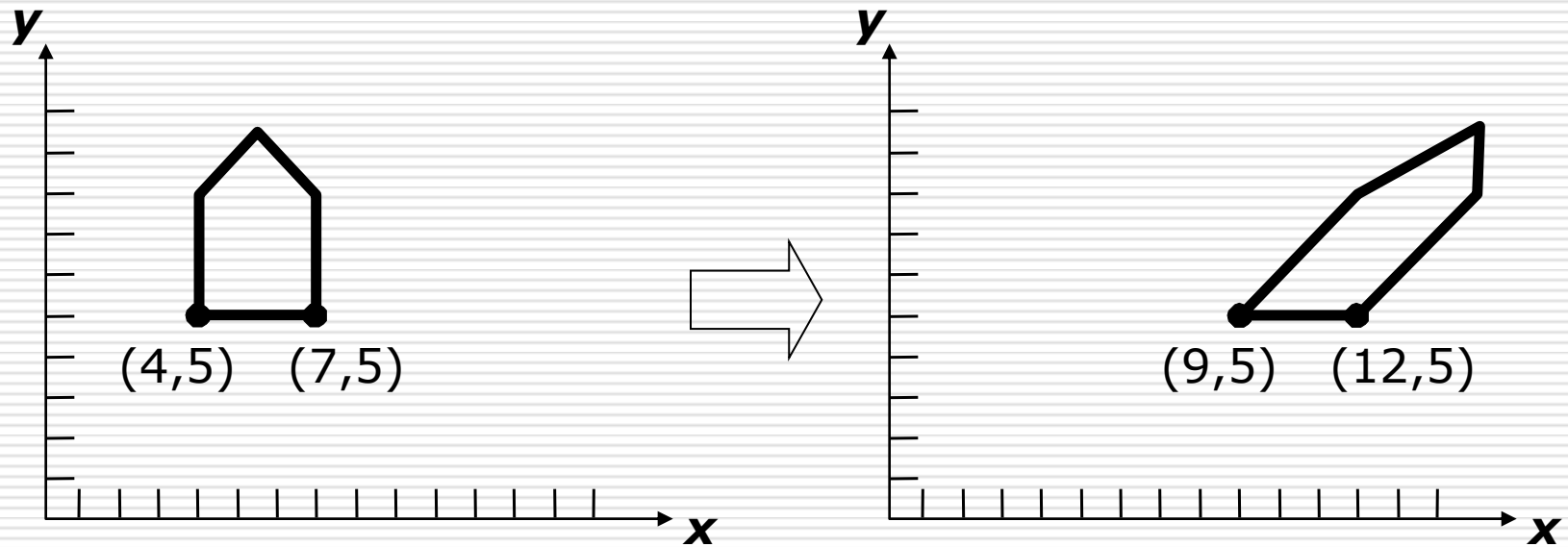
2D Reflection



$$P' = RE_x \bullet P$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

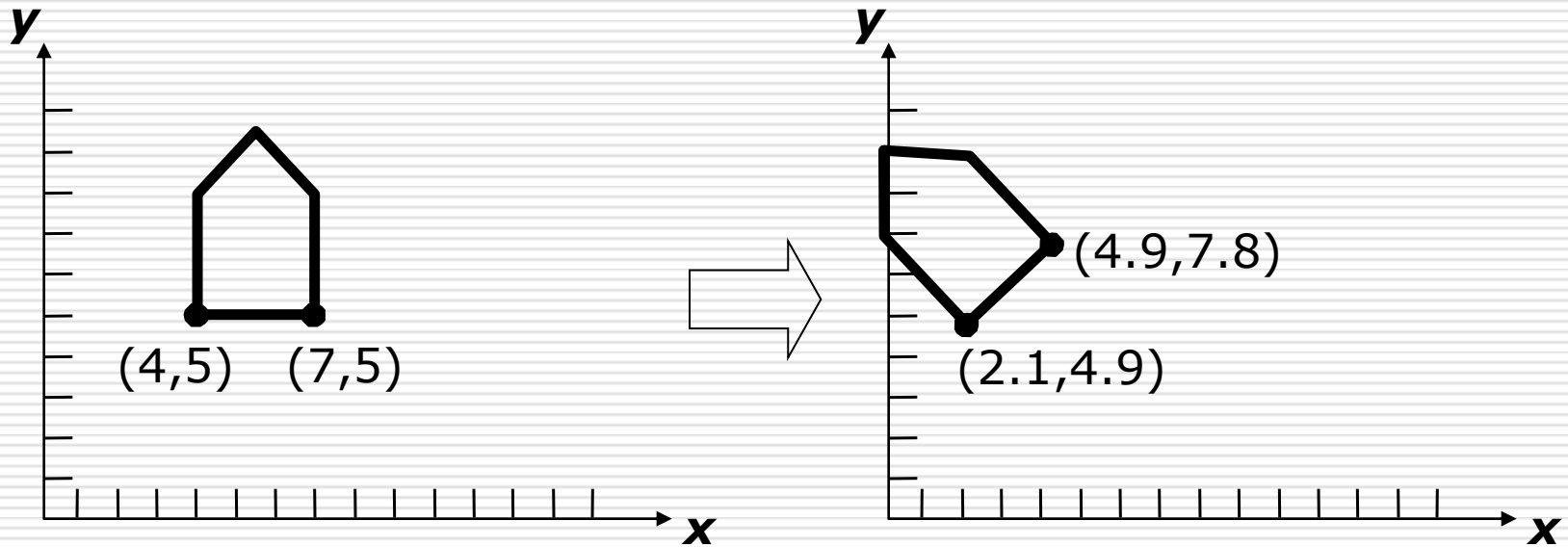
2D Shearing



$$P' = SH_x \bullet P$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & sh_x \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

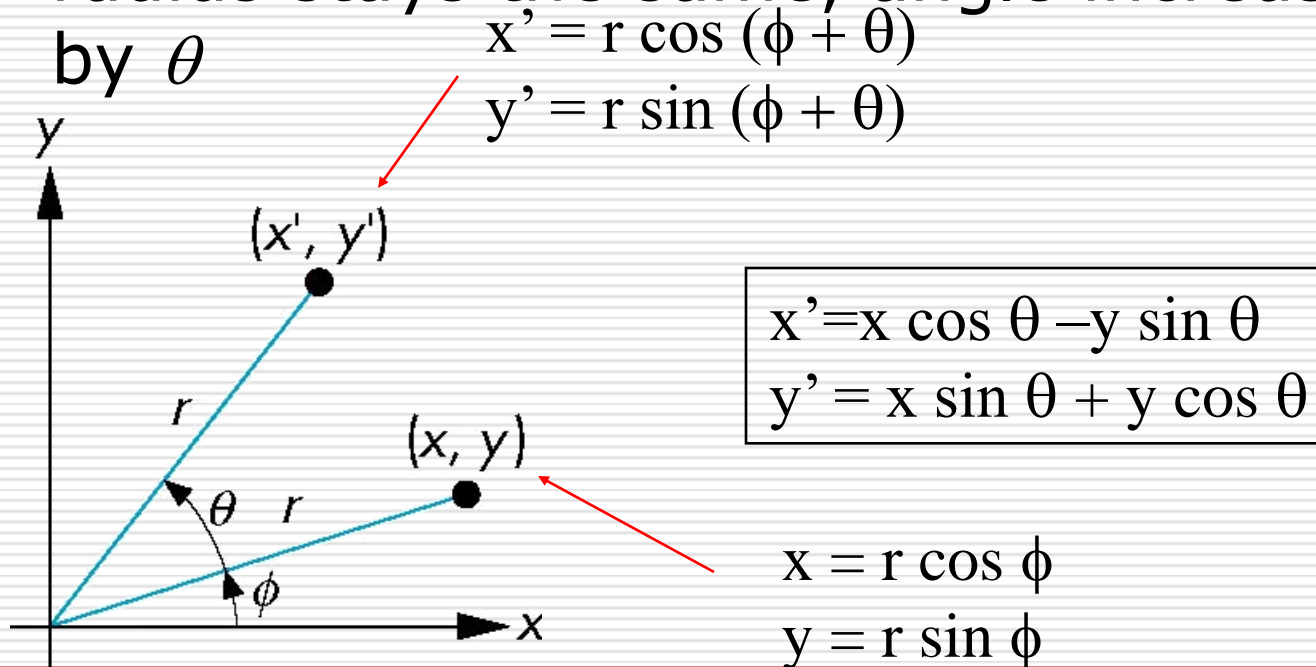
2D Rotation



$$P' = R \bullet P$$
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \bullet \begin{bmatrix} x \\ y \end{bmatrix}$$

2D Rotation

- Consider rotation about the origin by θ degrees
- radius stays the same, angle increases by θ



Limitations of a 2X2 matrix

- Scaling
 - Rotation
 - Reflection
 - Shearing
-
- What do we miss?

Homogeneous Coordinates

□ Why & What is

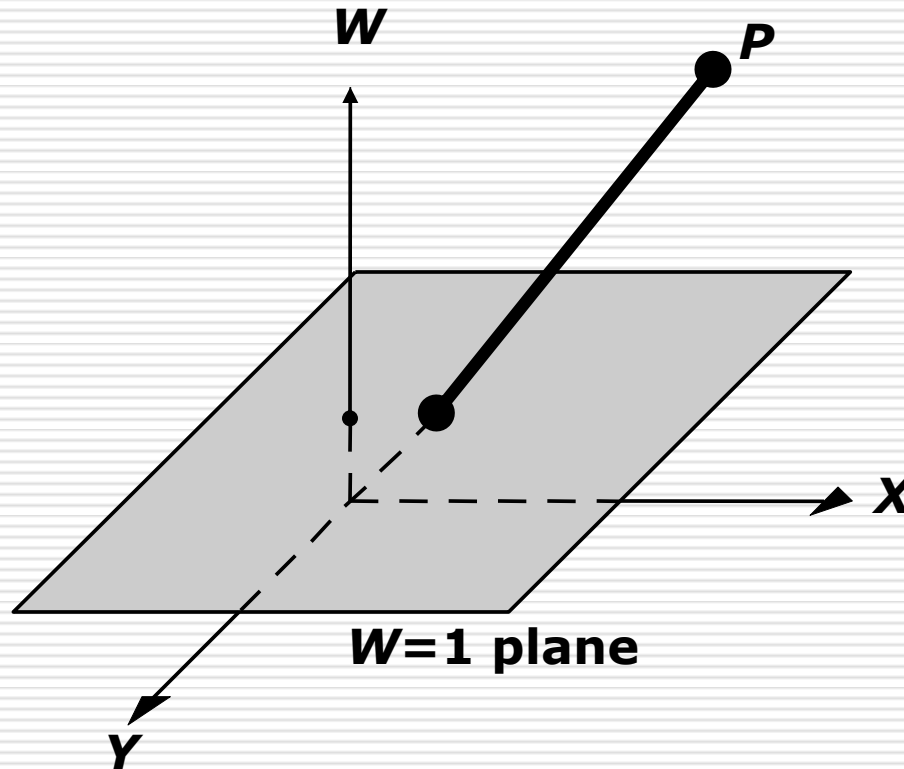
homogeneous coordinates ?

- if points are expressed in homogeneous coordinates, all three transformations can be treated as multiplications.

$$(x, y) \rightarrow (x, y, W)$$

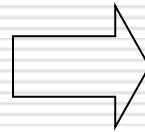
↑
usually 1
can not be 0

Homogeneous Coordinates



Homogeneous Coordinates for 2D Translation

$$P' = P + T$$
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} d_x \\ d_y \end{bmatrix}$$



$$P' = T(d_x, d_y) \bullet P$$
$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & d_x \\ 0 & 1 & d_y \\ 0 & 0 & 1 \end{bmatrix} \bullet \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$P' = T(d_{x1}, d_{y1}) \bullet P$$

$$P'' = T(d_{x2}, d_{y2}) \bullet P'$$

Homogeneous Coordinates for 2D Translation

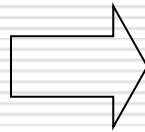
$$\begin{aligned} P'' &= T(d_{x2}, d_{y2}) \bullet (T(d_{x1}, d_{y1}) \bullet P) \\ &= (T(d_{x2}, d_{y2}) \bullet T(d_{x1}, d_{y1})) \bullet P \end{aligned}$$

$$\begin{aligned} T(d_{x2}, d_{y2}) \bullet T(d_{x1}, d_{y1}) &= \begin{bmatrix} 1 & 0 & d_{x2} \\ 0 & 1 & d_{y2} \\ 0 & 0 & 1 \end{bmatrix} \bullet \begin{bmatrix} 1 & 0 & d_{x1} \\ 0 & 1 & d_{y1} \\ 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} 1 & 0 & d_{x1} + d_{x2} \\ 0 & 1 & d_{y1} + d_{y2} \\ 0 & 0 & 1 \end{bmatrix} \end{aligned}$$

Homogeneous Coordinates for 2D Scaling

$$P' = S \bullet P$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$



$$P' = S(s_x, s_y) \bullet P$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \bullet \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$S(s_{x2}, s_{y2}) \bullet S(s_{x1}, s_{y1}) = \begin{bmatrix} s_{x2} & 0 & 0 \\ 0 & s_{y2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \bullet \begin{bmatrix} s_{x1} & 0 & 0 \\ 0 & s_{y1} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} s_{x1} \bullet s_{x2} & 0 & 0 \\ 0 & s_{y1} \bullet s_{y2} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

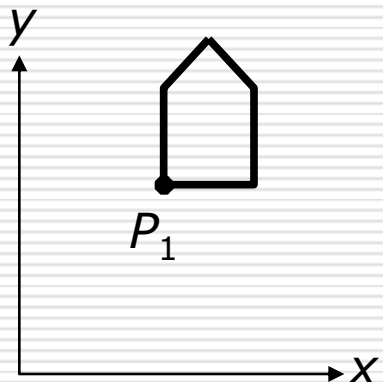
Homogeneous Coordinates for 2D Rotation

$$\begin{aligned} P' = R \bullet P \\ \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \bullet \begin{bmatrix} x \\ y \end{bmatrix} \end{aligned} \Rightarrow \begin{aligned} P' = R(\theta) \bullet P \\ \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \bullet \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \end{aligned}$$

Properties of Transformations

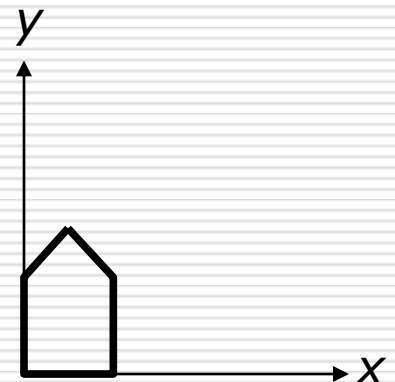
- rigid-body transformations
 - rotation & translation
 - preserving angles and lengths
- affine transformations
 - rotation & translation & scaling
 - preserving parallelism of lines

