Computer Graphics

Bing-Yu Chen
National Taiwan University

Geometrical Transformations

- Mathematical Preliminaries
- 2D Transformations
- Homogeneous Coordinates & Matrix Representation
- The Window-to-Viewport Transformation
- 3D Transformations
- Quaternions

Vectors

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

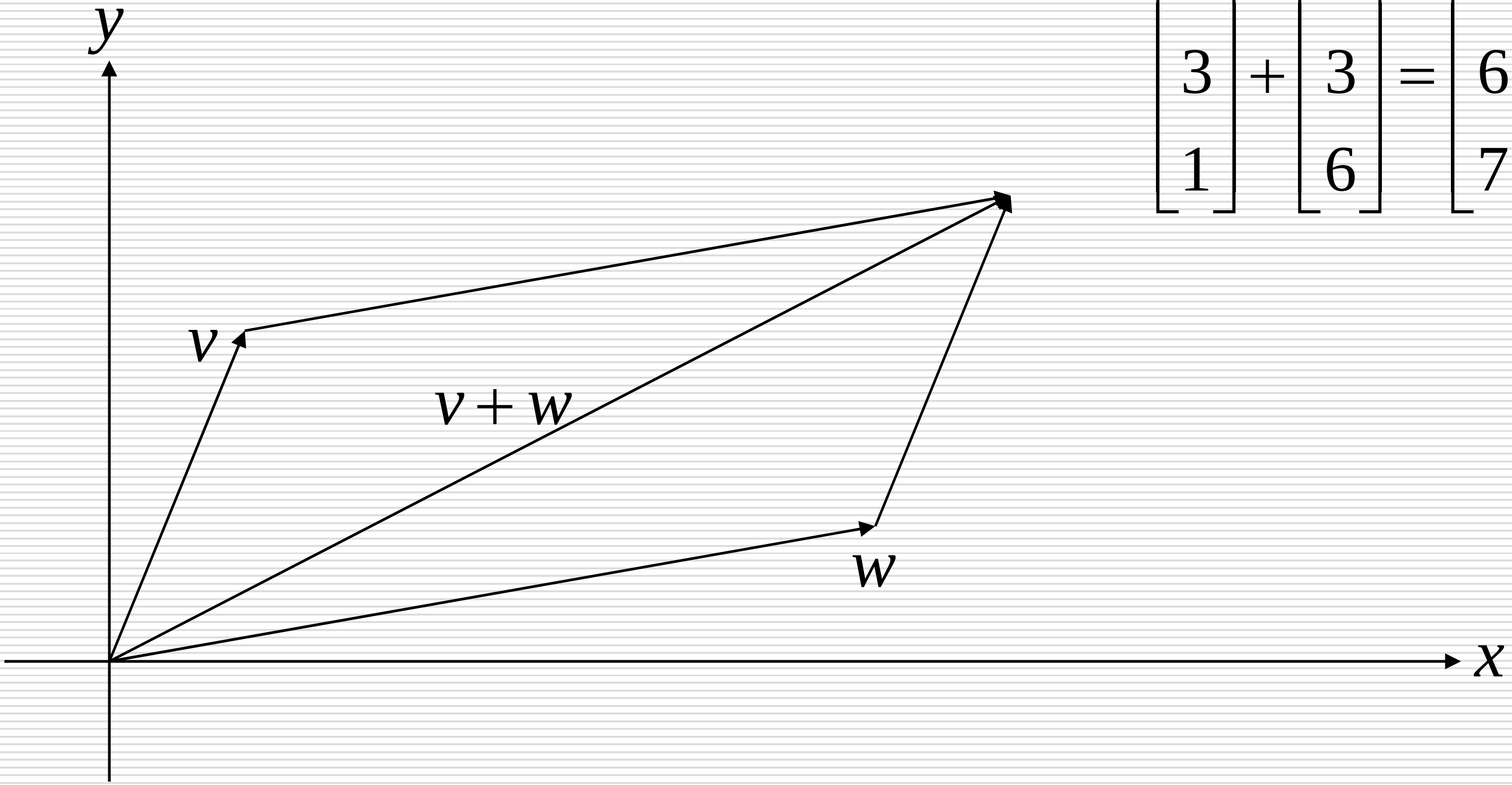
Vectors

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

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

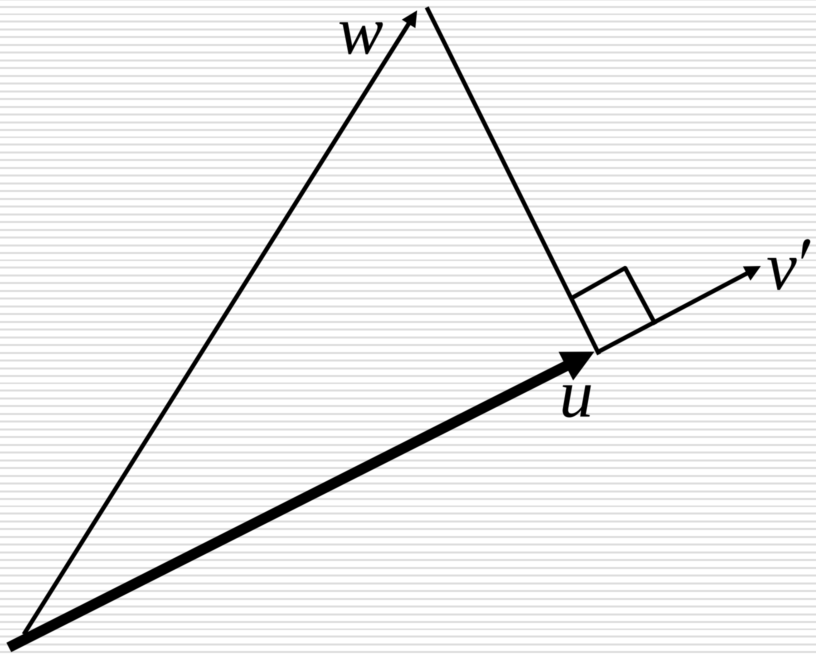
$$\Rightarrow u \bullet v = x_1 y_1 + \ldots + x_n y_n$$

- 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
  - $\cos^{-1}\left(v \bullet w / \|v\| \|w\|\right)$

