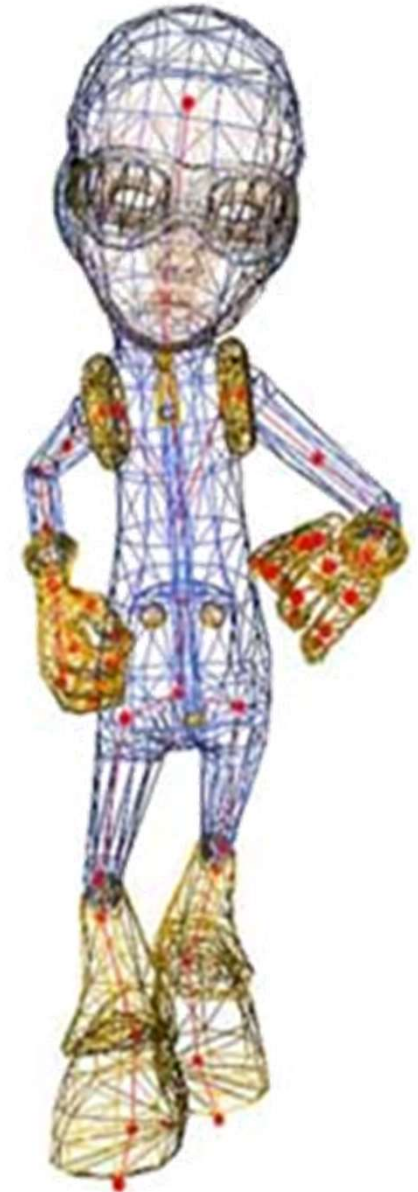
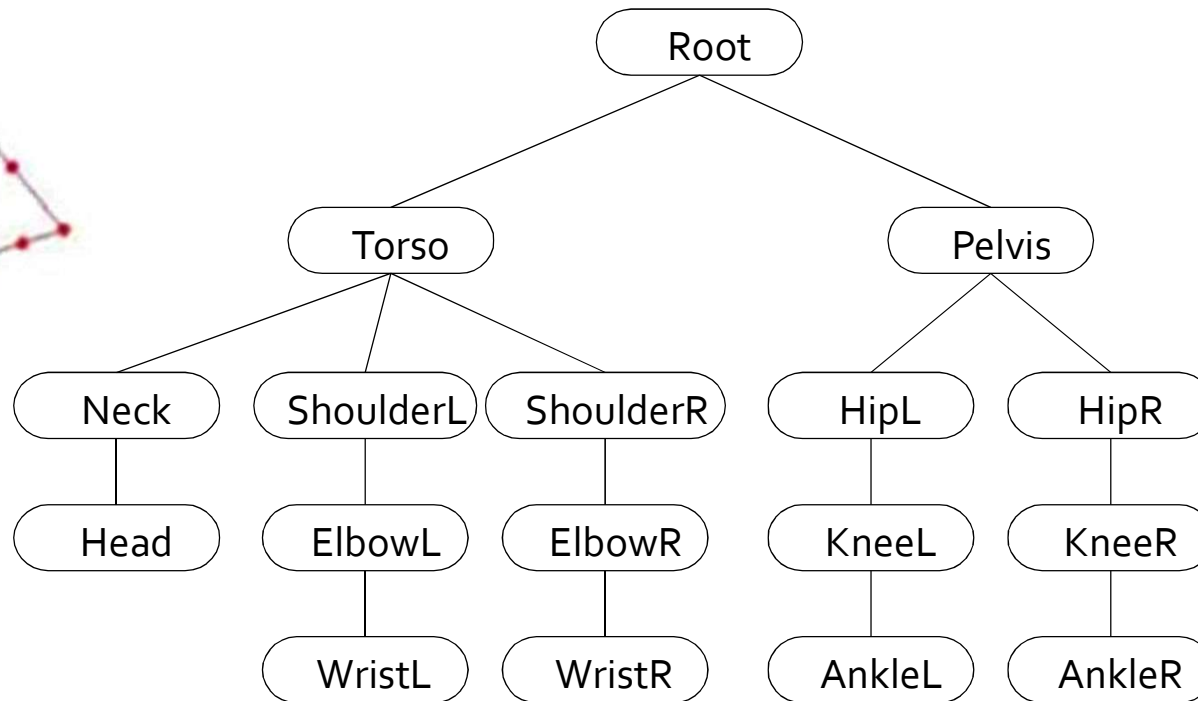
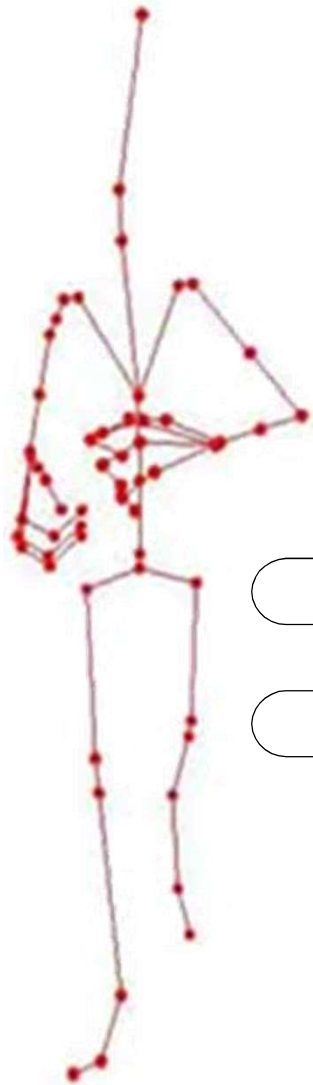