Composition of 2D Transformations



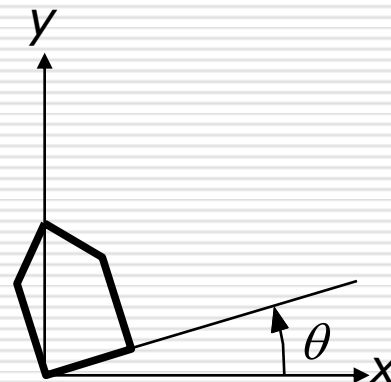
Original

$$P_1 = (x_1, y_1)$$



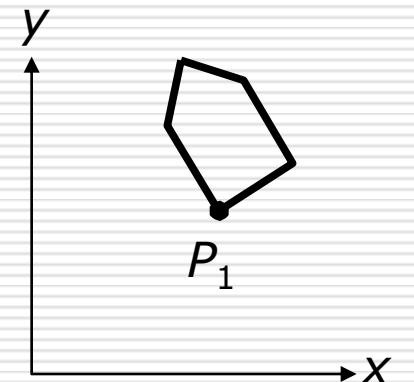
After translation
of P_1 to origin

$$T(-x_1, -y_1)$$



After rotation

$$R(\theta)$$



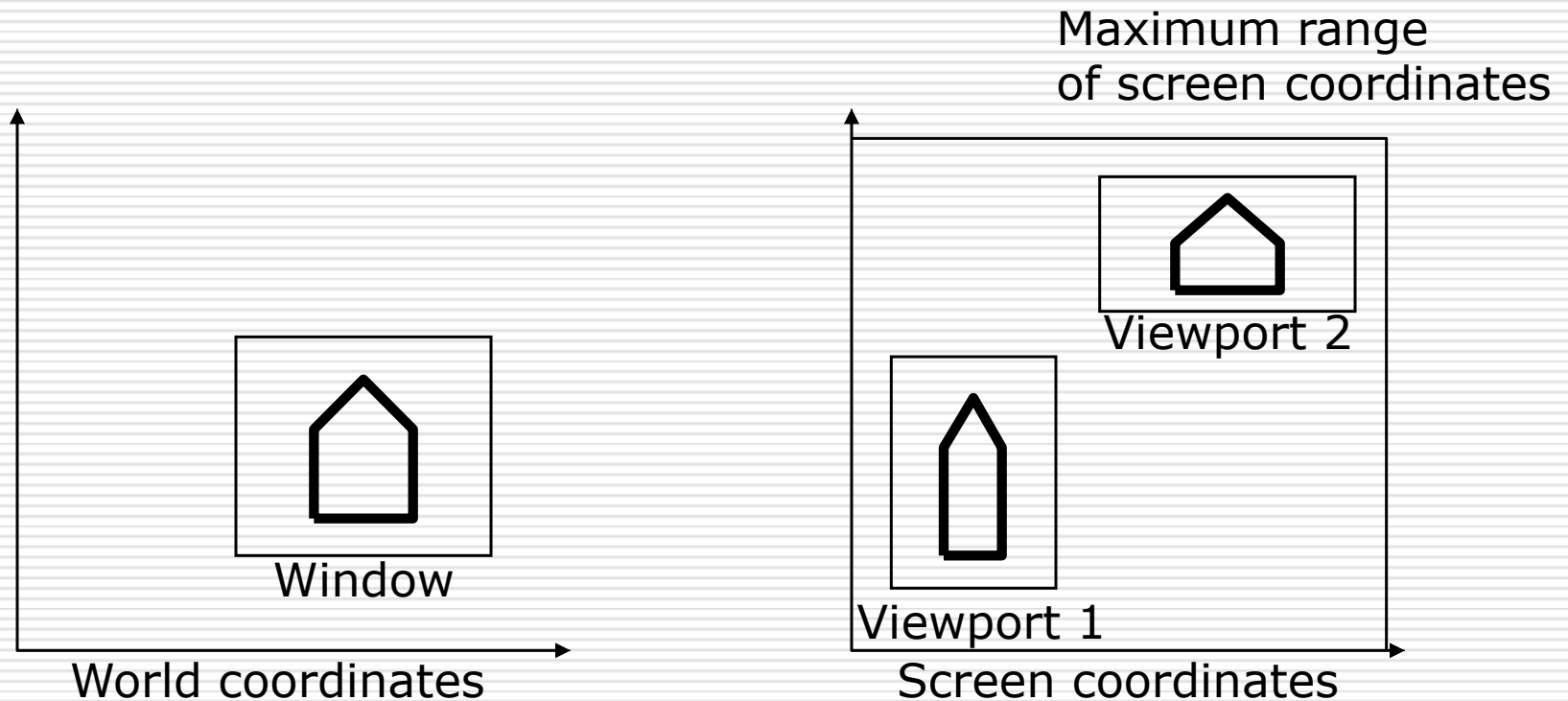
After translation
to original P_1

$$T(x_1, y_1)$$

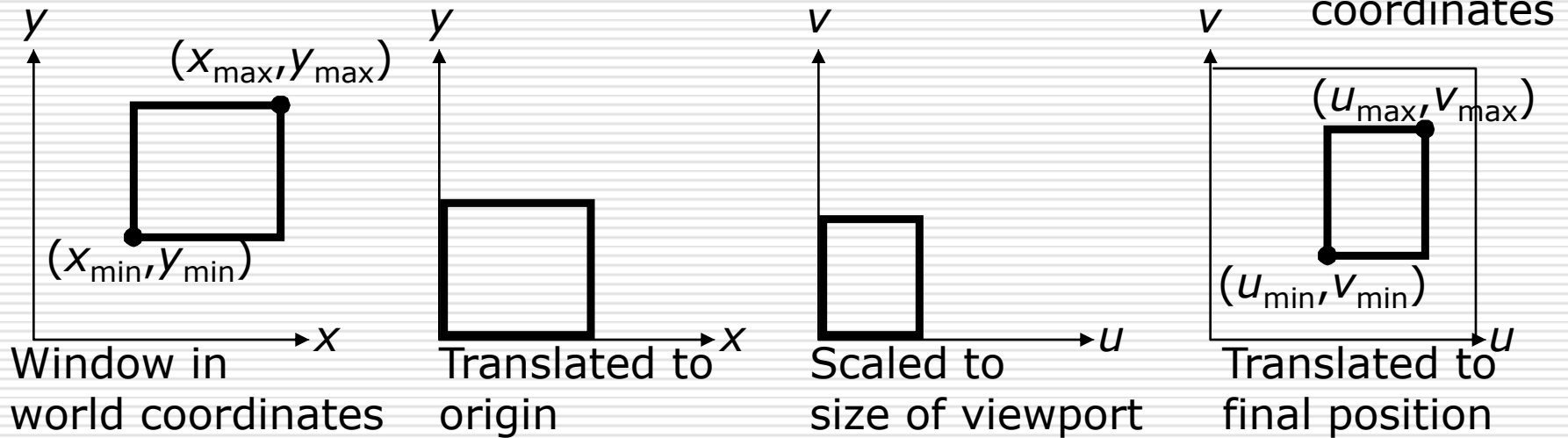
Composition of 2D Transformations

$$\begin{aligned} T(x_1, y_1) \bullet R(\theta) \bullet T(-x_1, -y_1) &= \begin{bmatrix} 1 & 0 & x_1 \\ 0 & 1 & y_1 \\ 0 & 0 & 1 \end{bmatrix} \bullet \begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \bullet \begin{bmatrix} 1 & 0 & -x_1 \\ 0 & 1 & -y_1 \\ 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} \cos \theta & -\sin \theta & x_1(1 - \cos \theta) + y_1 \sin \theta \\ \sin \theta & \cos \theta & y_1(1 - \cos \theta) - x_1 \sin \theta \\ 0 & 0 & 1 \end{bmatrix} \end{aligned}$$

The Window-to-Viewport Transformation

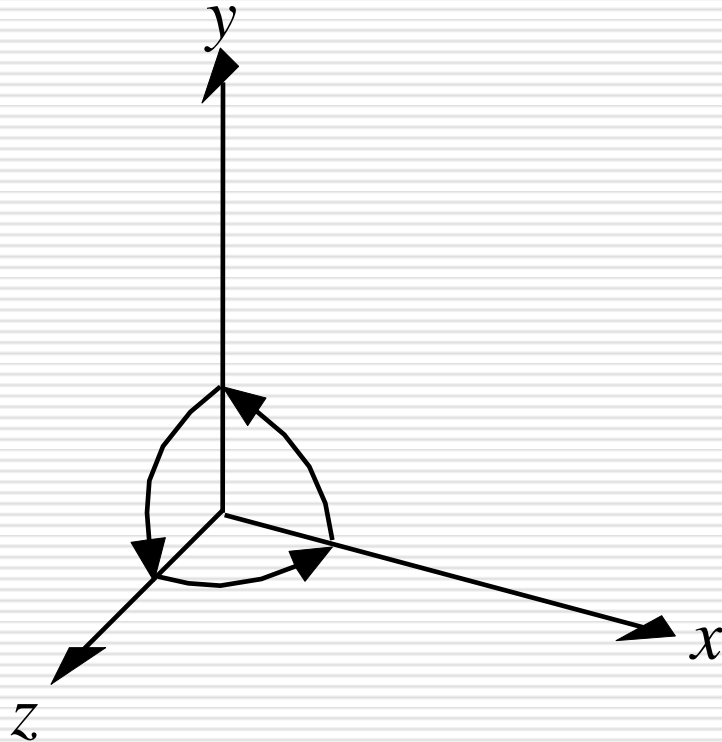


The Window-to-Viewport Transformation



$$M_{wv} = T(u_{\min}, v_{\min}) \bullet S\left(\frac{u_{\max} - u_{\min}}{x_{\max} - x_{\min}}, \frac{v_{\max} - v_{\min}}{y_{\max} - y_{\min}}\right) \bullet T(-x_{\min}, -y_{\min})$$

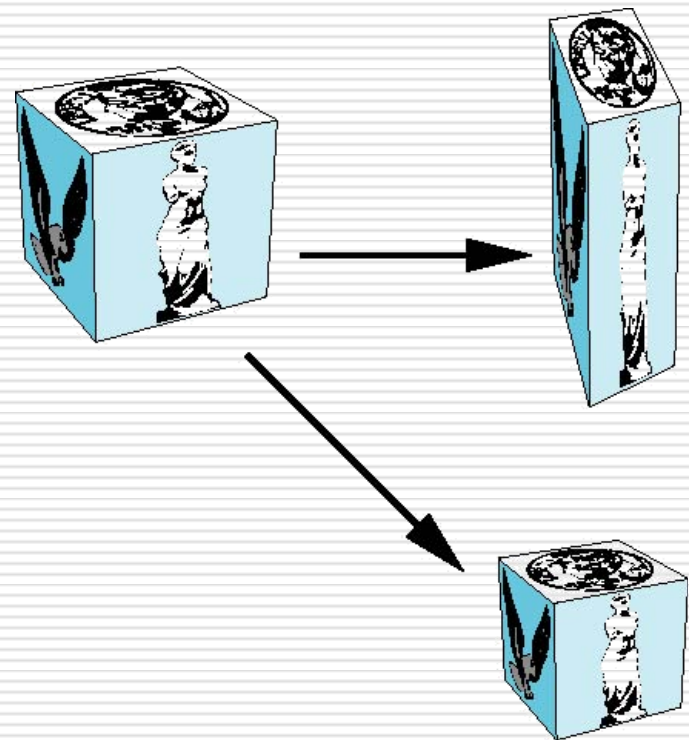
Right-handed Coordinate System



3D Translation & 3D Scaling

$$T(d_x, d_y, d_z) = \begin{bmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$S(s_x, s_y, s_z) = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



3D Reflection & 3D Shearing

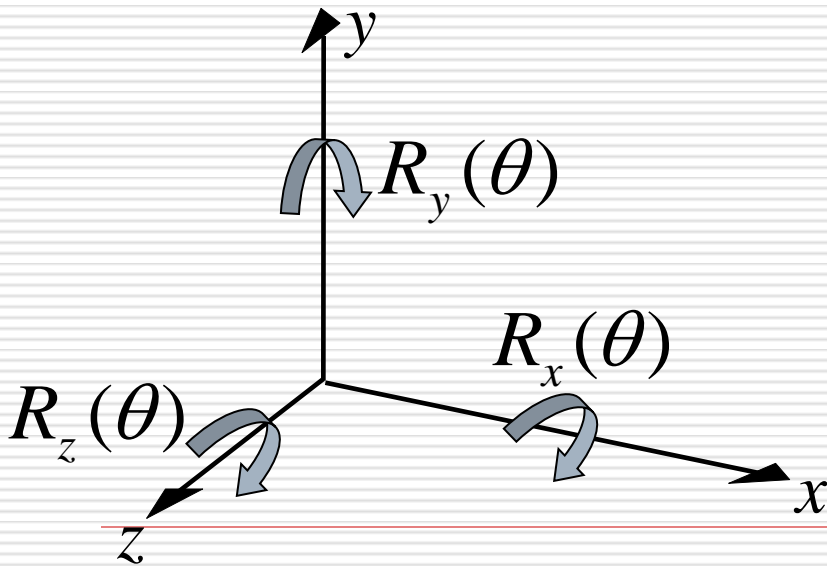
$$RE_x = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad RE_y = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$SH_{xy}(sh_x, sh_y) = \begin{bmatrix} 1 & 0 & sh_x & 0 \\ 0 & 1 & sh_y & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

3D Rotations

$$R_z(\theta) = \begin{bmatrix} \cos \theta & -\sin \theta & 0 & 0 \\ \sin \theta & \cos \theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

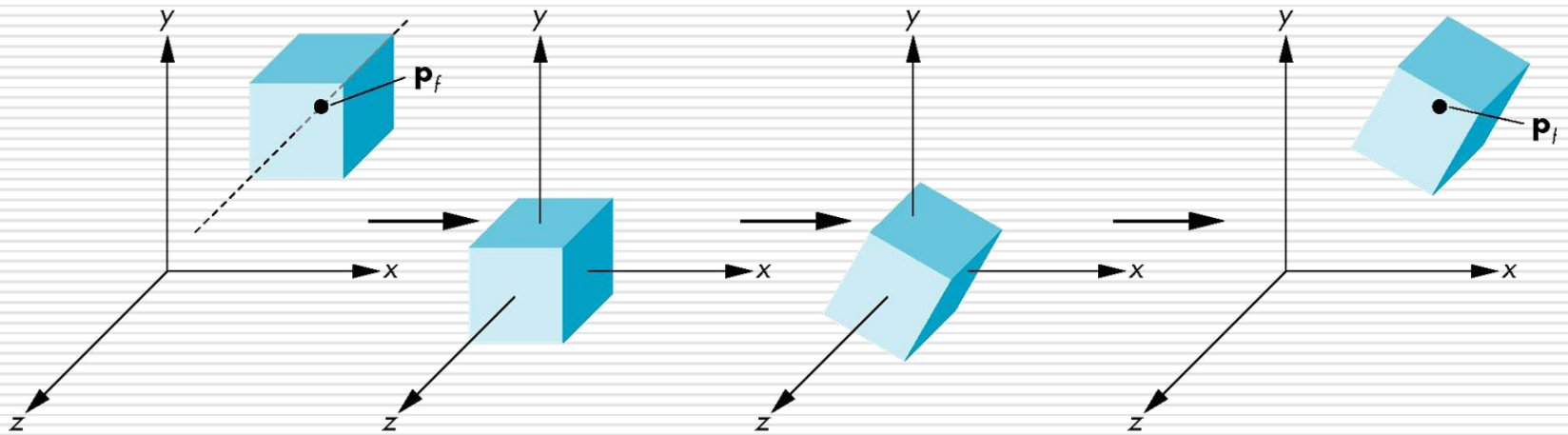
$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \theta & -\sin \theta & 0 \\ 0 & \sin \theta & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



$$R_y(\theta) = \begin{bmatrix} \cos \theta & 0 & \sin \theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \theta & 0 & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation About a Fixed Point other than the Origin

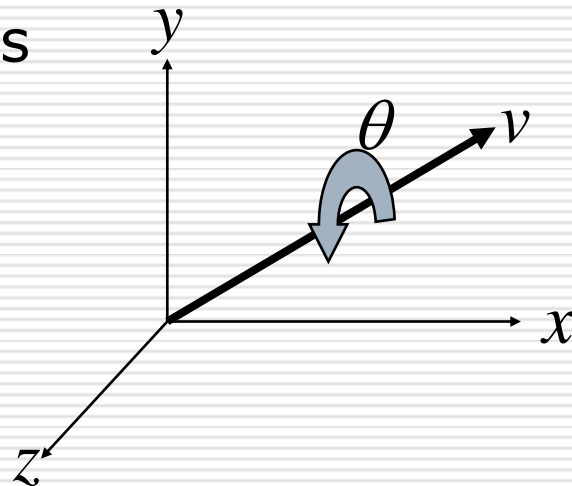
- ❑ Move fixed point to origin
- ❑ Rotate
- ❑ Move fixed point back
- ❑ $M = T(P_f) \bullet R(\theta) \bullet T(-P_f)$



General Rotation About the Origin

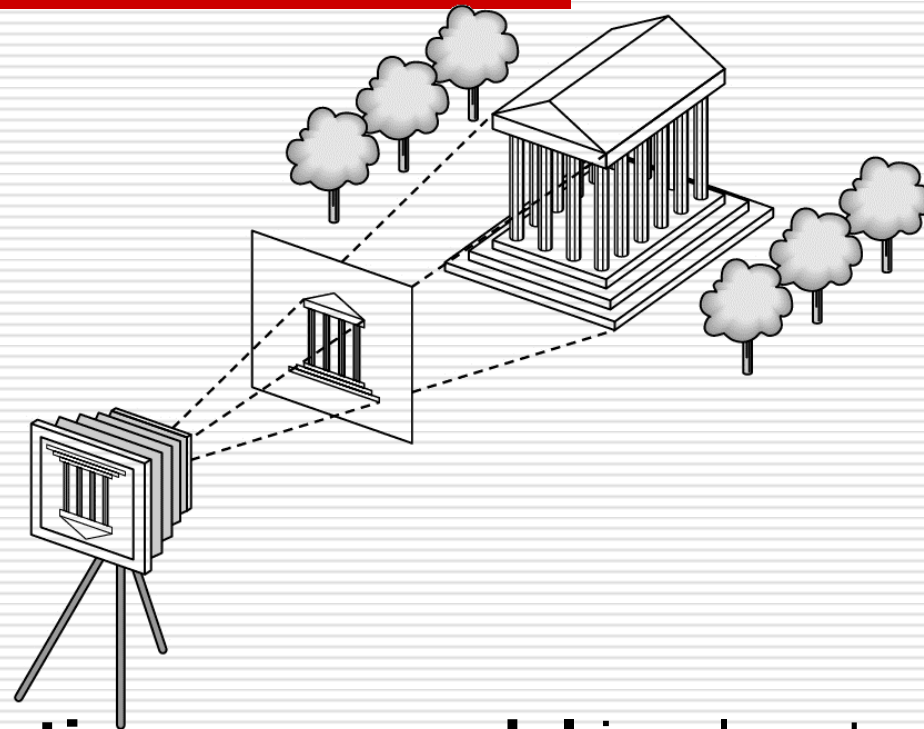
- A rotation by θ about an arbitrary axis can be decomposed into the concatenation of rotations about the x , y , and z axes
 - $\theta_x, \theta_y, \theta_z$ are called the Euler angles

$$R(\theta) = R_z(\theta_z) \bullet R_y(\theta_y) \bullet R_x(\theta_x)$$



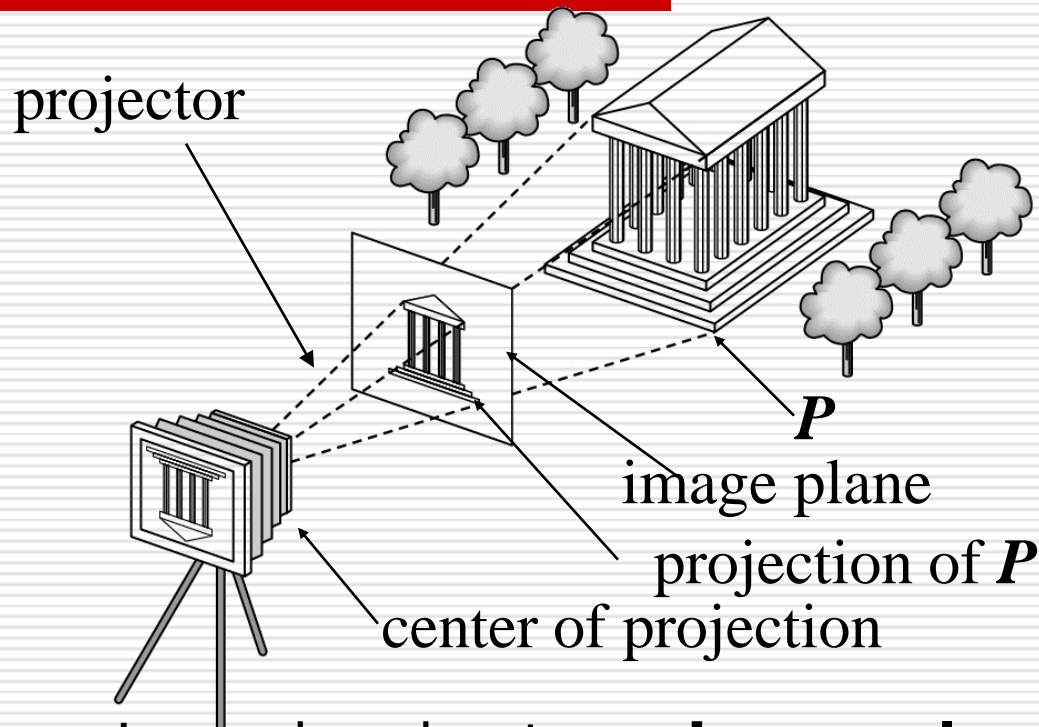
- Note that rotations do not commute
 - We can use rotations in another order but with different angles.