Projection



$$\|u\| = \|w\|\cos\theta$$

$$= \|w\|\left(\frac{v' \bullet w}{\|v'\|\|w\|}\right)$$

$$= v' \bullet w$$

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

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

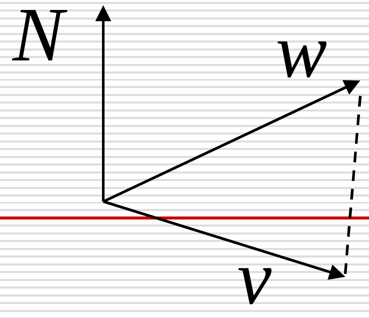
- Matrix-matrix multiplication

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

Cross Product of Vectors

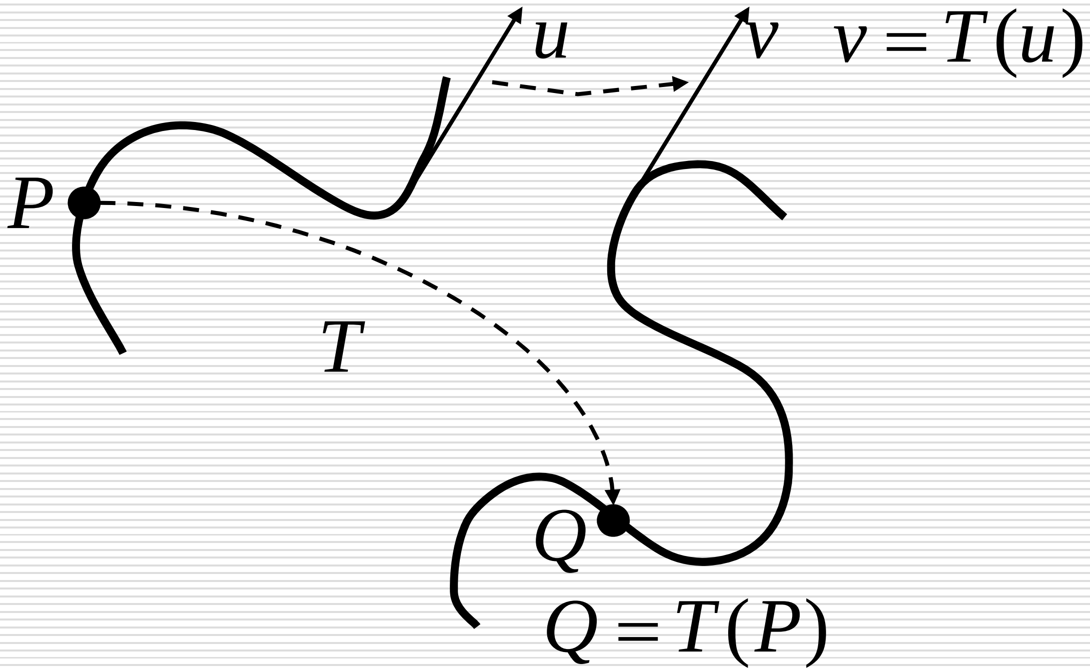
- Definition
  - $x = v \times w$
    $= (v_2 w_3 - v_3 w_2)i + (v_3 w_1 - v_1 w_3)j + (v_1 w_2 - v_2 w_1)k$
  - where i, j, and k are standard unit vectors:
    $i = (1,0,0) \quad j = (0,1,0) \quad k = (0,0,1)$
- Application
  - A normal vector to a polygon is calculated from 3 (non-collinear) vertices of the polygon.

$N = v \times w$

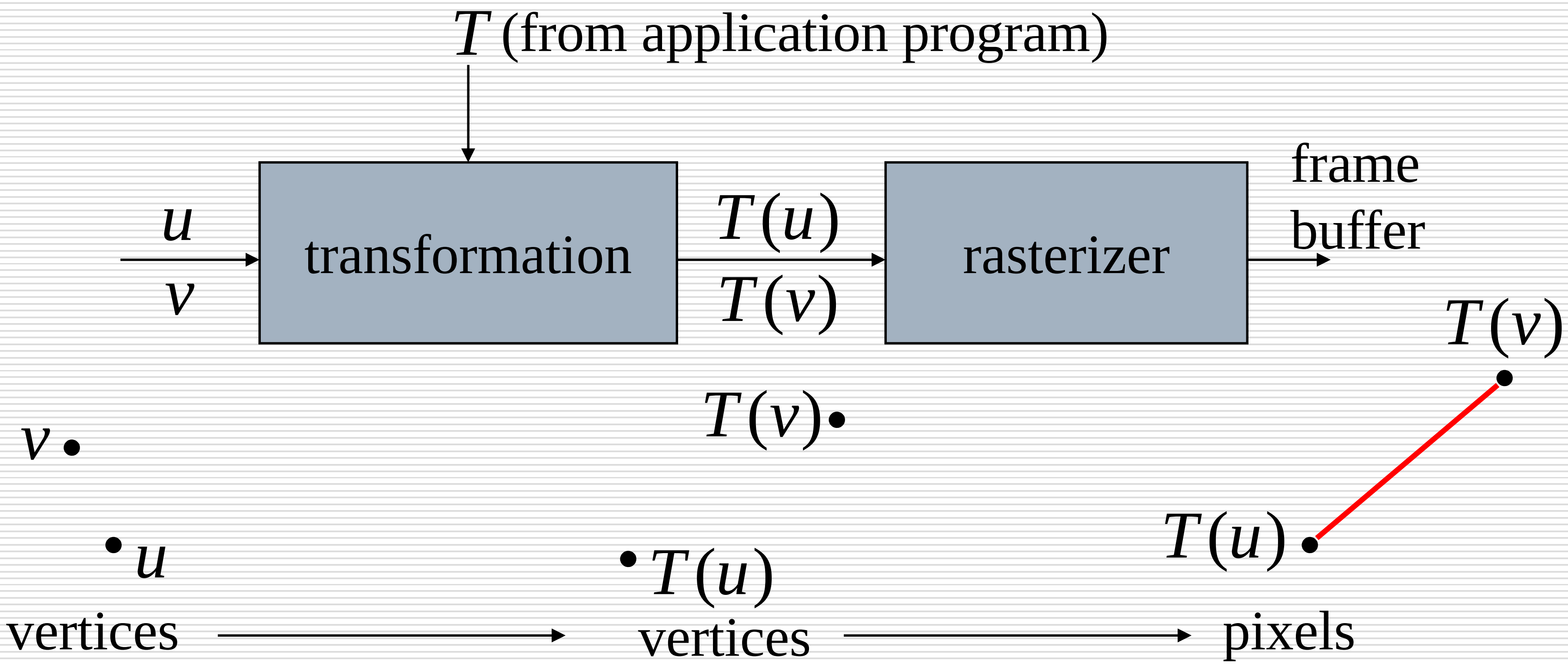


General Transformations

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



Pipeline Implementation



Representation

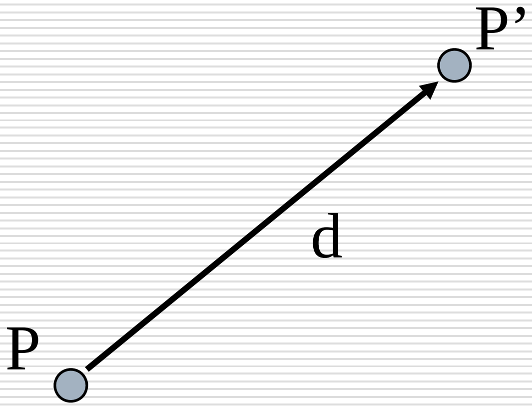
- We can represent a **point**, $\mathbf{p} = (x, y)$ in the plane
  - as a column vector $\begin{bmatrix} x \\ y \end{bmatrix}$
  - as a row vector $\begin{bmatrix} x & y \end{bmatrix}$