# Transformations

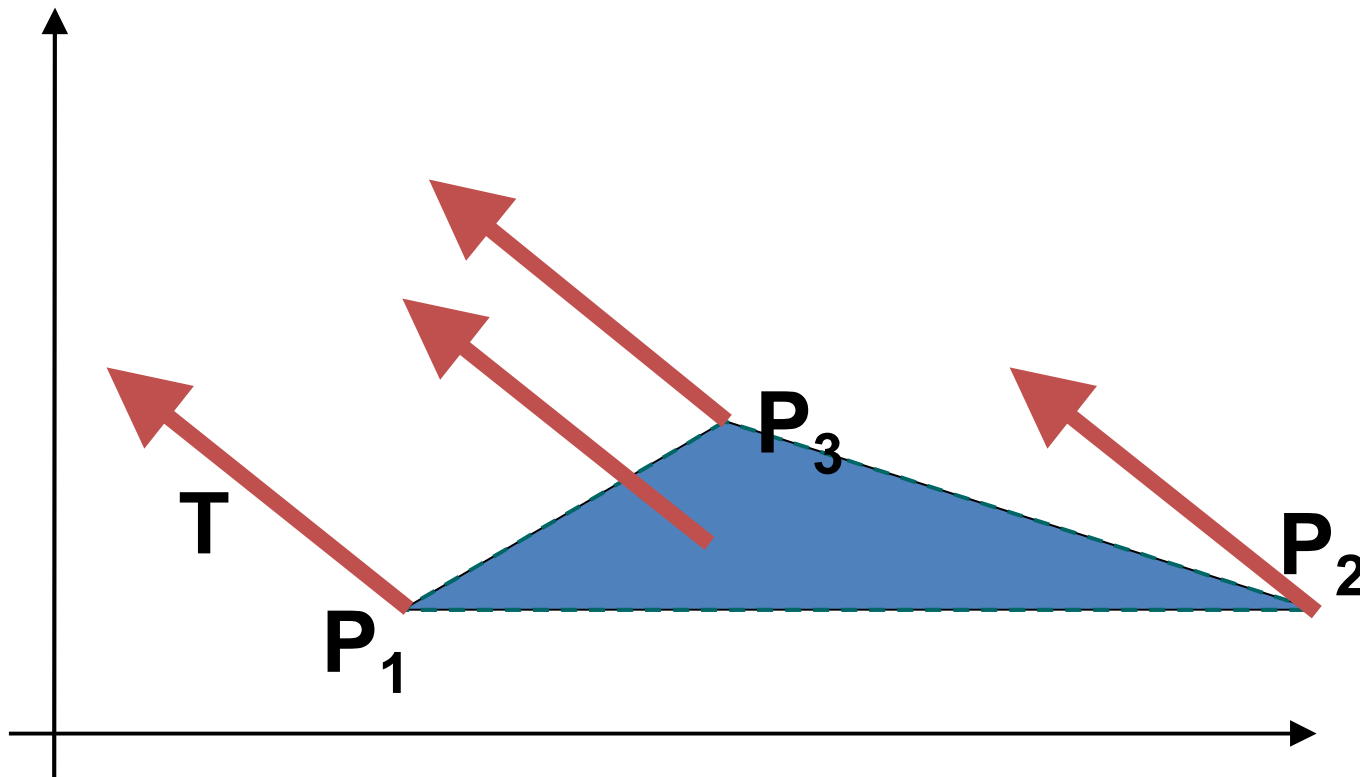


# Hierarchical (Transformation) Models



# Rigid body transformation

- Object transformed by transforming boundary points



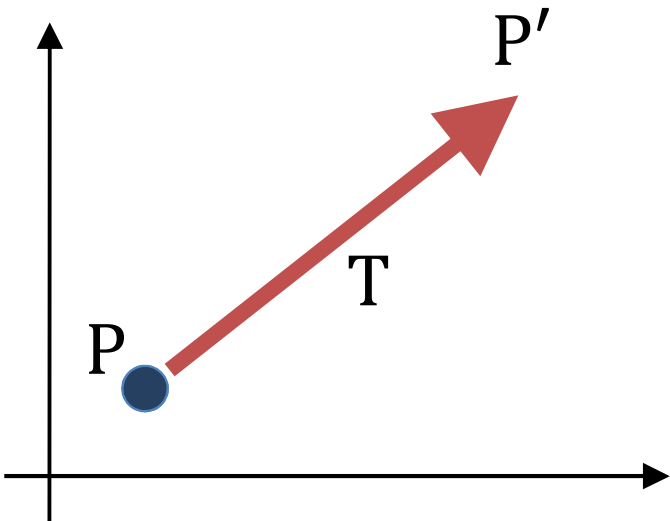
# Translation

- Translating a point from position  $P$  to position  $P'$  with translation vector  $T$

$$P = \begin{pmatrix} x \\ y \end{pmatrix} \quad P' = \begin{pmatrix} x' \\ y' \end{pmatrix} \quad T = \begin{pmatrix} t_x \\ t_y \end{pmatrix}$$

$$x' = x + t_x \quad y' = y + t_y$$

$$P' = P + T$$



# Translation

- For convenience we usually describe objects in relation to their own coordinate system
- We can *translate* or move points to a new position by adding offsets to their coordinates:

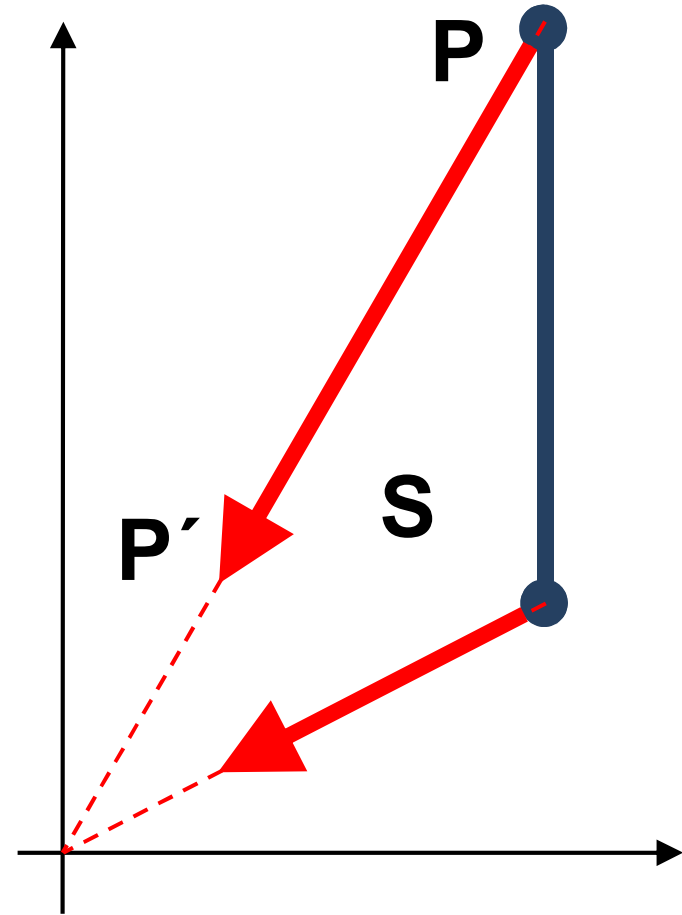
$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \end{pmatrix} \qquad \begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \\ t_z \end{pmatrix}$$