3D Synthetic Camera Model



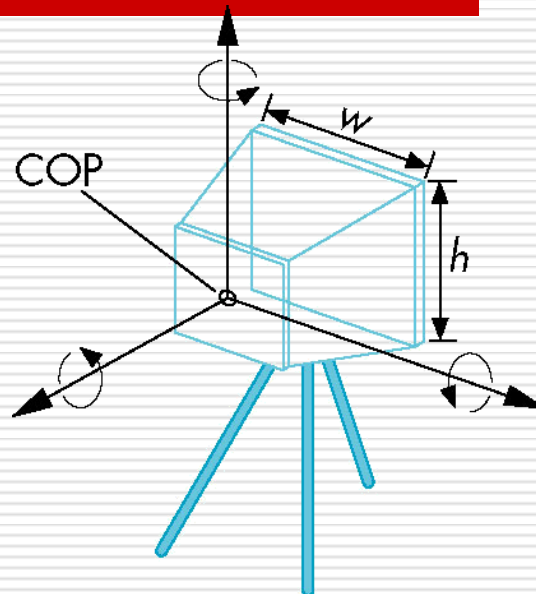
- The **synthetic camera model** involves two components, specified *independently*:
 - objects (a.k.a **geometry**)
 - viewer (a.k.a **camera**)
-

Imaging with the Synthetic Camera



- The image is rendered onto an **image plane** or **project plane** (usually in front of the camera).
- **Projectors** emanate from the **center of projection** (COP) at the center of the lens (or pinhole).
- The image of an object point P is at the intersection of the projector through P and the image plane.

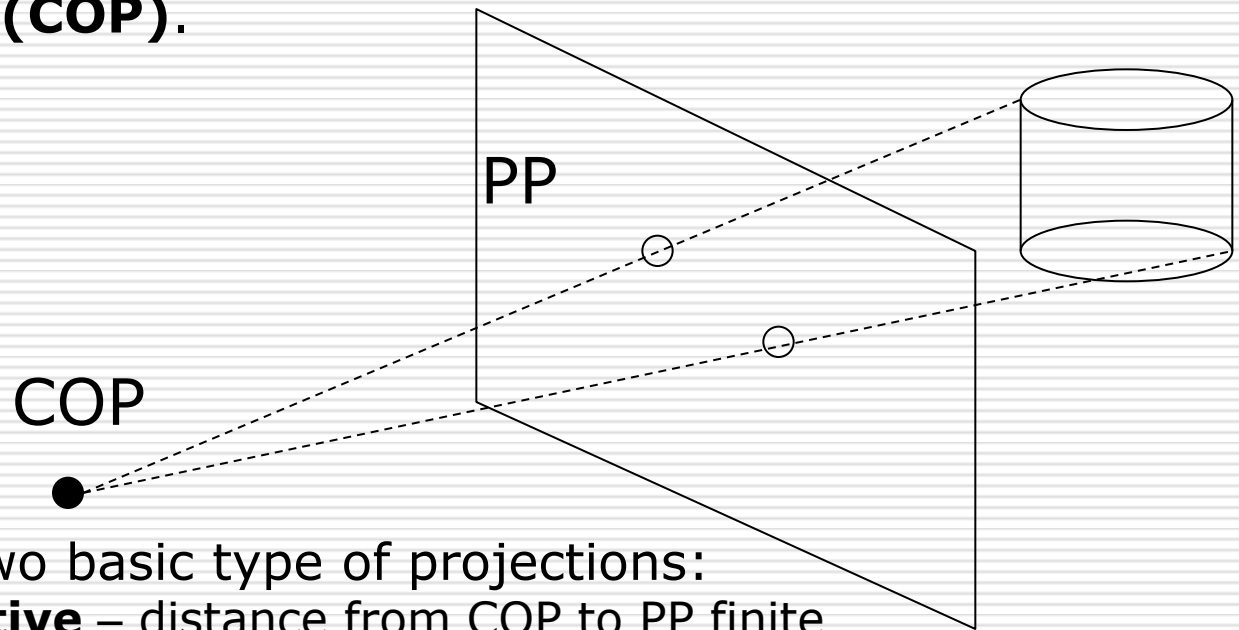
Specifying a Viewer



- Camera specification requires four kinds of parameters:
 - *Position*: the COP.
 - *Orientation*: rotations about axes with origin at the COP.
 - *Focal length*: determines the size of the image on the film plane, or the **field of view**.
 - *Film plane*: its width and height, and possibly orientation.

Projections

- **Projections** transform points in n -space to m -space, where $m < n$.
- In 3D, we map points from 3-space to the **projection plane (PP)** along projectors emanating from the **center of projection (COP)**.

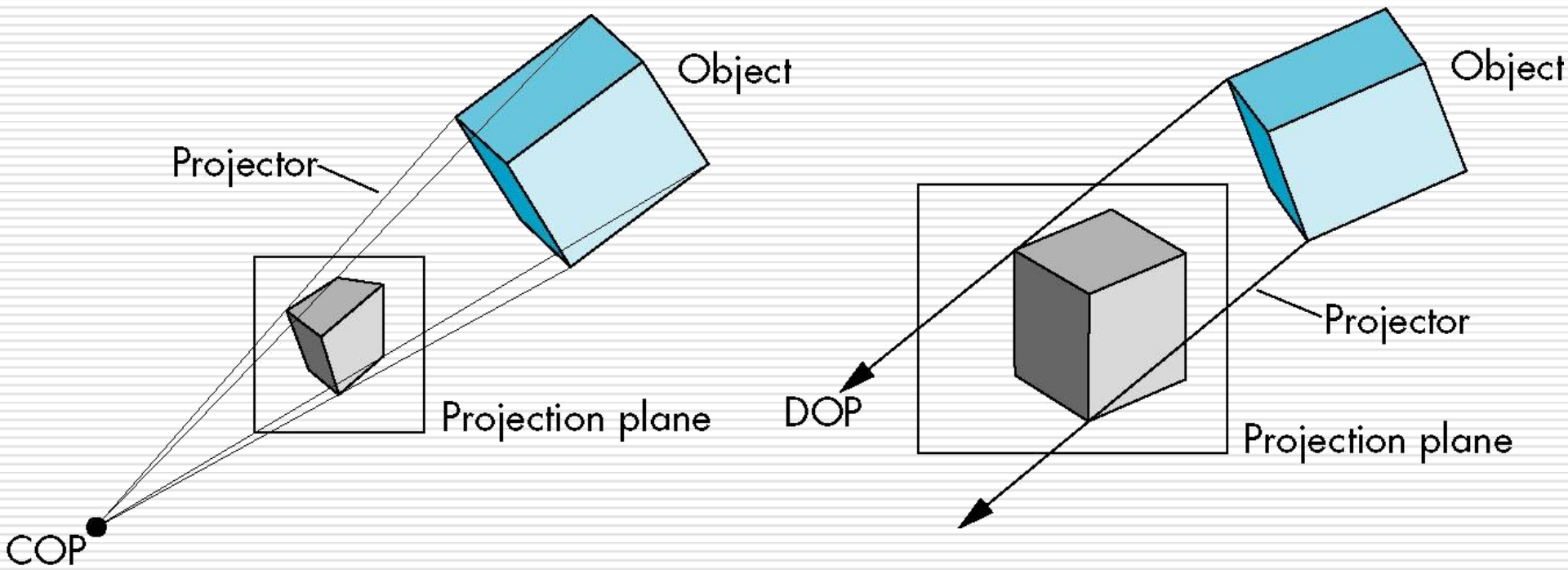


- There are two basic type of projections:
 - **Perspective** – distance from COP to PP finite
 - **Parallel** – distance from COP to PP infinite

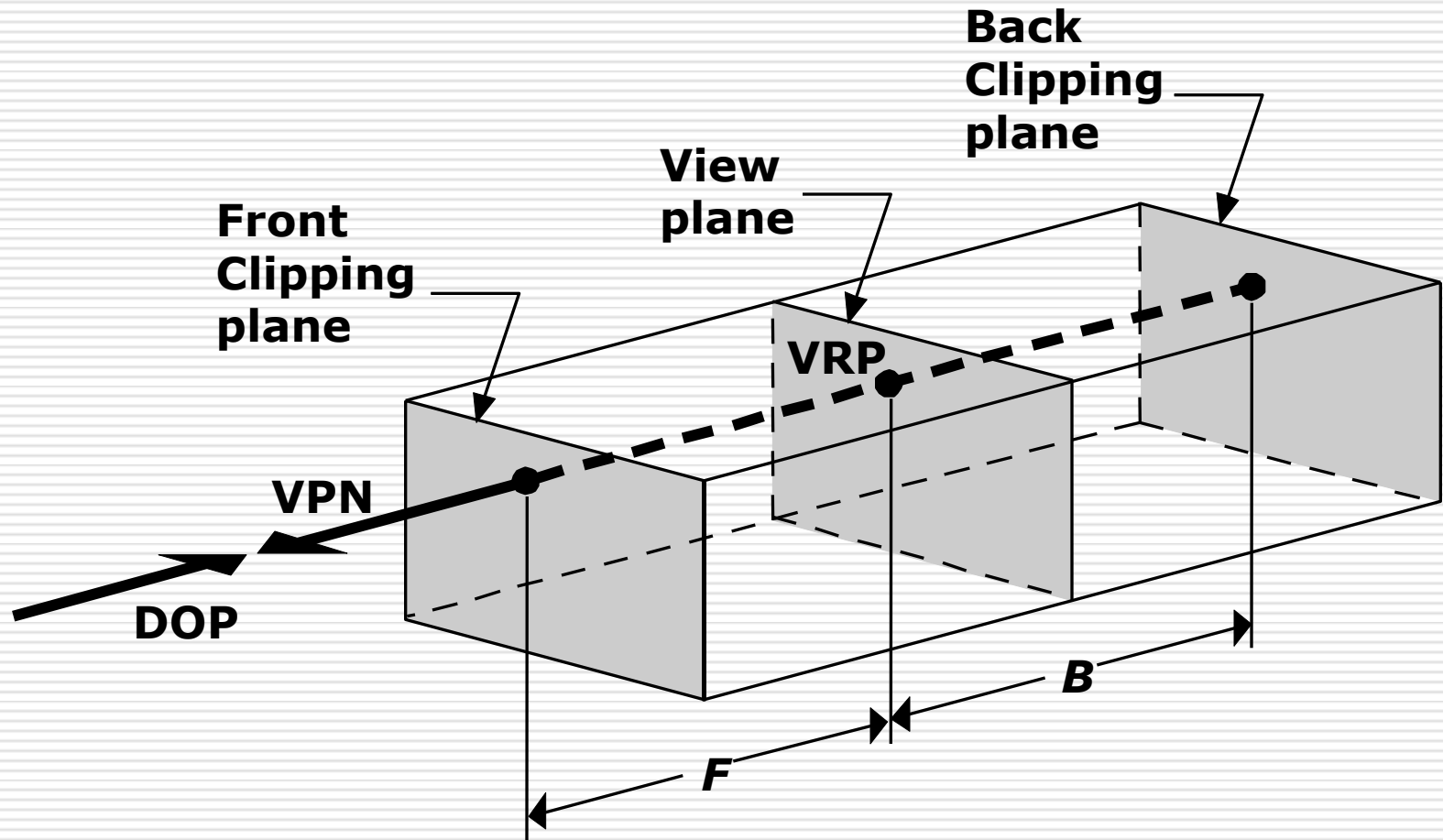
Perspective vs. Parallel Projections

- ❑ Computer graphics treats all projections the same and implements them with a single pipeline
- ❑ Classical viewing developed different techniques for drawing each type of projection
- ❑ Fundamental distinction is between parallel and perspective viewing even though mathematically parallel viewing is the limit of perspective viewing

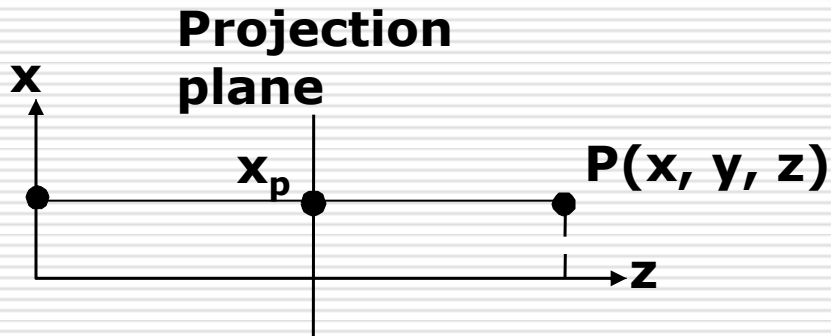
Perspective vs. Parallel Projections



Truncated View Volume for an Orthographic Parallel Projection

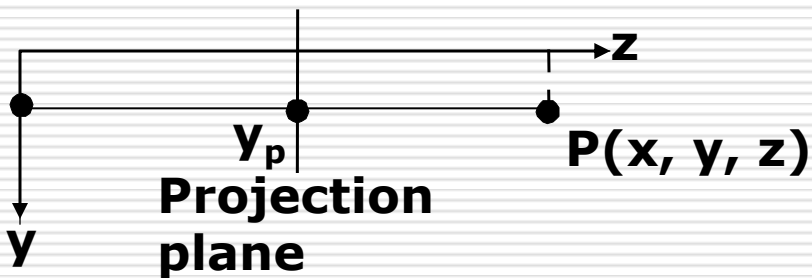


The Mathematics of Orthographic Parallel Projection



View along y axis

View along x axis

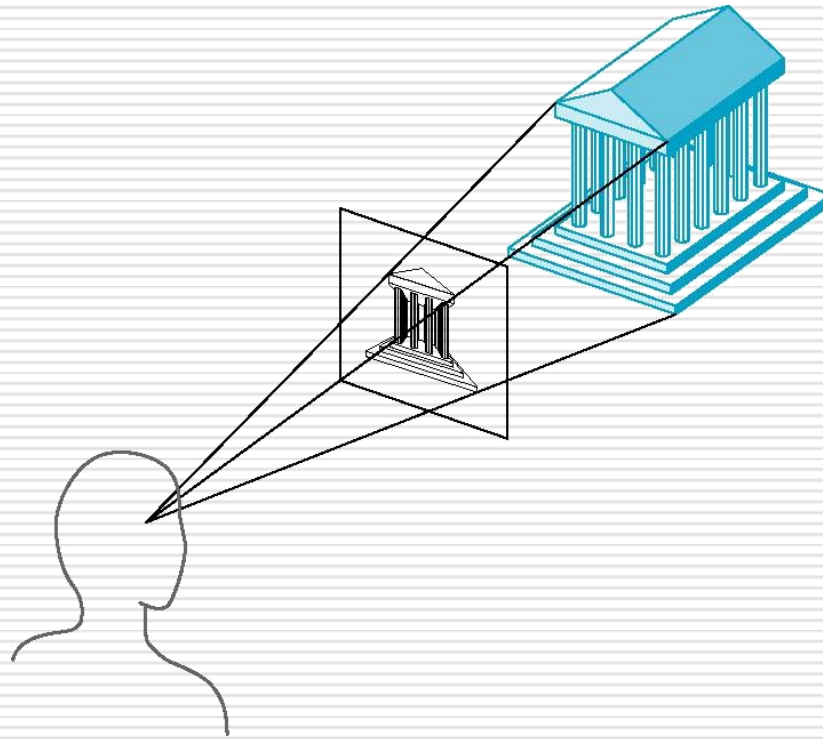


$$x_p = x; y_p = y; z_p = 0$$

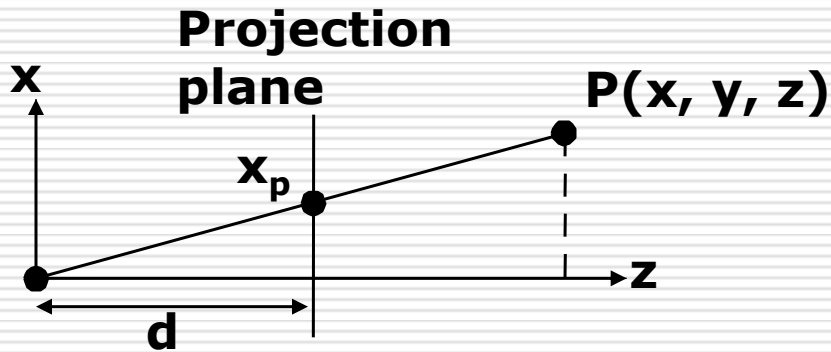
$$M_{ort} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Perspective Projection

Projectors converge at center of projection

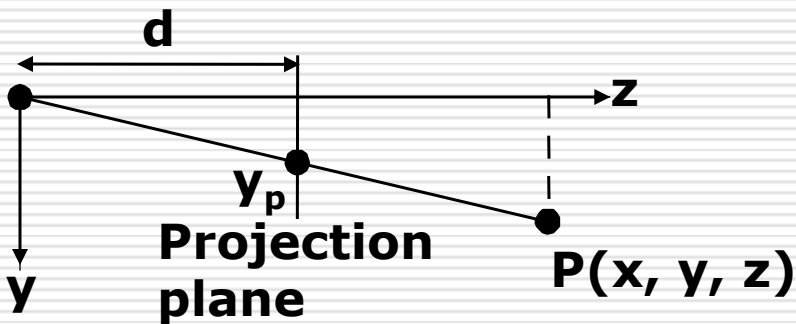


Perspective Projection (Pinhole Camera)



View along y axis

View along x axis



$$\frac{x_p}{d} = \frac{x}{z}; \frac{y_p}{d} = \frac{y}{z}$$

$$x_p = \frac{x}{z/d}; y_p = \frac{y}{z/d}$$

$$M_{per} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix}$$

Perspective Division

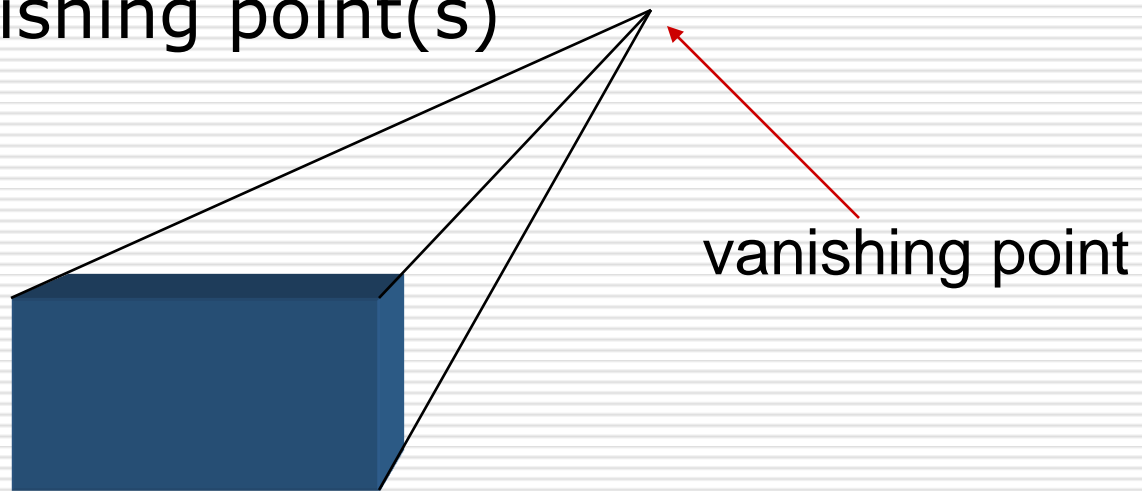
$$\begin{bmatrix} x_p \\ y_p \\ z_p \\ 1 \end{bmatrix} = \begin{bmatrix} X \\ Y \\ Z \\ W \end{bmatrix} = M_{per} \bullet P = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix} \bullet \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ \frac{z}{d} \end{bmatrix}$$