2D Transformations

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

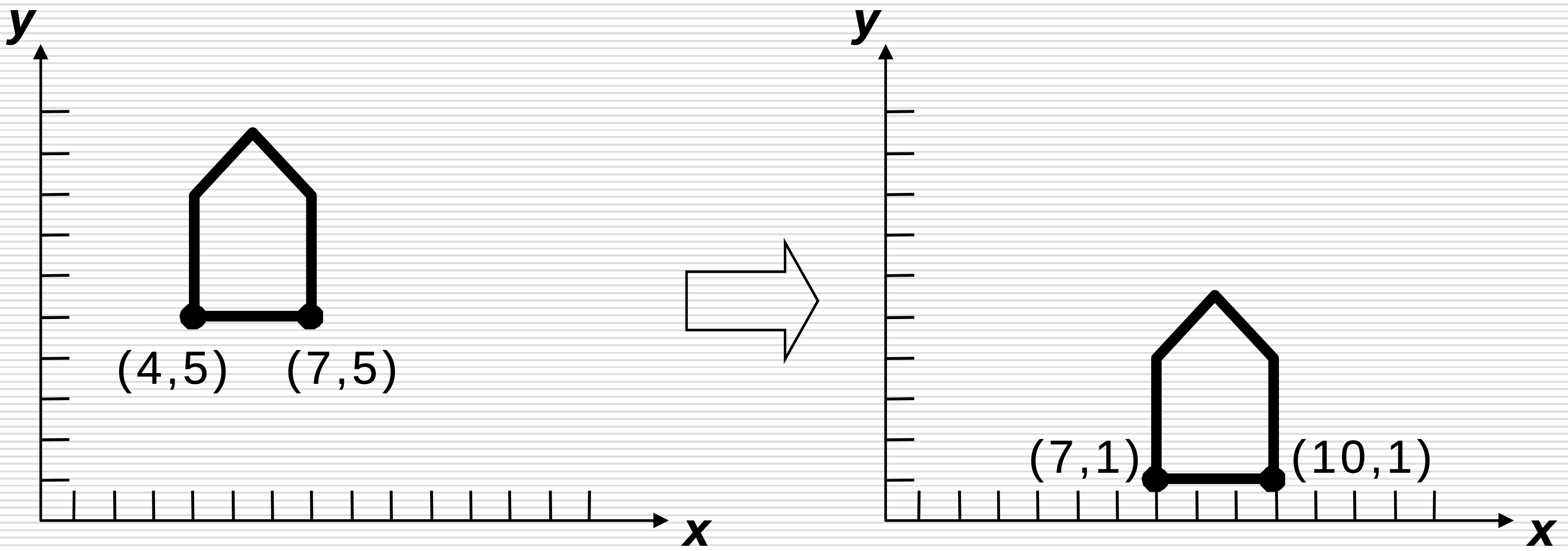
Translation

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



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

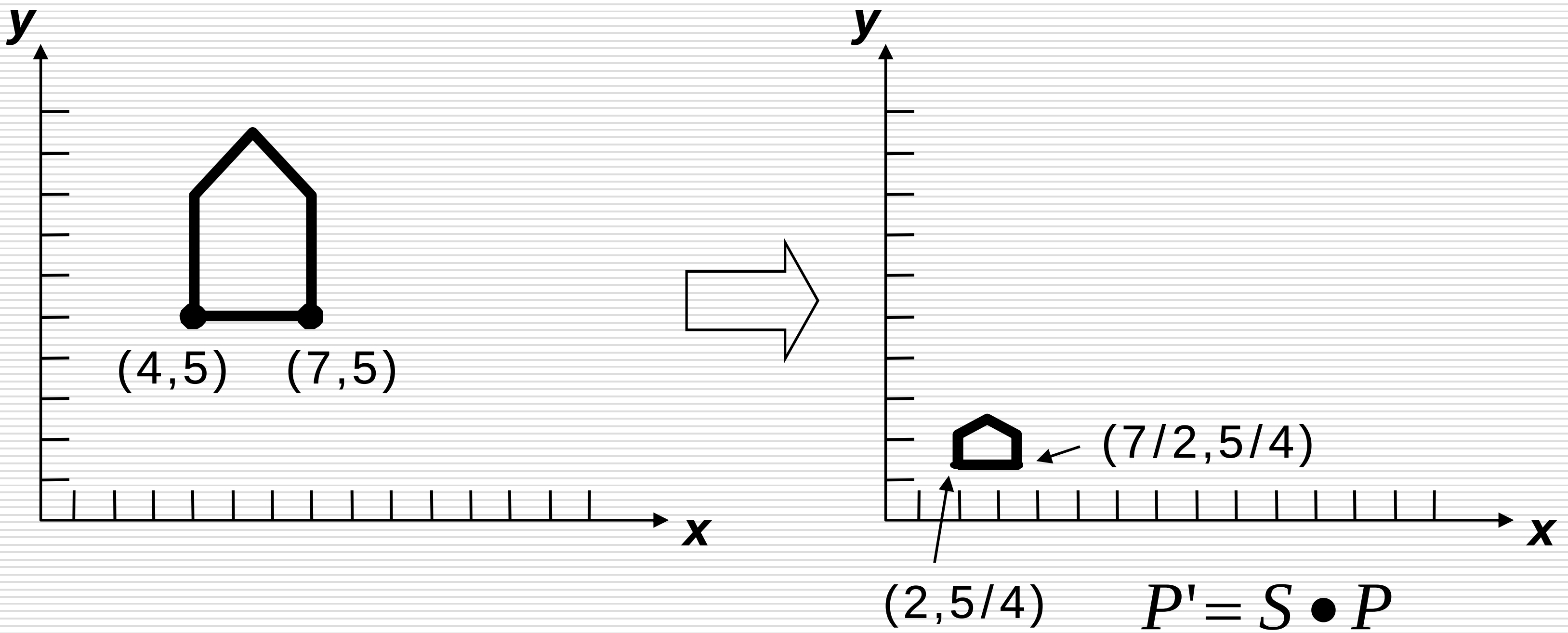
2D Translation



$$P' = P + T$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} d_x \\ d_y \end{bmatrix}$$