- Note that this translates all points uniformly

# Scaling

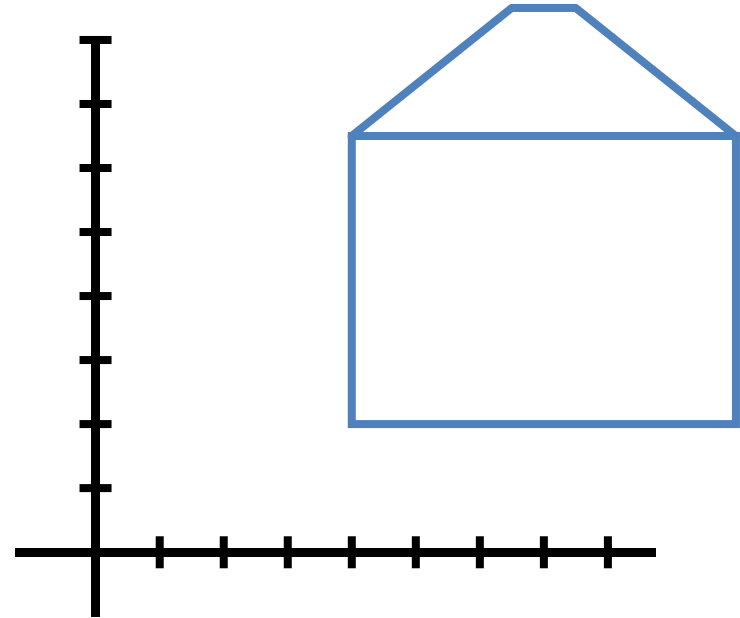
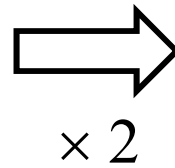
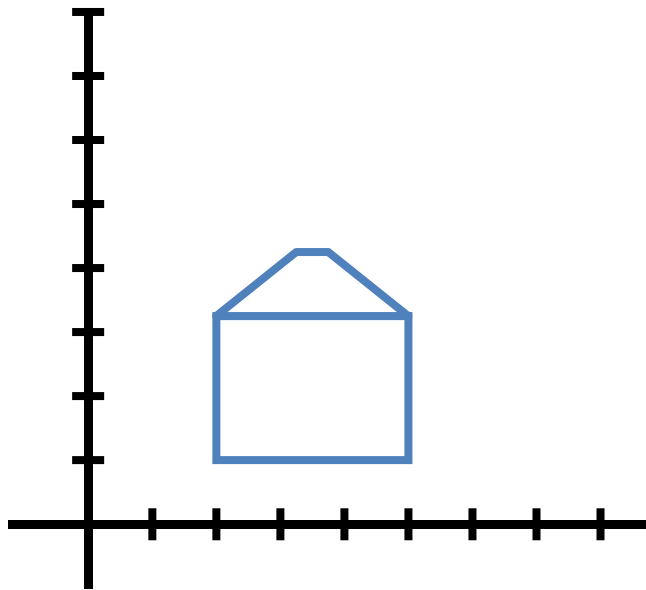
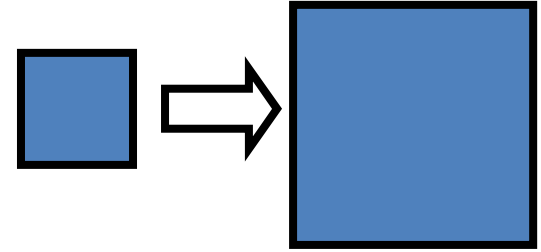
$$x' = x \cdot s_x, \quad y' = y \cdot s_y$$

*example: a line scaled using  $s_x=s_y=0.33$  is reduced in size and moved closer to the coordinate origin*



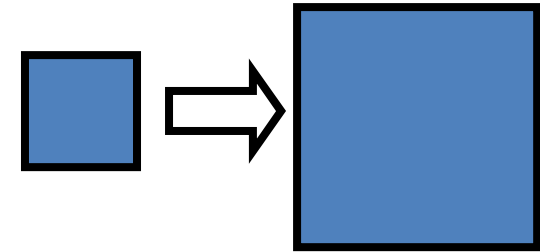
# Scaling

- Uniform scaling:  $S_x = S_y$

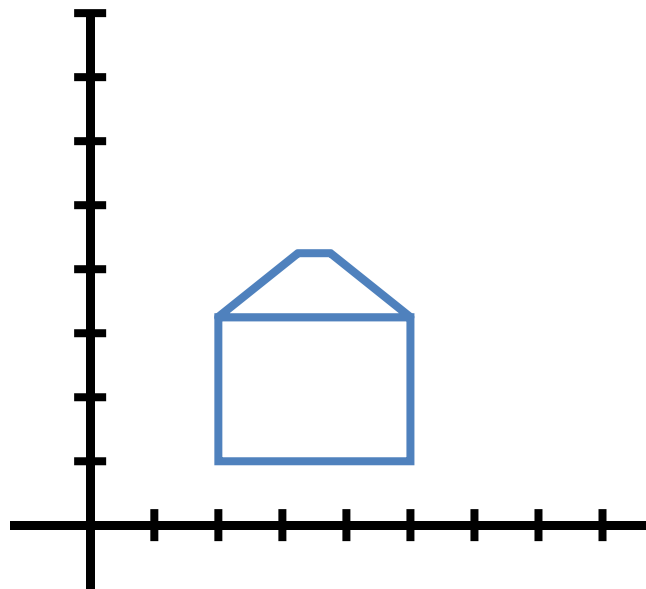
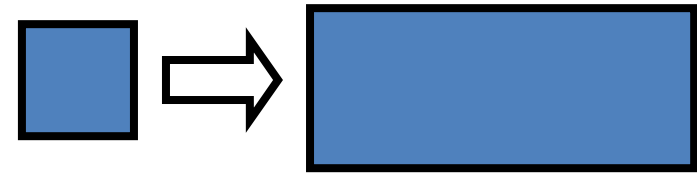



