Composite **Transformations**

Unit 2-Lecture 3

Composite Transformations

- Composite 2D Translation

$$T = \mathbf{T}(t_{x1}, t_{y1}) \cdot \mathbf{T}(t_{x2}, t_{y2})$$

$$= \mathbf{T}(t_{x1} + t_{x2}, t_{y1} + t_{y2})$$

$$\begin{pmatrix} 1 & 0 & t_{x2} \\ 0 & 1 & t_{y2} \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & t_{x1} \\ 0 & 1 & t_{y1} \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & t_{x1} + t_{x2} \\ 0 & 1 & t_{y1} + t_{y2} \\ 0 & 0 & 1 \end{pmatrix}$$

Composite Transformations

- Composite 2D Scaling

$$T = \mathbf{S}(s_{x1}, s_{y1}) \cdot \mathbf{S}(s_{x2}, s_{y2})$$
$$= \mathbf{S}(s_{x1}s_{x2}, s_{y1}s_{y2})$$

$$\begin{pmatrix} s_{x2} & 0 & 0 \\ 0 & s_{y2} & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} s_{x1} & 0 & 0 \\ 0 & s_{y1} & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} s_{x1} \cdot s_{x2} & 0 & 0 \\ 0 & s_{y1} \cdot s_{y2} & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Composite Transformations

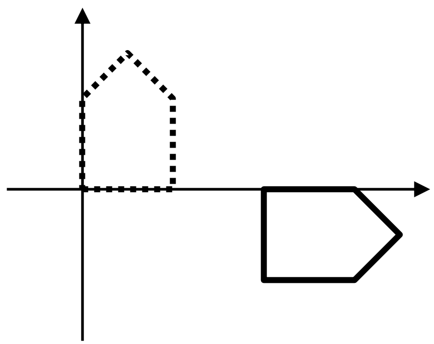
- Composite 2D Rotation

$$T = \mathbf{R}(\theta_2) \cdot \mathbf{R}(\theta_1)$$
$$= \mathbf{R}(\theta_2 + \theta_1)$$

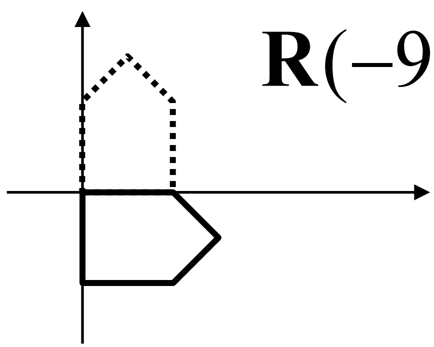
$$\begin{pmatrix} \cos\theta_2 & -\sin\theta_2 & 0 \\ \sin\theta_2 & \cos\theta_2 & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} \cos\theta_1 & -\sin\theta_1 & 0 \\ \sin\theta_1 & \cos\theta_1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} \cos(\theta_2+\theta_1) & -\sin(\theta_2+\theta_1) & 0 \\ \sin(\theta_2+\theta_1) & \cos(\theta_2+\theta_1) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Composing Transformations

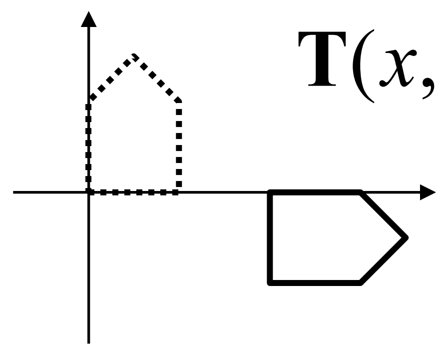
- Suppose we want,



- We have to compose two transformations



$\mathbf{R}(-90°)$



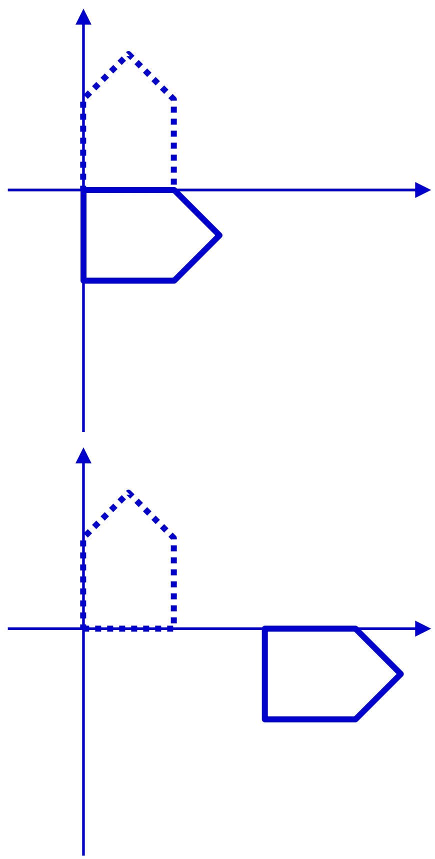
$\mathbf{T}(x,3)$

Composing Transformations

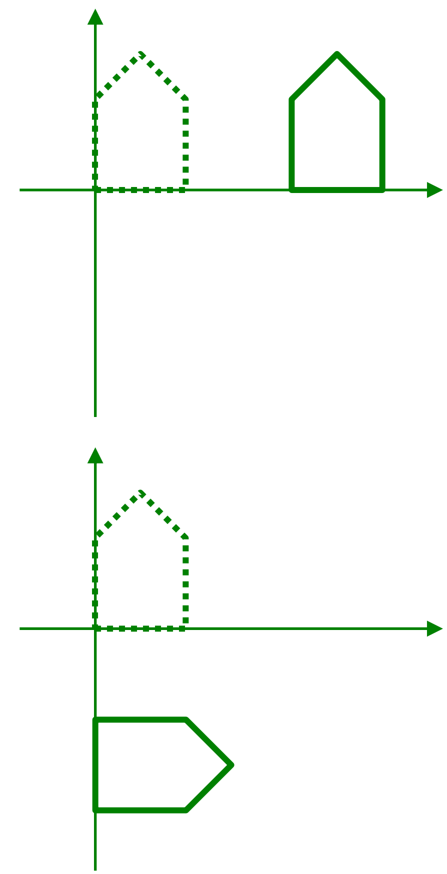
- Matrix multiplication is not commutative

$$\mathbf{T}(x,3) \cdot \mathbf{R}(-90°) \neq \mathbf{R}(-90°)\mathbf{T}(x,3)$$

Translation followed by rotation



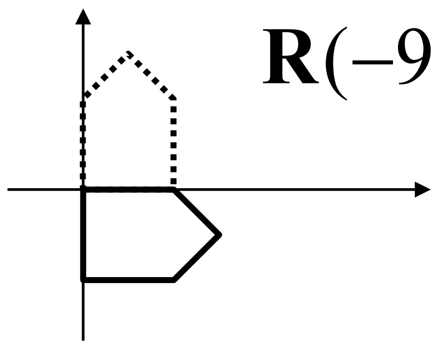
Translation followed by rotation



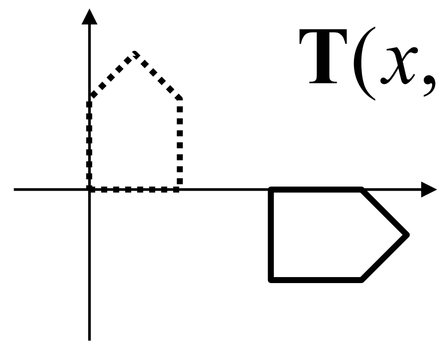
Composing Transformations

$$T = \mathbf{T}(x,3) \cdot \mathbf{R}(-90°)$$ (Column major convention)