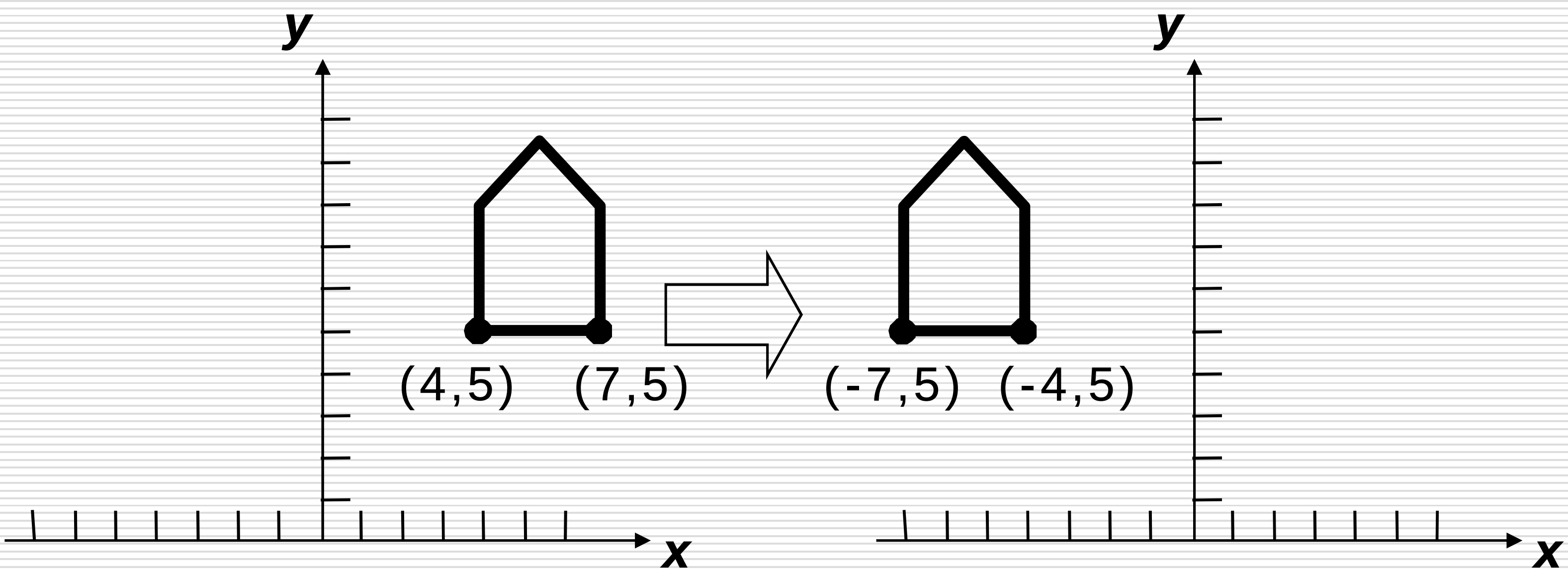
2D Scaling



$$P' = S \bullet P$$

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

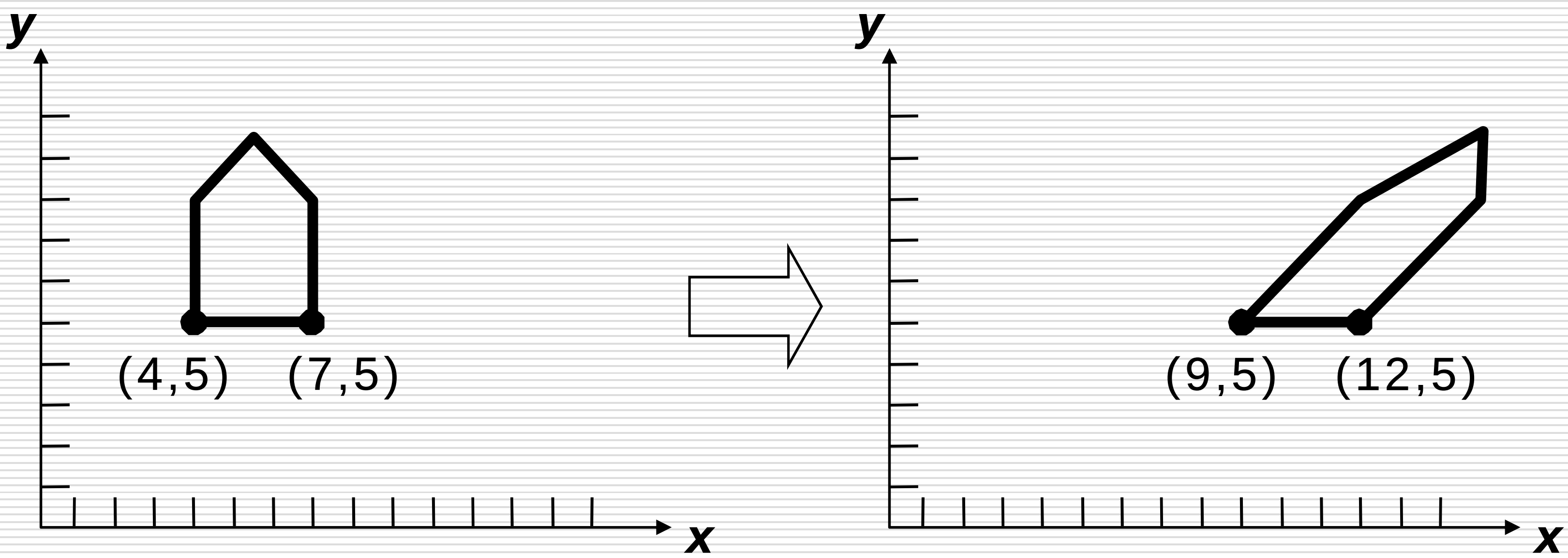
2D Reflection



$$P' = RE_x \bullet P$$

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

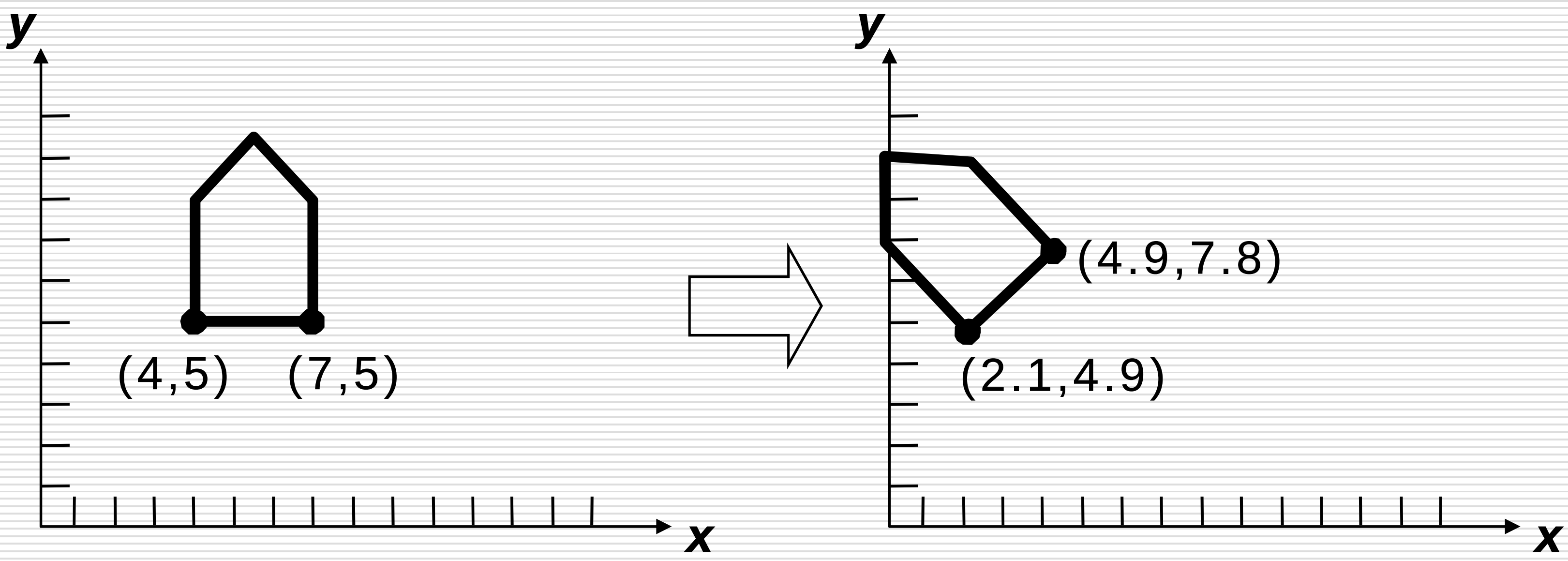
2D Shearing



$$P' = SH_x \bullet P$$

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

2D Rotation



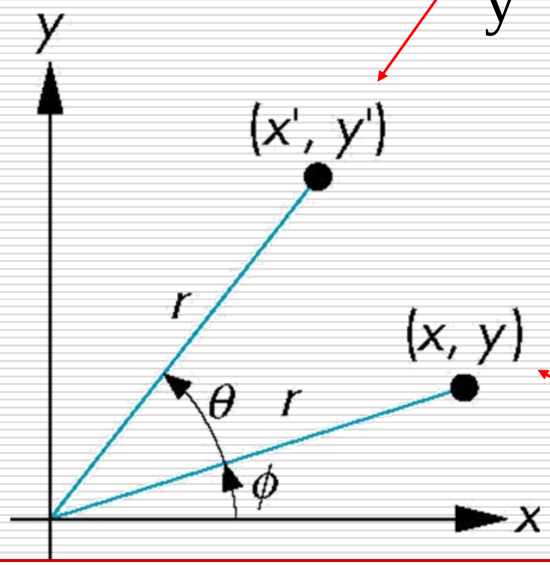
$$P' = R \bullet P$$

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

2D Rotation

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

$$x' = r\cos(\phi + \theta)$$
$$y' = r\sin(\phi + \theta)$$



$$x' = x\cos\theta - y\sin\theta$$
$$y' = x\sin\theta + y\cos\theta$$

$$x = r\cos\phi$$
$$y = r\sin\phi$$

Limitations of a 2X2 matrix

- Scaling
- Rotation
- Reflection
- Shearing

- What do we miss?

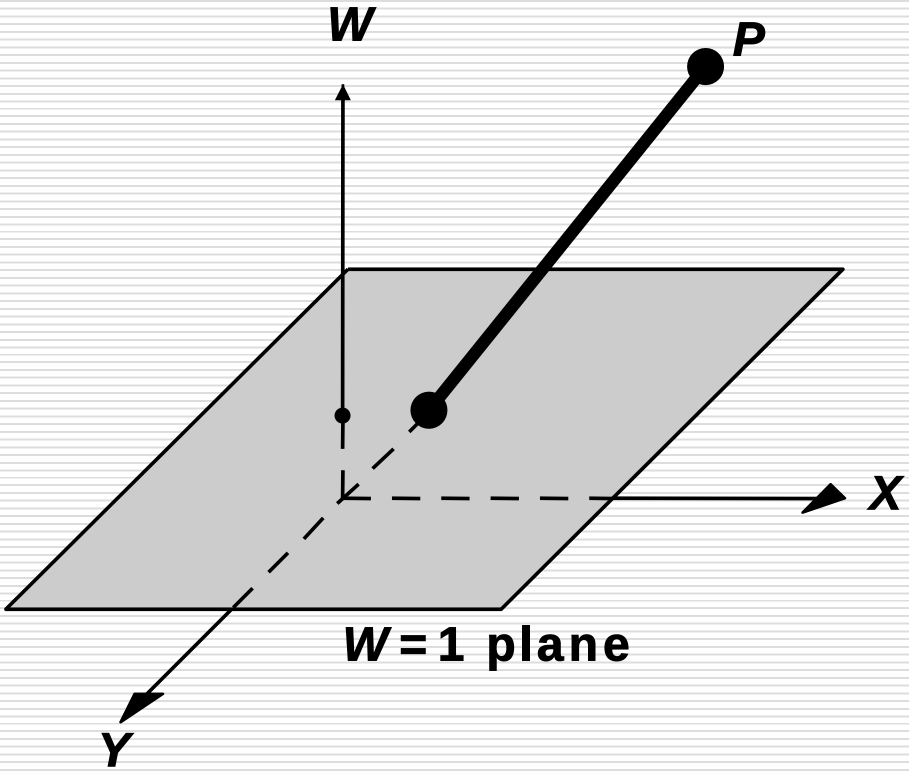
Homogeneous Coordinates

- Why & What is **homogeneous coordinates** ?
  - if points are expressed in homogeneous coordinates, all three transformations can be treated as multiplications.