However $W \neq 1$, so we must divide by W to return from homogeneous coordinates

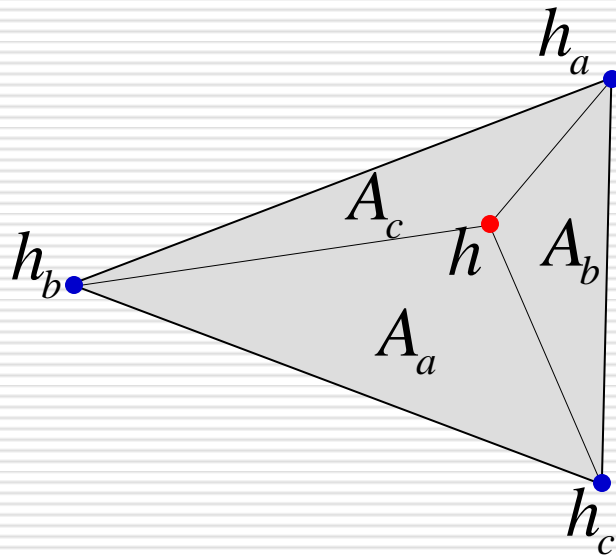
$$(x_p, y_p, z_p) = \left(\frac{X}{W}, \frac{Y}{W}, \frac{Z}{W} \right) = \left(\frac{x}{z/d}, \frac{y}{z/d}, d \right)$$

Vanishing Points

- Parallel lines (not parallel to the projection plan) on the object converge at a single point in the projection (the *vanishing point*)
- Drawing simple perspectives by hand uses these vanishing point(s)



Triangular Coordinate System



$$h = \frac{A_a}{A} h_a + \frac{A_b}{A} h_b + \frac{A_c}{A} h_c$$

$$\text{where } A = A_a + A_b + A_c$$

if ($A_a < 0 \parallel A_b < 0 \parallel A_c < 0$) than
the point is outside the triangle

Triangular Coordinate System - Application

- Terrain following
 - Interpolating the height of arbitrary point within the triangle
- Hit test
 - Intersection of a ray from camera to a screen position with a triangle
- Ray cast
 - Intersection of a ray with a triangle
- Collision detection
 - Intersection

Intersection

- Ray cast
- Containment test

Ray Cast – The Ray

- ❑ Cast a ray to calculate the intersection of the ray with models
- ❑ Use parametric equation for a ray

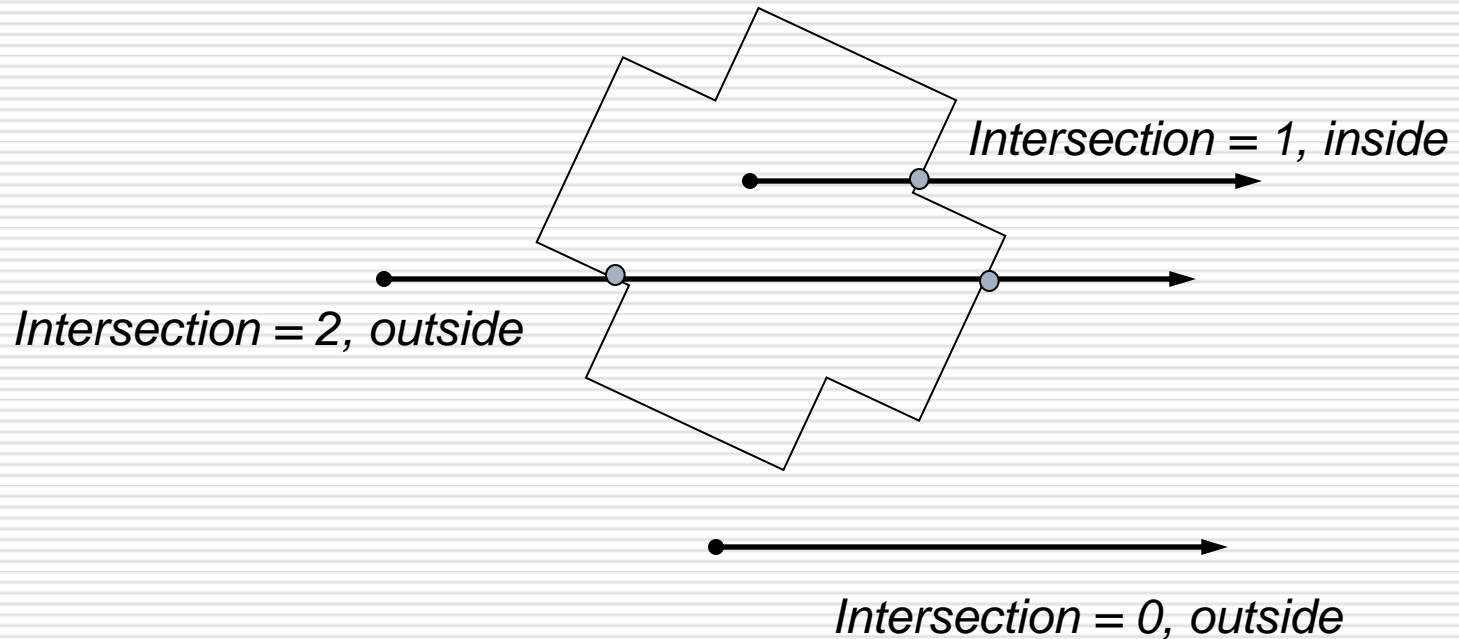
$$\begin{cases} x = x_0 + (x_1 - x_0)t, \\ y = y_0 + (y_1 - y_0)t, \\ z = z_0 + (z_1 - z_0)t, \quad t \geq 0 \end{cases}$$

- ❑ When $t = 0$, the ray is on the start point (x_0, y_0, z_0)
- ❑ Only the $t \geq 0$ is the answer candidate
- ❑ The smallest positive t is the answer

Ray Cast – The Plane

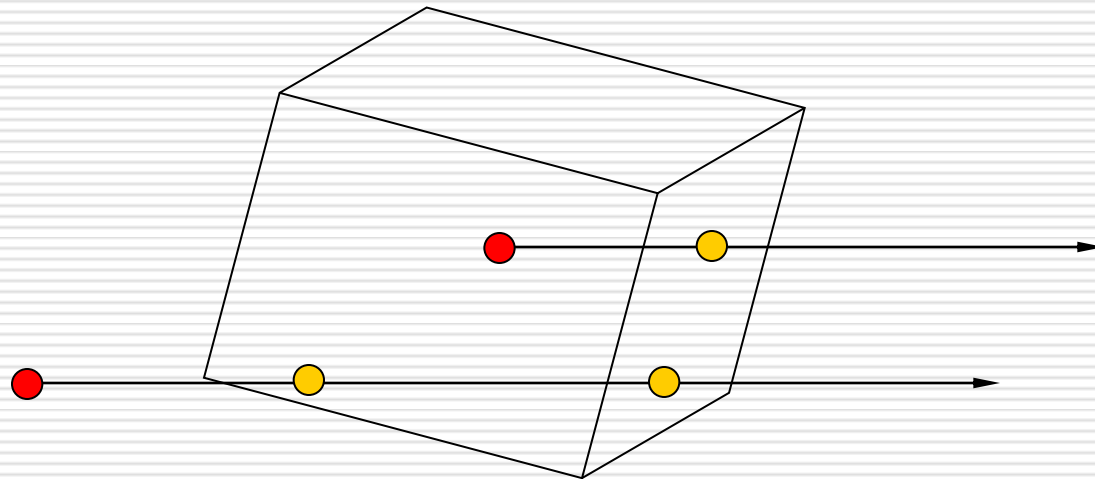
- ❑ Each triangle in the 3D models has its plane equation.
- ❑ Use $Ax + By + Cz + D = 0$ as the plane equation.
- ❑ (A, B, C) is the plane normal vector.
- ❑ $|D|$ is the distance of the plane to origin.
- ❑ Substitute the ray equation into the plane.
- ❑ Solve the t to find the intersect point.
- ❑ Check the intersect point within the triangle or not by using “Triangle Area Test”.

2D Containment Test



- if the no. of intersection is odd, the point is inside, otherwise, is outside

3D Containment Test

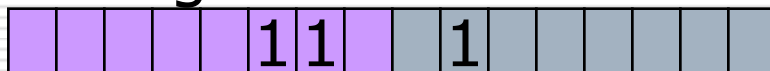


- if the no. of intersection is odd, the point is inside, otherwise, is outside

Fixed Point Arithmetic

- Fixed point arithmetic: n bits (signed) integer
 - Example : $n = 16$ gives range $-32768 \leq \tilde{a} \leq 32767$
 - We can use fixed scale to get the decimals.

8 integer bits



8 fractional bits

$$a = \tilde{a} \cdot 2^{-8}$$

$$\tilde{a} = 1600 \rightarrow a = 6.25$$

Fixed Point Arithmetic

- Multiplication requires rescaling

$$e = a \cdot c = \tilde{a} \cdot 2^{-8} \cdot \tilde{c} \cdot 2^{-8} = \tilde{e} \cdot 2^{-8}$$

$$\Rightarrow \tilde{e} = (\tilde{a} \cdot \tilde{c}) \cdot 2^{-8}$$

- Addition just like normal

$$e = a + c = \tilde{a} \cdot 2^{-8} + \tilde{c} \cdot 2^{-8} = (\tilde{a} + \tilde{c}) \cdot 2^{-8}$$

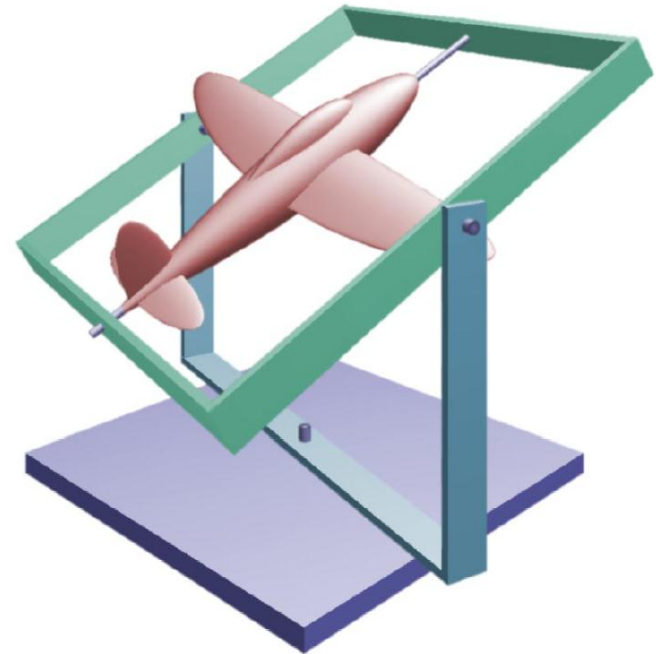
$$\Rightarrow \tilde{e} = \tilde{a} + \tilde{c}$$

Fixed Point Arithmetic - Application

- Compression for floating-point real numbers
 - 4 bytes reduced to 2 bytes
 - Lost some accuracy but affordable
- Network data transfer
- Software 3D rendering
(without hardware-assistant)

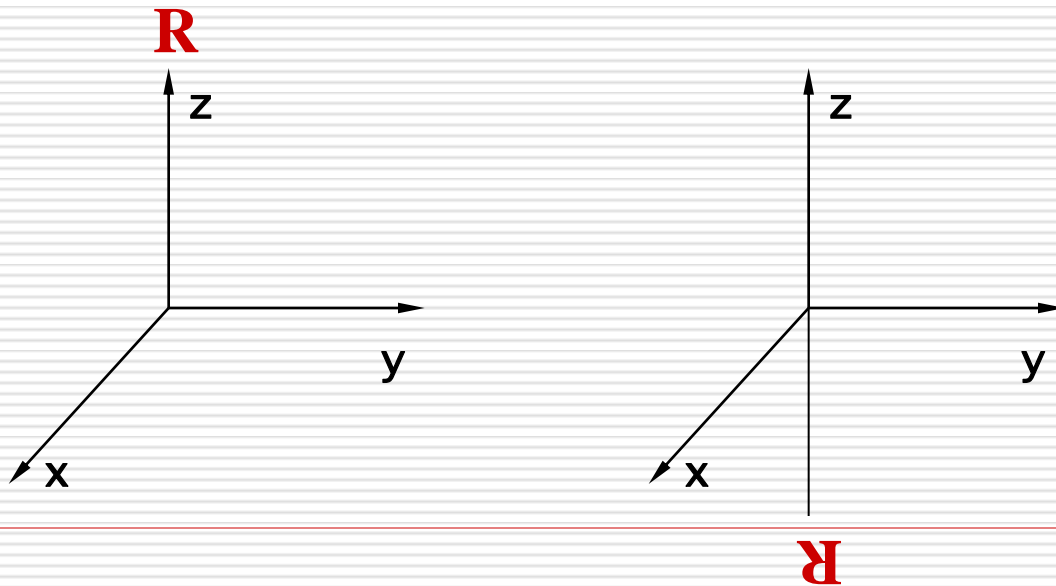
Euler Angles

- An Euler angle is a rotation about a single axis.
- A rotation is described as a sequence of rotations about three mutually orthogonal coordinates axes fixed in space
 - X-roll, Y-roll, Z-roll
- There are 6 possible ways to define a rotation.
 - 3!



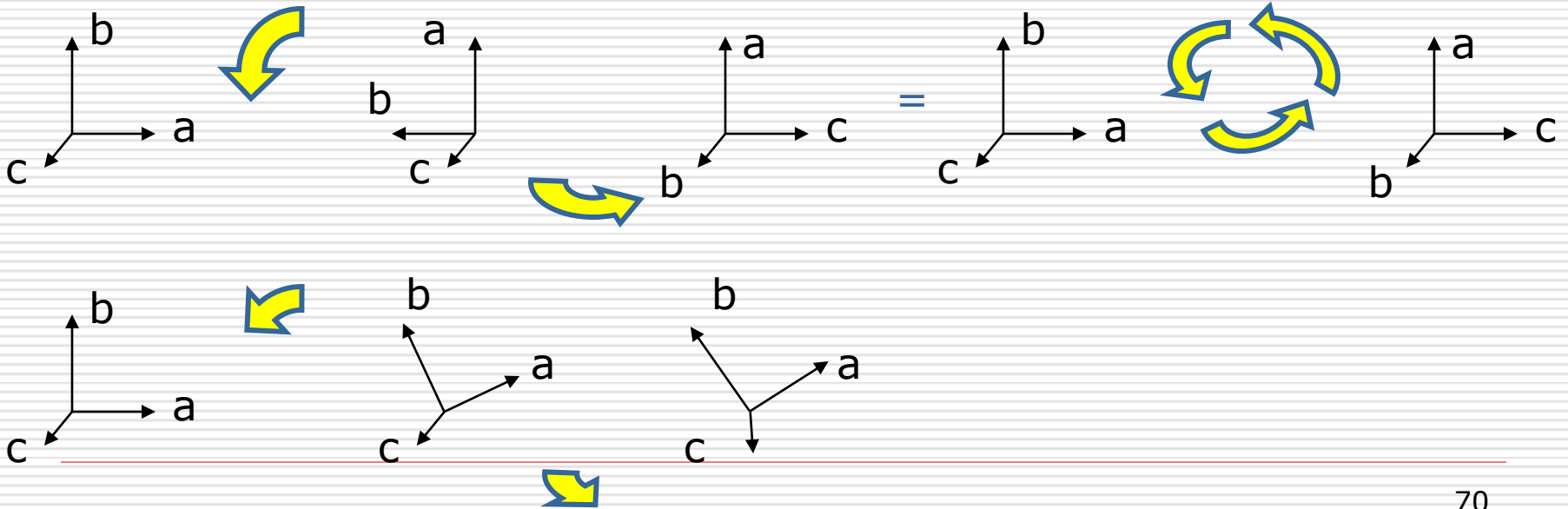
Euler Angles & Interpolation

- ❑ Interpolation happening on each angle
- ❑ Multiple routes for interpolation
- ❑ More keys for constrains



Interpolating Euler Angles

- Natural orientation representation:
 - 3 angles for 3 degrees of freedom
- Unnatural interpolation:
 - A rotation of 90° first around Z and then around Y = 120° around $(1, 1, 1)$.
 - But 30° around Z then Y differs from 40° around $(1, 1, 1)$.