– R-to-L : interpret operations w.r.t. fixed coordinates

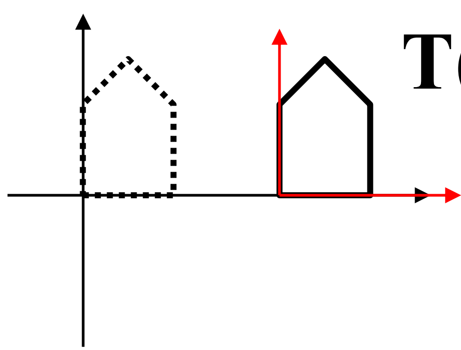


$\mathbf{R}(-90°)$

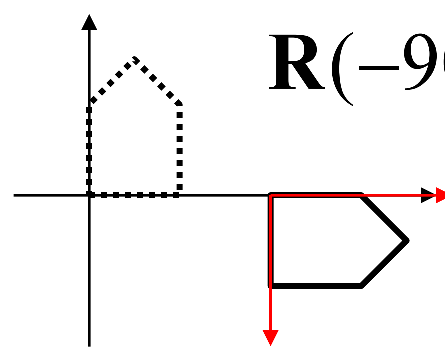


$\mathbf{T}(x,3)$

– L-to-R : interpret operations w.r.t local coordinates



$\mathbf{T}(x,3)$



$\mathbf{R}(-90°)$

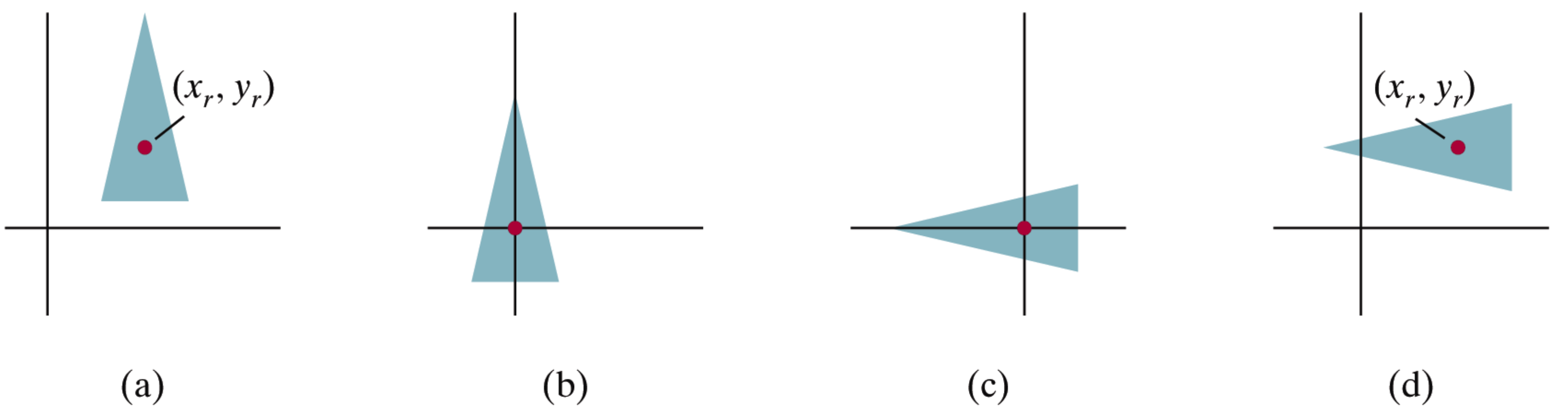
Pivot-Point Rotation

- Rotation with respect to a pivot point (x,y)

$$T(x,y)\cdot R(\theta)\cdot T(-x,-y)$$

$$=\begin{pmatrix}1 & 0 & x\\ 0 & 1 & y\\ 0 & 0 & 1\end{pmatrix}\cdot\begin{pmatrix}\cos\theta & -\sin\theta & 0\\ \sin\theta & \cos\theta & 0\\ 0 & 0 & 1\end{pmatrix}\cdot\begin{pmatrix}1 & 0 & -x\\ 0 & 1 & -y\\ 0 & 0 & 1\end{pmatrix}$$

$$=\begin{pmatrix}\cos\theta & -\sin\theta & x(1-\cos\theta)+y\sin\theta\\ \sin\theta & \cos\theta & y(1-\cos\theta)-x\sin\theta\\ 0 & 0 & 1\end{pmatrix}$$



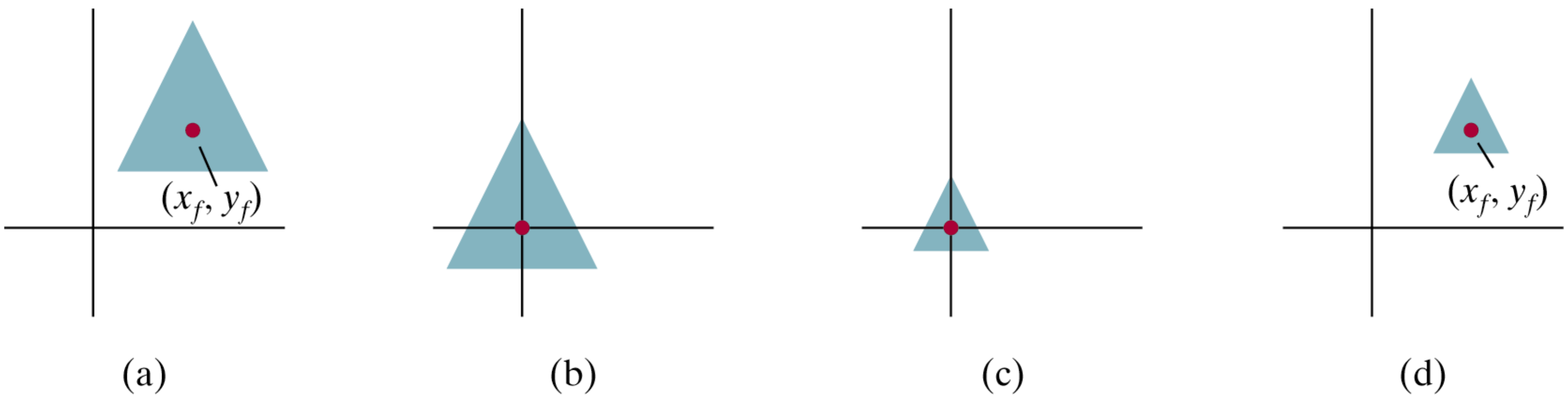
Fixed-Point Scaling

- Scaling with respect to a fixed point (x,y)

$$T(x,y)\cdot S(s_x,s_y)\cdot T(-x,-y)$$

$$=\begin{pmatrix}1 & 0 & x\\ 0 & 1 & y\\ 0 & 0 & 1\end{pmatrix}\cdot\begin{pmatrix}s_x & 0 & 0\\ 0 & s_y & 0\\ 0 & 0 & 1\end{pmatrix}\cdot\begin{pmatrix}1 & 0 & -x\\ 0 & 1 & -y\\ 0 & 0 & 1\end{pmatrix}$$