# Scaling

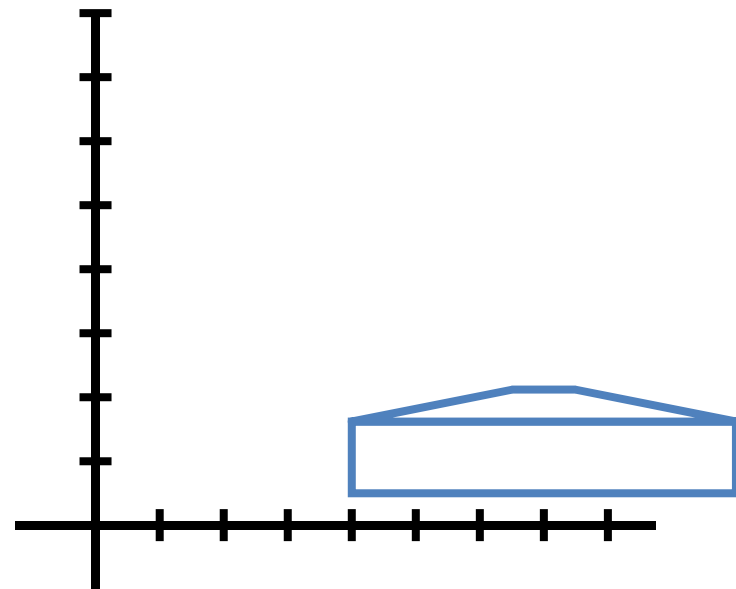
- Uniform scaling:  $S_x = S_y$



- Differential scaling:  $S_x \neq S_y$



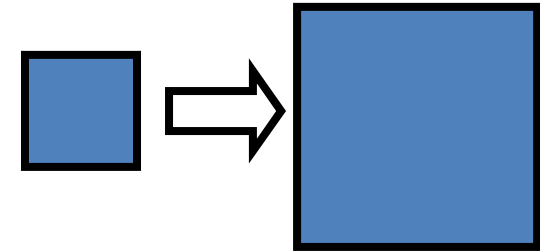
  
 $X \times 2,$   
 $Y \times 0.5$



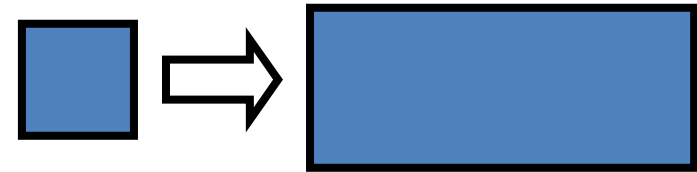


# Scaling

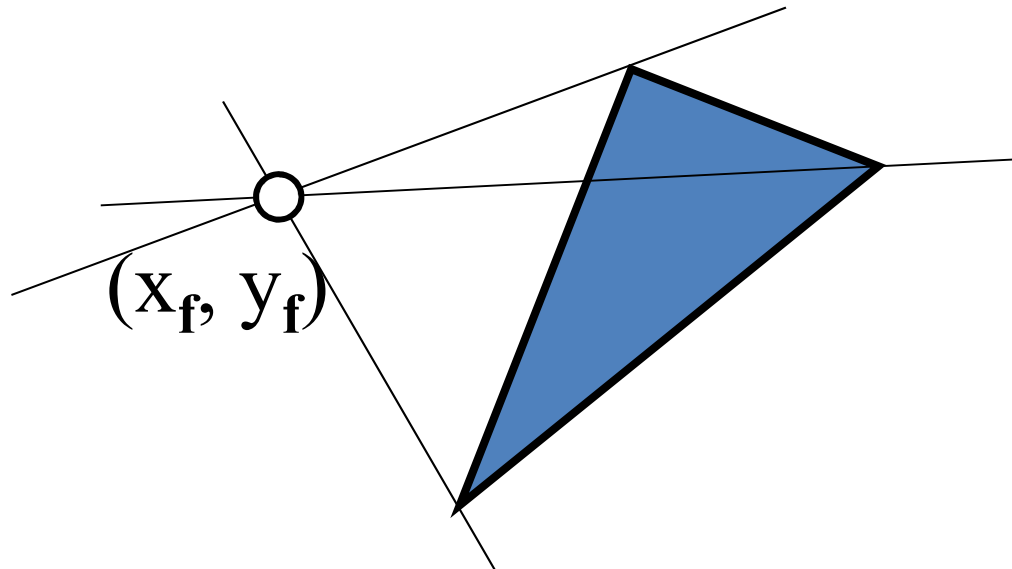
- uniform scaling:  $S_x = S_y$



- differential scaling:  $S_x \neq S_y$

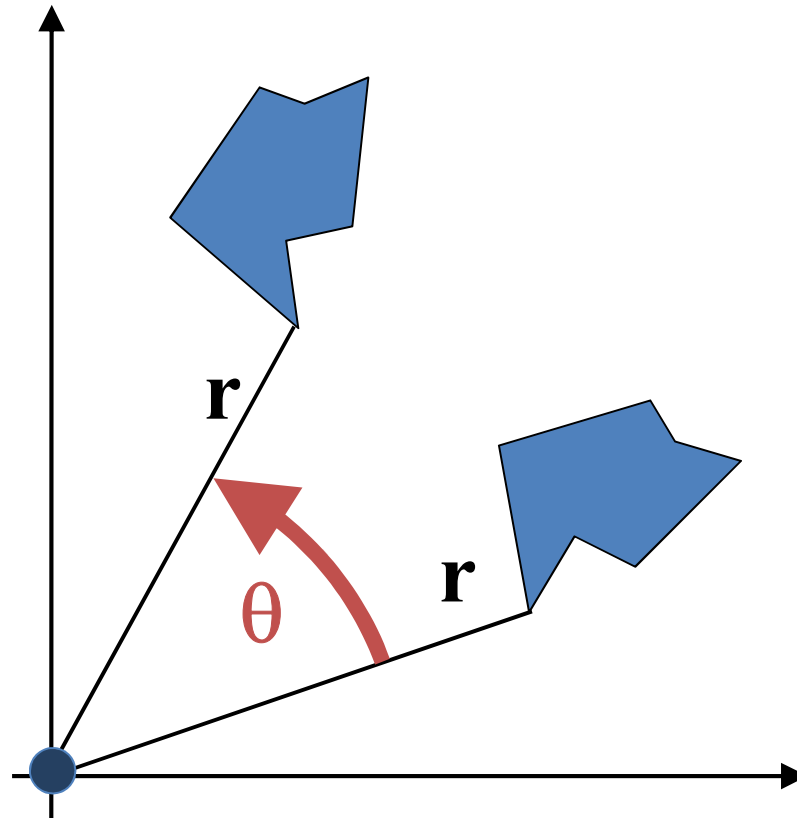


- fixed point:



# Rotation

- Rotation of an object by an angle  $\theta$  around the origin



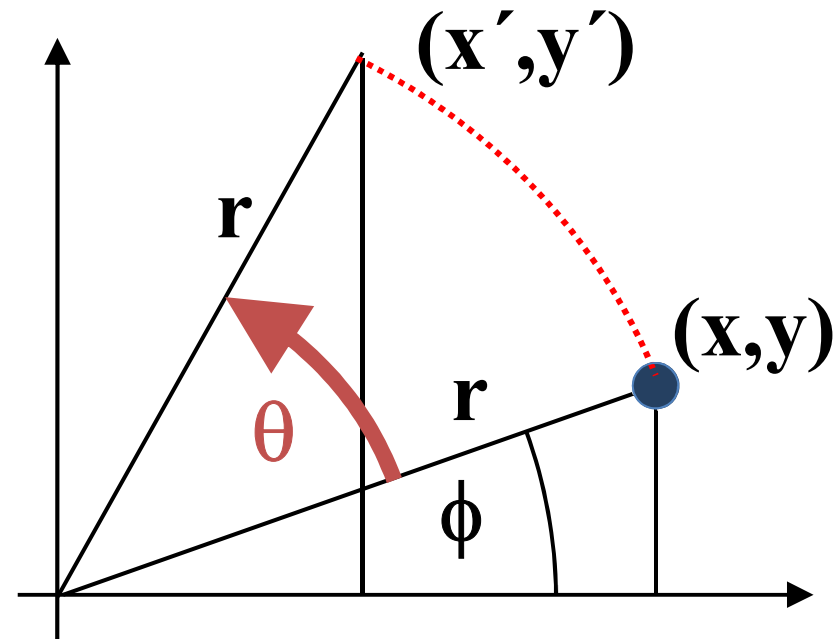
# Rotation

- Positive angle  $\Rightarrow$  ccw rotation

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

$$\begin{aligned} x' &= r \cdot \cos(\phi + \theta) \\ &= \underline{r \cdot \cos \phi} \cdot \cos \theta - \underline{r \cdot \sin \phi} \cdot \sin \theta \\ &= \underline{x} \cdot \cos \theta - \underline{y} \cdot \sin \theta \end{aligned}$$

$$\begin{aligned} y' &= r \cdot \sin(\phi + \theta) \\ &= \underline{r \cdot \cos \phi} \cdot \sin \theta + \underline{r \cdot \sin \phi} \cdot \cos \theta \end{aligned}$$

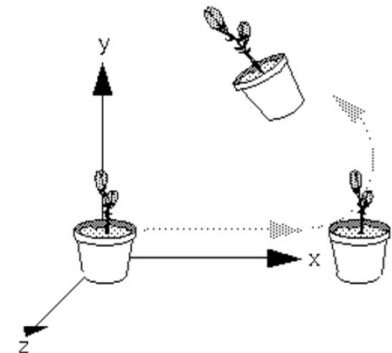
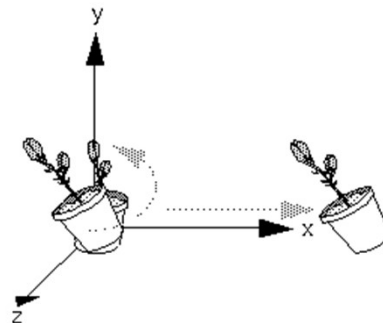


$$\begin{aligned} x' &= x \cdot \cos \theta - y \cdot \sin \theta \\ y' &= x \cdot \sin \theta + y \cdot \cos \theta \end{aligned}$$

# Transformation Matrices

# Why Matrices?

- All transformations representable by a matrix multiplication
  - Uniform way of representing transformations
  - Matrix multiplications are **associative**
    - $(M_1 \cdot M_2) \cdot M_3 = M_1 \cdot (M_2 \cdot M_3)$
    - Composite Transformations can be premultiplied
  - Not **commutative**  $M_1 \cdot M_2 \neq M_2 \cdot M_1$  which is also true for transformations



# Scaling Matrix

- Operation

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} s_x x \\ s_y y \end{pmatrix} \quad \begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} s_x x \\ s_y y \\ s_z z \end{pmatrix}$$

- Matrix form

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \underbrace{\begin{pmatrix} s_x & 0 \\ 0 & s_y \end{pmatrix}}_{\text{scaling matrix}} \begin{pmatrix} x \\ y \end{pmatrix} \quad \begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \underbrace{\begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & s_z \end{pmatrix}}_{\text{scaling matrix}} \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

# 2-D Rotation

- Operation

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

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

- Matrix form

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

- 3-D is more complicated

- Need to specify an
- Simple cases: rotation about X, Y, Z axes

# 3-D Rotation

- *What does the 3-D rotation matrix look like for a rotation about the Z-axis?*
  - Build it coordinate-by-coordinate

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



# 3-D Rotation

- *What does the 3-D rotation matrix look like for a rotation about the Y-axis?*
  - Build it coordinate-by-coordinate

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

# 3-D Rotation

- *What does the 3-D rotation matrix look like for a rotation about the X-axis?*
  - Build it coordinate-by-coordinate

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

# Rotation Matrices

- Remember basis!
- What does a z-rotation by  $0^\circ$ ,  $90^\circ$  do to the basis?  
(draw transformation of basis vectors)