Smooth Rotation

- From a practical standpoint, we often want to use transformations to move and reorient an object smoothly
 - Problem: find a sequence of model-view matrices $\mathbf{M}_0, \mathbf{M}_1, \dots, \mathbf{M}_n$ so that when they are applied successively to one or more objects we see a smooth transition
- For orientating an object, we can use the fact that every rotation corresponds to part of a great circle on a sphere
 - Find the axis of rotation and angle
 - Virtual trackball

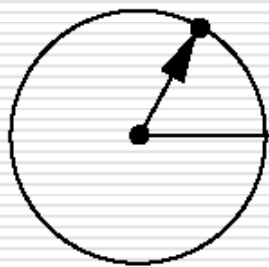
Incremental Rotation

- Consider the two approaches
 - For a sequence of rotation matrices $\mathbf{R}_0, \mathbf{R}_1, \dots, \mathbf{R}_n$, find the Euler angles for each and use $\mathbf{R}_i = \mathbf{R}_{iz} \mathbf{R}_{iy} \mathbf{R}_{ix}$
 - Not very efficient
 - Use the final positions to determine the axis and angle of rotation, then increment only the angle
- Quaternions can be more efficient than either

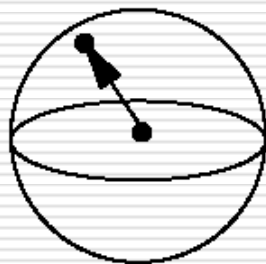
Solution:

Quaternion Interpolation

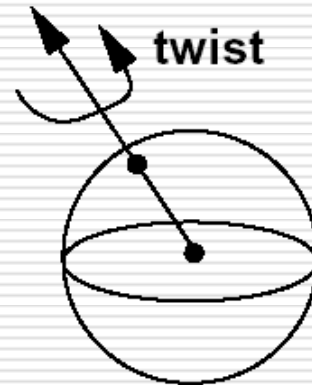
- ❑ Interpolate orientation on the unit sphere
- ❑ By analogy: 1-, 2-, 3-DOF rotations as constrained points on 1-, 2-, 3-spheres



1-DOF



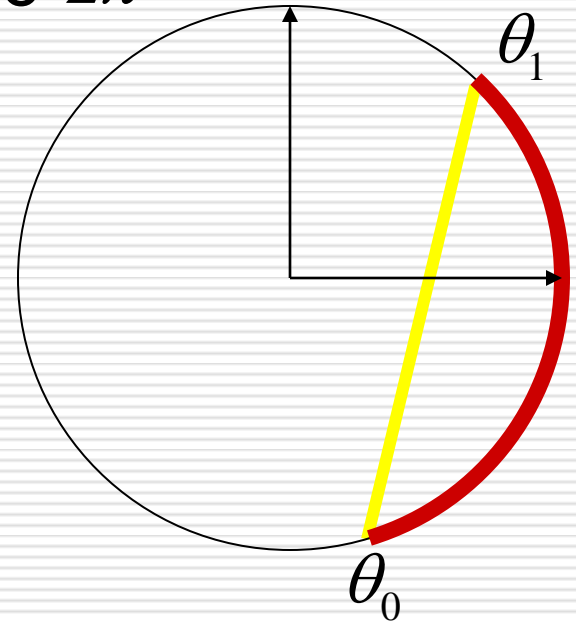
2-DOF



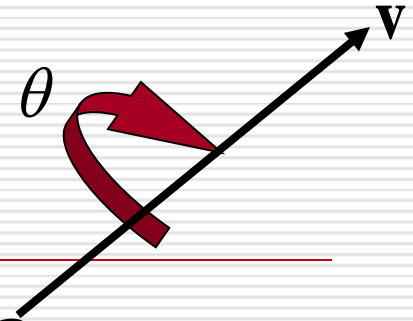
3-DOF

1D-Sphere and Complex Plane

- Interpolate orientation in 2D
- 1 angle
 - but messy because modulo 2π
- Use interpolation in (complex) 2D plane
- Orientation = complex argument of the number



Quaternions



- Quaternions are unit vectors on 3-sphere (in 4D)
- Right-hand rotation of θ radians about \mathbf{v} is $q = [\cos(\theta/2), \sin(\theta/2) \bullet \mathbf{v}]$
 - often noted $[\mathbf{w}, \mathbf{v}]$
- Requires one real and three imaginary components $\mathbf{i}, \mathbf{j}, \mathbf{k}$
 - $q = q_0 + q_1\mathbf{i} + q_2\mathbf{j} + q_3\mathbf{k} = [\mathbf{w}, \mathbf{v}]; \mathbf{w} = q_0, \mathbf{v} = (q_1, q_2, q_3)$
 - where $\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = \mathbf{ijk} = -1$
 - \mathbf{w} is called **scalar** and \mathbf{v} is called **vector**

Basic Operations Using Quaternions

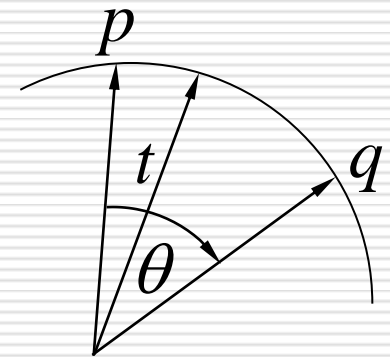
- Addition $q + q' = [\mathbf{w} + \mathbf{w}', \mathbf{v} + \mathbf{v}']$
- Multiplication $q \bullet q' = [\mathbf{w} \bullet \mathbf{w}' - \mathbf{v} \bullet \mathbf{v}', \mathbf{v} \times \mathbf{v}' + \mathbf{w} \bullet \mathbf{v}' + \mathbf{w}' \bullet \mathbf{v}]$
- Conjugate $q^* = [\mathbf{w}, -\mathbf{v}]$
- Length $|q| = (\mathbf{w}^2 + |\mathbf{v}|^2)^{1/2}$
- Norm $N(q) = |q|^2 = \mathbf{w}^2 + |\mathbf{v}|^2$
- Inverse $q^{-1} = q^* / |q|^2 = q^* / N(q)$
- Unit Quaternion
 - q is a unit quaternion if $|q| = 1$ and then $q^{-1} = q^*$
- Identity
 - $[1, (0, 0, 0)]$ (when involving multiplication)
 - $[0, (0, 0, 0)]$ (when involving addition)

SLERP-Spherical Linear intERPolation

- Interpolate between two quaternion rotations along the shortest arc.

- $$\text{SLERP}(p, q, t) = \frac{p \bullet \sin((1-t) \bullet \theta) + q \bullet \sin(t \bullet \theta)}{\sin(\theta)}$$

- where $\cos(\theta) = \mathbf{w}_p \bullet \mathbf{w}_q + \mathbf{v}_p \bullet \mathbf{v}_q$



- If two orientations are too close, use linear interpolation to avoid any divisions by zero.
-

Parametric Polynomial Curves

- We will use parametric curves where the functions are all polynomials in the parameter.

$$x(u) = \sum_{k=0}^n a_k u^k$$

$$y(u) = \sum_{k=0}^n b_k u^k$$

- Advantages:
 - easy (and efficient) to compute
 - infinitely differentiable

Parametric Cubic Curves

- Fix $n = 3$
- The cubic polynomials that define a curve segment $Q(t) = [x(t) \ y(t) \ z(t)]^T$ are of the form

$$x(t) = a_x t^3 + b_x t^2 + c_x t + d_x,$$

$$y(t) = a_y t^3 + b_y t^2 + c_y t + d_y,$$

$$z(t) = a_z t^3 + b_z t^2 + c_z t + d_z, \quad 0 \leq t \leq 1.$$

Parametric Cubic Curves

- The curve segment can be rewrite as

$$Q(t) = [x(t) \quad y(t) \quad z(t)]^T = C \bullet T$$

- where $T = [t^3 \quad t^2 \quad t \quad 1]^T$

$$C = \begin{bmatrix} a_x & b_x & c_x & d_x \\ a_y & b_y & c_y & d_y \\ a_z & b_z & c_z & d_z \end{bmatrix}$$

Tangent Vector

$$\begin{aligned}\frac{d}{dt}Q(t) &= Q'(t) = \left[\frac{d}{dt}x(t) \quad \frac{d}{dt}y(t) \quad \frac{d}{dt}z(t) \right]^T \\ &= \frac{d}{dt}C \bullet T = C \bullet \left[3t^2 \quad 2t \quad 1 \quad 0 \right]^T \\ &= \left[3a_x t^2 + 2b_x t + c_x \quad 3a_y t^2 + 2b_y t + c_y \quad 3a_z t^2 + 2b_z t + c_z \right]^T\end{aligned}$$

Three Types of Parametric Cubic Curves

□ Hermite Curves

- defined by two **endpoints** and two endpoint **tangent vectors**

□ Bézier Curves

- defined by two **endpoints** and two **control points** which control the endpoint' **tangent vectors**

□ Splines

- defined by four **control points**

Parametric Cubic Curves

- $Q(t) = C \bullet T$
- rewrite the coefficient matrix as $C = G \bullet M$
 - where M is a 4x4 **basis matrix**, G is called the **geometry matrix**

■ so

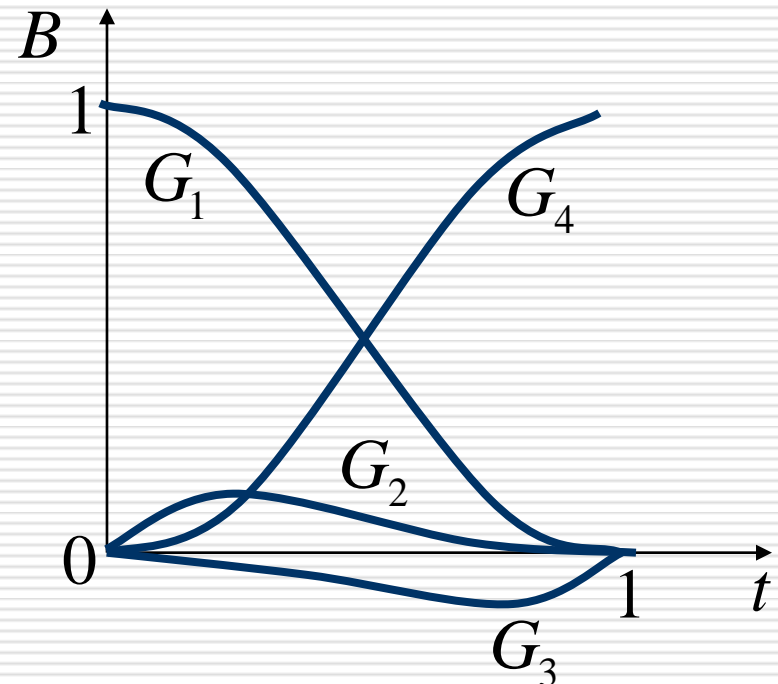
$$Q(t) = \begin{bmatrix} x(t) \\ y(t) \\ z(t) \end{bmatrix} = \begin{bmatrix} G_1 & G_2 & G_3 & G_4 \end{bmatrix} \begin{bmatrix} m_{11} & m_{21} & m_{31} & m_{41} \\ m_{12} & m_{22} & m_{32} & m_{42} \\ m_{13} & m_{23} & m_{33} & m_{43} \\ m_{14} & m_{24} & m_{34} & m_{44} \end{bmatrix} \begin{bmatrix}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

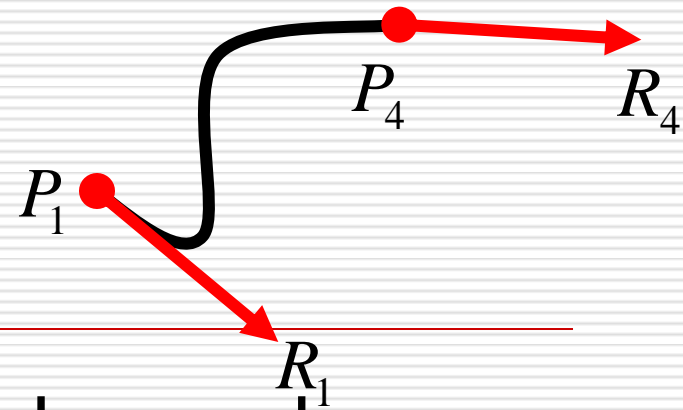
4 endpoints or tangent vectors

Parametric Cubic Curves

□ $Q(t) = G \bullet M \bullet T = G \bullet B$

where $B = M \bullet T$ is called the **blending functions**

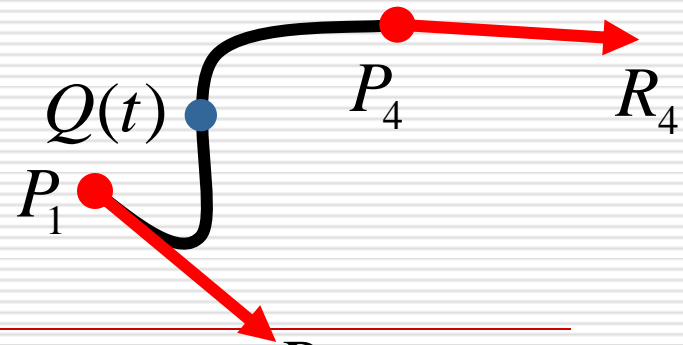




Hermite Curves

- Given the endpoints P_1 and P_4 and tangent vectors at them R_1 and R_4
- What is
 - Hermite basis matrix M_H
 - Hermite geometry vector G_H
 - Hermite blending functions B_H
- by definition

$$G_H = [P_1 \quad P_4 \quad R_1 \quad R_4]$$



Hermite Curves

□ since $Q(0) = P_1 = G_H \cdot M_H \cdot [0 \ 0 \ 0 \ 1]^T R_1$

$$Q(1) = P_4 = G_H \cdot M_H \cdot [1 \ 1 \ 1 \ 1]^T$$

$$Q'(0) = R_1 = G_H \cdot M_H \cdot [0 \ 0 \ 1 \ 0]^T$$

$$Q'(1) = R_4 = G_H \cdot M_H \cdot [3 \ 2 \ 1 \ 0]^T$$

$$G_H = [P_1 \ P_4 \ R_1 \ R_4] = G_H \cdot M_H \cdot \begin{bmatrix} 0 & 1 & 0 & 3 \\ 0 & 1 & 0 & 2 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \end{bmatrix}$$

Hermite Curves

□ so

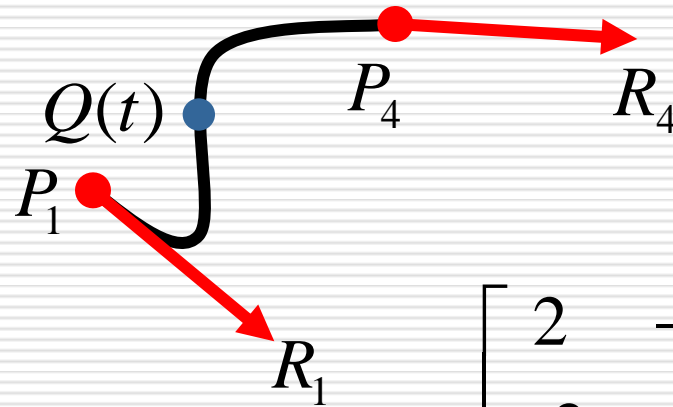
$$M_H = \begin{bmatrix} 0 & 1 & 0 & 3 \\ 0 & 1 & 0 & 2 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \end{bmatrix}^{-1} = \begin{bmatrix} 2 & -3 & 0 & 1 \\ -2 & 3 & 0 & 0 \\ 1 & -2 & 1 & 0 \\ 1 & -1 & 0 & 0 \end{bmatrix}$$

□ and $Q(t) = G_H \bullet M_H \bullet T = G_H \bullet B_H$

$$B_H = \begin{bmatrix} 2t^3 - 3t^2 + 1 & -2t^3 + 3t^2 & t^3 - 2t^2 + t & t^3 - t^2 \end{bmatrix}^T$$

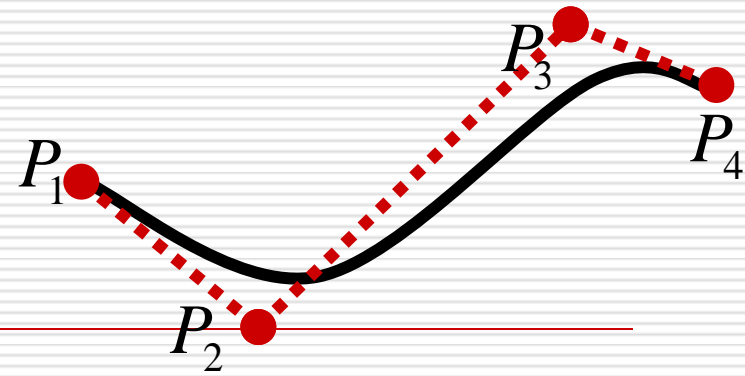
Computing a point

- Given two endpoints P_1 and P_4 and two tangent vectors at them R_1 and R_4



so

$$Q(t) = [P_1 \quad P_4 \quad R_1 \quad R_4] \begin{bmatrix} 2 & -3 & 0 & 1 \\ -2 & 3 & 0 & 0 \\ 1 & -2 & 1 & 0 \\ 1 & -1 & 0 & 0 \end{bmatrix} \begin{bmatrix} t^3 \\ t^2 \\ t \\ 1 \end{bmatrix}$$



Bézier Curves

- Given the endpoints P_1 and P_4 and two control points P_2 and P_3 which determine the endpoints' tangent vectors, such that

$$R_1 = Q'(0) = 3(P_2 - P_1)$$

$$R_4 = Q'(1) = 3(P_4 - P_3)$$

- What is

- **Bézier basis matrix** M_B
- **Bézier geometry vector** G_B
- **Bézier blending functions** B_B

Bézier Curves

□ by definition $G_B = [P_1 \ P_2 \ P_3 \ P_4]$

□ then $G_H = [P_1 \ P_4 \ R_1 \ R_4]$

$$= [P_1 \ P_2 \ P_3 \ P_4] \begin{bmatrix} 1 & 0 & -3 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & -3 \\ 0 & 1 & 0 & 3 \end{bmatrix} = G_B \bullet M_{HB}$$

□ so $Q(t) = G_H \bullet M_H \bullet T = (G_B \bullet M_{HB}) \bullet M_H \bullet T$

$$= G_B \bullet (M_{HB} \bullet M_H) \bullet T = G_B \bullet M_B \bullet T$$

Bézier Curves

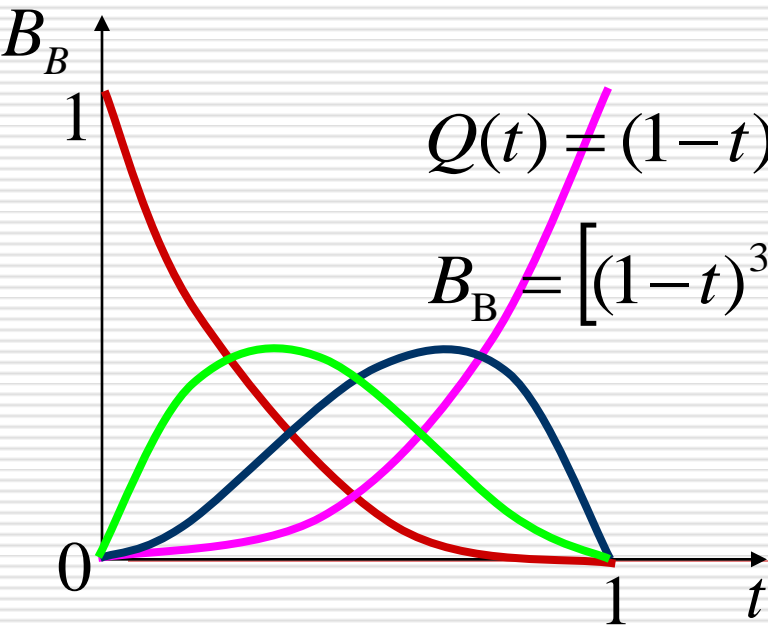
□ and

$$M_B = M_{HB} \bullet M_H = \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

$$Q(t) = (1-t)^3 P_1 + 3t(1-t)^2 P_2 + 3t^2(1-t) P_3 + t^3 P_4$$

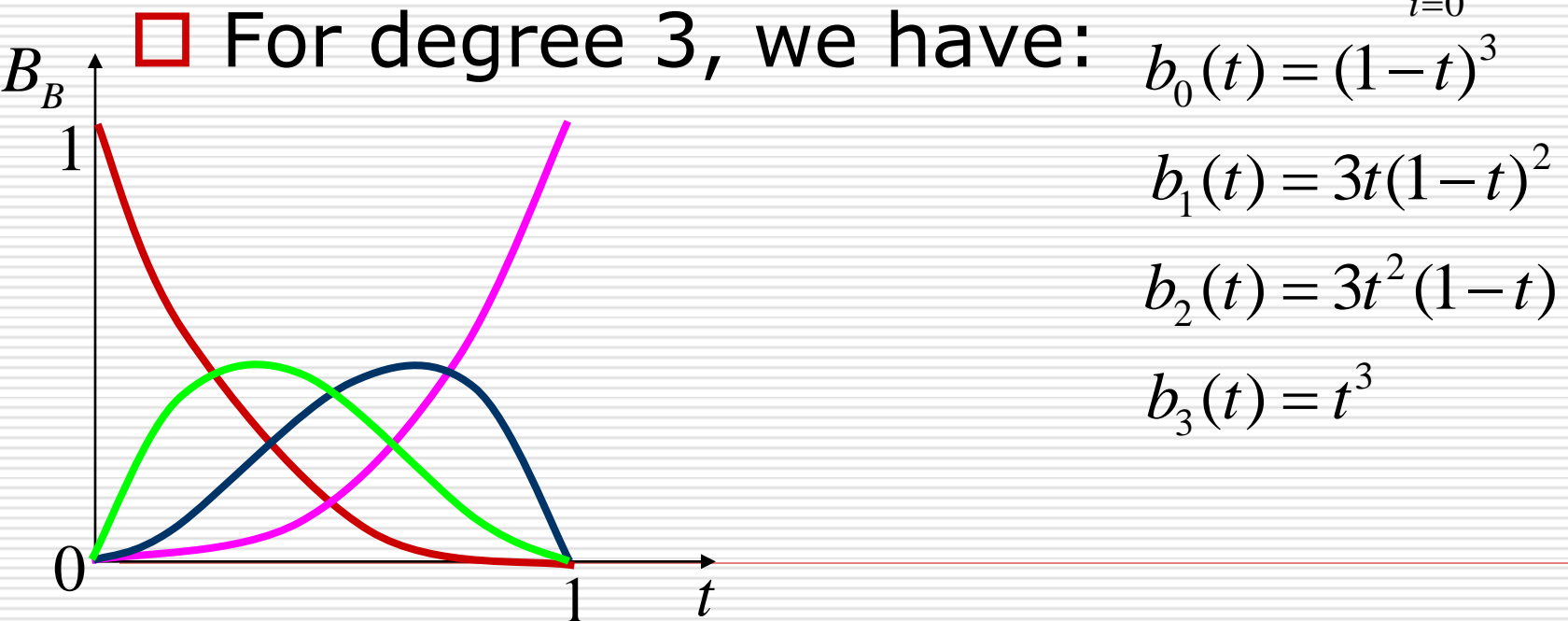
$$B_B = \left[(1-t)^3 \quad 3t(1-t)^2 \quad 3t^2(1-t) \quad t^3 \right]^T$$