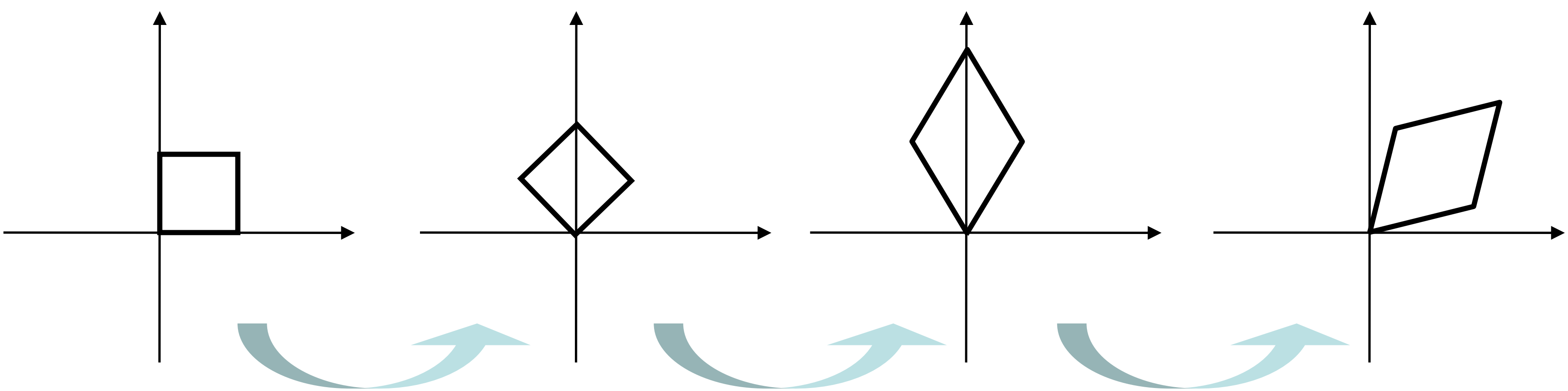
$$=\begin{pmatrix}s_x & 0 & (1-s_x)\cdot x\\ 0 & s_y & (1-s_y)\cdot y\\ 0 & 0 & 1\end{pmatrix}$$



Scaling Direction

- Scaling along an arbitrary axis

$$R^{-1}(\theta) \cdot S(s_x, s_y) \cdot R(\theta)$$



$R(\theta)$ $\quad$ $S(s_x, s_y)$ $\quad$ $R^{-1}(\theta)$

Properties of Affine Transformations

- Any ***affine transformation*** between 3D spaces can be represented as a combination of a ***linear transformation*** followed by ***translation***
- An affine transf. maps ***lines*** to ***lines***
- An affine transf. maps ***parallel lines*** to ***parallel lines***
- An affine transf. preserves ***ratios of distance*** along a line
- An affine transf. does not preserve absolute distances and angles

Review of Affine Frames

- A ***frame*** is defined as a set of vectors $\{\mathbf{v}_i \mid i=1, \ldots, N\}$ and a point $\mathbf{o}$
  - Set of vectors $\{\mathbf{v}_i\}$ are bases of the associate vector space
  - $\mathbf{o}$ is an origin of the frame
  - N is the dimension of the affine space
  - Any point $\mathbf{p}$ can be written as

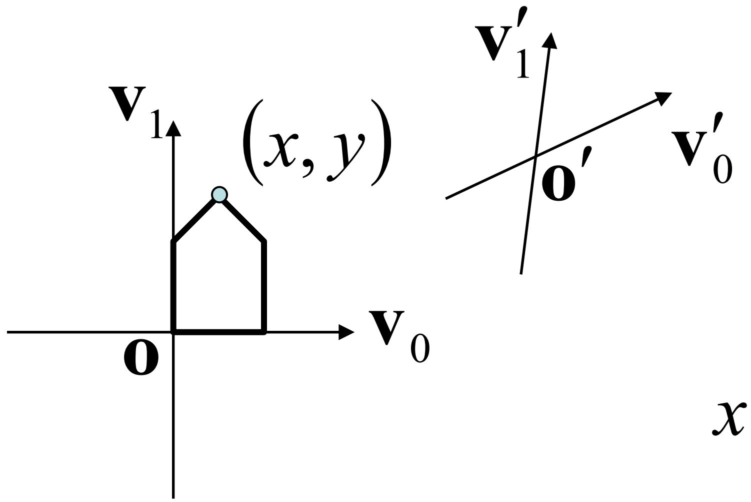
$$\mathbf{p} = \mathbf{o} + c_1\mathbf{v}_1 + c_2\mathbf{v}_2 + \cdots + c_N\mathbf{v}_N$$

  - Any vector $\mathbf{v}$ can be written as

$$\mathbf{v} = c_1\mathbf{v}_1 + c_2\mathbf{v}_2 + \cdots + c_N\mathbf{v}_N$$

Changing Frames

- Affine transformations as a change of frame

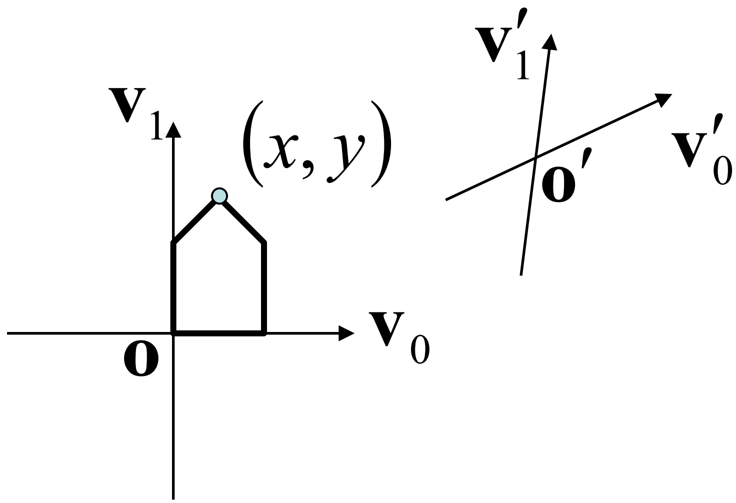


$$x'\mathbf{v}_0' + y'\mathbf{v}_1' + \mathbf{o}' = x\mathbf{v}_0 + y\mathbf{v}_1 + \mathbf{o}$$

$$\begin{pmatrix} \mathbf{v}_0' & \mathbf{v}_1' & \mathbf{o}' \end{pmatrix} \begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} \mathbf{v}_0 & \mathbf{v}_1 & \mathbf{o} \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