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

(W:) usually 1
can not be 0

Homogeneous Coordinates



Homogeneous Coordinates for 2D Translation

$$P' = P + T$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} d_x \\ d_y \end{bmatrix}$$

$\Rightarrow$

$$P' = T(d_x, d_y) \bullet P$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & d_x \\ 0 & 1 & d_y \\ 0 & 0 & 1 \end{bmatrix} \bullet \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

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

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

Homogeneous Coordinates for 2D Translation

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

$$= (T(d_{x2}, d_{y2}) \bullet T(d_{x1}, d_{y1})) \bullet P$$

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

$$= \begin{bmatrix} 1 & 0 & d_{x1} + d_{x2} \\ 0 & 1 & d_{y1} + d_{y2} \\ 0 & 0 & 1 \end{bmatrix}$$

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

$$P' = R \bullet P$$

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

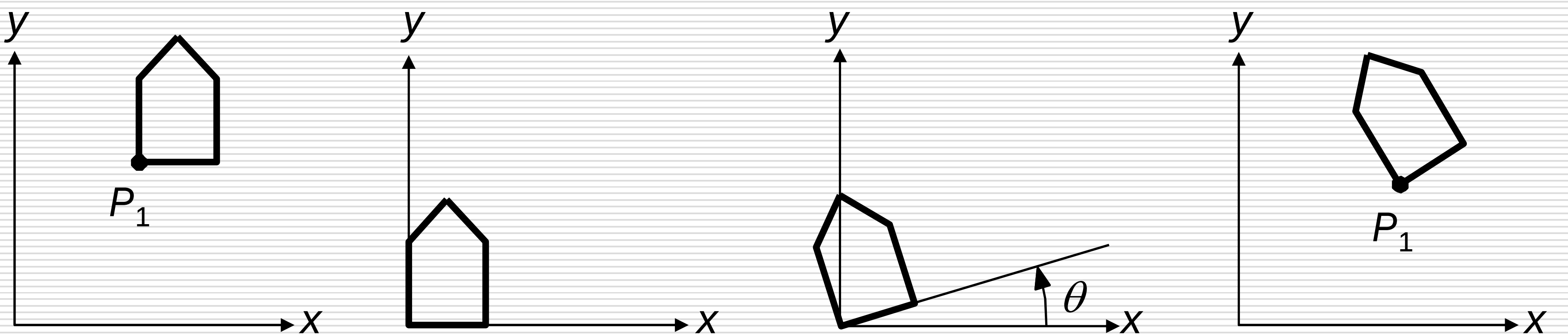
$$P' = R(\theta) \bullet P$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \bullet \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

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 | After translation of P_1 to origin | After rotation | After translation to original P_1

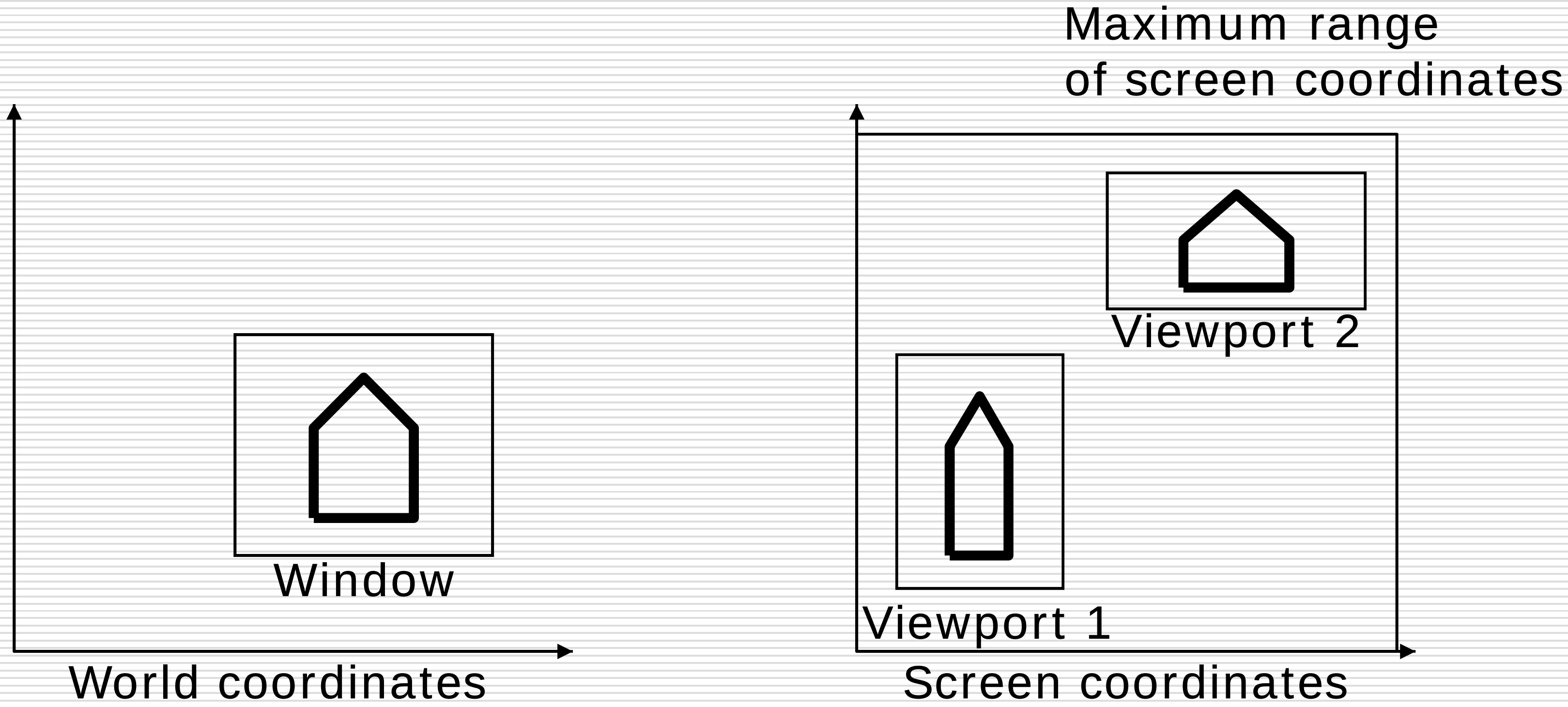
$$P_1 = (x_1, y_1) \qquad T(-x_1, -y_1) \qquad R(\theta) \qquad T(x_1, y_1)$$