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

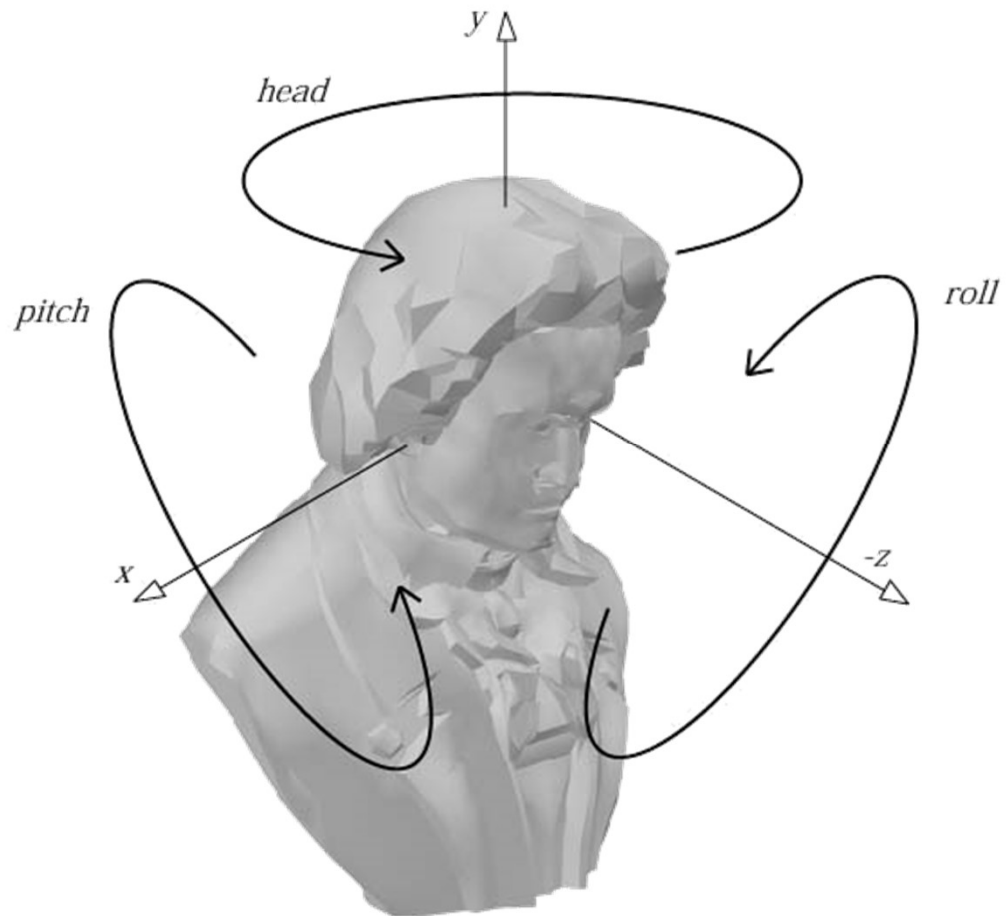
# Rotation Matrices

- Rotation matrix is **orthogonal**
  - Columns/rows linearly independent
  - Columns/rows sum to 1
- The inverse of an orthogonal matrix is just its transpose:

$$\begin{bmatrix} a & b & c \\ d & e & f \\ h & i & j \end{bmatrix}^{-1} = \begin{bmatrix} a & b & c \\ d & e & f \\ h & i & j \end{bmatrix}^T = \begin{bmatrix} a & d & h \\ b & e & i \\ c & f & j \end{bmatrix}$$

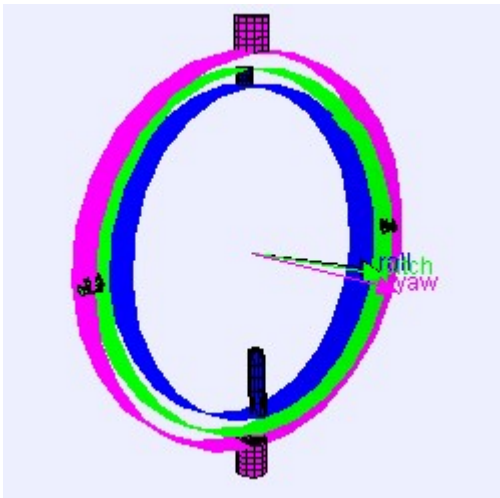
# The Euler Transform

- $E(h, p, r) = \mathbf{R}_z(r)\mathbf{R}_x(p)\mathbf{R}_y(h)$
- $h$  = head
- $p$  = pitch
- $r$  = roll
- Just 3 rotations



# Gimbal Lock

- Usually 3 rotation matrices allow for arbitrary orientation in space
- Case where rank of  $E(h, p, r)$  drops below 3
- Example  $h = 0, p = \frac{\pi}{2}$
- Roll and yaw have same effect

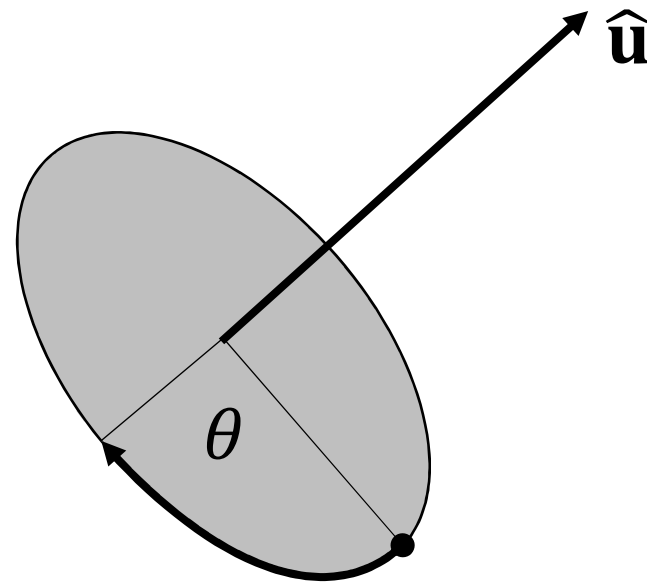


# Quaternions

- $\mathbf{q} = (q_x, q_y, q_z, q_w) = (\mathbf{q}_v, q_w)$   
 $= i q_x + j q_y + k q_z + q_w$
- Extension of imaginary numbers (not commutative)  
 $ii = jj = kk = -1$   
 $ij = -ji = k$   
 $jk = -kj = i$   
 $ki = -ik = j$
- Focus on unit quaternions  
 $\|\hat{\mathbf{q}}\| = q_x^2 + q_y^2 + q_z^2 + q_w^2 = 1$

# Quaternions and Rotations

- A rotation of an angle  $\theta$  about an axis  $\hat{\mathbf{u}} = (x, y, z)$  can be represented by  $\hat{\mathbf{q}} = (\hat{\mathbf{u}} \sin \frac{\theta}{2}, \cos \frac{\theta}{2})$ 
  - Compact
  - Avoids Gimbal Lock



- $\mathbf{q}^* := (-\mathbf{q}_v, q_w)$
- A (pure) vector  $\mathbf{p} = (p_x, p_y, p_z, 0) = \mathbf{i}p_x + \mathbf{j}p_y + \mathbf{k}p_z$  can be rotated by  $\hat{\mathbf{q}}\mathbf{p}\hat{\mathbf{q}}^*$



# Translation Matrices?

- But how to represent translation as a matrix?
- Answer: with **homogeneous coordinates**

# Homogeneous Coordinates

# Homogeneous Coordinates

- *Homogeneous coordinates*: represent coordinates with an additional dimension

- Points:

$$\begin{pmatrix} x \\ y \end{pmatrix} \cong \begin{pmatrix} x/w \\ y/w \\ 1 \end{pmatrix} = \begin{pmatrix} x \\ y \\ w \end{pmatrix}$$