Changing Frames

- Affine transformations as a change of frame



$$\mathbf{v}_0 = a_0\mathbf{v}_0' + a_1\mathbf{v}_1'$$

$$\mathbf{v}_1 = b_0\mathbf{v}_0' + b_1\mathbf{v}_1'$$

$$\mathbf{o} = c_0\mathbf{v}_0' + c_1\mathbf{v}' + \mathbf{o}'$$

$$\begin{pmatrix}\mathbf{v}_0' & \mathbf{v}_1' & \mathbf{o}\end{pmatrix}\begin{pmatrix}x'\\ y'\\ 1\end{pmatrix} = \begin{pmatrix}\mathbf{v}_0' & \mathbf{v}_1' & \mathbf{o}\end{pmatrix}\begin{pmatrix}a_0 & b_0 & c_0\\ a_1 & b_1 & c_1\\ 0 & 0 & 1\end{pmatrix}\begin{pmatrix}x\\ y\\ 1\end{pmatrix}$$

$$\begin{pmatrix}x'\\ y'\\ 1\end{pmatrix} = \begin{pmatrix}a_0 & b_0 & c_0\\ a_1 & b_1 & c_1\\ 0 & 0 & 1\end{pmatrix}\begin{pmatrix}x\\ y\\ 1\end{pmatrix}$$

Changing Frames

- In case the xyz system has standard bases

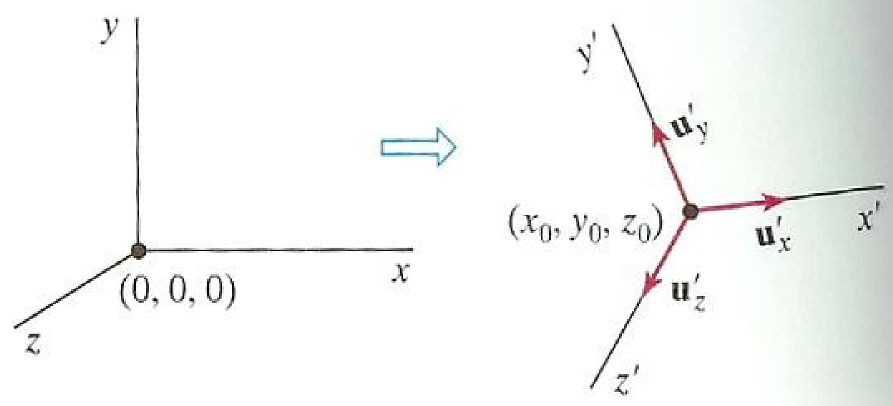


FIGURE 5-54 An $x'y'z'$ coordinate system defined within an xyz system. A scene description is transferred to the new coordinate reference using a transformation sequence that superimposes the $x'y'z'$ frame on the xyz axes.

Rigid Transformations

- A ***rigid transformation*** T is a mapping between affine spaces
  - T maps vectors to vectors, and points to points
  - T preserves distances between all points
  - T preserves cross product for all vectors (to avoid reflection)
- In 3-spaces, T can be represented as

$$T(\mathbf{p}) = \mathbf{R}_{3\times 3}\mathbf{p}_{3\times 1} + \mathbf{T}_{3\times 1}, \quad \text{where}$$

$$\mathbf{R}\mathbf{R}^T = \mathbf{R}^T\mathbf{R} = \mathbf{I} \quad \text{and} \quad \det \mathbf{R} = 1$$

Rigid Body Rotation

- Rigid body transformations allow only rotation and translation

- Rotation matrices form SO(3)
  - Special orthogonal group

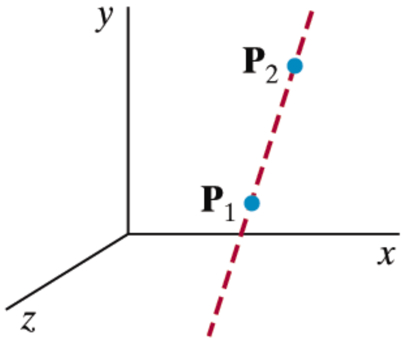
$$\mathbf{R}\mathbf{R}^T = \mathbf{R}^T\mathbf{R} = \mathbf{I}$$ (Distance preserving)

$$\det \mathbf{R} = 1$$ (No reflection)

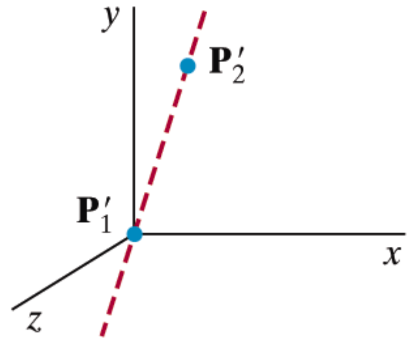
Rigid Body Rotation

- R is normalized
  - The squares of the elements in any row or column sum to 1
- R is orthogonal $\mathbf{R}\mathbf{R}^T = \mathbf{R}^T\mathbf{R} = \mathbf{I}$
  - The dot product of any pair of rows or any pair columns is 0
- The rows (columns) of R correspond to the vectors of the principle axes of the rotated coordinate frame

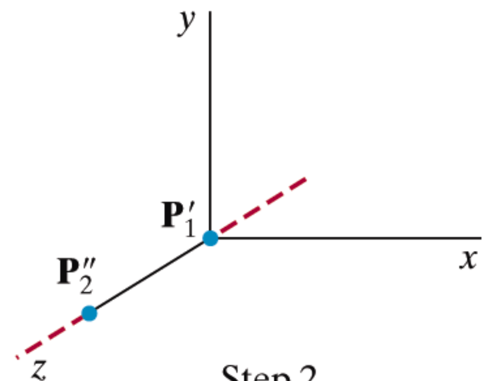
3D Rotation About Arbitrary Axis



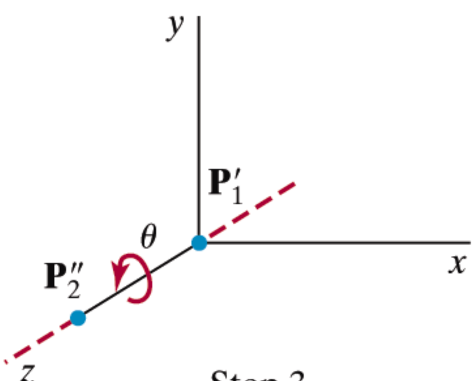
Initial
Position



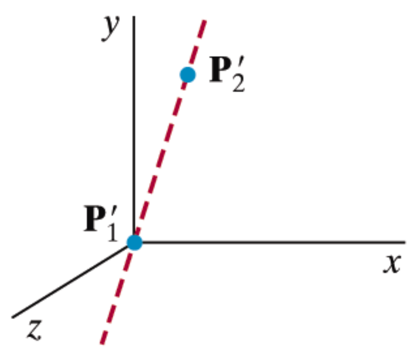
Step 1
Translate
$\mathbf{P}_1$ to the Origin