↑
Bernstein polynomials



Bernstein Polynomials

- The coefficients of the control points are a set of functions called the **Bernstein polynomials**: $Q(t) = \sum_{i=0}^n b_i(t)P_i$



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

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

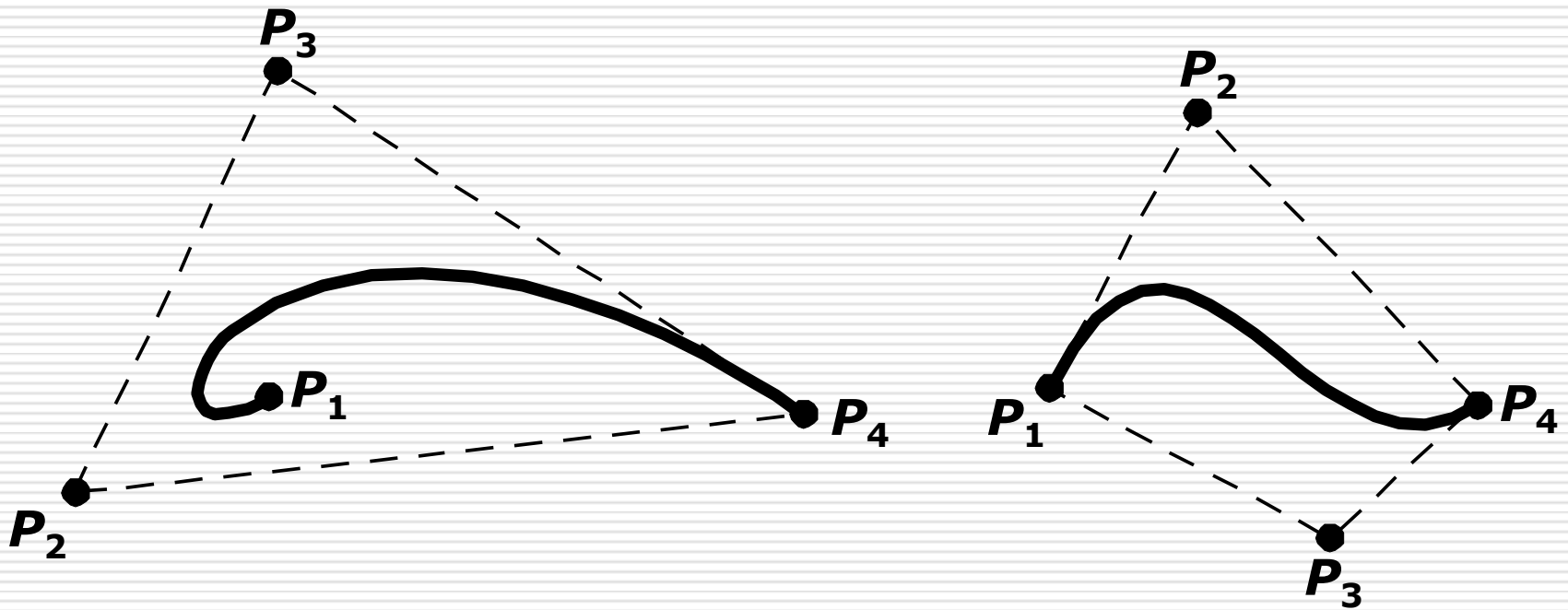
$$b_2(t) = 3t^2(1-t)$$

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

Bernstein Polynomials

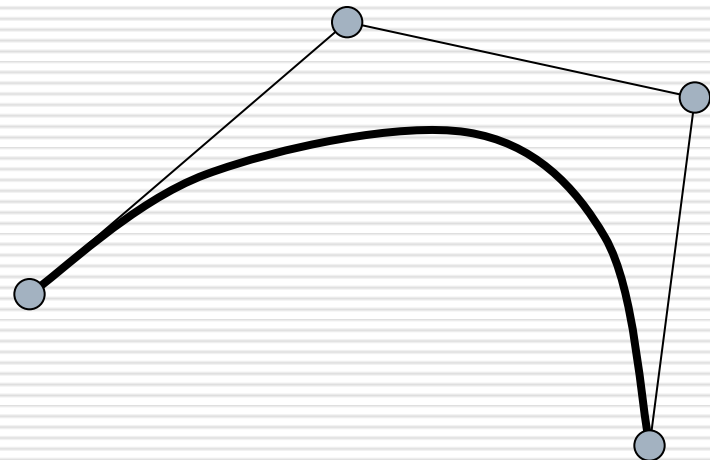
- Useful properties on the interval $[0,1]$:
 - each is between 0 and 1
 - sum of all four is exact 1
 - a.k.a., a “partition of unity”
- These together imply that the curve lies within the **convex hull** of its control points.

Convex Hull



Subdividing Bézier Curves

- $Q(t) = (1-t)^3 P_1 + 3t(1-t)^2 P_2 + 3t^2(1-t)P_3 + t^3 P_4$
- How to draw the curve ?
- How to convert it to be line-segments ?

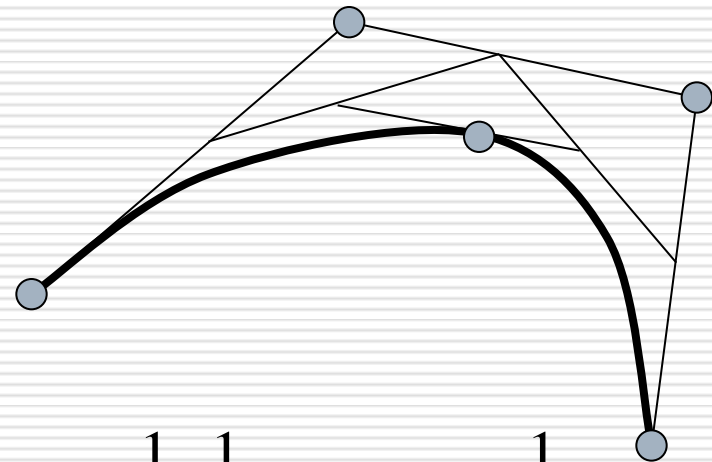


Subdividing Bézier Curves (de Casteljau's algorithm)

- $Q(t) = (1-t)^3 P_1 + 3t(1-t)^2 P_2 + 3t^2(1-t)P_3 + t^3 P_4$
- How to draw the curve ?
- How to convert it to be line-segments ?

$$Q\left(\frac{1}{2}\right) = \frac{1}{8}P_1 + \frac{3}{8}P_2 + \frac{3}{8}P_3 + \frac{1}{8}P_4$$

$$= \frac{1}{2}\left(\frac{1}{2}\left(\frac{1}{2}(P_1 + P_2) + \frac{1}{2}(P_2 + P_3)\right) + \frac{1}{2}\left(\frac{1}{2}(P_3 + P_4) + \frac{1}{2}(P_2 + P_3)\right)\right)$$



Display Bézier Curves

```
DisplayBezier(P1,P2,P3,P4)
```

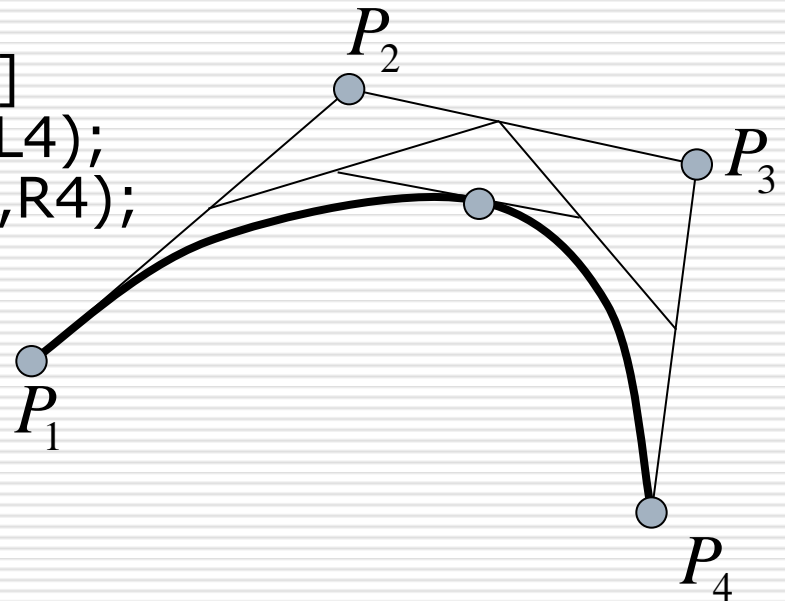
```
begin
```

```
  if (FlatEnough(P1,P2,P3,P4))  
    Line(P1,P4);
```

```
  else
```

```
    Subdivide(P[[]])=>L[[]],R[[]]  
    DisplayBezier(L1,L2,L3,L4);  
    DisplayBezier(R1,R2,R3,R4);
```

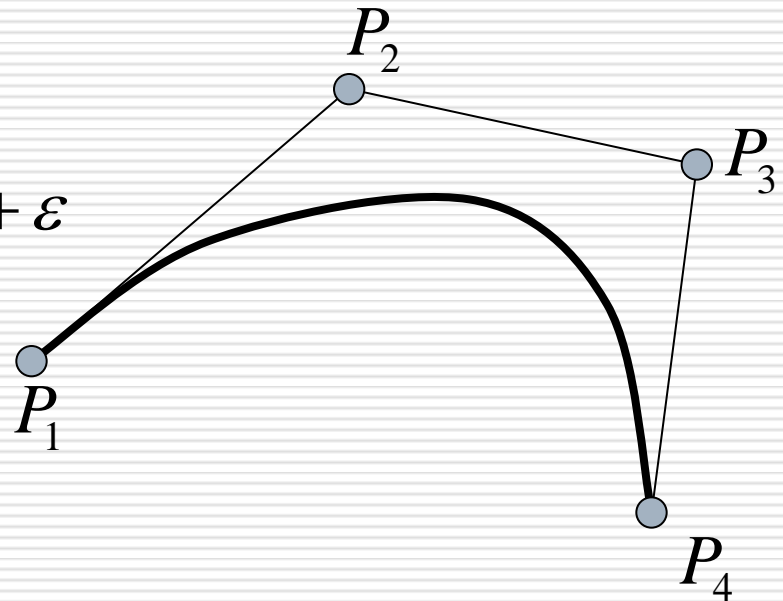
```
end;
```



Testing for Flatness

- Compare total length of control polygon to length of line connecting endpoints

$$\frac{|P_1 - P_2| + |P_2 - P_3| + |P_3 - P_4|}{|P_1 - P_4|} < 1 + \varepsilon$$

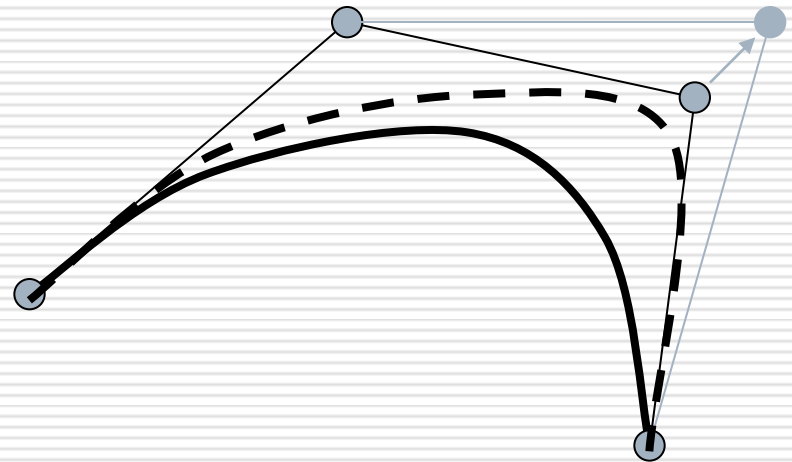


What do we want for a curve?

- Local control
- Interpolation
- Continuity

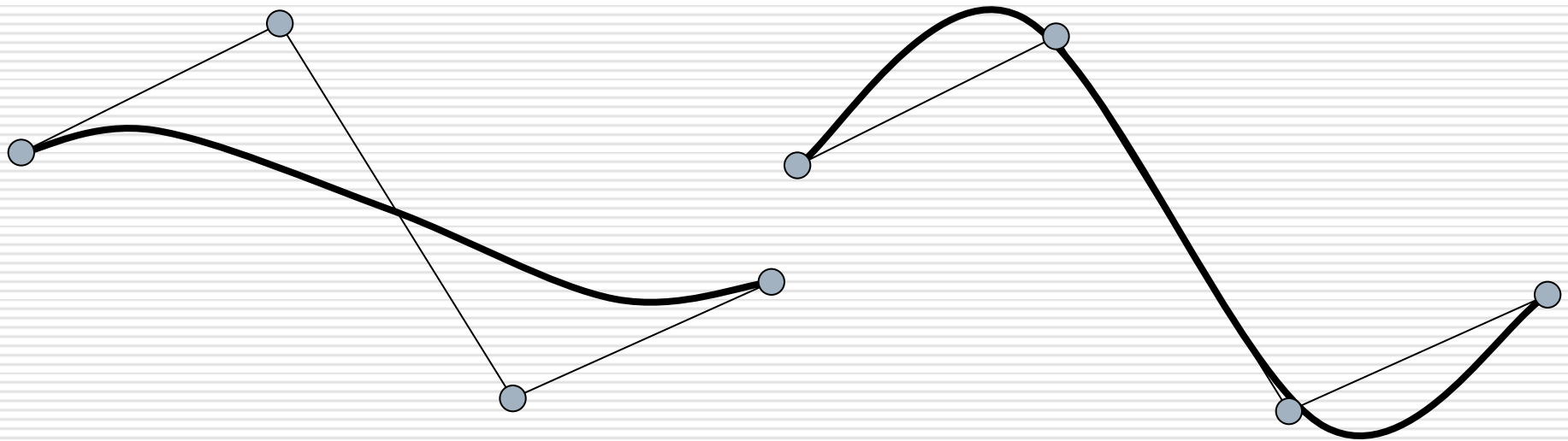
Local Control

- ❑ One problem with Bézier curve is that every control points affect every point on the curve (except for endpoints). Moving a single control point affects the whole curve.
- ❑ We'd like to have local control, that is, have each control point affect some well-defined neighborhood around that point.

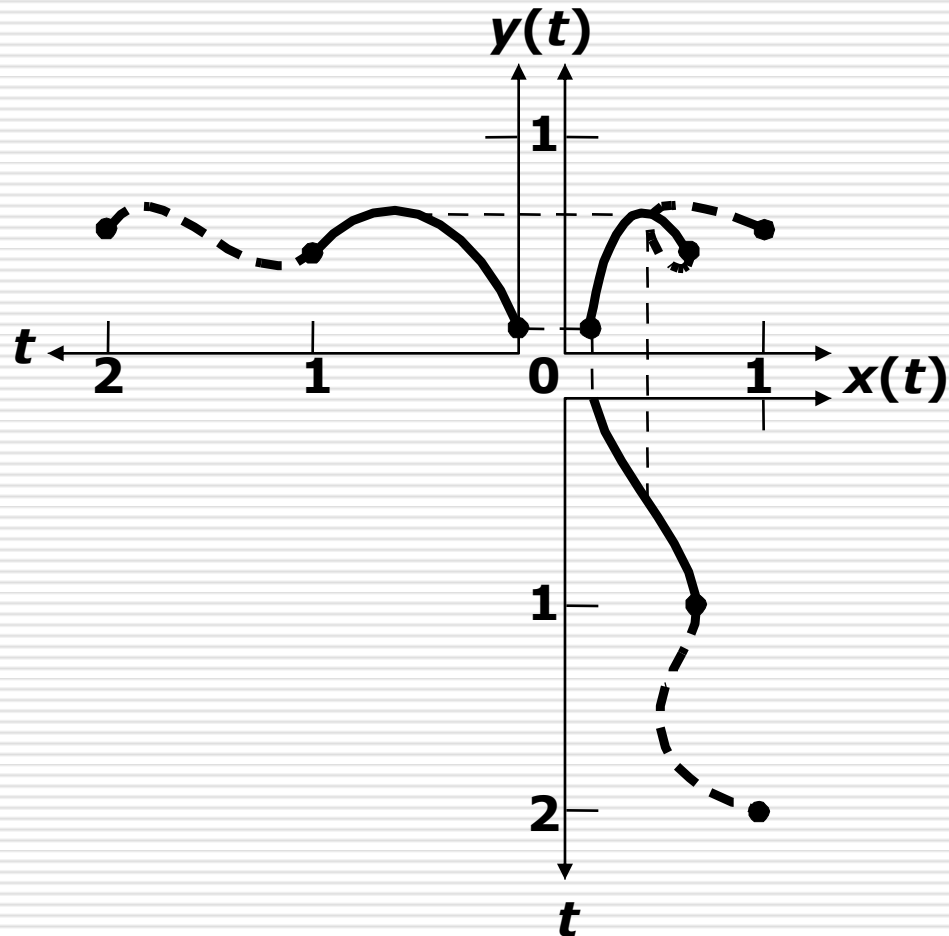


Interpolation

- Bézier curves are approximating. The curve does not necessarily pass through all the control points. We'd like to have a curve that is interpolating, that is, that always passes through every control points.