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} \cong \begin{pmatrix} x/w \\ y/w \\ z/w \\ 1 \end{pmatrix} = \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix}$$

Directions:

$$\begin{pmatrix} x \\ y \end{pmatrix} \cong \begin{pmatrix} x \\ y \\ 0 \end{pmatrix}$$

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} \cong \begin{pmatrix} x \\ y \\ z \\ 0 \end{pmatrix}$$

# Translation Matrix

- Operation

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \end{pmatrix} \qquad \begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \\ t_z \end{pmatrix}$$

- Matrix form

$$\begin{pmatrix} x + t_x \\ y + t_y \\ z + t_z \\ w \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix}$$

# Homogeneous Coordinates

- Homogeneous coordinates seem unintuitive, but they make graphics operations much easier
- Our transformation matrices are now 4x4:

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

# Homogeneous Coordinates

- Homogeneous coordinates seem unintuitive, but they make graphics operations much easier
- Our transformation matrices are now 4x4:

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

# Homogeneous Coordinates

- Homogeneous coordinates seem unintuitive, but they make graphics operations much easier
- Our transformation matrices are now 4x4:

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

# Homogeneous Coordinates

- Homogeneous coordinates seem unintuitive, but they make graphics operations much easier
- Our transformation matrices are now 4x4:

$$\mathbf{S} = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



# Inverse Matrices

- translation

$$T^{-1}(t_x, t_y) = T(-t_x, -t_y)$$

- rotation

$$R^{-1}(\theta) = R(-\theta)$$

- scaling

$$S^{-1}(s_x, s_y) = S(1/s_x, 1/s_y)$$

# Composite Transformations

- How to rotate/scale around a given point?
- How to create hierarchical transformations?

# Composite Transformations

n transformations are applied after each other on a point P, these transformations are represented by matrices  $M_1, M_2, \dots, M_n$ .

$$P' = M_1 \times P$$

$$P'' = M_2 \times P'$$

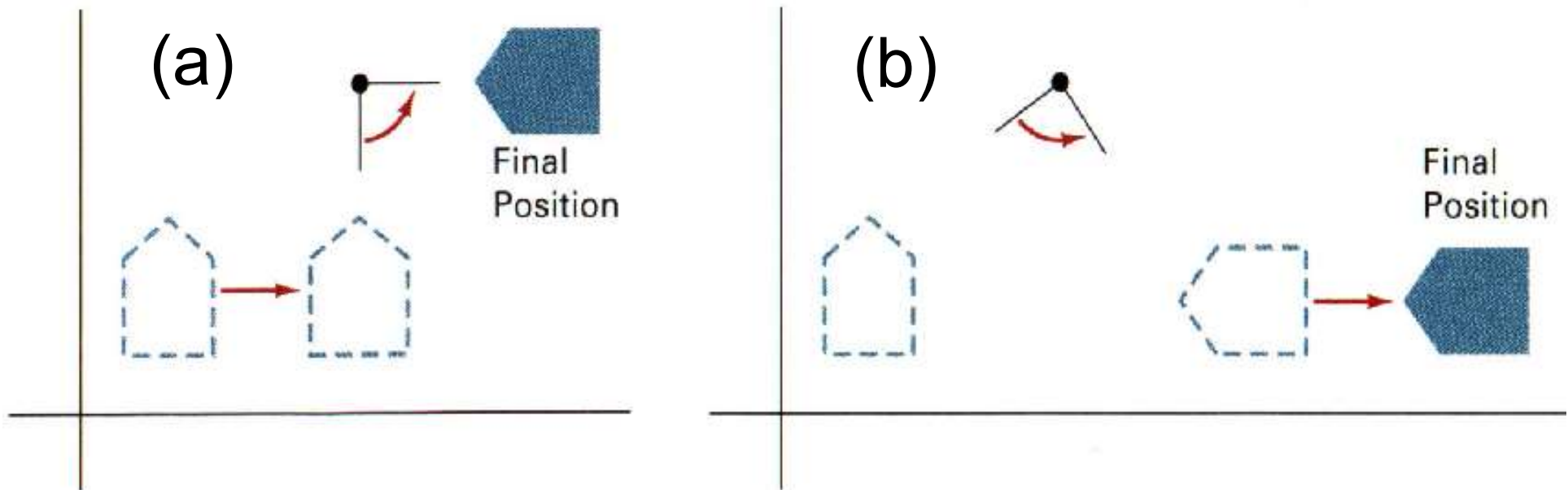
...

$$P^{(n)} = M_n \times P^{(n-1)}$$

shorter:  $P^{(n)} = (M_n \times \dots (M_2 \times (M_1 \times P)) \dots )$

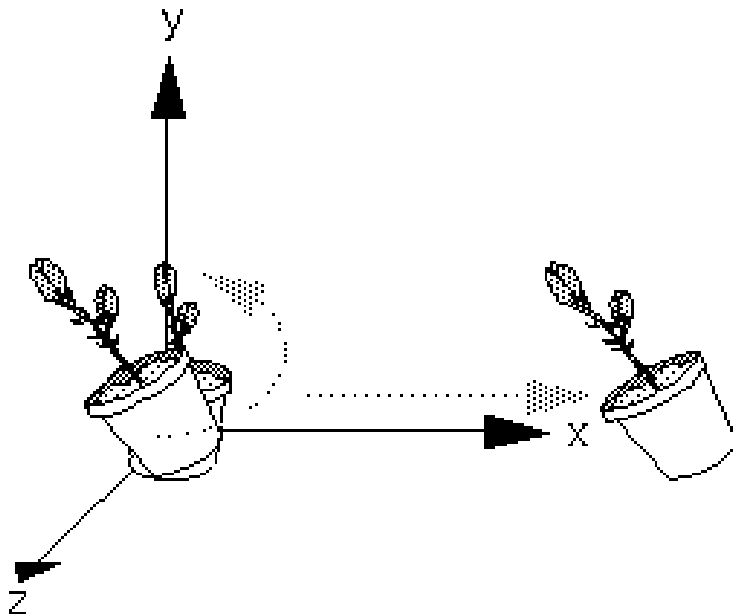
# Transformations are not commutative!

- Reversing the order in which a sequence of transformations is performed may affect the transformed position of an object.
- In (a), an object is first translated, then rotated. In (b), the object is rotated first, then translated.