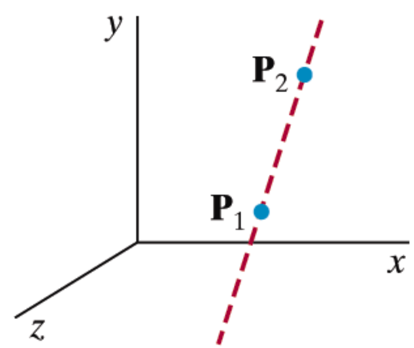
Step 2
Rotate $\mathbf{P}_2'$
onto the z Axis



Step 3
Rotate the
Object Around the
z Axis



Step 4
Rotate the Axis
to its Original
Orientation



Step 5
Translate the
Rotation Axis
to its Original
Position

3D Rotation About Arbitrary Axis

1. Translation : rotation axis passes through the origin $T(-x_1, -y_1, -z_1)$

2. Make the rotation axis on the z-axis

$$R_x(\alpha) \cdot R_y(\beta)$$

3. Do rotation $R_z(\theta)$

4. Rotation & translation

$$T^{-1} \cdot R_y^{-1}(\beta) \cdot R_x^{-1}(\alpha)$$

3D Rotation About Arbitrary Axis

- Rotate **u** onto the *z*-axis

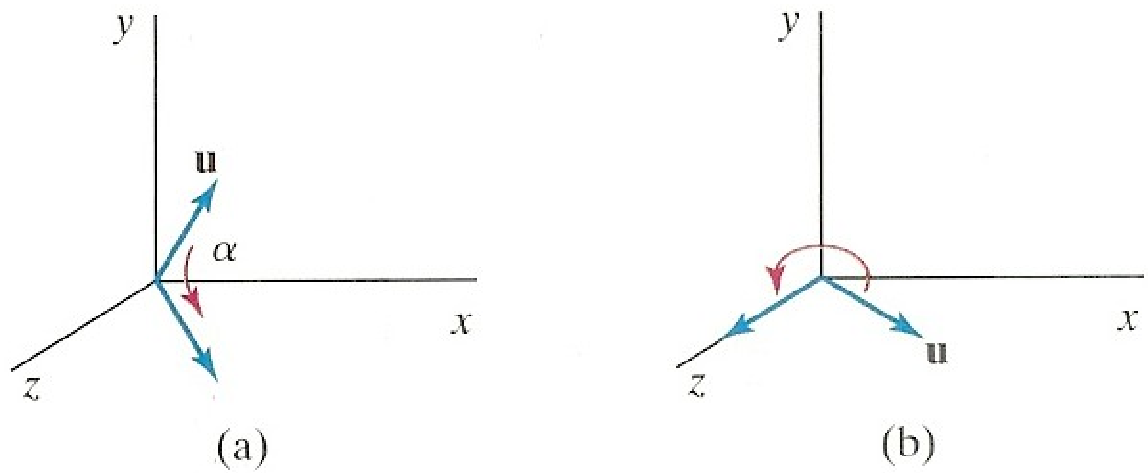


FIGURE 5-45 Unit vector **u** is rotated about the x axis to bring it into the xz plane (a), then it is rotated around the y axis to align it with the z axis (b).

3D Rotation About Arbitrary Axis

- Rotate **u** onto the z-axis
  - **u'**: Project **u** onto the yz-plane to compute angle α
  - **u''**: Rotate **u** about the x-axis by angle α
  - Rotate **u''** onto the z-axis

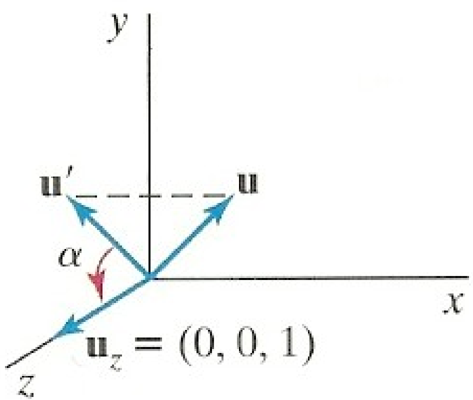


FIGURE 5-46 Rotation of **u** around the x axis into the xz plane is accomplished by rotating $\mathbf{u}'$ (the projection of **u** in the yz plane) through angle α onto the z axis.

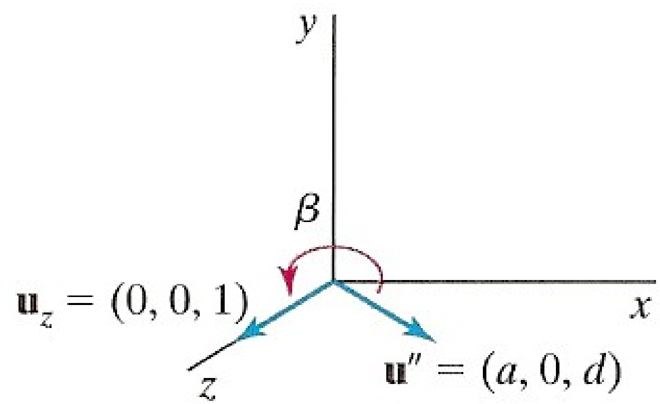
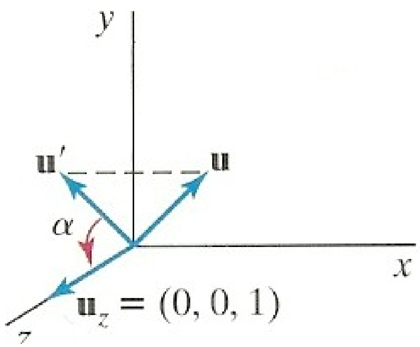


FIGURE 5-47 Rotation of unit vector $\mathbf{u}''$ (vector **u** after rotation into the xz plane) about the y axis. Positive rotation angle β aligns $\mathbf{u}''$ with vector $\mathbf{u}_z$.