Composition of 2D Transformations

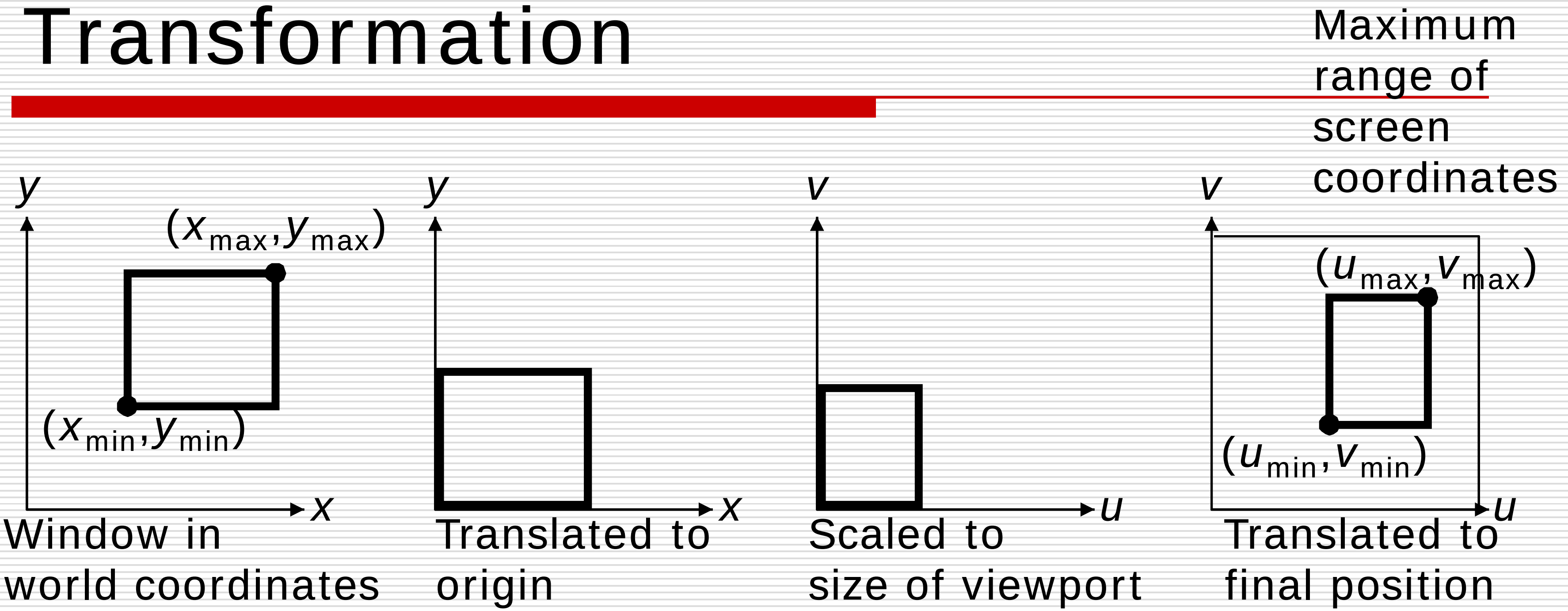
$$T(x_1, y_1) \bullet R(\theta) \bullet T(-x_1, -y_1) = \begin{bmatrix} 1 & 0 & x_1 \\ 0 & 1 & y_1 \\ 0 & 0 & 1 \end{bmatrix} \bullet \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \bullet \begin{bmatrix} 1 & 0 & -x_1 \\ 0 & 1 & -y_1 \\ 0 & 0 &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

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

The Window-to-Viewport Transformation

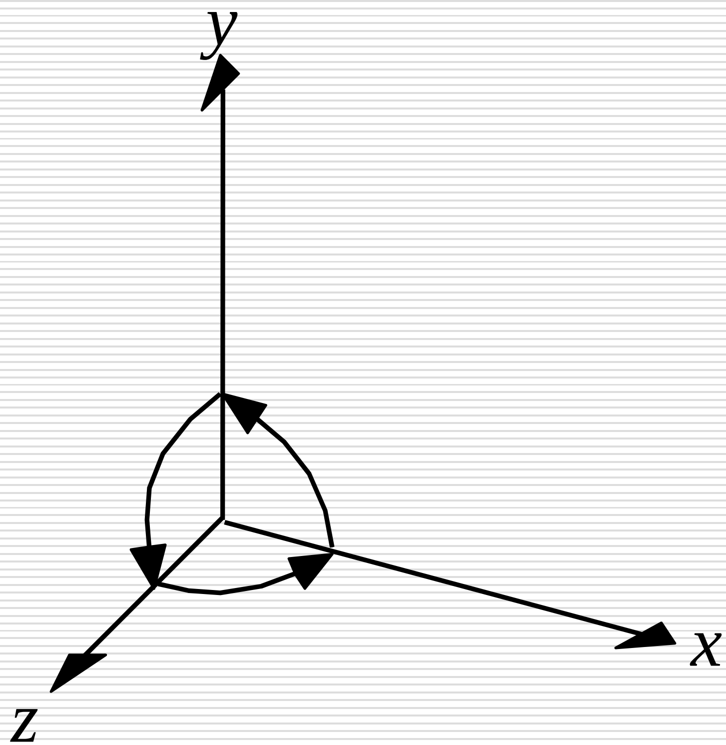


The Window-to-Viewport Transformation



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

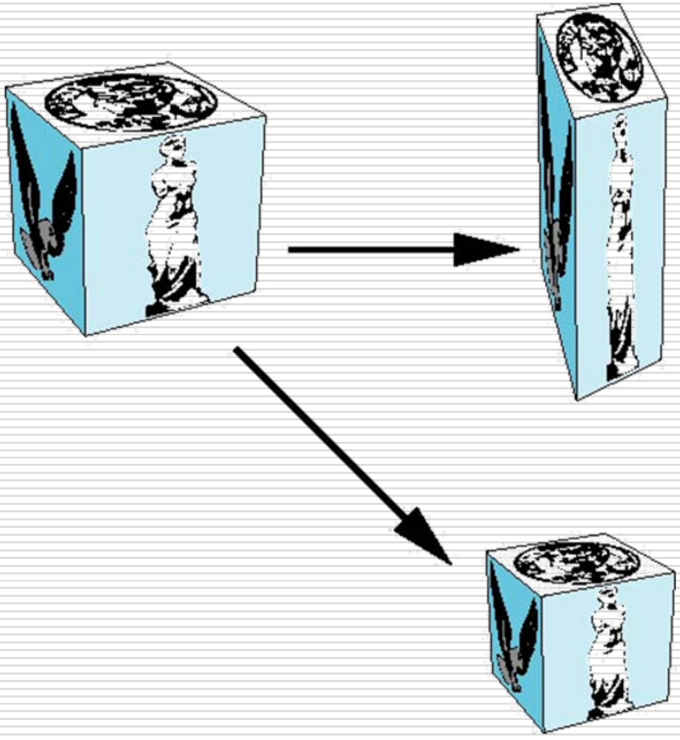
Right-handed Coordinate System



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

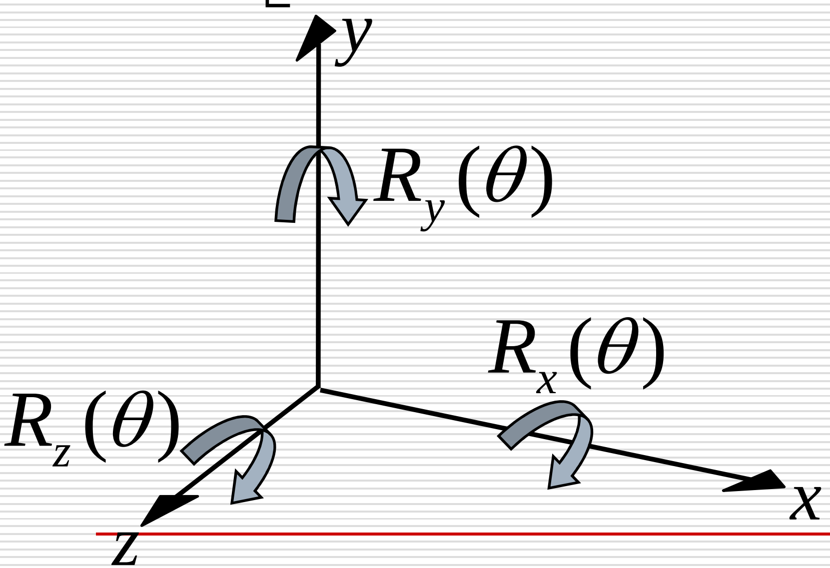
$$RE_x = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad RE_y = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