Continuity between Curve Segments



Continuity between Curve Segments

- G^0 geometric continuity
 - two curve segments join together

- G^1 geometric continuity
 - the directions (*but not necessarily the magnitudes*) of the two segments' tangent vectors are equal at a join point

Continuity between Curve Segments

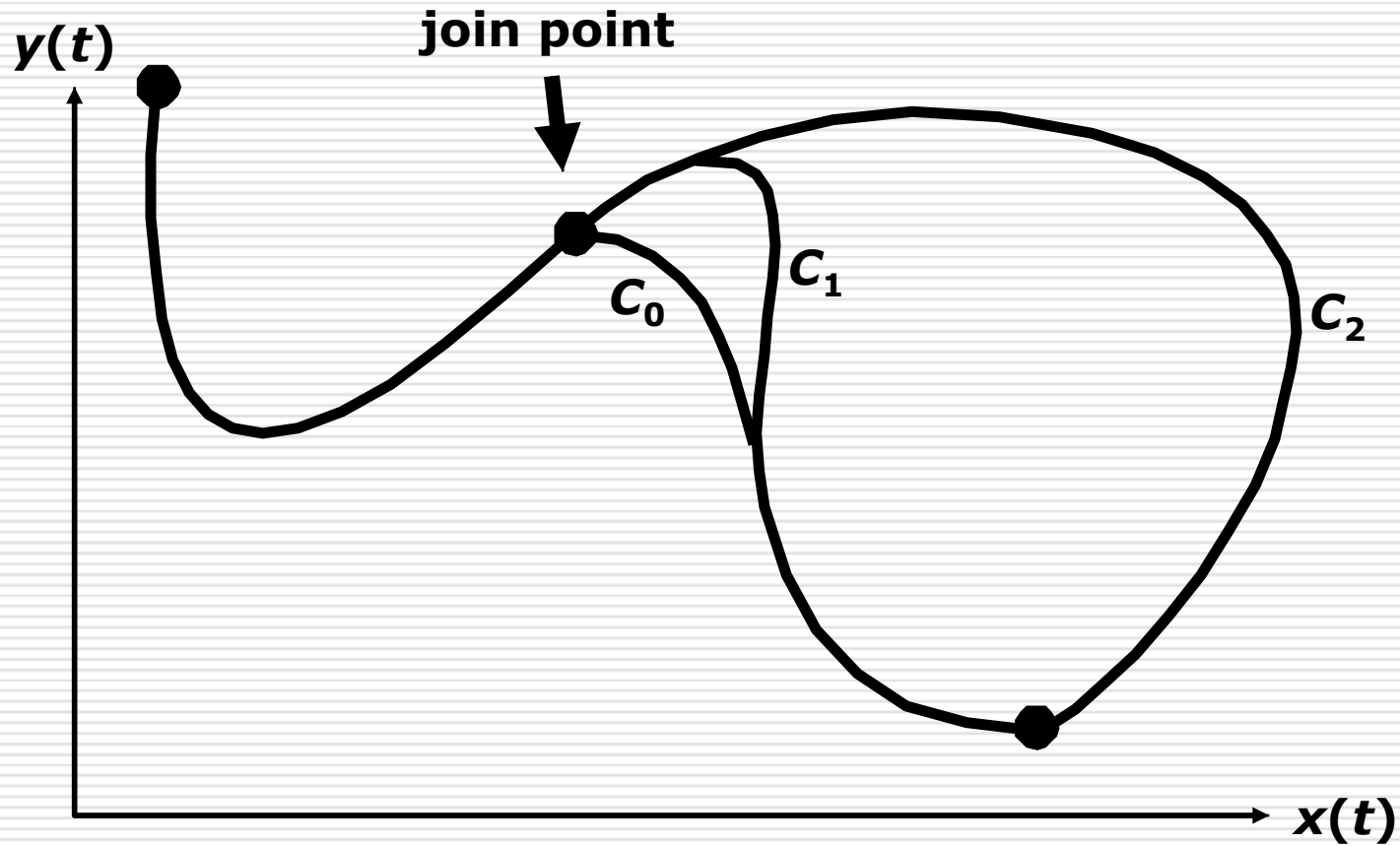
□ C^1 continuous

- the tangent vectors of the two cubic curve segments are equal (*both directions and magnitudes*) at the segments' join point

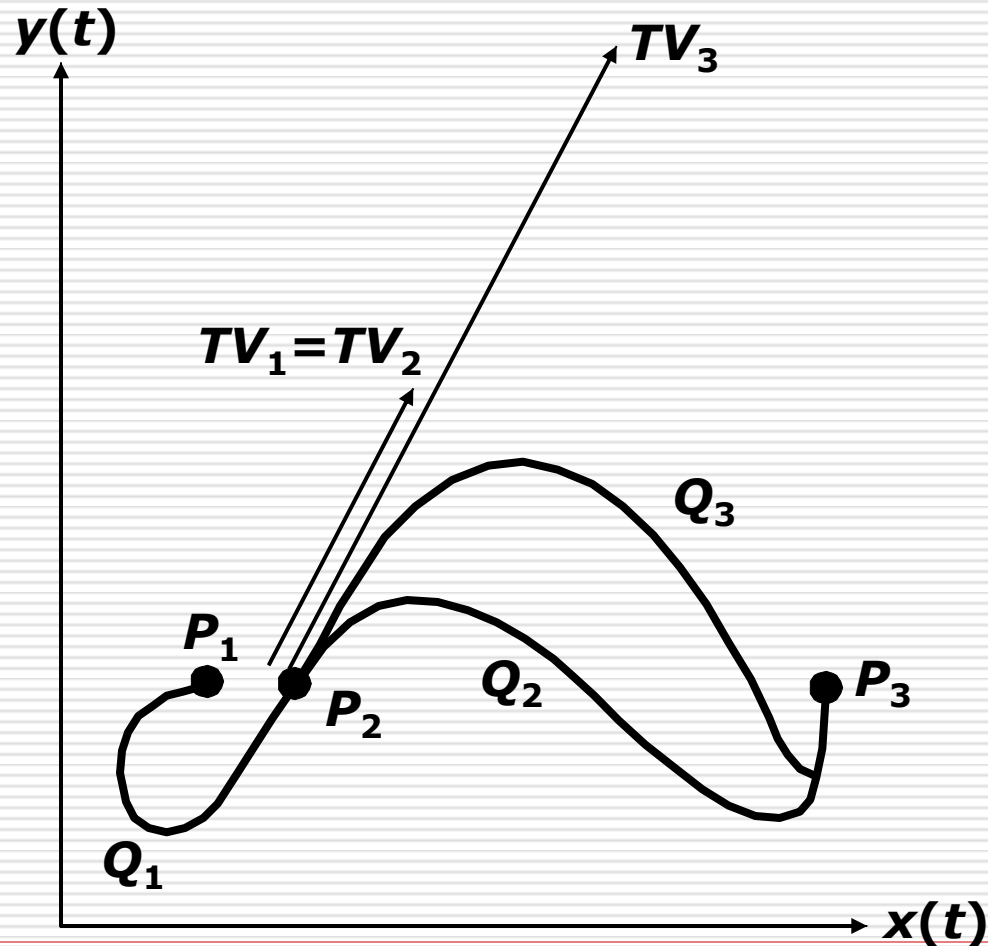
□ C^n continuous

- the direction and magnitude of $d^n / dt^n [Q(t)]$ through the n th derivative are equal at the join point

Continuity between Curve Segments



Continuity between Curve Segments



Bézier Curves → Splines

- Bézier curves have C-infinity continuity on their interiors, but we saw that they do not exhibit local control or interpolate their control points.
- It is possible to define points that we want to interpolate, and then solve for the Bézier control points that will do the job.
- But, you will need as many control points as interpolated points -> high order polynomials -> wiggly curves. (And you still won't have local control.)

Bézier Curves → Splines

- We will splice together a curve from individual Bézier segments. We call these curves **splines**.
- When splicing Bézier together, we need to worry about continuity.

Ensuring C^0 continuity

- Suppose we have a cubic Bézier defined by (V_1, V_2, V_3, V_4) , and we want to attach another curve (W_1, W_2, W_3, W_4) to it, so that there is C^0 continuity at the joint.

$$C^0 : Q_V(1) = Q_W(0)$$

- What constraint(s) does this place on (W_1, W_2, W_3, W_4) ?

$$Q_V(1) = Q_W(0) \Rightarrow V_4 = W_1$$

Ensuring C^1 continuity

- Suppose we have a cubic Bézier defined by (V_1, V_2, V_3, V_4) , and we want to attach another curve (W_1, W_2, W_3, W_4) to it, so that there is C^1 continuity at the joint.
 $C^0 : Q_V(1) = Q_W(0)$

$$C^1 : Q'_V(1) = Q'_W(0)$$

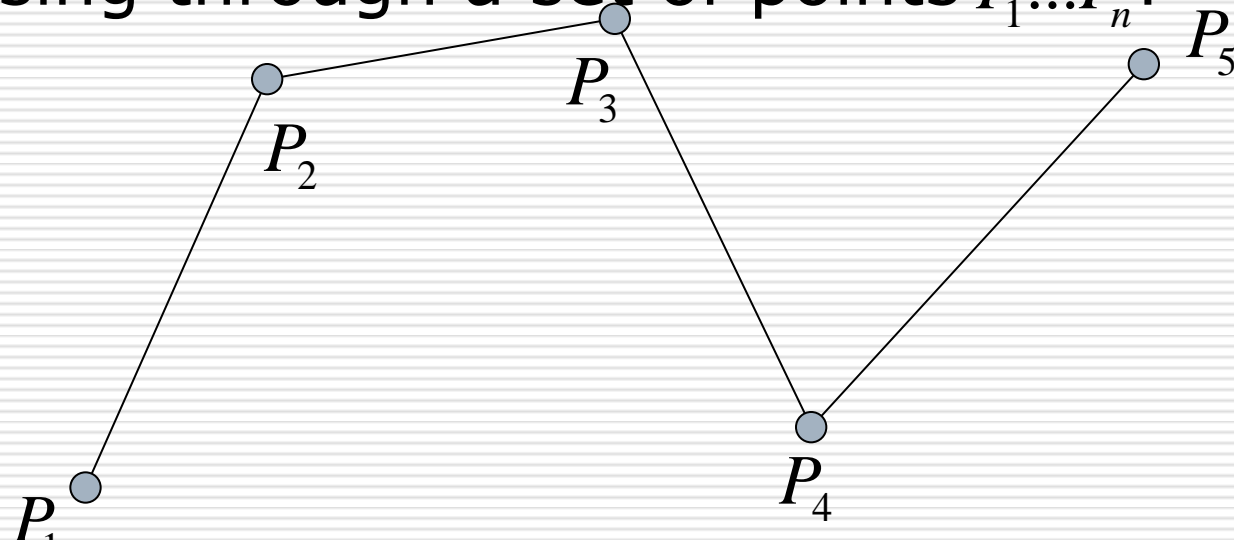
- What constraint(s) does this place on (W_1, W_2, W_3, W_4) ?

$$Q_V(1) = Q_W(0) \Rightarrow V_4 = W_1$$

$$Q'_V(1) = Q'_W(0) \Rightarrow V_4 - V_3 = W_2 - W_1$$

The C^1 Bézier Spline

- How then could we construct a curve passing through a set of points $P_1 \dots P_n$?



- We can specify the Bézier control points directly, or we can devise a scheme for placing them automatically...

Catmull-Rom Spline

- If we set each derivative to be one half of the vector between the previous and next controls, we get a **Catmull-Rom Spline**.

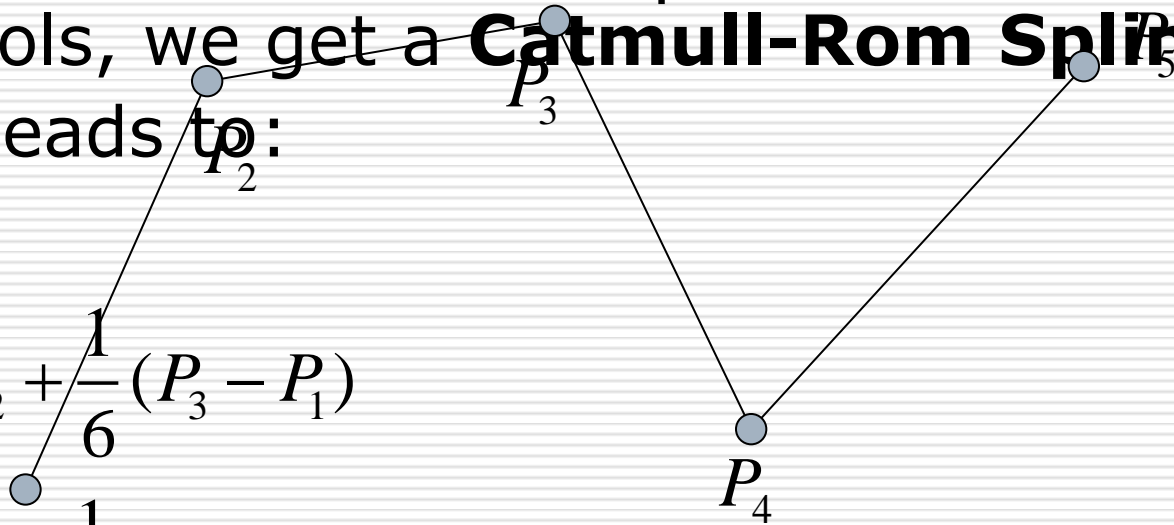
- This leads to:

$$V_1 = P_2$$

$$V_2 = P_2 + \frac{1}{6}(P_3 - P_1)$$

$$V_3 = P_3 - \frac{1}{6}(P_4 - P_2)$$

$$V_4 = P_3$$



Catmull-Rom Basis Matrix

$$\begin{aligned}
 Q(t) &= G_B \bullet M_B \bullet T \\
 &= G_B \bullet \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \bullet T \quad G_B = \begin{bmatrix} V_1 \\ V_2 \\ V_3 \\ V_4 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ -\frac{1}{6} & 1 & \frac{1}{6} & 0 \\ 0 & \frac{1}{6} & 1 & -\frac{1}{6} \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} P_1 \\ P_2 \\ P_3 \\ P_4 \end{bmatrix} \\
 Q(t) &= \begin{bmatrix} P_1 & P_2 & P_3 & P_4 \end{bmatrix} \frac{1}{2} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 2 & -5 & 4 & -1 \\ -1 & 0 & 1 & 0 \\ 0 & 2 & 0 & 0 \end{bmatrix} \begin{bmatrix} t^3 \\ t^2 \\ t \\ 1 \end{bmatrix}
 \end{aligned}$$

Ensuring C^2 continuity

- Suppose we have a cubic Bézier defined by (V_1, V_2, V_3, V_4) , and we want to attach another curve (W_1, W_2, W_3, W_4) to it, so that there is C^2 continuity at the joint.

$$Q_V(1) = Q_W(0) \Rightarrow V_4 = W_1$$

$$Q'_V(1) = Q'_W(0) \Rightarrow V_4 - V_3 = W_2 - W_1$$

$$Q''_V(1) = Q''_W(0) \Rightarrow V_2 - 2V_3 + V_4 = W_1 - 2W_2 + W_3$$

↓

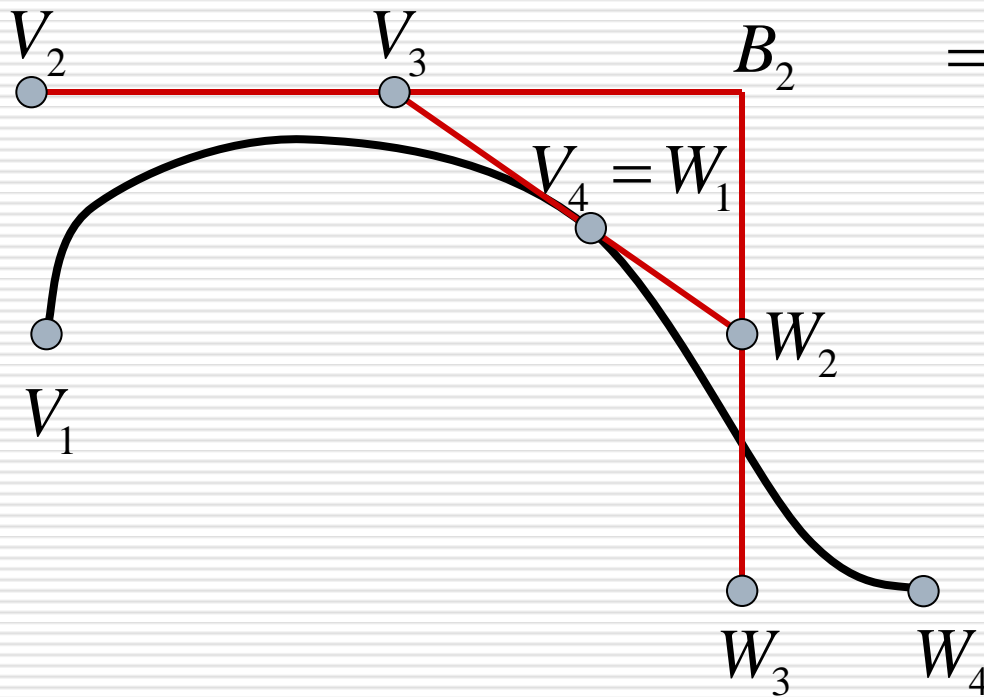
$$W_3 = V_2 - 4V_3 + 4V_4$$

B-Spline

- Instead of specifying the Bézier control points themselves, let's specify the corners of the A-frames in order to build a C^2 continuous spline.