3D Rotation About Arbitrary Axis

- Rotate **u'** about the x-axis onto the *z*-axis
  - Let **u**=(a,b,c) and thus **u'**=(0,b,c)
  - Let $\mathbf{u}_z$=(0,0,1)



$$\cos\alpha = \frac{\mathbf{u}'\cdot\mathbf{u}_z}{\|\mathbf{u}'\|\,\|\mathbf{u}_z\|} = \frac{c}{\sqrt{b^2+c^2}}$$

$$\mathbf{u}'\times\mathbf{u}_z = \mathbf{u}_x\|\mathbf{u}'\|\,\|\mathbf{u}_z\|\sin\alpha = \mathbf{u}_x\cdot b$$

$$\Rightarrow \quad \sin\alpha = \frac{b}{\|\mathbf{u}'\|\,\|\mathbf{u}_z\|} = \frac{b}{\sqrt{b^2+c^2}}$$

3D Rotation About Arbitrary Axis

- Rotate **u'** about the x-axis onto the *z*-axis
  - Since we know both cos α and sin α, the rotation matrix can be obtained

$$\mathbf{R}_x(\alpha) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{c}{\sqrt{b^2+c^2}} & \frac{-b}{\sqrt{b^2+c^2}} & 0 \\ 0 & \frac{b}{\sqrt{b^2+c^2}} & \frac{c}{\sqrt{b^2+c^2}} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

  - Or, we can compute the signed angle α

$$\mathrm{atan2}\left(\frac{c}{\sqrt{b^2+c^2}}, \frac{b}{\sqrt{b^2+c^2}}\right)$$

  - Do not use acos() since its domain is limited to [-1,1]

Euler angles

- Arbitrary orientation can be represented by three rotation along x,y,z axis

$$R_{XYZ}(\gamma,\beta,\alpha)=R_z(\alpha)R_y(\beta)R_x(\gamma)$$

$$=\begin{bmatrix} \mathrm{C}\alpha\mathrm{C}\beta & \mathrm{C}\alpha\mathrm{S}\beta\mathrm{S}\gamma - \mathrm{S}\alpha\mathrm{C}\gamma & C\alpha S\beta C\gamma + S\alpha S\gamma & 0 \\ S\alpha C\beta & S\alpha S\beta S\gamma + C\alpha C\gamma & S\alpha S\beta C\gamma - C\alpha S\gamma & 0 \\ -S\beta & C\beta S\gamma & C\beta C\gamma & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Gimble

- Hardware implementation of Euler angles
- Aircraft, Camera

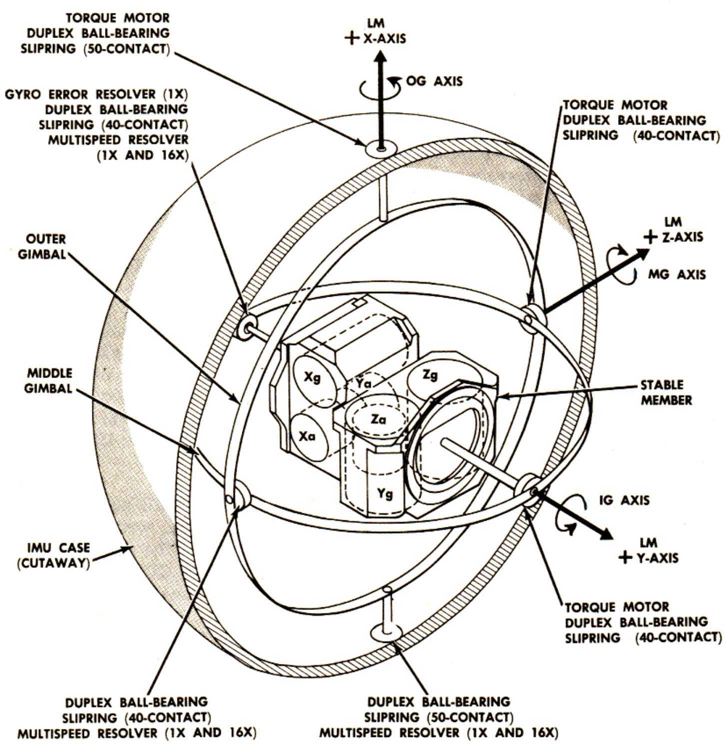
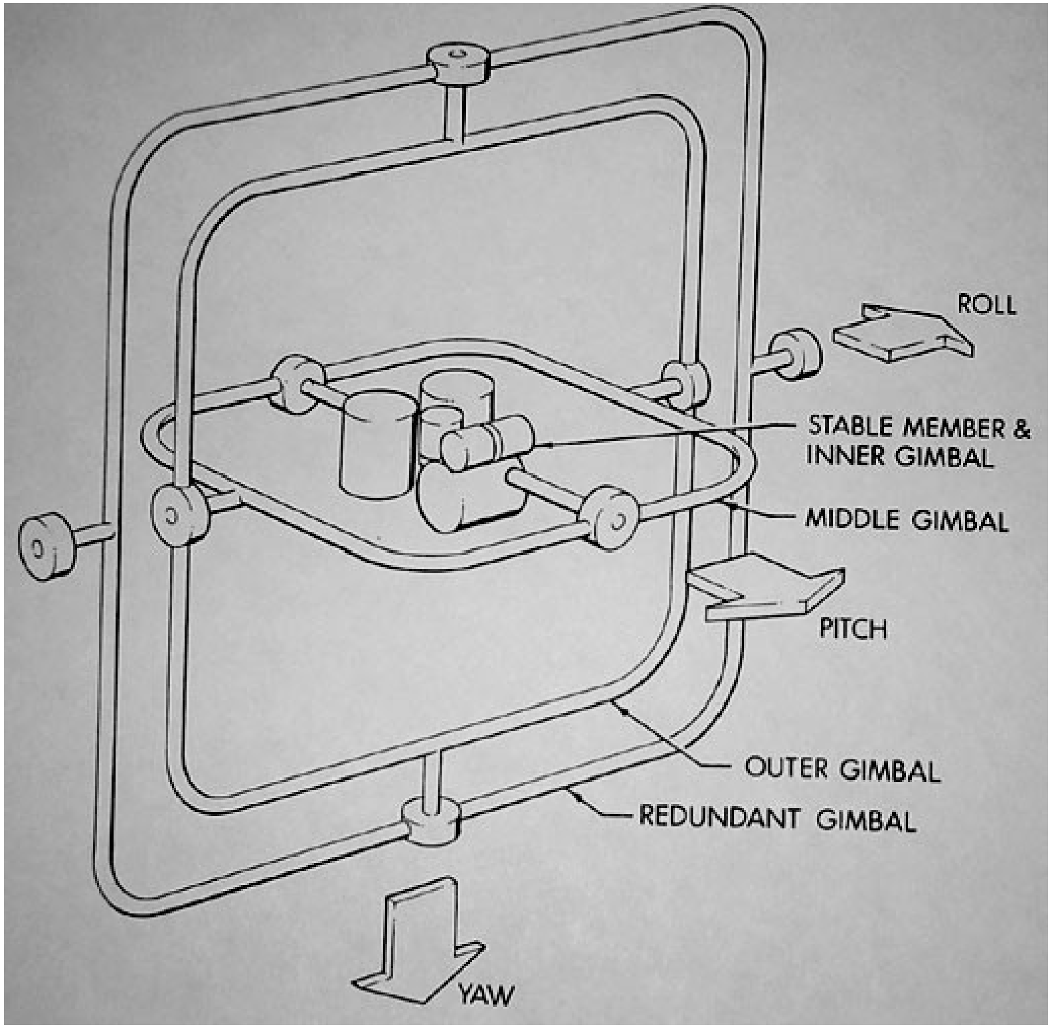