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

3D Rotations

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

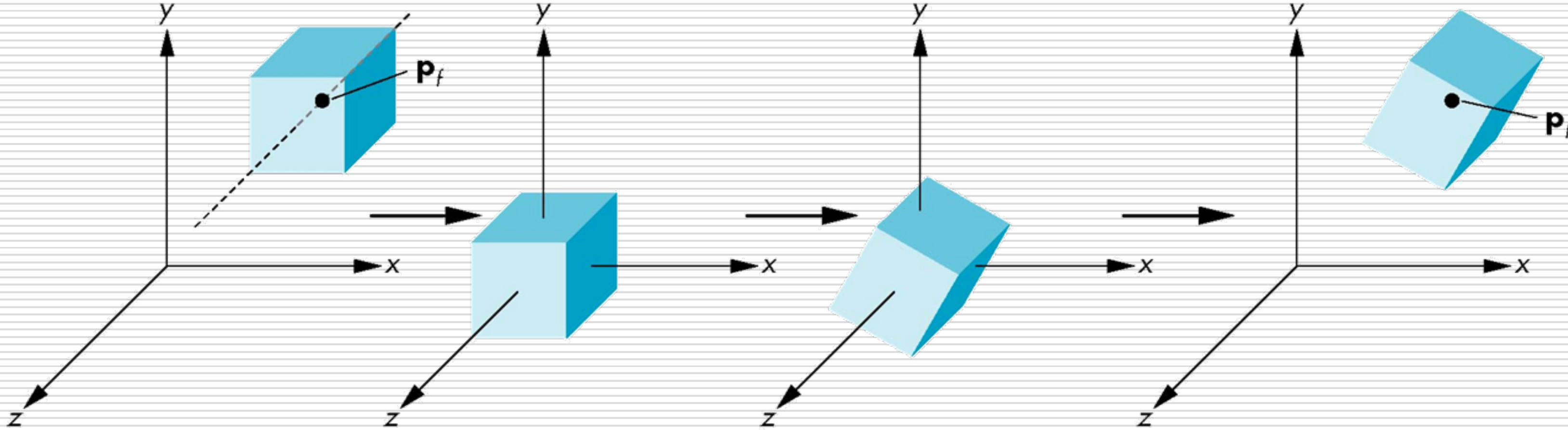
$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



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

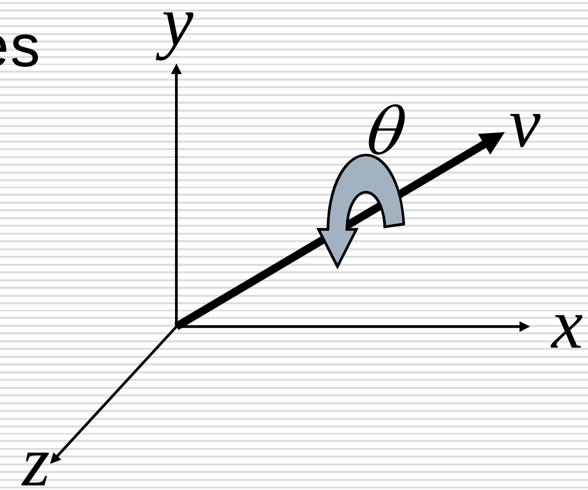
Rotation About a Fixed Point other than the Origin

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



Rotation About an Arbitrary Axis

- A rotation by θ about an arbitrary axis can be decomposed into the concatenation of rotations about the x, y, and z axes
  - θ_x, θ_y, θ_z are called the Euler angles

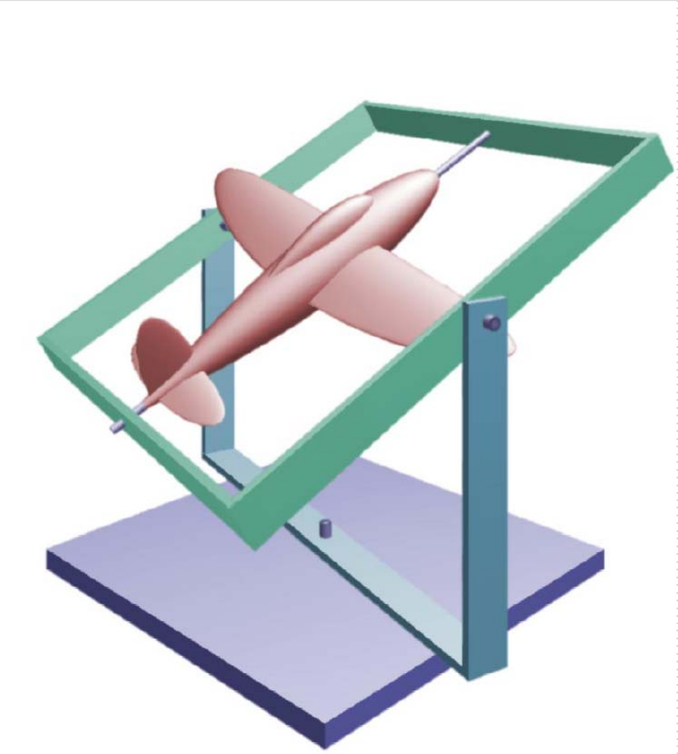


$$R(\theta) = R_z(\theta_z) \bullet R_y(\theta_y) \bullet R_x(\theta_x)$$

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

Euler Angles

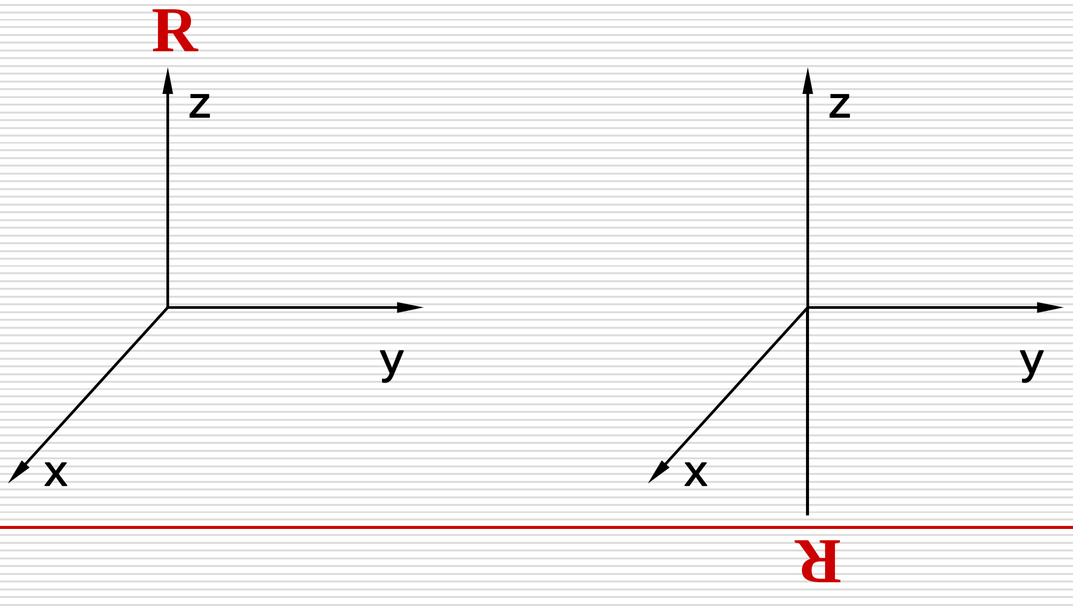
- An Euler angle is a rotation about a single axis.
- A rotation is described as a sequence of rotations about three mutually orthogonal coordinates axes fixed in space
  - X-roll, Y-roll, Z-roll



- There are 6 possible ways to define a rotation.
  - 3!