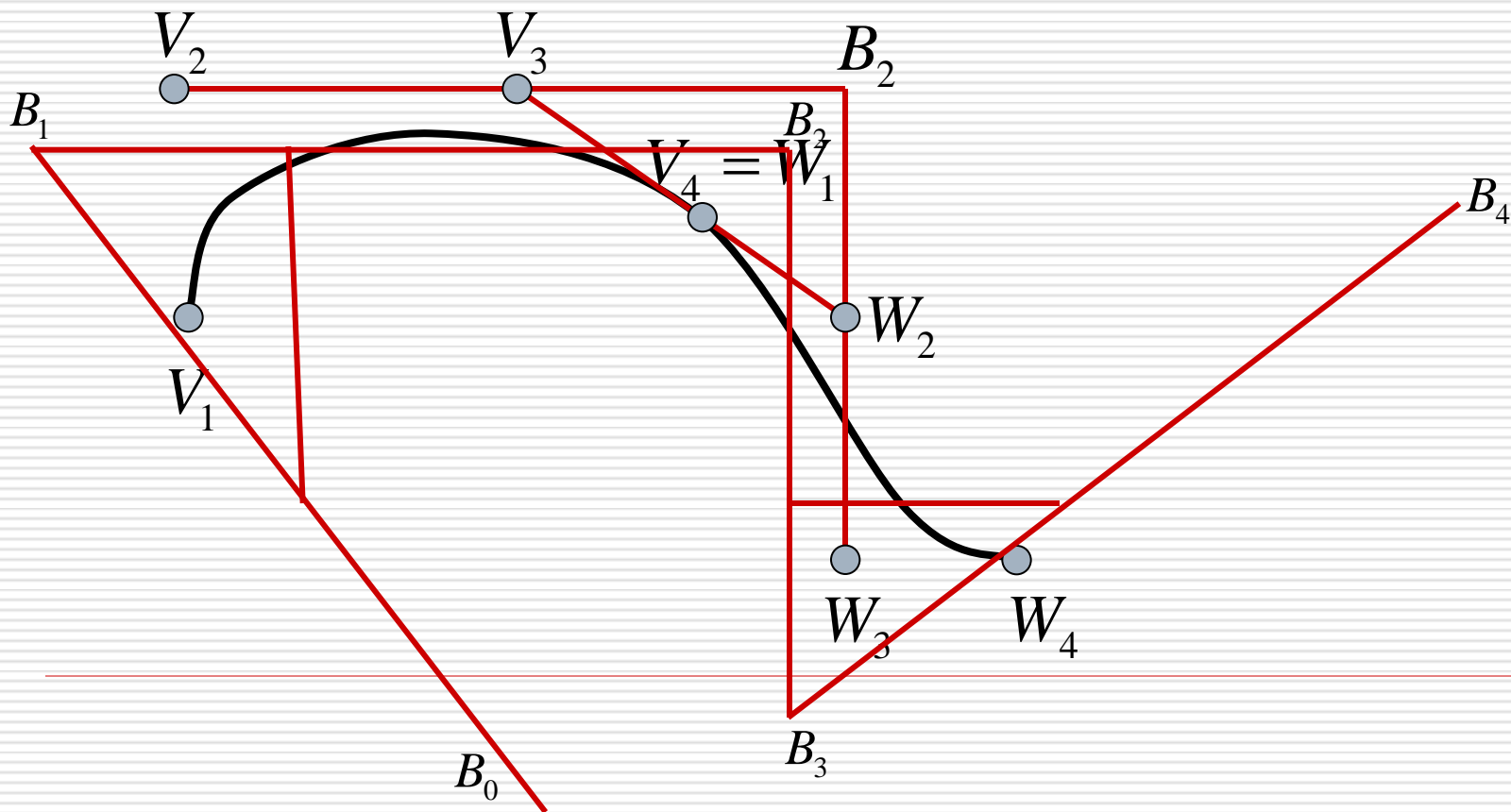
B-Spline

$$\begin{aligned}W_3 &= V_2 - 4V_3 + 4V_4 \\ &= 2(2V_4 - V_3) - (2V_3 - V_2) \\ &= 2W_2 - B_2\end{aligned}$$



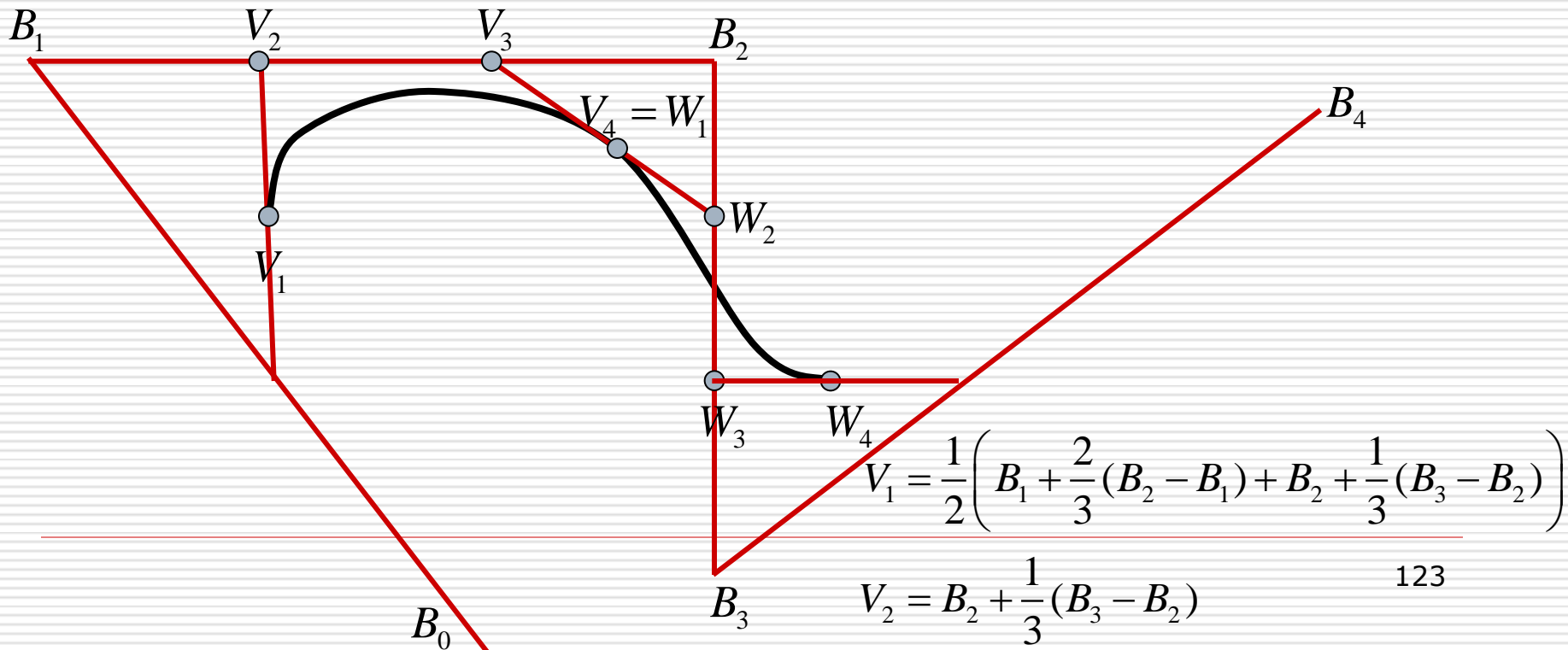
B-Spline

- Instead of specifying the Bézier control points themselves, let's specify the corners of the A-frames in order to build a C^2 continuous spline.



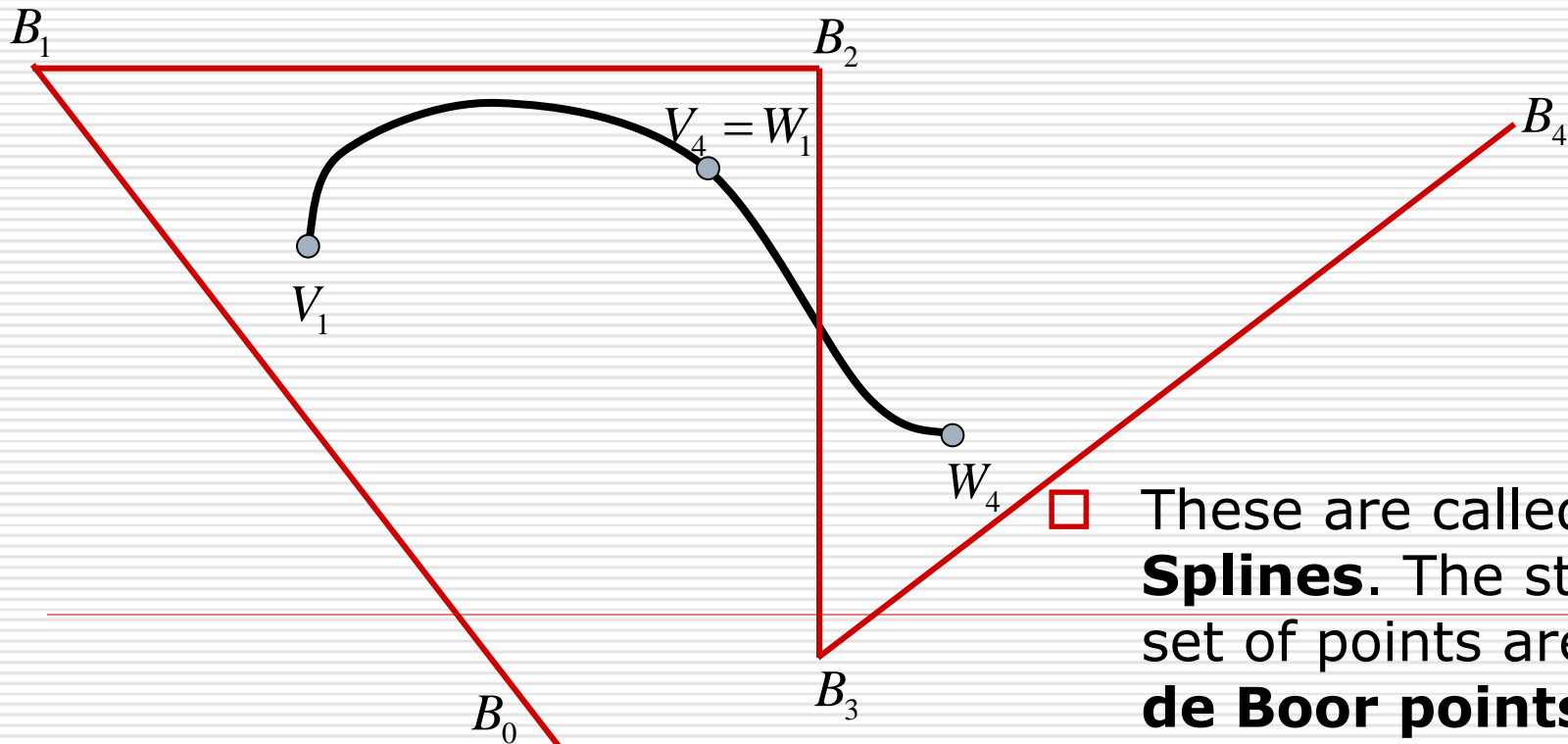
B-Spline

- Instead of specifying the Bézier control points themselves, let's specify the corners of the A-frames in order to build a C^2 continuous spline.



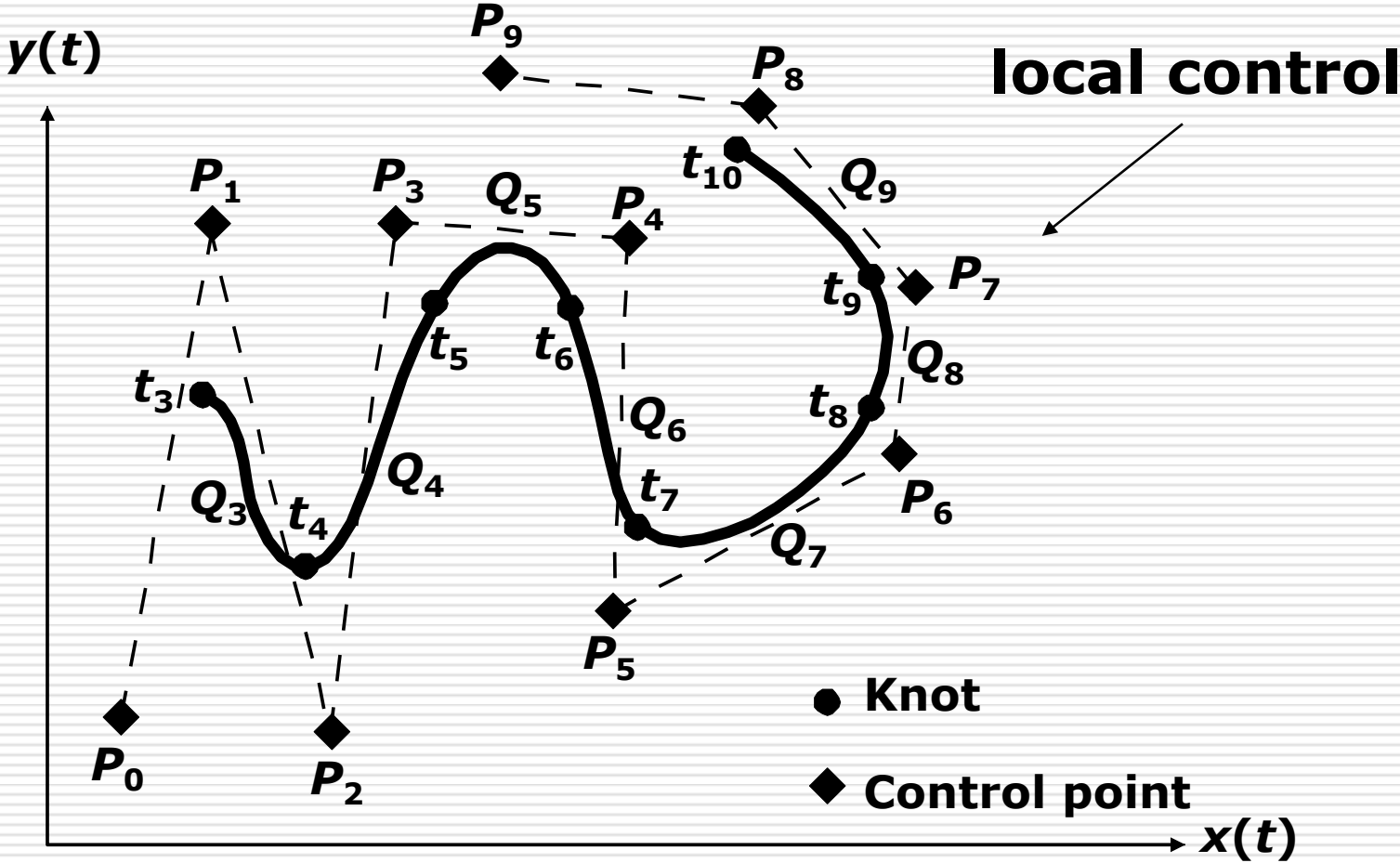
B-Spline

- Instead of specifying the Bézier control points themselves, let's specify the corners of the A-frames in order to build a C^2 continuous spline.



- These are called **B-Splines**. The starting set of points are called **de Boor points**.

B-Spline



Uniform NonRational B-Splines

□ cubic B-Spline

- has $m+1$ control points $P_0, P_1, \dots, P_m, m \geq 3$
- has $m-2$ cubic polynomial curve segments Q_3, Q_4, \dots, Q_m

□ uniform

- the knots are spaced at equal intervals of the parameter t

□ non-rational

- not rational cubic polynomial curves

Uniform NonRational B-Splines

□ curve segment Q_i is defined by points $P_{i-3}, P_{i-2}, P_{i-1}, P_i$, thus

□ **B-Spline geometry matrix**

$$G_{Bs_i} = [P_{i-3} \quad P_{i-2} \quad P_{i-1} \quad P_i], \quad 3 \leq i \leq m$$

□ if $T_i = [(t-t_i)^3 \quad (t-t_i)^2 \quad (t-t_i) \quad 1]^T$

□ then $Q_i(t) = G_{Bs_i} \bullet M_{Bs} \bullet T_i, \quad t_i \leq t \leq t_{i+1}$

Uniform NonRational B-Splines

□ so **B-Spline basis matrix**

$$M_{Bs} = \frac{1}{6} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 0 & 4 \\ -3 & 3 & 3 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

□ **B-Spline blending functions**

$$B_{Bs} = \frac{1}{6} \left[(1-t)^3 \quad 3t^3 - 6t^2 + 4 \quad -3t^3 + 3t^2 + 3t + 1 \quad t^3 \right]^T, \quad 0 \leq t \leq 1$$

NonUniform NonRational B-Splines

- the **knot-value sequence** is a nondecreasing sequence
- allow **multiple knot** and the number of identical parameter is the **multiplicity**
 - Ex. (0,0,0,0,1,1,2,3,4,4,5,5,5,5)
- so

$$Q_i(t) = P_{i-3} \bullet B_{i-3,4}(t) + P_{i-2} \bullet B_{i-2,4}(t) + P_{i-1} \bullet B_{i-1,4}(t) + P_i \bullet B_{i,4}(t)$$

NonUniform NonRational B-Splines

- where $B_{i,j}(t)$ is j th-order blending function for weighting control point P_i

$$B_{i,1}(t) = \begin{cases} 1, & t_i \leq t \leq t_{i+1} \\ 0, & \text{otherwise} \end{cases}$$

$$B_{i,2}(t) = \frac{t-t_i}{t_{i+1}-t_i} B_{i,1}(t) + \frac{t_{i+2}-t}{t_{i+2}-t_{i+1}} B_{i+1,1}(t)$$

$$B_{i,3}(t) = \frac{t-t_i}{t_{i+2}-t_i} B_{i,2}(t) + \frac{t_{i+3}-t}{t_{i+3}-t_{i+1}} B_{i+1,2}(t)$$

$$B_{i,4}(t) = \frac{t-t_i}{t_{i+3}-t_i} B_{i,3}(t) + \frac{t_{i+4}-t}{t_{i+4}-t_{i+1}} B_{i+1,3}(t)$$

Knot Multiplicity & Continuity

- since $Q(t_i)$ is within the convex hull of P_{i-3} , P_{i-2} , and P_{i-1}
- if $t_i = t_{i+1}$, $Q(t_i)$ is within the convex hull of P_{i-3} , P_{i-2} , and P_{i-1} and the convex hull of P_{i-2} , P_{i-1} , and P_i , so it will lie on $\overline{P_{i-2}P_{i-1}}$
- if $t_i = t_{i+1} = t_{i+2}$, $Q(t_i)$ will lie on P_{i-1}
- if $t_i = t_{i+1} = t_{i+2} = t_{i+3}$, $Q(t_i)$ will lie on both P_{i-1} and P_i , and the curve becomes broken

Knot Multiplicity & Continuity

- multiplicity 1 : C^2 continuity
- multiplicity 2 : C^1 continuity
- multiplicity 3 : C^0 continuity
- multiplicity 4 : no continuity

NURBS: NonUniform Rational B-Splines

□ rational

- $x(t)$, $y(t)$, and $z(t)$ are defined as the ratio of two cubic polynomials

□ rational cubic polynomial curve segments are ratios of polynomials

$$x(t) = \frac{X(t)}{W(t)} \quad y(t) = \frac{Y(t)}{W(t)} \quad z(t) = \frac{Z(t)}{W(t)}$$

□ can be Bézier, Hermite, or B-Splines