Figure 2.1-24. IMU Gimbal Assembly

Euler Angles

- Rotation about three orthogonal axes
  - 12 combinations
    - XYZ, XYX, XZY, XZX
    - YZX, YZY, YXZ, YXY
    - ZXY, ZXZ, ZYX, ZYZ
- ***Gimble lock***
  - Coincidence of inner most and outmost gimbles' rotation axes
  - Loss of degree of freedom



Euler Angles

- Euler angles are ambiguous
  - Two different Euler angles can represent the same orientation

$$R_1 = (r_x, r_y, r_z) = (\theta, \frac{\pi}{2}, 0) \quad \text{and} \quad R_2 = (0, \frac{\pi}{2}, -\theta)$$

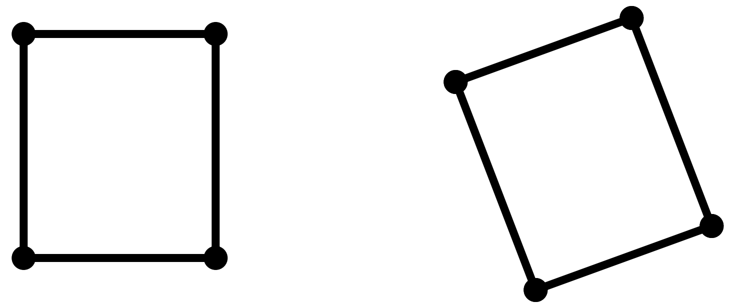
  - This ambiguity brings unexpected results of animation where frames are generated by interpolation.

Taxonomy of Transformations

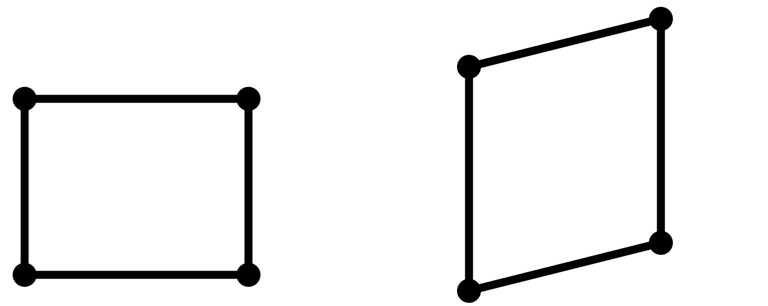
- **Linear** transformations
  - 3x3 matrix
  - Rotation + scaling + shear
- **Rigid** transformations
  - SO(3) for rotation
  - 3D vector for translation
- **Affine** transformation
  - 3x3 matrix + 3D vector or 4x4 homogenous matrix
  - Linear transformation + translation
- **Projective** transformation
  - 4x4 matrix
  - Affine transformation + perspective projection

Taxonomy of Transformations

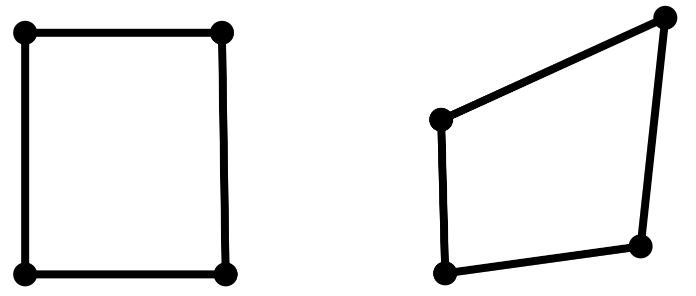
Rigid



Affine



Projective

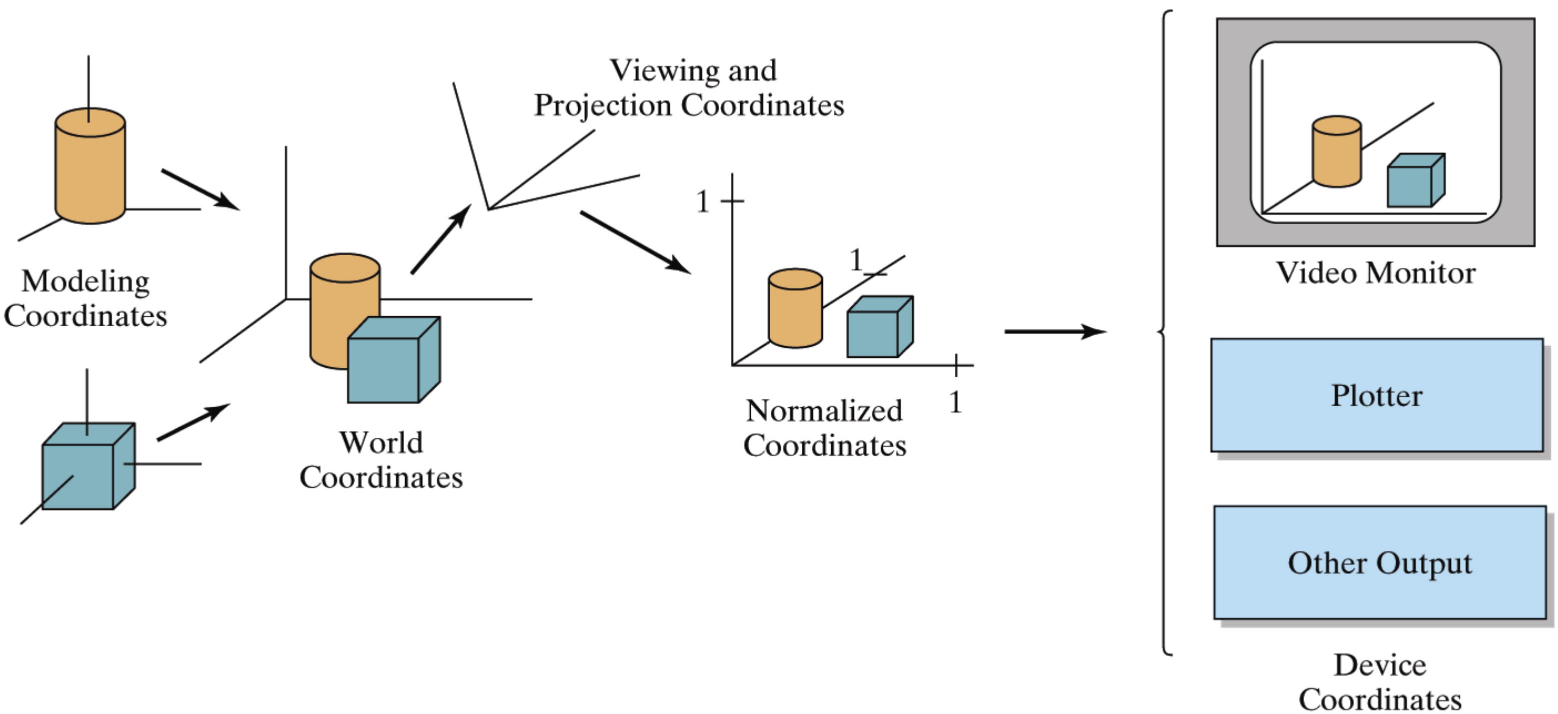


Composition of Transforms

- What is the composition of linear/affine/rigid transformations ?
- What is the linear (or affine) combination of linear (or affine) transformations ?
- What is the linear (or affine) combination of rigid transformations ?