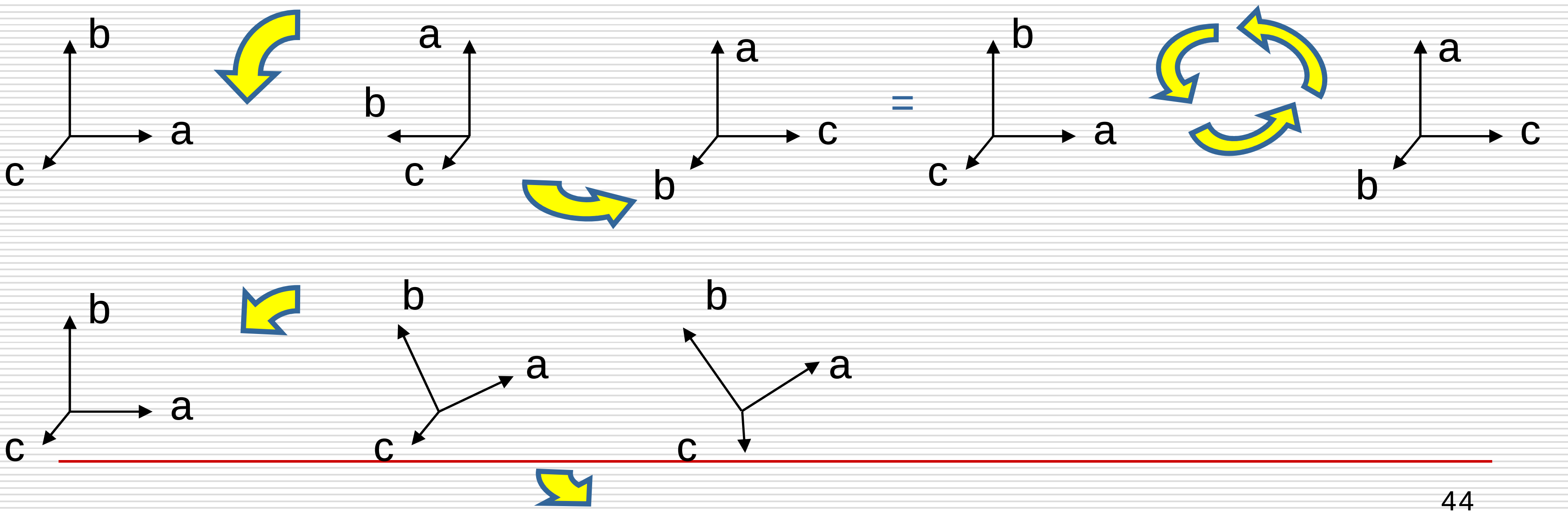
Euler Angles & Interpolation

- 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

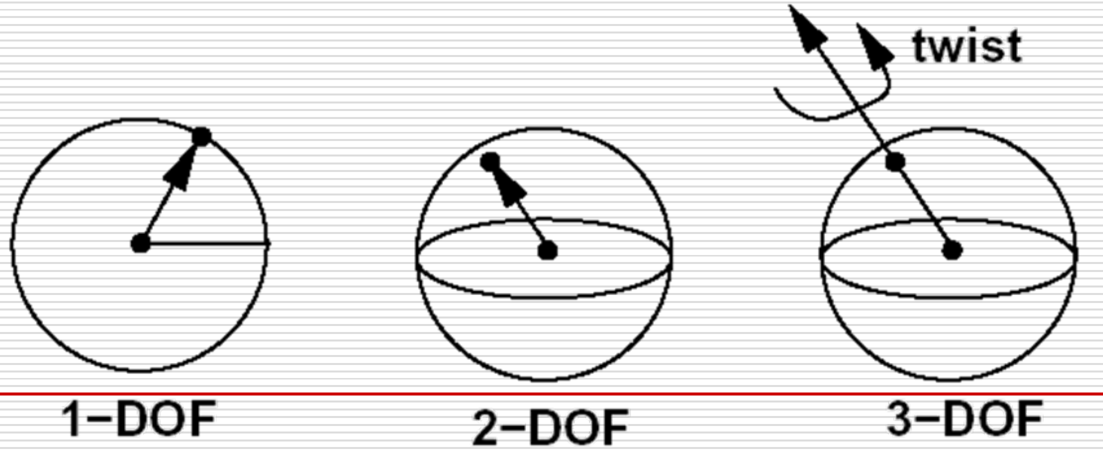
- From a practical standpoint, we are often want to use transformations to move and reorient an object smoothly
  - Problem: find a sequence of model-view matrices $\mathbf{M}_0, \mathbf{M}_1, \ldots, \mathbf{M}_n$ so that when they are applied successively to one or more objects we see a smooth transition
- 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 $\mathbf{R}_0,\ \mathbf{R}_1, \ldots, \mathbf{R}_n$, 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

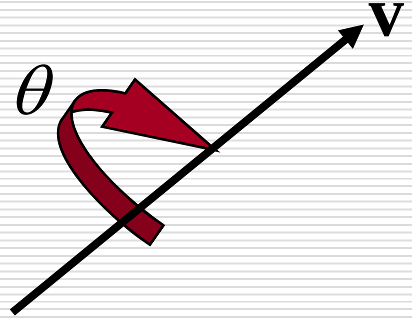
Solution: Quaternion Interpolation

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



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 $[\mathbf{w}, \mathbf{v}]$



Quaternions

- Extension of imaginary numbers from 2 to 3 dimensions
- 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**
- Quaternions can express rotations on sphere smoothly and efficiently. Process:
  - Model-view matrix $\rightarrow$ Quaternion
  - Carry out operations with Quaternions
  - Quaternion $\rightarrow$ Model-view matrix

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 $\mathrm{N}(q) = |q|^2 = \mathbf{w}^2 + |\mathbf{v}|^2$
- Inverse $q^{-1} = q^* / |q|^2 = q^* / \mathrm{N}(q)$
- 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)

Angle and Axis & Eular Angles

- Angle and Axis
  - $q = [\cos(\theta/2), \sin(\theta/2) \bullet \mathbf{v}]$

- Eular Angles
  - $q = q_{\text{yaw}} \bullet q_{\text{pitch}} \bullet q_{\text{roll}}$
    - $q_{\text{roll}} = [\cos(\psi/2), (\sin(\psi/2), 0, 0)]$
    - $q_{\text{pitch}} = [\cos(\theta/2), (0, \sin(\theta/2), 0)]$
    - $q_{\text{yaw}} = [\cos(\phi/2), (0, 0, \sin(\phi/2))]$

Matrix-to-Quaternion Conversion

```
MatToQuat (float m[4][4], QUAT * quat) {
        float tr, s, q[4];
        int i, j, k;
        int nxt[3] = {1, 2, 0};
        tr = m[0][0] + m[1][1] + m[2][2];
        if (tr > 0.0) {
                s = sqrt (tr + 1.0);
                quat->w = s / 2.0;
                s = 0.5 / s;
                quat->x = (m[1][2] - m[2][1]) * s;
                quat->y = (m[2][0] - m[0][2]) * s;
                quat->z = (m[0][1] - m[1][0]) * s;
        } else {
                i = 0;
                if (m[1][1] > m[0][0]) i = 1;
                if (m[2][2] > m[i][i]) i = 2;
                j = nxt[i];
                k = nxt[j];
                s = sqrt ((m[i][i] - (m[j][j] + m[k][k])) + 1.0);
                q[i] = s * 0.5;
                if (s != 0.0) s = 0.5 / s;
                q[3] = (m[j][k] - m[k][j]) * s;
                q[j] = (m[i][j] + m[j][i]) * s;
                q[k] = (m[i][k] + m[k][i]) * s;
                quat->x = q[0];
                quat->y = q[1];
                quat->z = q[2];
                quat->w = q[3];
        }
}
```

Quaternion-to-Matrix Conversion

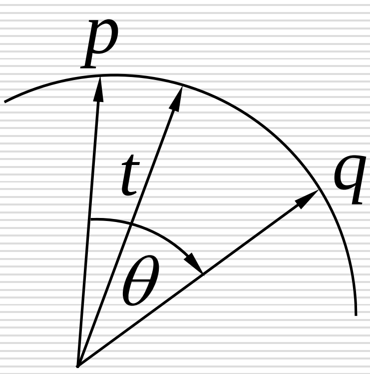
```
QuatToMatrix (QUAT * quat, float m[4][4]) {
    float wx, wy, wz, xx, yy, yz, xy, xz, zz, x2, y2, z2;
    x2 = quat->x + quat->x; y2 = quat->y + quat->y;
    z2 = quat->z + quat->z;
    xx = quat->x * x2; xy = quat->x * y2; xz = quat->x * z2;
    yy = quat->y * y2; yz = quat->y * z2; zz = quat->z * z2;
    wx = quat->w * x2; wy = quat->w * y2; wz = quat->w * z2;
    m[0][0] = 1.0 - (yy + zz); m[1][0] = xy - wz;
    m[2][0] = xz + wy; m[3][0] = 0.0;
    m[0][1] = xy + wz; m[1][1] = 1.0 - (xx + zz);
    m[2][1] = yz - wx; m[3][1] = 0.0;
    m[0][2] = xz - wy; m[1][2] = yz + wx;
    m[2][2] = 1.0 - (xx + yy); m[3][2] = 0.0;
    m[0][3] = 0; m[1][3] = 0;
    m[2][3] = 0; m[3][3] = 1;
}
```

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.