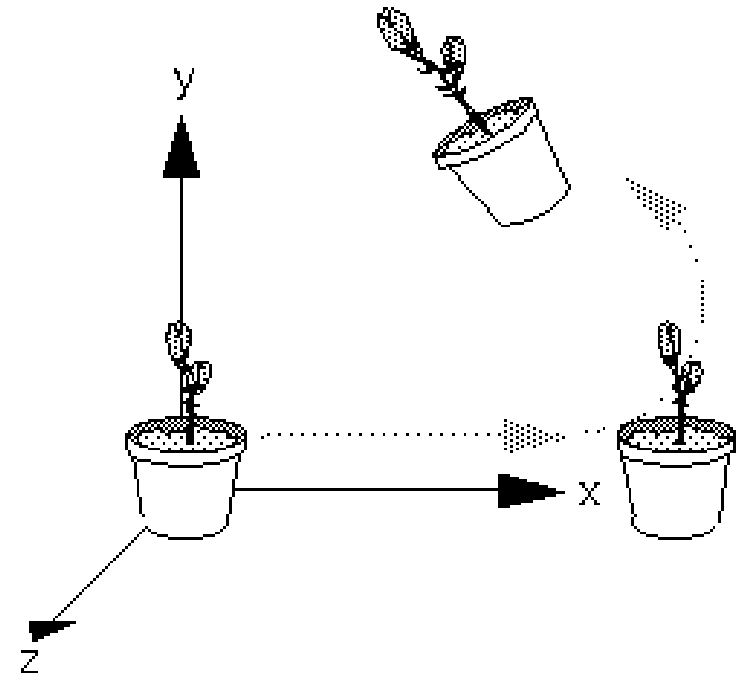


# Transformation Order

- Order matters



$\text{Trans} * \text{Rot} * v$



$\text{Rot} * \text{Trans} * v$

# Translation Matrices

- Now that we can represent translation as a matrix, we can composite it with other transformations
- Ex: rotate  $90^\circ$  about **X**, then 10 units down **Z**:

$$\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 10 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(90^\circ) & -\sin(90^\circ) & 0 \\ 0 & \sin(90^\circ) & \cos(90^\circ) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

# Translation Matrices

- Now that we can represent translation as a matrix, we can composite it with other transformations
- Ex: rotate  $90^\circ$  about **X**, then 10 units down **Z**:

$$\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 10 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

# Translation Matrices

- Now that we can represent translation as a matrix, we can composite it with other transformations
- Ex: rotate  $90^\circ$  about **X**, then 10 units down **Z**:

$$\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 10 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

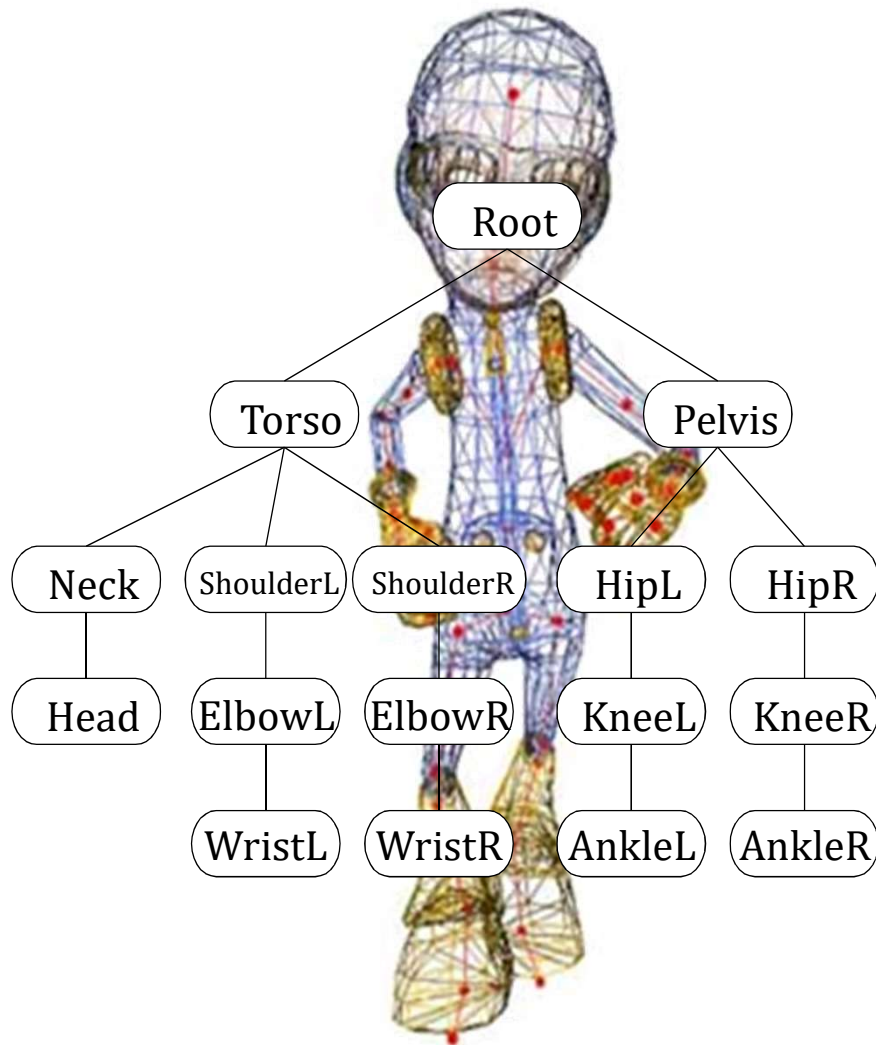


# Translation Matrices

- Now that we can represent translation as a matrix, we can composite it with other transformations
- Ex: rotate  $90^\circ$  about **X**, then 10 units down **Z**:

$$\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = \begin{bmatrix} x \\ -z \\ y + 10 \\ w \end{bmatrix}$$

# Hierarchical (Transformation) Models



- Transformations for vertices of right toes
  - **Root \* Pelvis \* HipR \* KneeR \* AnkleR \* v**

# Column-major vs Row-major

- Both are widely used
- Just a transpose  $\mathbf{T}\mathbf{v} \hat{=} \mathbf{v}^T \mathbf{T}^T$  or  $\mathbf{RST}\mathbf{v} \hat{=} \mathbf{v}^T \mathbf{T}^T \mathbf{S}^T \mathbf{R}^T$
- Storage
  - OpenGL expects matrices stored in column-major fashion
  - Matrices stored in row-major are transposed if feed to OpenGL