OpenGL Geometric Transformations

- `glMatrixMode(GL_MODELVIEW);`

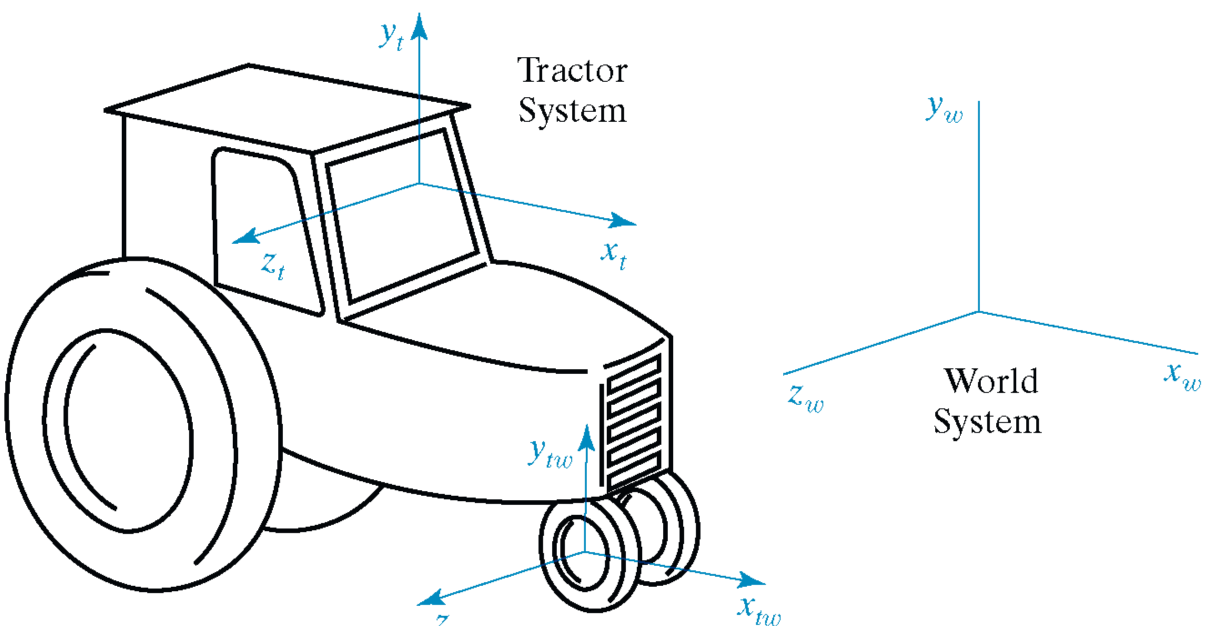


OpenGL Geometric Transformations

- Construction
  - `glLoadIdentity();`
  - `glTranslatef(tx, ty, tz);`
  - `glRotatef(theta, vx, vy, vz);`
    - `(vx, vy, vz)` is automatically normalized
  - `glScalef(sx, sy, sz);`
  - `glLoadMatrixf(Glfloat elems[16]);`
- Multiplication
  - `glMultMatrixf(Glfloat elems[16]);`
  - The current matrix is postmultiplied by the matrix
  - Row major

Hierarchical Modeling

- A hierarchical model is created by nesting the descriptions of subparts into one another to form a tree organization



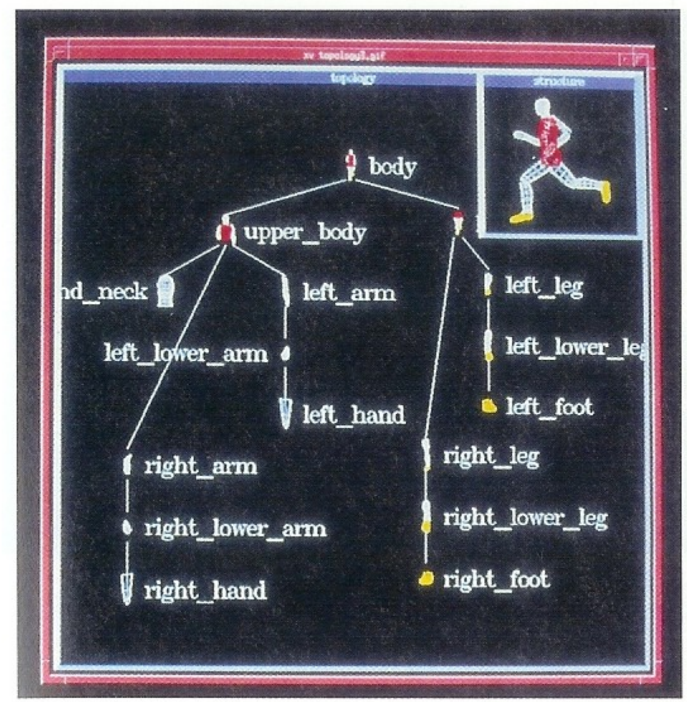


FIGURE 14-4 An object hierarchy generated using the PHIGS Toolkit package developed at the University of Manchester. The displayed object tree is itself a PHIGS structure. (*Courtesy of T. L. J. Howard, J. G. Williams, and W. T. Hewitt, Department of Computer Science, University of Manchester, United Kingdom.*)

OpenGL Matrix Stacks

- Four matrix modes
  - Modelview, projection, texture, and color
  - glGetIntegerv(GL_MAX_MODELVIEW_STACK_DEPTH, stackSize);
- Stack processing
  - The top of the stack is the "current" matrix
  - glPushMatrix(); // Duplicate the current matrix at the top
  - glPopMatrix(); // Remove the matrix at the top

Programming Assignment #1

- Create a hierarchical model using matrix stacks
- The model should consists of three-dimensional primitives such as polygons, boxes, cylinders, spheres and quadrics.
- The model should have a hierarchy of at least three levels
- Animate the model to show the hierarchical structure
  - Eg) a car driving with rotating wheels
  - Eg) a runner with arms and legs swing
- Make it aesthetically pleasing or technically illustrative